МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПУТЕЙ СООБЩЕНИЯ (МИИТ)

Кафедра «Прикладная математика-2»

В.Х. ХАХАНЯН

ЭЛЕМЕНТЫ ТЕОРИИ СЛОЖНОСТИ АЛГОРИТМОВ И ВЫЧИСЛЕНИЙ

Рекомендовано редакционно–издательским советом
Университета
в качестве учебного пособия для студентов
специальности АКБ

МОСКВА - 2010

УДК 510.633
510.635
510.557
Х 27

Хаханян В.Х. Элементы теории сложности алгоритмов и вычислений. Учебное пособие для студентов специальности АКБ. Издание второе, сокращённое и исправленное. - М.: МИИТ, 2010. – 146 с.

В пособии даётся изложение основных результатов по теории сложности алгоритмов и приводятся классификация класса разрешимых массовых проблем и примеры задач класса P, класса NP и класса NP-полных задач. Пособие рассчитано на студентов специальности АКБ и соответствует программе читаемого по данной специальности курса «Сложность алгоритмов и вычислений».

Рецензенты:
кандидат физико-математических наук, доцент кафедры математической логики и теории алгоритмов механико-математического факультета МГУ им. М.В.Ломоносова В.Н.Крупский и кандидат физико-математических наук, заведующий кафедрой «Вычислительная математика» (МИИТ) В.Н. Деснянский.

ПРЕДИСЛОВИЕ

Настоящее учебное пособие имеет целью обеспечение учебной литературой курса «Сложность алгоритмов», читаемого для студентов специальности АКБ в V семестре. Пособие состоит из предисловия, введения и 6-ти глав, в которые входят основные понятия теории сложности алгоритмов и вычислений. Необходимость написания данного пособия вызвана тем, что достаточно богатая литература по теории сложности содержит изучаемые сведения в очень разбросанном для курса виде (переводные учебники явно не учитывают специфики и часов читаемого курса, отечественные же учебники и пособия не согласуются как с часами, выделенными на изучение курса, так и по программе и их изложение рассчитано в первую очередь на университеты и пединституты). Всё это вызывает серьёзные трудности у студентов специальности АКБ при изучении данного курса, вызванные ещё и тем, что соответствующей учебной литературы по изучаемому курсу в библиотеке учебных пособий нет.

Кратко содержание пособия можно описать так: вначале идёт введение в специфику изучаемого предмета (Введение и Глава 1), затем определяются класс P задач, решаемых за полиномиальное время на детерминированной машине Тьюринга (Глава 2) , класс NP трудноразрешаемых задач (Глава 3), класс NP-полных задач (Глава 4), примеры NP-полных задач (Глава 5) и методы доказательства NP-полноты (Глава 6). В конце пособия приводится список литературы, использованной автором при написании данного пособия.

Необходимо отметить, что при написании пособия автор в сильной степени использовал замечательную переводную книгу (1), которая почти не устарела (из этой книги автор взял большие фрагменты цитат), пособия (4) и (5), написанные для факультета ВМК МГУ им. Ломоносова, а также книгу (3)

курса лекций, читаемых на механико-математическом факультете МГУ им. Ломоносова и препринт (2).

Содержание курса «Сложность алгоритмов» связано как с изучаемым в IV семестре курсом «Математическая логика и теория алгоритмов», так и с курсом по теории чисел, изучаемым в одном из последних семестров, который непосредственно выводит изучающего специальность студента на современный уровень криптографии (криптосистема без передачи ключей, криптосистема с открытым ключом, электронная подпись и затем экспоненциальный открытый ключ, дискретный логарифм; подробности см. в (6)), при овладении которым необходимо знать оценки сложности используемых алгоритмов.

Автор надеется, что данное пособие закроет пробел в студенческой литературе по курсу «Сложность алгоритмов» и будет полезно изучающим данный курс студентам.

Во втором издании были исправлено ошибки и опечатки, расширены рисунки и сокращен текст для более полного соответствия пособия часам читаемого курса.

ВВЕДЕНИЕ

Математика разрабатывает эффективные методы решения широких классов задач. Развитие теории и практики решения дискретных и комбинаторных задач показали, что общность метода и его эффективность находятся в известном противоречии. Важно знать, можно ли в принципе надеяться на создание достаточно общих и эффективных методов или нужно идти по пути разбиения задач на всё более узкие классы и для этих классов разрабатывать эффективные алгоритмы.

Неудачи исследователей на этом пути привели к необходимости анализа сложности задач. Возникло интенсивно развивающееся «сложностное» направление исследований. Число публикации в этой области быстро

растет и проблематика эта весьма актуальна, ибо такие задачи часто возникают на практике и их решение средствами ЭВМ наталкивается на трудности больших затрат машинного времени.

Все конечные дискретные задачи допускают решение с помощью процесса перебора, например, задачи коммивояжер или изоморфизм подграфу (см.ниже). Такие задачи называются переборными. Но число шагов переборного метода растет экспоненциально в зависимости от размеров задачи. Для ряда задач удаётся построить эффективные методы решения. Приведём несколько примеров.

1. Арифметика остатков. Операции сложение и умножение в кольцах вычетов по модулю m вычисляются за полиномиальное время с помощью **целочисленной арифметики** (для соответствующих доказательств см. (3)).

2. Связность в графе. В графе, заданном матрицей смежности, по заданным двум вершинам выяснить, связаны ли они. Нетрудно видеть, что задача сводится к вычислению степени n матрицы смежностей, а это вычисление имеет полиномиальную сложность. (для доказательства см. (3)).

3. Вычисление степени числа (см. (2)).

Алгоритм 1 Тривиальное вычисление $y = x^n \mod m$

Вход: Натуральные $n, x, m \geq 1$

Выход: $y = x^n \mod m$

$y \leftarrow 1$

for all $i \in 1..n$ **do**

$y \leftarrow y \times x \mod m$ {Деление по модулю m каждый раз "не вредит" конечному результату, т.к. кольцо Z_m замкнуто относительно умножения.}

end for

return y

Приведённый алгоритм слишком прямолинеен и его можно улучшить (сложность приведенного алгоритма есть

O(n)). Для этого достаточно рассмотреть двоичное разложение числа n:

$$n = \sum_{i=1}^{k} a_i 2^i, \text{ где } a_i \in \{0,1\},$$

и теперь заметить, что

$$x^n = x^{\sum_{i=1}^{k} a_i 2^i} = \prod_{i=1}^{k} x^{a_i 2^i} = \prod_{\{i:a_i>0\}} x^{2^i}$$

Итак, достаточно провести k ≤ (logn + 1) итераций по двоичному разложению n, чтобы на каждой i-й итерации, в зависимости от i-го бита в разложении, проводить умножение результата на сомножитель x^{2^i}, который легко получить из сомножителя предыдущей итерации (ключевое соотношение):

$$x^{2^i} \leftarrow \left(x^{2^{(i-1)}}\right)^2.$$

Анализ же этого алгоритма показывает сложность O(log n). Разница в сложности **экспоненциальна!** Самостоятельно приведите алгоритм для вычисления степени числа (см. (2)).

4. Аналогичная **задача для вычисления n!** имеет сложность O(n) и неизвестно, можно ли улучшить (и на сколько!) эту сложность (попробуйте самостоятельно привести соответствующий алгоритм, см. также (2)).

Следует отметить, что число таких задач, решаемых за полиномиальное «время», невелико. Как мы увидим далее, большинство встречающихся задач не удаётся решить с помощью полиномиальных алгоритмов. Приведем пример такой задачи

5. Дискретный логарифм. Пусть p – нечетное простое число и пусть для заданных a и b выполняется соотношение $a^x \equiv b \mod p$. Найти x.

Эта задача обратная к задаче возведения в степень по модулю. Можем ли мы опять ожидать легкого решения? Оказывается нет, вычисление дискретного алгоритма является очень сложной задачей. На этом важном свойстве (свойстве

односторонней вычислимости модульной экспоненты) основано множество современных алгоритмов ассиметричной криптографии (также см. (2) и (6)).

6. Умножение 0-1 матриц. Для заданных матриц A и B, состоящих из 0 и 1, нужно вычислить их произведение. Если в качестве операций разрешить только любые битовые, то существует алгоритм, перемножающий матрицы порядка n с числом таких операций $O(n^{\log 7}\log^2 n)$ (см., например, (4), стр.22-24, Теорема 3.1).

Анализ трудностей, встретившихся на пути создания эффективных методов решения дискретных задач, привел к постановке центральной теоретико-методологической проблемы - можно ли исключить перебор при решении дискретных задач? Речь идет о принципиальной возможности найти нужное решение, не перебирая всех или почти всех вариантов в задаче. Эта проблема имеет не только чисто математическое, но и глубокое познавательное значение. Оно заключается в том, что при поиске эффективных точных методов решения широкого класса дискретных задач надо учитывать возможность отсутствия таких методов признав, что существуют «труднорешаемые задачи». До настоящего времени эта проблема остается открытой.

Феномен труднорешаемых задач не нов для математики. После уточнения понятия алгоритма было обнаружено наличие алгоритмически неразрешимых проблем. В качестве известного примера укажем десятую проблему Гильберта: «Существует ли алгоритм, который по данному многочлену $p(x_1,\ldots,x_n)$ с целыми коэффициентами распознает, имеет ли уравнение $p = 0$ решение в целых числах». Было показано, что такого алгоритма не существует.

В переборных задачах имеется конечное множество вариантов, среди которых нужно найти решение. Но с ростом n число вариантов часто быстро растет и задача становится труднорешаемой, т. е. практически неразрешимой. Поэтому в конечной области аналогом алгоритмической неразрешимости

является перебор экспоненциального числа вариантов, а аналогом алгоритмической разрешимости существование алгоритма существенно более экономичного, чем перебор. Переборная задача решаема эффективно, если имеется алгоритм, решающий ее за время, ограниченное полиномом от «размера задачи».

Концепция эффективной (полиномиальной) сводимости переборных задач была развита в работах С. Кука, Р. Карпа, Л. Левина. Роль основного инструмента играет понятие сводимости задач, возникшее в математической логике. Переборная задача Π_1 сводится к переборной задаче Π_2, если метод решения задачи Π_2 можно преобразовать в метод решения задачи Π_1. Сводимость называется полиномиальной, если подобное преобразование можно осуществить за полиномиальное время. Главными объектами теории являются: класс NP всех переборных задач и класс P переборных задач, разрешимых за полиномиальное время на машине Тьюринга. Очевидно, что $P \subseteq NP$. Центральным в теории сводимости является вопрос о том, совпадают ли классы P и NP. Такова математическая формулировка проблемы элиминации перебора.

В классе NP выявлены универсальные или NP-полные задачи, к которым полиномиально сводится любая задача из NP. В этом смысле универсальные задачи дают эталон сложности. В настоящее время установлена универсальность многих задач, эквивалентных между собой относительно полиномиальной сводимости. Если удастся доказать, что хотя бы какая-то одна NP-полная задача принадлежит классу P, то тем самым было бы доказано, что $P = NP$ и можно в принципе надеяться на построение эффективных алгоритмов для различных классов дискретных задач. Если же классы P и NP различны, то придется разрабатывать эффективные алгоритмы для все более узких классов задач. Возможна и такая ситуация, что гипотезу $P \neq NP$ (принимаемую многими

математиками) нельзя ни доказать, ни опровергнуть (аналогично континуум - гипотезе).

Большой практический опыт решения дискретных задач дает основание считать, что NP-полные задачи и задачи из P сильно отличаются по трудоемкости решения, но в строгом смысле до сих пор это различие и, следовательно, различие между классами P и NP, не доказано. Это объясняется тем, что классы P и NP определяются с помощью понятия времени работы вычислительного устройства с потенциально неограниченной памятью. Однако хорошо известно, что время выполнения алгоритма на машине плохо поддается описанию и анализу общематематическими средствами.

Излагаемая ниже и ставшая уже классической теория полиномиальной сводимости была построена для задач распознавания свойств. Но можно построить аналогичную теорию (полиномиальной) сводимости для экстремальных дискретных задач. Более того, при построении указанной теории можно не пользоваться моделью вычислительного устройства.

Как уже отмечалось, настоящие пособие написано на основе ряда переводных и отечественных книг и использует приводимые там результаты, давая в ряде случаев подробные доказательства, которые при первом чтении можно опустить и рассчитано в первую очередь на слушателя-новичка в изучении теории сложности алгоритмов и вычислений.

ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ СЛОЖНОСТИ

Найдется немного научных терминов, так быстро завоевавших широкую известность, как понятие «NP-полная задача». Возникшее в начале 70-х годов, оно стало символом громадных трудностей, с которыми все чаще приходится сталкиваться разработчикам алгоритмов по мере того, как они берутся за решение задач постоянно возрастающей

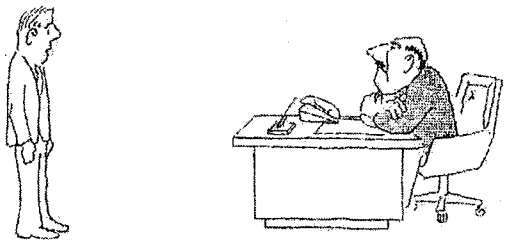
размерности и усложняющейся структуры. Разнообразные задачи, встречающиеся в математике, теоретическом программировании и исследовании операций оказались NP-полными и список таких задач быстро пополняется. NP-полные задачи широко распространены. Для всех, кто в своей работе соприкасается с вычислительными аспектами, очень важно понимать смысл и значение этого понятия.

Начинающие изучать теорию сложности алгоритмов могут достичь такого понимания, если бегло прочтут вводную часть данного пособия, концентрируя внимание на неформальных обсуждениях определений и методов и обращаясь к формальным рассуждениям тогда, когда это необходимо для полной ясности. Сначала давайте разберём один пример (см. (1)).

Представьте себе, что вы занимаетесь исследовательской работой в одной из отраслей промышленности. Однажды шеф вызывает вас к себе в кабинет и доверительно сообщает, что компания собирается приступить к производству и продаже «машин» в условиях жесткой конкуренции. По этой причине нужен хороший метод, выявляющий возможность (или невозможность) создания новых изделий: узлов машины с заданными техническими характеристиками, причем этот метод должен выдавать ответ для любого заданного набора технических характеристик. В случае положительного ответа требуется представить проект соответствующей машины. Так как вы отвечаете за разработку алгоритмов в фирме, то вам и предстоит отыскать требуемый метод и построить соответствующий эффективный алгоритм.

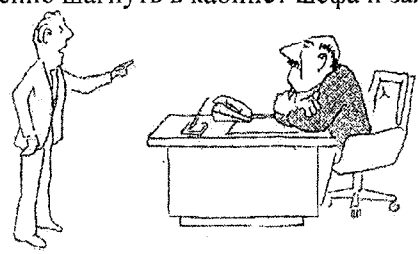
Получив в Отделе машин необходимые разъяснения по постановке задачи, вы спешите в свой кабинет, запасаетесь справочниками и с большим энтузиазмом погружаетесь в решение задачи. Через несколько недель кабинет завален грудами скомканных черновиков, а ваш энтузиазм заметно поубавился. Вам все еще не удалось придумать алгоритма, существенно лучшего, чем перебор всех возможных

вариантов. За подобное решение вы едва ли удостоитесь похвалы шефа, поскольку перебор всех вариантов только для одного набора технических характеристик потребует нескольких лет машинного времени, а в Отделе машин уже по тринадцати узлам машины наблюдается отставание от намеченного графика. Естественно, что у вас нет желания возвращаться в кабинет шефа со словами:



«Я не могу найти эффективного алгоритма, боюсь, что я для этого слишком туп».

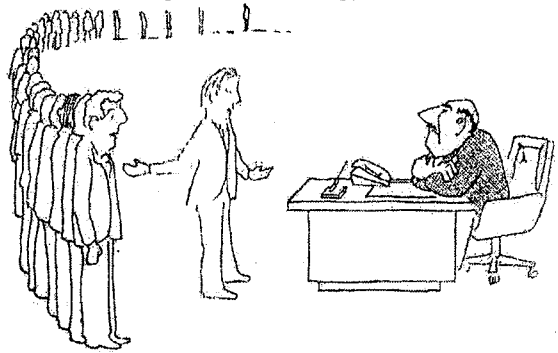
Для вашего положения в компании было бы гораздо лучше, если бы вы смогли доказать, что поставленная задача в принципе трудно решаема и что ни один алгоритм вообще не в состоянии решить ее быстро. В этом случае вы могли бы уверенно шагнуть в кабинет шефа и заявить:



«Я не могу найти эффективного алгоритма, потому что такого алгоритма не существует!»

К сожалению, доказать, что задача труднорешаема, может быть не менее трудно, чем найти эффективные алгоритмы. Даже выдающиеся теоретики оказывались беспомощными в

попытках получить подобное доказательство для трудных задач. Однако достойный выход из положения есть, т.к. теория NP-полных задач предлагает много простых методов доказательства того, что та или иная конкретная задача столь же трудна, как и большое число других задач, признанных очень трудными и уже много лет не поддающихся усилиям специалистов. Вы могли бы доказать NP-полноту задачи о машине и таким образом установить её эквивалентность всем другим трудным задачам этого класса. В этом случае можно было бы смело отправиться к шефу и сообщить:



«Я не могу найти эффективного алгоритма, но этого не может сделать никто из этих знаменитостей».

Ваше начальство будет знать, что нет смысла увольнять вас и брать на работу другого специалиста по алгоритмам.

Любое начальство вряд ли одобрило бы написание этого пособия, если бы единственной его целью было защитить разработчиков алгоритмов. На самом деле выявление NP-полноты лишь начало работы над задачей. Проблемы отдела машин не исчезнут только потому, что поставленная задача оказалась NP-полной. Знание этого факта действительно дает ценную информацию о том, какие подходы к ее решению окажутся наиболее разумными.

В этом случае не следует делать ставку на поиски эффективного алгоритма. Лучше сосредоточить свое

внимание на иных, более скромных подходах. Можно бы начать разработку эффективных алгоритмов, позволяющих решать различные частные случаи поставленной общей задачи. Можно заняться отысканием алгоритмов, хотя и не гарантирующих быстрого решения, однако работающих быстро в большинстве случаев. Можно даже ослабить постановку задачи и искать быстрый алгоритм проектирования машин, у которого заданным требованиям удовлетворяет большая часть характеристик. Короче говоря, основное предназначение теории NP-полных задач в том, чтобы помочь разработчикам алгоритмов и направить их усилия на выбор таких подходов к решению задач, которые приведут к практически полезным алгоритмам. Введем теперь основные понятия теории сложности и рассмотрим их применимость.

ЗАДАЧИ, АЛГОРИТМЫ И СЛОЖНОСТЬ

Чтобы в дальнейшем изучать такие понятия, как «труднорешаемые задачи» и «эквивалентные по сложности задачи», необходимо договориться о значении нескольких основных терминов, не давая формальных определений.

Начнем с понятия задачи. *Массовой задачей* (или просто *задачей*) называется некоторый общий вопрос, на который следует дать ответ. Обычно задача содержит ряд *параметров*, или свободных переменных, конкретные значения которых не определены. Задача П определяется следующей информацией:
(1) общим списком всех ее параметров и
(2) заданием тех свойств, которым должно удовлетворять решение задачи.

Индивидуальная задача I получается из массовой задачи П, если всем параметрам задачи П присвоить конкретные значения.

В качестве примера рассмотрим классическую задачу о коммивояжере. Параметры этой задачи состоят из конечного

набора «городов» $C=\{c_1,c_2,\ldots,c_m\}$ и «расстояний» $d(c_i,c_j)$ между парами c_i,c_j из С. Решением является упорядоченный набор $<c_{\pi(1)},c_{\pi(2)},\ldots,c_{\pi(m)}>$ заданных городов, который минимизирует величину $\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)})$. Последнее есть длина маршрута, начинающегося в городе $c_{\pi(1)}$, проходящего последовательно через все города и возвращающегося в $c_{\pi(1)}$ непосредственно из последнего города $c_{\pi(m)}$. Индивидуальная задача о коммивояжере, показанная на рисунке 1.1 ниже, задается следующим образом:

$C=\{c_1, c_2, c_3, c_4\}$, $d(c_1, c_3)=10$, $d(c_1, c_3)=5$,

$d=(c_1, c_4)=9$, $d(c_2, c_3)=6$, $d(c_2, c_4)=9$, $d(c_3, c_4)=3$.

Последовательность *<c1, c2, c4, c3>* представляет собой решение задачи, поскольку соответствующий маршрут имеет минимальную возможную длину, равную 27.

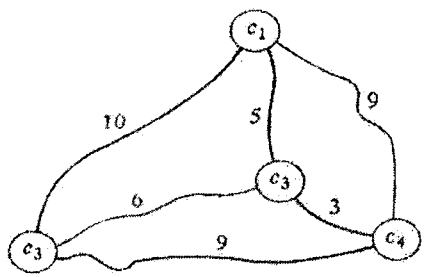


Рис. 1.1. Индивидуальная задача о коммивояжере и маршрут минимальной возможной длины, равной 27.

Под *алгоритмом* будем понимать общую, выполняемую шаг за шагом процедуру решения задачи (можно считать ее программой для ЭВМ, написанной на формальном машинном языке). Будем говорить, что алгоритм *решает* массовую задачу П, если он применим к любой индивидуальной задаче I из П и обязательно дает решение задачи I. Термин «решение»

понимается здесь строго в соответствии с данным выше определением. Нельзя сказать, например, что алгоритм решает задачу о коммивояжере, если он не выдаст маршрут минимальной длины хотя бы для одной индивидуальной задачи.

Вообще говоря, нужен наиболее эффективный алгоритм решения задачи. В широком смысле понятие эффективности связано с вычислительными ресурсами, необходимыми для работы алгоритма. Однако обычно под *самым эффективным* алгоритмом понимается самый быстрый. Поскольку ограничения по времени часто являются доминирующим фактором, определяющим пригодность данного алгоритма, основное внимание будет сосредоточено на этом виде ресурсов.

Время работы алгоритма удобно выражать в виде функции от одной переменной, характеризующей *размер* индивидуальной задачи, т. е. объем входных данных, требуемых для описания задачи. Такой подход удобен, т.к. в дальнейшем сравнительная сложность задач будет оцениваться через их размеры. Часто размер задачи измеряется неформально. В задаче о коммивояжере для этой цели обычно используется число городов. Однако в задаче с m городами кроме номеров этих городов на объем входной информации влияют также $m(m—1)/2$ величин, определяющих расстояния между городами. Если нам предстоит иметь дело с временными характеристиками в точной математической постановке, то мы должны так определить размер задачи, чтобы все эти факторы были учтены.

Для этого обратим внимание на то, что описание индивидуальной задачи, которое мы даем в терминах входа, можно рассматривать как конечную цепочку (слово) символов, выбранных из конечного входного алфавита. Существуют различные пути описания данной индивидуальной задачи. Предположим, что заранее выбран некоторый определенный способ и что с каждой массовой

задачей связана некоторая фиксированная *схема кодирования*, которая отображает индивидуальные задачи в соответствующие цепочки символов. Входная длина индивидуальной задачи I из П определяется как число символов в цепочке, полученной применением к задаче I схемы кодирования для массовой задачи П. Именно это число, т. е. входная длина, и используется в качестве формальной характеристики размера индивидуальной задачи.

Например, различные конкретные задачи о коммивояжере можно описать с помощью алфавита {c, [,], /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, при этом предыдущий пример будет закодирован в виде такой цепочки символов: c[1]c[2]c[3]c[4]//10/5/9//6/9//3. Кодирование индивидуальных задач аналогично. Данная кодирующая схема для задачи о коммивояжере дает входную длину 32.

Временная сложность алгоритма отражает требующиеся для его работы затраты времени. Это функция, которая каждой входной длине n ставит в соответствие максимальное (по всем индивидуальным задачам данной длины) время, затрачиваемое алгоритмом на решение индивидуальных задач этой длины.

Эта функция не будет полностью определена до тех пор, пока не зафиксирована схема кодирования, определяющая входную длину индивидуальной задачи и не выбрано вычислительное устройство (или его модель), определяющее время работы. Однако, как будет видно из дальнейшего, подобные детали окажут незначительное влияние на различия между классами, возникающими в теории NP-полных задач. Поэтому в дальнейшем читателю рекомендуется зафиксировать мысленно какую-либо конкретную схему кодирования для каждой задачи, выбрать некоторое конкретное вычислительное устройство или его модель и рассматривать затем временную сложность алгоритмов в

соответствии с получающимися входными длинами и соответствующими затратами времени.

ПОЛИНОМИАЛЬНЫЕ АЛГОРИТМЫ И ТРУДНОРЕШАЕМЫЕ ЗАДАЧИ

Разные алгоритмы имеют различную временную сложность и выяснение того, какие алгоритмы достаточно эффективны, а какие совершенно неэффективны, всегда будет зависеть от конкретной ситуации. Однако теоретики, занимающиеся разработкой и анализом алгоритмов, предлагают для сравнения эффективности алгоритмов один простой подход, позволяющий существенно прояснить ситуацию. Речь идет о различии между полиномиальными и экспоненциальными алгоритмами.

Будем говорить, что функция $f(n)$ есть $O(g(n))$, если существует константа c, такая, что $|f(n)| \leq c|g(n)|$ для всех значений $n \geq 0$. Полиномиальным алгоритмом (или алгоритмом полиномиальной временной сложности) называется алгоритм, у которого временная сложность равна $O(p(n))$, где $p(n)$ - полином, а n - входная длина. Те алгоритмы, временная сложность которых не поддается подобной оценке, называются экспоненциальными (это определение включает и такие функции, как $n^{\log n}$, которые хотя и не являются полиномиальными, но и не считаются экспоненциальными).

Различие между двумя указанными типами алгоритмов становится особенно заметным при решении задач большого размера. Таблица на рис. 1.2 позволяет сравнить скорости роста нескольких типичных полиномиальных и экспоненциальных функций (обратите внимание на чрезвычайно быстрый рост двух приведённых экспонент).

Функция временной сложности	Размер n					
	10	20	30	40	50	60
n	0,00001 сек	0,00002 сек	0,00003 сек	0,00004 сек	0,00005 сек	0,00006 сек
n^2	0,0001 сек	0,0004 сек	0,0009 сек	0,0016 сек	0,0025 сек	0,0036 сек
n^3	0,001 сек	0,008 сек	0,027 сек	0,064 сек	0,125 сек	0,216 сек
n^5	0,1 сек	3,2 сек	24,3 сек	1,7 мин	5,2 мин	13,0 мин
2^n	0,001 сек	1,0 сек	17,9 мин	12,7 дней	35,7 лет	366 столетий
3^n	0,059 сек	58 мин	6,5 лет	3855 столетий	2×10^8 столетий	$1{,}3\times10^{13}$ столетий

Рис. 1.2. Сравнение нескольких полиномиальных и экспоненциальных функций временной сложности.

Различие между полиномиальными и экспоненциальными алгоритмами проявляется еще более убедительно, если проанализировать влияние увеличения быстродействия ЭВМ на время работы алгоритмов. Рис. 1.3 показывает, насколько увеличатся размеры задач, решаемых за один час машинного времени, если, благодаря совершенствованию технологии, быстродействие ЭВМ возрастет в 100 или 1000 раз по сравнению с современными (на 1980 год) машинами. Заметим, что для функции $f(n) = 2^n$ увеличение скорости вычислений в 1000 раз приводит лишь к тому, что размер наибольшей задачи, разрешимой за один час, возрастет только на 10, в то время как для функции $f(n) = n^5$ этот размер возрастет почти в 4 раза.

Размеры наибольшей задачи, разрешимой за один час

Функция временной сложности	*На современных ЭВМ*	*На ЭВМ, в 100 раз более быстрых*	*На ЭВМ, в 1000 раз более быстрых*
n	N_1	$100\ N_1$	$1000\ N_1$
n^2	N_2	$10\ N_2$	$31{,}6\ N_2$
n^3	N_3	$4{,}64\ N_3$	$10\ N_3$
n^5	N_4	$2{,}5\ N_4$	$3{,}98\ N_4$
2^n	N_5	$N_5+6{,}64$	$N_5+9{,}97$
3^n	N_6	$N_6+4{,}19$	$N_6+6{,}29$

Рис. 1.3. Влияние технического совершенствования ЭВМ на полиномиальные и экспоненциальные алгоритмы.

Эти таблицы демонстрируют тот факт, что полиномиальные алгоритмы обычно считаются более предпочтительными по сравнению с экспоненциальными. Такая точка зрения, различающая, с одной стороны, полиномиальные алгоритмы, а с другой стороны, экспоненциальные, и будет являться отправным пунктом в нашем определении *труднорешаемых* задач и теории NP-полных задач.

Большинство экспоненциальных алгоритмов есть просто варианты полного перебора, в то время как полиномиальные алгоритмы обычно можно построить лишь тогда, когда удается более глубоко проникнуть в суть решаемой задачи (см. пример **3** выше). Имеется широко распространенное соглашение, согласно которому задача не считается хорошо решаемой до тех пор, пока для нее не получен полиномиальный алгоритм. Поэтому мы будем называть задачу *труднорешаемой*, если для ее решения не существует полиномиального алгоритма.

Конечно, это формальное определение следует рассматривать только как одну из возможных трактовок понятия *труднорешаемая задача.* Различие между эффективными (полиномиальными) алгоритмами и неэффективными (экспоненциальными) алгоритмами может принять иной характер, когда размеры решаемых задач невелики. Функция $f(n)=2^n$ ведет себя лучше, чем $f(n) = n^5$ при $n \leq 20$ по отношению к размерам наибольшей задачи, разрешимой за 1 час. Известны некоторые экспоненциальные алгоритмы, хорошо зарекомендовавшие себя на практике. Дело в том, что временная сложность определена нами как мера поведения алгоритма в наихудшем случае, и тот факт, что какой-то алгоритм имеет временную сложность порядка 2^n, означает только, что решение по крайней мере одной индивидуальной задачи размера n требует времени порядка 2^n. Может оказаться, что большинство индивидуальных задач требует для своего решения значительно меньших затрат времени, и такого рода ситуация имеет место для нескольких хорошо известных алгоритмов. Было установлено, что симплекс-метод для решения задач линейного программирования имеет экспоненциальную временную сложность, но в то же время многочисленные свидетельства подтверждают, что этот метод хорошо работает на практике. Другой пример: алгоритмы ветвей и границ столь успешно решают задачу о рюкзаке, что многие исследователи считают эту задачу хорошо решаемой, хотя эти алгоритмы имеют экспоненциальную временную сложность.

Но подобные примеры очень редки. Хотя экспоненциальные алгоритмы известны для многих задач, немногие из них считаются приемлемыми для практических целей. Однако исследователи не отказались от попыток найти для соответствующих задач полиномиальные алгоритмы. Сам факт успешного применения экспоненциальных алгоритмов давал основание предположить, что более глубокое их исследование может привести к дальнейшему улучшению

методов. В настоящее время не получено удовлетворительных объяснений, почему эти алгоритмы работают успешно, и не известны методы, позволяющие заранее прогнозировать хорошую работу того или иного экспоненциального алгоритма в практической ситуации (возможно, что данная ситуация уже исследована более подробно).

С другой стороны, полиномиальные алгоритмы часто позволяют делать такого рода прогнозы, поскольку полиномиальные функции значительно более адекватно оценивают время работы алгоритмов. Хотя алгоритмы, имеющие временную сложность типа n^{100} или $10^{99}n^2$, не могут считаться эффективными с практической точки зрения, естественно возникающие полиномиальные задачи обычно требуют для своего решения времени порядка n^2 или n^3, причем коэффициенты при старших членах полиномов не слишком велики. Такие алгоритмы можно считать эффективными, и именно это свойство заставляет отдавать предпочтение полиномиальным алгоритмам как средству решения задач.

Определение трудно решаемой задачи создает базу для теории, обладающей значительной общностью и большими возможностями. Понятие *труднорешаемой* задачи оказывается по существу независимым от конкретной схемы кодирования и модели ЭВМ, используемых при определении временной сложности.

Схема кодирования	*Цепочки символов (слова)*	*Длина*
Списки вершин и ребер	V[1]V[2]V[3]V[4](V[1]V[2])(V[2]V[3])	36
Список соседей	(V[2])(V[1]V[3])(V[2])( )	24
Строки матрицы инцидений	0100/1010/0010/0000	19

Рис. 1.4. Описания графа $G = (V, E)$, где $V = \{v_1, v_2, v_3, v_4\}$, $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$, при трех разных схемах кодирования.

Рассмотрим сначала схемы кодирования. Предположим, что решаемая задача определена на графе $G = (V, E)$, где V-множество вершин, E - множество ребер и каждое ребро понимается как неупорядоченная пара вершин. Условия этой задачи могут быть описаны (см. рис. 1.4) либо просто списками всех вершин и ребер, либо с помощью матрицы инциденций графа, либо составлением для каждого узла списка всех узлов, имеющих с данным общее ребро (списки «соседей»). Эти схемы кодирования для одного и того же графа могут дать входы разной длины. Однако легко проверить (см. рис. 1.5), что получаемые входы отличаются друг от друга не более чем полиномиальным образом

Схема кодирования	*Нижняя оценка*	*Верхняя оценка*
Списки вершин и ребер	$4v + 10e$	$4v + 10e + (v + 2e)\cdot\lceil\log_{10} v\rceil$
Список соседей	$2v + 8e$	$2v + 8e + 2e\cdot\lceil\log_{10} v\rceil$
Матрица инциденций	$v^2 + v - 1$	$v^2 + v - 1$

Рис. 1.5. Общие оценки длины входа для трех схем кодирования графа $G = (V, E)$, представленных на рис. 1.4, где $|V| = v$, $|E| = e$. Поскольку $e < v^2$, эти оценки показывают, что длины входа отличаются друг от друга не более чем полиномиальным образом ($\lceil x\rceil$ обозначает наименьшее целое число, не меньшее, чем x).

(проверьте!), т.е. любой алгоритм, имеющий полиномиальную временную сложность при одной из этих схем кодирования, будет также обладать полиномиальной временной сложностью при всех остальных схемах.

Стандартные схемы кодирования, используемые на практике для любой конкретной задачи, по-видимому, всегда будут отличаться друг от друга не более чем полиномиальным

образом. Трудно представить себе разумную схему кодирования какой-либо задачи, которая отличалась бы более чем полиномиальным образом от стандартных схем. Хотя мы не можем формально выразить такое понятие, как «разумная схема кодирования», следующие два условия охватывают важные требования, связанные с этим понятием:
(1) код любой индивидуальной задачи I не должен содержать избыточной информации или символов;
(2) числа, входящие в условия задачи I, должны быть представлены в двоичной системе счисления (или десятичной, или восьмеричной, или иметь любое другое основание, но только не 1).

Если мы ограничимся рассмотрением тех схем кодирования, которые удовлетворяют (1) и (2), то выяснение вопроса, является ли данная задача трудно решаемой, не будет зависеть от выбора конкретной схемы кодирования.

Моделируемая машина B	Моделирующая машина A		
	1МТ	kМТ	МПД
Машина Тьюринга с 1 лентой (1МТ)	–	$O(T(n))$	$O(T(n)\log T(n))$
Машина Тьюринга с k лентами (kМТ)	$O(T^2(n))$	–	$O(T(n)\log T(n))$
Машина произвольного доступа (МПД)	$O(T^3(n))$	$O(T^2(n))$	–

Рис. 1.6 Время, требуемое машиной A для моделирования работы алгоритма сложности $T(n)$ на машине B.

Аналогичные замечания можно сделать относительно выбора модели ЭВМ. Все известные в настоящее время реалистические модели ЭВМ (например, одноленточные и многоленточные машины Тьюринга и др.) эквивалентны относительно полиномиальной временной сложности.

Вероятно, и любая иная «разумная» модель ЭВМ будет эквивалентна по сложности всем перечисленным моделям (см. рис. 1.6).

Говоря «разумная модель» мы имеем в виду, что объем работы, выполняемой машиной в единицу времени, ограничен сверху полиномом. Например, модель со способностью выполнять параллельно произвольно много операций, не будет считаться «разумной». Если ограничиться рассмотрением стандартных моделей реальных ЭВМ, то класс труднорешаемых задач не будет зависеть от выбора конкретной модели.

ЗАДАЧИ, ТРУДНОРЕШАЕМОСТЬ КОТОРЫХ ДОКАЗУЕМА

После обсуждения формального содержания понятия «труднорешаемая задача», дадим краткий обзор наших знаний о существовании труднорешаемых задач.

Имеется два аспекта труднорешаемости, которые отражены в определении. Первый аспект состоит в том, что для отыскания решения требуется экспоненциальное время. Второй заключается в том, что искомое решение настолько велико, что не может быть представлено в виде выражения, длина которого ограничена полиномом от длины входа.

Вторая ситуация возникает, например, если в задаче о коммивояжере в качестве дополнительного параметра фигурирует число B и требуется найти все маршруты длины, не превосходящей B. Легко построить индивидуальную задачу о коммивояжере, в которой имеется экспоненциальное число маршрутов длины, не превосходящей B, поэтому не существует алгоритма с полиномиальной временной сложностью, который все их перечисляет.

Труднорешаемостью этого вида нельзя пренебрегать и очень важно ее обнаружить. В большинстве случаев, однако, ее наличие видно из постановки задачи. Этот аспект

труднорешаемости рассматривается как указание на то, что постановка задачи не реалистична, поскольку мы запрашиваем больше информации, чем сможем использовать. В связи с этим мы будем рассматривать только первый тип труднорешаемости. Будут изучаться только такие задачи, длина решения которых ограничена полиномиальной функцией от длины входа.

Первые результаты о труднорешаемости задач (классические результаты о неразрешимости) были получены А. Тьюрингом. Тьюринг доказал, что некоторые задачи неразрешимы в том смысле, что вообще не существует алгоритма их решения. Он доказал, что невозможно указать алгоритм, который по произвольной программе определял бы, остановится ли эта программа на произвольно заданном входе или нет. Неразрешимые задачи не могут быть решены никаким алгоритмом и труднорешаемы в самом сильном смысле.

Первые примеры труднорешаемых разрешимых задач были получены в начале 60-х годов. Однако эти примеры включали только искусственные задачи, специально построенные, чтобы обладать соответствующими свойствами. Только в начале 70-х годов удалось показать, что некоторые естественные разрешимые задачи труднорешаемы.

В число этих задач вошло большое число изучавшихся ранее задач из теории автоматов, формальной теории языков и математической логики. Было показано, что эти задачи не могут быть решены за полиномиальное время даже с помощью «недетерминированного» вычислительного устройства, выполняющего параллельно неограниченное количество независимых вычислений. В дальнейшем мы увидим, что эта модель вычислительного устройства играет важную роль в теории NP-полных задач.

Все известные в настоящее время задачи, труднорешаемость которых доказана, попадают в один из классов: они либо вовсе неразрешимы, либо труднорешаемы

недетерминированной машиной. Однако большинство представляющихся труднорешаемыми практических задач в действительности разрешимы и, более того, могут быть решены за полиномиальное время с помощью недетерминированного вычислительного устройства. Известные методы недостаточно сильны, чтобы установить труднорешаемость этих задач.

NP-ПОЛНЫЕ ЗАДАЧИ

Пока продолжается поиск более мощных методов доказательства труднорешаемости задач, параллельно ведутся работы по сравнению сложности различных задач. Как уже было сказано, обнаружение таких взаимосвязей между задачами часто может дать разработчику алгоритмов полезную информацию.

Основной метод, используемый для доказательства того, что две задачи близки, состоит в сведении их друг к другу с помощью конструктивного преобразования, которое отображает любую индивидуальную задачу первого типа в эквивалентную ей индивидуальную задачу второго типа. Такое преобразование позволяет превратить любой алгоритм решения второй задачи о соответствующий алгоритм решения первой задачи.

Много примеров подобной сводимости известно уже давно, Например, Эдмондс свел задачи из теории графов (найти минимальное вершинное покрытие графа и найти минимальное независимое множество вершин графа) к задаче о покрытии. Было дано описание хорошо известной сводимости задачи о коммивояжере к задаче о кратчайшем пути, в которой допускаются ребра отрицательной длины.

Эти первые сводимости хотя и были изолированы и относились к узкому кругу задач, предвосхитили дух результатов, которые доказываются в теории NP-полных задач.

Фундамент теории NP-полных задач был заложен в работе С. Кука, опубликованной в 1971 г. под названием «Сложность процедур вывода теорем». В этой короткой, но элегантной работе Кук получил несколько важных результатов.

Во-первых, он подчеркнул важность понятия «сводимость за полиномиальное время», т. е. сводимость, которая выполняется с помощью алгоритма с полиномиальной временной сложностью (если одна задача сводится за полиномиальное время к другой, то любой полиномиальный алгоритм решения второй задачи может быть превращен в полиномиальный алгоритм решения первой).

Во-вторых, он обратил внимание на класс задач распознавания свойств (класс NP), которые могут быть решены за полиномиальное время на недетерминированном вычислительном устройстве (задачей распознавания свойств называется задача, решениями которой могут быть ответы либо «да», либо «нет»). Большинство не поддающихся решению задач, которые встречаются на практике, после переформулировки их в виде задач распознавания попадают в этот класс.

В-третьих, С. Кук доказал, что одна конкретная задача из NP, называемая задачей о выполнимости, обладает тем свойством, :что всякая другая задача из класса NP может быть сведена к ней за полиномиальное время (такие задачи называются универсальными). Если задача о выполнимости может быть решена за полиномиальное время, то и любая задача из класса NP полиномиально разрешима, а если какая-то задача из NP труднорешаема, то и задача о выполнимости также должна быть труднорешаемой. Таким образом задача о выполнимости - самая трудная в классе NP.

Наконец, С. Кук предположил, что и другие задачи из класса NP могут быть, как и задача о выполнимости, самыми трудными представителями класса NP. Он показал это для

следующей задачи «содержит ли заданный граф G полный подграф с заданным числом вершин?»

Вслед за этим был доказано, что многие хорошо известные комбинаторные задачи, включая задачу о коммивояжере, сформулированные в виде задачи распознавания, столь же трудны, как задача о выполнимости. Касс эквивалентности, состоящий из самых трудных задач из NP, получил название «класс NP-полных задач» (т.н. универсальный класс).

Оригинальные идеи Кука позволили свести много разнообразных вопросов о сложности в единый вопрос: «Верно ли, что NP-полные задачи труднорешаемы?» В настоящем пособии приводится много различных задач, NP-полнота которых уже установлена.

Вопрос о труднорешаемости считается одним из основных открытых вопросов современной математики и теоретической кибернетики. Вопреки предположению, что все NP-полные задачи труднорешаемы, прогресс как в доказательстве, так и в опровержении этого далеко идущего предположения весьма незначителен. Несмотря на отсутствие доказательства того, что из NP-полноты следует труднорешаемость, NP-полнота задачи означает, что для ее решения полиномиальным алгоритмом требуется, по крайней мере, крупное открытие.

ТЕОРИЯ NP-ПОЛНЫХ ЗАДАЧ

Мы изложим формальный аппарат теории NP-полных задач. Чтобы эта теория приобрела строгие математические черты, необходимо дать формальные эквивалентны многих интуитивных понятий, например таких, как «задача» и «алгоритм». Основная цель состоит в выявлении связи между формальными терминами и интуитивными понятиями, которые чаще всего используются. Овладев этой связью, мы сможем далее вести изложение главным образом на

неформальном уровне, обращаясь к формальному только тогда, когда это необходимо.

Начнём с обсуждения задач распознавания свойств и представления этих задач в виде языков, что даст возможность отождествить решение задачи с распознаванием соответствующего языка. Основной моделью процесса вычисления будет одноленточная машина Тьюринга, которая используется для определения класса P (Polynomial — полиномиальный) всех языков, детерминированно распознаваемых за полиномиальное время. Машина Тьюринга затем дополняется гипотетической способностью «угадывания», и эта дополненная модель используется для определения класса NP всех языков, «недетерминировано» распознаваемых за полиномиальное время (Nondetenninislically Polynomial). После обсуждения взаимосвязи классов P и NP определяется понятие полиномиального преобразования одного языка в другой. Это понятие применяется для определения наиболее важного класса — класса NP-полных задач. Завершим мы формулировкой и доказательством фундаментальной теоремы Кука, которая даст нам первую NP-полную задачу.

ЗАДАЧИ РАСПОЗНАВАНИЯ, ЯЗЫКИ И КОДИРОВАНИЕ

Теория NP-полных задач строится только для *задач распознавания свойств*. Такие задачи имеют только два возможных решения – «да» или «нет». Задача распознавания П состоит из двух множеств: множества $D_П$ всех возможных *индивидуальных задач* и множества $Y_П$ ($Y_П \subset D_П$) *индивидуальных задач с ответом* «да». Содержательные задачи распознавания, как правило, обладают различными дополнительными структурными свойствами, которые будут учитываться при описании задач.

Стандартная форма описания задач состоит из двух частей. В первой части дается описание *условия* задачи в терминах различных компонент - множеств, графов, функций, чисел и т. д. Во второй части в терминах условия формулируется *вопрос*, предполагающий один из двух ответов – «да» или «нет». Это описание определяет множества D_Π и Y_Π очевидным образом. Индивидуальная задача принадлежит D_Π в том и только в том случае, когда она может быть получена из стандартной формы описания подстановкой конкретных значений во все компоненты условия. Индивидуальная задача принадлежит Y_Π в том и только в том случае, когда ответом на вопрос задачи будет «да».
Рассмотрим пример.

ИЗОМОРФИЗМ ПОДГРАФУ
УСЛОВИЕ. Заданы два графа: $G_1=(V_1,E_1)$ и $G_2=(V_2,E_2)$.
ВОПРОС. Верно ли, что G_1 содержит подграф, изоморфный G_2? Другими словами, существуют ли:
(1) подмножества $V'\subseteq V_1$ и $E'\subset E_1$ такие, что $|V'| = |V_2|$, $|E'| = |E_2|$ и
(2) взаимно однозначная функция $f : V_2 \rightarrow V'$, такая, что $\{u, v\}\in E_2$ тогда и только тогда, когда $\{f(u), f(v)\}\in E'$?

Задача распознавания, соответствующая задаче о коммивояжере, может быть сформулирована следующим образом.

КОММИВОЯЖЕР
УСЛОВИЕ. Заданы конечное множество $C = \{c_1, c_2, \ldots, c_m\}$ «городов», «расстояния» $d(c_i, c_j)\in Z^-$ для каждой пары городов $c_i, c_j\in C$ и граница $B\in Z^+$ (здесь Z^+ обозначает положительные целые числа).
ВОПРОС. Существует ли «маршрут», проходящий через все города из C, длина которого не превосходит B? Другими словами, существует ли последовательность $\langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(m)}\rangle$ элементов C, такая, что

$$\sum_{i=1}^{m-1} d\left(c_{\pi(i)}, c_{\pi(i+1)}\right) + d\left(c_{\pi(m)}, c_{\pi(1)}\right) \leqslant B?$$

Второй пример служит также для иллюстрации важного приема - получения из оптимизационной задачи соответствующей задачи распознавания. Если в оптимизационной задаче среди всех структур данного типа ищется структура, имеющая минимальную «стоимость» (например, среди всех маршрутов ищется маршрут минимальной длины), то этой задаче можно сопоставить задачу распознавания, в которой в качестве дополнительного параметра фигурирует числовая граница B, а вопрос ставится о существовании структуры данного типа, стоимость которой *не превосходит* B (например, маршрут длины не более B). Аналогичным образом задача распознавания может быть получена из задачи максимизации. Для этого достаточно условие «не превосходит» заменить условием «не меньше».

Относительно подобного соответствия между задачами распознавания и задачами оптимизации отметим, что поскольку значение функции легко оценить, то задача распознавания не может быть сложнее соответствующей задачи оптимизации. Если для задачи о коммивояжере можно за полиномиальное время найти маршрут минимальной длины, то совершенно ясно, как за полиномиальное время решить соответствующую задачу распознавания. Для этого нужно только найти маршрут минимальной длины, вычислить его длину и сравнить ее с заданной границей B. Т.к. КОММИВОЯЖЕР есть NP-полная задача, то отсюда следует, что и оптимизационная задача о коммивояжере столь же сложна. Таким образом, хотя в теории NP-полных задач рассматриваются только задачи распознавания, выводы этой теории вполне применимы к задачам оптимизации.

Теория NP-полных задач ограничивается только задачами распознавания. Это объясняется тем, что задачи распознавания имеют очень естественный формальный эквивалент, удобный для изучения методами теории вычислений. Этот эквивалент называется «языком» и определяется следующим образом.

Для любого Σ (алфавита) обозначим через Σ^* множество всех слов, составленных из символов алфавита Σ. Например, если $\Sigma = \{0, 1\}$, то Σ^* состоит из пустого слова, а также слов 0, 1, 00, 01, 10, 11, 000, 001 и всех других конечных слов, состоящих из нулей и единиц. Любое подмножество $L \subseteq \Sigma^*$ будем называть *языком* в алфавите Σ. Таким образом, множество {01,001,111,1101010} является языком в алфавите 0, 1. Языками также являются множество двоичных записей квадратов целых чисел, само множество {0, 1} и т.д.

Соответствие между задачами распознавания и языками устанавливается с помощью схем кодирования, которые применяются для представления индивидуальной задачи. Напомним, что схема кодирования *e* задачи П описывает каждую индивидуальную задачу из П подходящим словом в некотором фиксированном алфавите Σ. Таким образом, задача П и схема кодирования *e* задачи П разбивают слова из Σ^* на три класса: слова, не являющиеся кодами индивидуальных задач из П; слова, являющиеся кодами индивидуальных задач из П с отрицательным ответом на вопрос, и слова, являющиеся кодами индивидуальных задач из П с положительным ответом на вопрос. Третий класс слов и есть тот язык, который ставится в соответствие задаче П при кодировании *e* и обозначается через L[П,*e*]:

$$L[\Pi, e] = \left\{ x \in \Sigma^*; \begin{array}{c} \Sigma \text{ есть алфавит схемы кодирования } e, \\ a\ x - \text{код индивидуальной задачи} \\ I \in Y_{\Pi} \text{ при схеме кодирования } e \end{array} \right\}.$$

При применении формальной теории NP-полноты к задачам распознавания будем говорить, что некоторый результат имеет место для задачи П при схеме кодирования e, если этот результат имеет место для языка L[П, e].

Следуя общепринятой практике, мы не всегда будем соблюдать все эти формальности. Если ограничиться только «разумными схемами кодирования», то при введении любого нового понятия или свойства, которое формулируется в терминах языка, оно не будет зависеть от способа кодирования. Другими словами, если e и e' любые две разумные схемы кодирования задачи П, то рассматриваемое свойство языков L[П,e] и L[П,e'] выполняется (или не выполняется) для них одновременно. Это позволяет говорить неформально о том, что рассматриваемое свойство для задачи П выполняется или не выполняется. Но при использовании этого соглашения будем подразумевать, что можно указать схему кодирования e, что рассматриваемое свойство выполняется для языка L[П, e].

Если вести изложение в стиле, не зависящем от кодирования, то теряется связь с формальным понятием «длина входа». Поэтому необходим некоторый параметр, через который можно выразить временную сложность. Удобно считать, что с каждой задачей распознавания связана не зависящая от схемы кодирования функция *Length*: $D_{П} \rightarrow Z^{+}$, которая «полиномиально эквивалентна» длине кода индивидуальной задачи, получаемой при любой разумной схеме кодирования. Полиномиальная эквивалентность понимается в следующем смысле: для любой разумной схемы кодирования e задачи П существуют два полинома p и p', такие, что если $I \in B$ и слово x есть код индивидуальной задачи I при кодировании e, то *Length* [I]$\leq p(|x|)$ и $|x| \leq p'$(*Length*[I]), где $|x|$ - длина слова x. Например, в задаче ИЗОМОРФИЗМ ПОДГРАФУ можно положить

$\mathbf{Length}\,[I] = |V_1| + |V_2|$: $G_1=(V_1,E_1)$ и $G_2=(V_2,E_2)$ будут графами, образующими рассматриваемую индивидуальную задачу. В задаче КОММИВОЯЖЕР можно положить

$$\mathbf{Length}\,[I] = m + \lceil \log_2 B \rceil + \max\{\lceil \log_2 d(c_i, c_j) \rceil : c_i, c_j \in C\}.$$

Так как любые две разумные схемы кодирования задачи П дают полиномиально эквивалентные длины входов, то диапазон для выбора функции *Length* очень широк, а получаемые результаты будут верны для любой функции *Length*, удовлетворяющей сформулированным выше условиям.

Отсутствие формального определения разумной схемы кодирования вызывает неудобства. Для преодоления этих неудобств можно потребовать, чтобы условие задачи всегда состояло из фиксированного набора основных теоретико-множественных объектов. Мы не будем здесь накладывать этого требования. Однако дадим краткое описание одного из способов определения стандартной схемы кодирования.

Эта стандартная схема кодирования отображает индивидуальные задачи в правильно построенные слова (ППС) в алфавите ψ = {0, 1, —,[,],(,),,}. ППС определяется рекурсивно:

(1) Двоичная запись целого числа k, рассматриваемая как слово, состоящее из нулей и единиц (со знаком « - » перед словом, если k отрицательно), есть ППС, представляющее целое число k.

(2) Если x есть ППС, представляющее целое число k, то $[x]$ есть ППС, которое может использоваться как «имя» (например, «имя» вершины графа, элемента множества или города в задаче о коммивояжере).

(3) Если $x_1, x_2,...,x_m$ - правильно построенные слова, представляющие объекты $X_1, X_2,...,X_m$, то $(x_1, x_2,...,x_m)$ есть ППС, представляющее последовательность $< X_1, X_2,...,X_m >$.

Для указания схемы кодирования конкретной задачи распознавания, представленной стандартной записью, заметим, что если каждая компонента условия представлена некоторым ППС, то все условия можно закодировать, пользуясь правилом (3). Таким образом, достаточно указать способ кодирования объекта каждого типа. При этом мы ограничимся целыми числами, «элементами без структуры» (вершины графа, элементы множества, города и т. д.), последовательностями, множествами, графами, конечными функциями и рациональными числами.

Правила (1) и (3) указывают, как кодировать целые числа и последовательности. Для кодирования элементов без структуры присвоим им, согласно правилу (2), различные «имена», соблюдая при этом следующее условие: если индивидуальная задача содержит не более N элементов без структуры, то используются имена, величина которых не превосходит N. Объекты остальных четырех типов кодируются следующим образом.

Множество объектов представляется в виде произвольно упорядоченной последовательности $<X_1, X_2,...,X_m>$ элементов этого множества и кодируется ППС, соответствующим этой последовательности.

Граф с множеством вершин V и множеством ребер E кодируется ППС (x, y), где x и y есть ППС, представляющие множества V и E соответственно (элементами E являются двухэлементные подмножества V, образующие ребра).

Конечная функция $f: \{u_1, u_2,...,u_m\} \rightarrow W$ кодируется ППС $((x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m))$, где x_i - ППС для объекта u_i, а y_i - ППС для объекта $f(u_i) \in W$, $1 \le i \le m$.

Рациональное число q кодируется ППС (x, y), где x и y - ППС для целых чисел a и b , таких, что $a/b = q$ и наибольший общий делитель a и b равен 1.

Приведенный выше список кодов объектов довольно хорошо иллюстрирует понятие разумного кодирования. Его достаточно в большинстве встречающихся случаев. Для

описания условий задач можно ограничиться только перечисленными объектами, потому что другие типы объектов всегда могут быть выражены через объекты, перечисленные выше. Мы ограничимся только разумными схемами кодирования

ГЛАВА 2. ДЕТЕРМИНИРОВАННЫЕ МАШИНЫ ТЬЮРИНГА И КЛАСС P ПОЛИНОМИАЛЬНЫХ ЗАДАЧ

Для формализации понятия «алгоритм», необходимо зафиксировать модель процесса вычисления. Такой моделью будет служить *детерминированная одноленточная машина Тьюринга* (сокращенно ДМТ), которая схематически изображена на рис.2.1 ниже. Машина Тьюринга состоит из

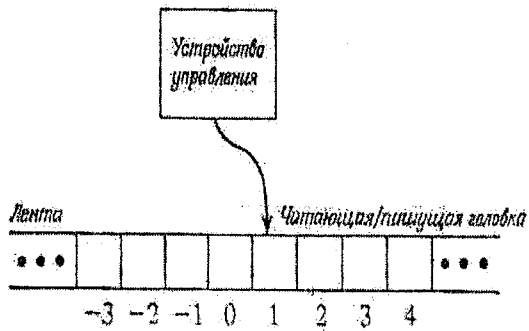


Рис. 2.1. Схематическое изображение детерминированной (одноленточной машины Тьюринга (ДМТ).

управляющего устройства с конечным числом состояний, читающей/пишущей головки, которая может считывать и записывать символы, и неограниченной в обе стороны *ленты,* разделенной на бесконечное число одинаковых ячеек, занумерованных целыми числами ...,-2,-1,0,1,2,3.....

Программа для ДМТ, или ДМТ - программа, определяется следующими компонентами:

(1) конечным множеством Г символов, записываемых на ленте,подмножеством $\Sigma \subset \Gamma$ входных символов и выделенным пустым множеством $b \subset \Gamma \backslash \Sigma$;

(2) конечным множеством *состояний* Q, в котором выделены *начальное состояние* q_0 и два *заключительных состояния* q_Y, q_N ;

(3) *функцией перехода* δ: $(Q \backslash \{q_Y, q_N\}) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$.

Таким образом, для пары состояния q_i и слова c_i определено, какой символ надо записать вместо c_i, в какое состояние перейти и куда передвинуться головке: вправо или влево.

Работа программы определяется так. *Входом* для детерминированной программы является слово $x \in \Sigma^*$. Слово записывается на ленте в ячейках с номерами 1, 2, ... , $|X|$ по одному символу в ячейке. Все другие ячейки в начальный момент времени содержат пустой символ и называются пустыми. Программа начинает работу, находясь в состоянии q_0, при этом читающая/пишущая головка находится над ячейкой с номером 1.

Далее процесс вычислений осуществляется шаг за шагом. Если текущее состояние q есть q_Y или q_N, то процесс вычисления заканчивается, при этом результатом будет «да», если $q = q_Y$ и «нет», если $q = q_N$. В противном случае текущее состояние принадлежит множеству $Q \backslash \{q_Y, q_N,\}$, при этом головка читает на ленте некоторый символ $s \in F$ и определено значение $\delta(q, s)$. Предположим, что $\delta(q, s) = (q', s', \Delta)$. В этом случае головка стирает s, пишет на этом месте s' и сдвигается на одну ячейку влево, если $\Delta = -1$, или на одну ячейку вправо, если $\Delta = +1$. Одновременно управляющее устройство переходит из состояния q в q'. На этом заканчивается один шаг процесса вычисления и программа готова к выполнению следующего шага.

Пример простой ДМТ - программы M представлен на рис. 2.2. Функция перехода δ определена таблицей, где величина, записанная в y-й строке и s-м столбце, есть значение $\delta(q, s)$. Рис. 2.3 иллюстрирует вычисление по программе M при входе x=10100. Здесь указаны состояние, расположение головки и содержимое непустых ячеек ленты до и после каждого шага.

Заметим, что это вычисление после восьми шагов оканчивается в состоянии q_y, поэтому на входе 10100 ответом будет «да». В общем случае будем говорить, что программа M, имеющая входной алфавит Σ, *принимает* $x \in \Sigma^*$ в том и только в том случае, когда, будучи примененной ко входу x она останавливается в состоянии q_y. Язык L_M, *распознаваемый* программой M, задаётся следующим образом:

$$L_M = \{x \in \Sigma^*: M \text{ принимает } x\}.$$

$$\Gamma = \{0, 1, b\}, \ \Sigma = \{0, 1\}$$

$$Q = \{q_0, q_1, q_2, q_3, q_Y, q_N\}$$

q	0	1	b
q_0	$(q_0, 0, +1)$	$(q_0, 1, +1)$	$(q_1, b, -1)$
q_1	$(q_2, b, -1)$	$(q_3, b, -1)$	$(q_N, b, -1)$
q_2	$(q_Y, b, -1)$	$(q_N, b, -1)$	$(q_N, b, -1)$
q_3	$(q_N, b, -1)$	$(q_N, b, -1)$	$(q_N, b, -1)$

$$\delta(q, s)$$

Рис. 2.2. Пример ДМТ-программы $M = (\Gamma, Q, \delta)$.

Нетрудно видеть, что программа, представленная на рис. 2.2, распознаёт язык

$$\left\{x \in \{0, 1\}^*: \begin{array}{l}\textit{два последних символа} \\ \textit{слова } x \textit{ являются нулями}\end{array}\right\}.$$

Отметим, что это определение распознавания языка не требует, чтобы программа M останавливалась при всех входах из Σ^*; M обязана останавливаться лишь при входах из L_M. Если x принадлежит $\Sigma^* \setminus L_M$, то работа программы М на x может либо закончиться в состоянии q_N, либо может бесконечно продолжаться без остановки. ДМТ-программа, соответствующая нашему пониманию алгоритма, должна останавливаться на всех словах входного алфавита. В этом смысле программа на рис. 2.2 является алгоритмической, так как, начиная работать на любом слове из символов 0, 1 она будет останавливаться.

Соответствие между распознаванием языков и решением задач распознавания определяется следующим образом. Будем говорить, что ДМТ-программа M *решает* задачу распознавания П при кодировании e, если M останавливается на всех словах, составленных из букв входного алфавита, и $L_M = L[H, e]$. Программа на рис. 2.2 иллюстрирует это соответствие. Рассмотрим следующую задачу распознавания из теории чисел.

ДЕЛИМОСТЬ НА ЧЕТЫРЕ

УСЛОВИЕ. Дано положительное целое число N.

ВОПРОС. Существует ли положительное целое число m, такое, что $N = 4m$?

При стандартном кодировании целое число N представляется словом из 0 и 1, т. е. двоичной записью этого числа. Так как положительное целое число делится на 4 тогда и только тогда, когда последние две цифры двоичной записи этого числа являются нулями, то программа, изображенная на рис. 2.2, «решает» при нашем стандартном кодировании задачу ДЕЛИМОСТЬ НА ЧЕТЫРЕ.

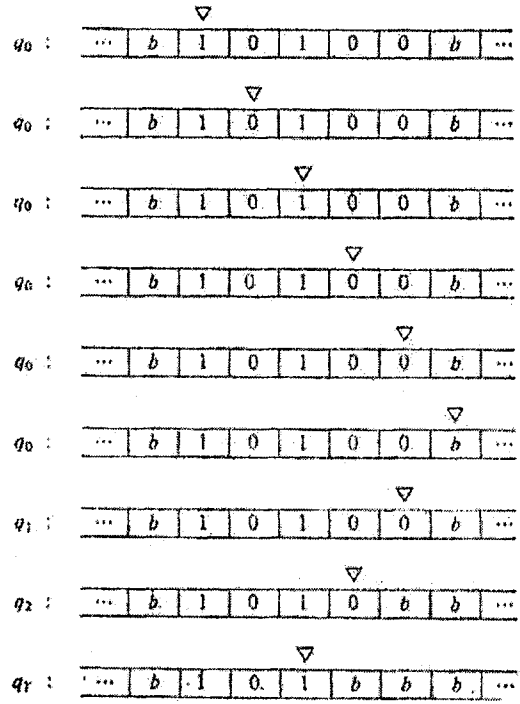


Рис. 2.3. Вычисление по программе M (см. рис. 2.2) при входе 10100.

Заметим, что ДМТ-программой можно пользоваться также и для вычисления функций. Пусть программа M, имеющая входной алфавит Σ и ленточный алфавит Γ, останавливается при любом входе из Σ^*. Тогда M вычисляет *функцию* f_M: $\Sigma^* \to \Gamma^*$, которая для каждого $x \in \Sigma^*$ определяется следующим образом. Если M, начиная работать при входе x, останавливается, то в качестве $f_M(x)$ берется слово,

составленное из символов, записанных после остановки машины в ячейках с номерами 1, 2, 3, ... , включая последнюю непустую ячейку. Программа *M,* представленная на рис. 2.2, вычисляет функцию f_M: $\{0,1\}^* \rightarrow \{0,1,b\}^*$, которая отображает, каждое слово $x \in \{0, 1\}^*$ в слово $f_м(x)$, получаемое из x удалением двух крайних справа символов (если $|x|<2$, то M выдает в качестве $f_м(x)$ пустое слово). Хорошо известно, что DMT-программы способны решать задачи гораздо более сложные, чем в рассмотренном примере.

Создание ДМТ-программ, реализующих более сложные алгоритмы, достигается за счёт увеличения числа состояний и числа символов в алфавите. Можно увидеть, что для любого сложного алгоритма работа машины Тьюринга будет занимать много времени, а соответствующая ДМТ-программа, в любом случае, становится гораздо более громоздкой, чем программа для современных компьютеров с большим количеством символов в алфавите и множеством состояний. Однако машина Тьюринга полезна тем, что, во-первых, любой алгоритм, который можно реализовать на произвольном вычислительном устройстве, можно реализовать и на машине Тьюринга. Во-вторых, схему работы машины Тьюринга легко себе представить без получения специальных знаний, чего нельзя сказать о компьютерах и программном обеспечении. Соответственно, несложно понять, какие алгоритмы могут быть реализованы на машине Тьюринга и к каким результатам может привести работа любой ДМТ-программы, т.е. любого алгоритма для машины Тьюринга.

Определим понятие «временная сложность». *Время,* требуемое ДМТ-программой M для вычисления при входе x, есть число шагов, выполняемых до момента остановки. Если программа M останавливается на всех входах $x \in \Sigma^*$, то *временную сложность* $T_M: Z^+ \rightarrow Z^+$ этой программы можно определить так:

$$T_M(n) = \max\left\{ m: \begin{array}{l} \text{существует такое слово } x \in \Sigma^*, |x| = n, \\ \text{что вычисление по программе } M \text{ на вхо-} \\ \text{де } x \text{ требует времени } m \end{array} \right\}$$

Детерминированная программа M называется *полиномиальной ДМТ-программой,* если существует такой полином p, что для всех $n \in Z^+$, $T_M(n) \le p(n)$.

Определим первый важный класс языков - класс P. Он определяется следующим образом:

$$P = \left\{ L: \begin{array}{l} \text{существует полиномиальная ДМТ-программа} \\ M, \text{ для которой } L = L_M \end{array} \right\}$$

Будем говорить, что задача распознавания П принадлежит классу P при кодировании e, если $L[П,e] \in$ P, т. е. существует полиномиальная ДМТ-программа, которая решает задачу П при кодировании e. Мы не будем приводить конкретные схемы кодирования и будем просто говорить, что задача распознавания П принадлежит классу P.

Приведём теперь примеры (см. также Введение) задач из класса P и некоторые общие методы получения алгоритмов для доказательства того, что задача принадлежит классу P.

МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО

УСЛОВИЕ Заданы n вершин и положительные целые веса дуг между ними.

ВОПРОС Чему равен наименьший возможный вес остовного дерева?

Замечание. Переформулируйте эту задачу в виде задачи распознавания.

Сначала кажется (см. задачу КОММИВОЯЖЕР), что задача МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО только усугубляет трудности задачи КОММИВОЯЖЕР: перебор по множеству всех замкнутых путей заменяется перебором по еще более необозримому множеству произвольных остовных деревьев. Тем не менее, эта интуиция в данном случае в корне

ошибочна: эффективные алгоритмы для задачи МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО существуют.

Алгоритм 6 Упрощенный алгоритм Прима

Вход: $m > 0, w_{ij} > 0, \forall i,j \in (1,\ldots,n)$ {Список ребер задается матрицей связности $\{w_{ij}\}$ — не оптимальный вариант}
Выход: Остов A минимального веса.

```
A ← {1} {можно выбрать произвольную вершину}
for all iteration ∈ 1..n do {достаточно, чтобы процесс сошелся}
  W_best ← +∞
  V_best ← null
  for all i ∈ {1..n : i ∉ A} do {по всем не включенным в A узлам}
    for all j ∈ A do {по всем включенным в A узлам}
      if w_ij > W_best then {w_ij более дешевая дорога из остова}
        W_best ← w_ij {улучшаем оценку}
        V_best = i {выбираем v_i для добавления}
      end if
    end for
  end for
  A ← A ∪ V_best {добавляем вершину с самым дешевым путем до остова}
end for
return A {Индексы вершин минимального остова G}
```

Опишем один из них, так называемый *алгоритм Прима.* В этом алгоритме минимальный остов строится постепенно, сначала выбирается произвольная вершина, которая включается в остов, затем, на каждой итерации, к текущему остову добавляется наиболее дешевое ребро *(u,v),* соединяющее какую-либо вершину из остова *u,* с какой-либо вершиной *v* не из остова. Попробуйте доказать корректность *алгоритма Прима*, приводимого выше. Нетрудно заметить, что сложность приведённого *алгоритма Прима* есть $O(n^3)$. Можно ли уменьшить сложность этого алгоритма (см. (2), стр. 13)?.

КРАТЧАЙШИЙ ПУТЬ В ГРАФЕ

УСЛОВИЕ Заданы n вершин в графе и длины дуг между ними.

ВОПРОС Найти наименьшую длину пути, ведущего из одной заданной вершины в другую заданную вершину?

Замечание. Сформулируйте и эту задачу как задачу распознавания.

Конечно, существует переборный алгоритм, решающий эту задачу. Однако алгоритм, приведённый ниже и имеющий сложность такую же, как *алгоритм Прима*, решает задачу КРАТЧАЙШИЙ ПУТЬ В ГРАФЕ (докажите корректность этого алгоритма, см., например, (2), стр. 11).

Алгоритм 5 Алгоритм КРАТЧАЙШИЙ ПУТЬ В ГРАФЕ в одну вершину

Вход: $n > 0, d_{ij} > 0, \forall i, j \in (1, \ldots, n)$

Выход: Кратчайший путь из любой вершины v_i в v_n.

for all $i \in 1..n$ **do**
  $P_i \equiv (Null, +\infty)$ {сбрасываем начальные оценки}
end for
for all $j \in \{j : d_{jm} < +\infty\}$ **do**
  $P_j \equiv (n, d_{jn})$ {устанавливаем оценки для соседей}
end for
for all *iteration* $\in 1..n$ **do** {достаточно, чтобы процесс сошелся}
  for all $i \in 1..n$ **do** {по всем узлам}
    for all $j \in \{j : d_{ij} < +\infty\}$ **do** {по всем соседям v_i}
      if $D_i > D_j + d_{ij}$ **then** {j-й сосед предлагает нам лучший путь}
        $D_i \leftarrow D_j + d_{ij}$ {улучшаем оценку}
        $P_i = (j, D_i)$ {направляем путь к j-му соседу}
      end if
    end for
  end for
end for
return $\forall i \quad (P_i(1), P_{P_i(1)}(1), P_{P_i(P_i(1))}(1), \ldots, n)$ {От всех узлов прокладываем путь к v_n}

Теперь мы обратимся к описанию известных методов получения полиномиальных алгоритмов и к примерам задач, на которых эти методы хорошо иллюстрируются.

ПОИСК В УПОРЯДОЧЕННОМ МАССИВЕ.

УСЛОВИЕ. Имеется упорядоченный линейно массив чисел $a_1 < a_2 < \ldots < a_n$ и элемент a, совпадающий с одним из a_i.

ВОПРОС. Чему равен номер элемента, с которым совпадает a?

Замечание. Сформулируйте также и эту задачу как задачу распознавания.

Конечно, есть простой алгоритм, решающий эту задачу за полиномиальное время (какой?). Однако имеет место следующая

Теорема 2.1. Существует алгоритм A, для которого максимальное число сравнений равно $\lceil log_2 n \rceil$.

Предъявите нужный алгоритм и докажите, что он правильно работает (для справки см. (4), стр. 5; соответствующий алгоритм называется «поиск в бинарном дереве» и определяется с помощью рекурсии).

Доказать утверждение типа «для любого алгоритма» обычно существенно труднее, чем утверждение типа «существует алгоритм». В первом случае нужно чётко описать весь класс рассматриваемых алгоритмов. Выше было указано, что любой алгоритм поиска в упорядоченном массиве из n элементов можно представить в виде бинарного дерева. Поэтому далее мы рассмотрим некоторые свойства бинарных деревьев.

Определение. *Глубиной $h(x)$ листа x* в корневом дереве D будем называть число ребер в (единственном) пути из корня дерева в лист x. *Высотой $h(D)$ дерева D* будем называть $\max\{h(x)\}$, где максимум берется по всем листьям дерева D. *Средней высотой $h_{ср}(D)$ дерева D* будем называть среднее арифметическое величин $h(x)$ по всем листьям дерева D.

Лемма 2.2. *Для любого бинарного дерева с n листьями выполнены неравенства:*

1) $h(D) \geq \lceil \log_2 n \rceil$;

2) $h_{cp}(D) \geq \log_2 n$.

Докажите эту лемму самостоятельно (для справки см. (4), стр. 6)

Следствие. Для всякого алгоритма A поиска в упорядоченном массиве сложность $A \geq \lceil \log_2 n \rceil$ (для доказательства достаточно представить A в виде бинарного дерева, в котором не менее n листьев).

СОРТИРОВКА

ВОПРОС. Дана последовательность различных элементов некоторого линейно упорядоченного множества

УСЛОВИЕ. Выстроить элементы в порядке возрастания.

Соответствующий алгоритм A упорядочения можно представить в виде бинарного корневого дерева, каждой вершине которого приписана пара сравниваемых элементов, а листьям приписаны ответы в виде перестановок. Обозначим через $L(n)=\min L_A(n)$, где min берётся по всем алгоритмам A, а $L_A(n)$ есть сложность алгоритма сортировки A и равна максимальному числу сравнений от начала работы до ответа.

Теорема 2.3. $L_A(n) \geq \log_2 n!$.

Для доказательства достаточно представить A в виде бинарного дерева и заметить, что любая последовательность элементов м.б. ответом и в силу этого должна быть приписана хотя бы одному листу, а тогда в дереве не менее n! листьев и его высота не меньше $\log_2 n!$ (отсюда немедленно следует, что сложность алгоритма сортировки (любого!) не менее $\log_2 n!$. В силу формулы Стирлинга имеем, $L(n) \geq (1-0(n)) \log_2 n$.

Рассмотрим теперь алгоритм «сортировка слиянием», чья оценка окажется близка сверху к полученной. Алгоритм «СОРТ» будет работать следующим рекурсивным образом: последовательность элементов делится на две почти равные

части A и B и каждую часть упорядочиваем, а затем сливаем отсортированные последовательности вместе в C (для деталей см. (4), стр.9). На каждом шаге слияния сравниваем первые элементы из полученных отсортированных подпоследовательностей A и B и меньший из них переносим очередным элементом в C (если A или B уже пусто, то оставшиеся элементы переносим в C по порядку). Если $L_{сл}(n)$ – число сравнений, то$L_{сл}(1)=0$, а $L_{сл}(n)=L_{сл}(\lfloor n/2 \rfloor)+ L_{сл}(\lceil n/2 \rceil)+ n-1$ при $n \geq 2$.

Лемма 2.4. *$L_{сл}(n)=n\log_2 n-n+1$ для $n=2^k$.*

Доказательство проводится индукцией по k (докажите сами!).

В качестве следствия получаем, что $L_{сл}(n) < 2n\log_2 n+1$ (докажите следствие сами: используйте индукцию или см. для справки (4), стр.9). Обратимся теперь к общим методам получения быстрых алгоритмов.

РЕКУРРЕНТНЫЕ МЕТОДЫ ПОСТРОЕНИЯ АЛГОРИТМОВ

Одно из важных направлений в построении быстрых алгоритмов – это рекуррентные методы. При этом решение задачи сводится к решению более простых подзадач такого же типа, которые, в свою очередь, сводятся к еще более простым подзадачам и т.д. Естественно, при этом должен быть некоторый базисный уровень, задачи которого решаются уже не рекуррентно, а непосредственно. Можно выделить два основных рекуррентных метода, которые используются для построения быстрых алгоритмов: метод динамического программирования и метод «разделяй и властвуй». Для этих основных методов можно привести очень простые и наглядные примеры, которые будут понятны любому читателю: задача об оптимальном порядке умножения матриц и алгоритм Карацубы для умножения чисел

1. Метод динамического программирования

В самом широком виде идея динамического программирования состоит в выделении в данной задаче с параметром n (характеризующим длину входа) подзадач с меньшими параметрами и решении подзадач в соответствии с увеличением параметра, начиная с самого меньшего (обычно 0 или 1). При этом задача с параметром $к$ решается, когда уже решены все подзадачи с параметром $к-1$ и меньше (иногда не $к-1$, а $к-c$, где c - константа). При этом большого числа подзадач удается часто избежать за счет того, что решение разных подзадач сводится к решению одних и тех же подзадач. Рассмотрим пример.

Задача об оптимальном порядке умножения матриц

Мы будем рассматривать здесь только обычный способ умножения двух матриц («строчка на столбец») и будем учитывать только число умножений элементов. При этом , если матрицы A и B имеют размеры m x n и n x p, то для вычисления $A \cdot B$ требуется, mnp умножений элементов. Известно, что для любых трех матриц $(AB)C$ = A(BC), то есть произведение матриц не зависит от расстановки скобок. Однако число операций умножения элементов может при этом оказаться разным.

Рассмотрим вычисление произведения n матриц

$$M = M_1 \times M_2 \times \ldots \times M_n,$$

где M_i - матрица с r_{i-1} строками и r_i столбцами. Порядок, в котором эти матрицы перемножаются, может существенно сказаться на общем числе операции, требуемых для вычисления M, независимо от алгоритма, применяемого для умножения матриц.

Пример Рассмотрим произведение

$$\underset{[10\times 20]}{M_1} \times \underset{[20\times 50]}{M_2} \times \underset{[50\times 1]}{M_3} \times \underset{[1\times 100]}{M_4}$$

где размеры каждой матрицы M_i указаны в скобках. Если вычислять M в порядке $M_1 \times (M_2 \times (M_3 \times M_4))$, то потребуется 125 000 операций, тогда как вычисление M в порядке $(M_1 \times (M_2 \times M_3)) \times M_4$ занимает лишь 2 200 операций.

Процесс перебора всех порядков, в которых можно вычислить рассматриваемое произведение n матриц с целью минимизировать число операций, имеет экспоненциальную сложность, что при больших n практически неприемлемо. Однако динамическое программирование приводит к алгоритму сложности $O(n^3)$. Пусть m_{ij} - минимальная сложность вычисления произведения $M_i \times M_{i+1} \times \ldots \times M_j$, $1 \le i \le j \le n$. Очевидно, что

$$m_{ij} = \begin{cases} 0, & \text{если } i = j, \\ \underset{i \le k < j}{\text{MIN}} (m_{ik} + m_{k+1,j} + r_{i-1} r_k r_j), & \text{если } j > i. \end{cases}$$

Здесь m_{ik} - минимальная сложность вычисления $M' = M_i \times M_{i+1} \times \ldots \times M_k$, а $m_{k+1,j}$ - минимальная сложность вычисления $M'' = M_{k+1} \times M_{k+2} \times \ldots \times M_j$. Третье слагаемое равно сложности умножения M' на M''. Заметим, что M' — матрица размера $r_{i-1} \times r_k$, а M'' - матрица размера $r_k \times r_j$

При динамическом программировании m_{ij} вычисляются в порядке возрастания разностей нижних индексов. Начинают с вычисления m_{ii} для всех i, затем $m_{i,i+1}$ для всех i, потом $m_{i,i+2}$ и т. д. При этом m_{ik} и $m_{k+1,j}$ будут уже вычислены, когда мы приступим к вычислению m_{ij} . Это следует из того, что при $i \le k < j$, разность j-i должна быть больше k-i, а также и j-(k+1). Соответствующий алгоритм приведён на рис 2.4.

```
   begin
1.   for i ← 1 until n do m_ii ← 0;
2.   for l ← 1 until n − 1 do
3.     for i ← 1 until n − 1 do
         begin
4.         j ← i + l;
5.         m_ij ← MIN_{i≤k<j} (m_ik + m_{k+1,j} + r_{i−1} * r_k * r_j)
         end;
6.   write m_1n
   end
```

Рис. 2.4 Алгоритм динамического программирования для определения порядка умножения матриц.

$m_{11}=0$	$m_{22}=0$	$m_{33}=0$	$m_{44}=0$
$m_{12}=10000$	$m_{23}=1000$	$m_{34}=5000$	
$m_{13}=1200$	$m_{24}=3000$		
$m_{14}=2200$			

Рис. 2.5 Сложности вычисления произведений $Mi \times Mi{+}1 \times \ldots \times Mj$.

Для оценки сложности тривиального (переборного) алгоритма и приведённого выше см. (4), стр. 10-12, однако не трудно сообразить, что сложность алгоритма динамического программирования не более чем полиномиальна (для другого примера см. (4), стр. 13, задача о кратчайших путях в ориентированном графе).

2. Метод «разделяй и властвуй». Теорема о рекуррентном неравенстве

Другой рекуррентный метод построения быстрых алгоритмов – это метод, который называют «разделяй и

властвуй». В нем также решение задачи сводится к решению подзадач, но, в отличие от метода динамического программирования, размерность подзадач отличается от размерности задачи не на константу, а в константу раз и подзадачи решаются независимо друг от друга. Для получения оценок сложности таких алгоритмов используется следующая теорема:

Теорема 2.5. *Пусть a, b и c — неотрицательные постоянные. Решение рекуррентных уравнений*

$$T(n) = \begin{cases} b & при\ n = 1, \\ aT(n/c) + bn & при\ n > 1, \end{cases}$$

где n - степень числа c, имеет вид,

$$T(n) = \begin{cases} O(n), & если\ a < c, \\ O(n \log n), & если\ a = c, \\ O(n^{\log_c a}), & если\ a > c. \end{cases}$$

Доказательство. Если n — степень числа, то

$$T(n) = bn \sum_{i=0}^{\log_c n} r^i, \text{ где } r = a/c.$$

Если $a < c$, то $\sum_{i=0}^{\infty} r^i$ сходится и, следовательно, $T(n)=O(n)$. Если $a=c$, то каждым членом этого ряда будет 1, а всего в нем $O(\log n)$ членов. Поэтому $T(n) = O(n \log n)$. Наконец, если $a > c$, то

$$bn \sum_{i=0}^{\log_c n} r^i = bn \frac{r^{1+\log_c n} - 1}{r - 1},$$

что составляет $O(a^{\log_c n})$ или $O(n^{\log_c a})$.

Из **Теоремы 2.5** вытекает, что разбиение задачи размера n (за линейное время) на две подзадачи размера $n/2$ дает алгоритм сложности $O(nlogn)$.Если бы подзадач было 3, 4 или 8, то получился бы алгоритм сложности порядка n^{log3}, n^2 или n^3 соответственно.

С другой стороны, разбиение задачи на 4 задачи размера $n/4$ дает алгоритм сложности $O(nlogn)$, а на 9 и 16 – порядка n^{log3} и n^2 соответственно. Поэтому асимптотически более быстрый алгоритм умножения целых чисел можно было бы получить, если бы удалось так разбить исходные целые числа на 4 части, чтобы суметь выразить исходное умножение через 8 или менее меньших умножений (см. алгоритм Тоома из (4), стр. 18-20). Верхние оценки порядка роста для рекуррентных соотношений см. (4), теорема 2.4.

Алгоритм Карацубы для умножения чисел.

Этот алгоритм по праву считается первым алгоритмом типа «разделяй и властвуй». Рассмотрим умножение двух n-разрядных двоичных чисел. Традиционный метод требует $O(n^2)$ операций. Однако в методе, изложенном ниже, достаточно порядка n^{log3}, т.е. примерно $n^{1,59}$ операций (логарифм выше берётся по основанию 2).

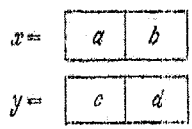


Рис. 2.6 Разбиение двоичных чисел

Пусть x и y - два n-разрядных двоичных числа. Снова будем считать для простоты, что n есть степень числа 2. Разобьем x и y на две равные части, как показано на рис. 2.6. Если рассматривать каждую из этих частей как $(n/2)$-разрядное число, то можно представить произведение чисел x и y в виде

$$xy = (a2^{n/2} + b)(c2^{n/2} + d) =$$
$$= ac2^n + (ad + bc)2^{n/2} + bd.$$

Равенство выше дает способ вычисления произведения x и y с помощью четырех умножений $(n/2)$-разрядных чисел и нескольких сложений и сдвигов (умножений на степень числа 2). Произведение z чисел x и y также можно вычислить по следующей программе:

```
begin
    u ← (a + b) * (c + d);
    v ← a * c;
    w ← b * d;
    z ← v * 2^n + (u − v − w) * 2^(n/2) + w
end
```

На время забудем, что из-за переноса $a+b$ и $c+d$ могут иметь $(n/2)+1$ разрядов и предположим, что они состоят лишь из $n/2$ разрядов. Наша схема для умножения двух n-разрядных чисел требует только трех умножений $(n/2)$-разрядных чисел и нескольких сложений и сдвигов. Для вычисления произведений u, v и w можно применять программу выше рекурсивно. Сложения и сдвиги занимают $O(n)$ времени. Следовательно, временная сложность умножения двух n-разрядных чисел ограничена сверху функцией

$$T(n) = \begin{cases} k & \text{при } n = 1, \\ 3T(n/2) + kn & \text{при } n > 1, \end{cases}$$

где k – постоянная, отражающая сложение и сдвиги в выражениях из программы. Решение рекуррентных уравнений выше ограничено сверху функцией $3kn^{\log 3} \approx 3kn^{1,59}$. На самом деле можно показать, что $T(n) = 3kn^{\log 3} - 2kn$.

Доказательство этого факта проведем индукцией по n, где n – степень числа 2. Базис, т. е. случай n=1, тривиален. Если функция $T(n) = 3kn^{\log 3} - 2kn$ удовлетворяет системе выше при n=m, то

$$\begin{aligned} T(2m) &= 3T(m) + 2km = \\ &= 3\left[3km^{\log 3} - 2km\right] + 2km = \\ &= 3k(2m)^{\log 3} - 2k(2m), \end{aligned}$$

так что она удовлетворяет этой системе и при n=2m. Отсюда следует, что $T(n) \le 3kn^{\log 3}$. Заметим, что попытка использовать в индукции $3kn^{\log 3}$ вместо $3kn^{\log 3} - 2kn$ не проходит.

Для завершении описания алгоритма умножения мы должны учесть, что числа $a+b$ и $c+d$, вообще говоря, имеют $(n/2)+1$ разрядов и поэтому произведение $(a+b)(c+d)$ нельзя вычислить непосредственным рекурсивным применением нашего алгоритма к задаче размера $n/2$. Вместо этого надо записать $a+b$ в виде $a_1 2^{n/2}+b_1$, где a_1=0 или 1. Аналогично запишем $c+d$ в виде $c_1 2^{n/2}+d_1$. Тогда произведение $(a+b)(c+d)$ можно представить в виде

$$a_1c_1 2^n + (a_1d_1 + b_1c_1) 2^{n/2} + b_1d_1.$$

Слагаемое b_1d_1 вычисляется с помощью рекурсивного применения нашего алгоритма умножения к задаче размера $n/2$. Остальные умножения в последнем выражении можно сделать за время $O(n)$, поскольку они содержат в качестве одного из аргументов либо единственный бит a_1 или c_1, либо степень числа 2.

Пример. Асимптотически быстрый алгоритм умножения целых чисел Карацубы можно применять не только к двоичным, но и к десятичным числам. Проиллюстрируем это на примере:

$$x = 3141 \qquad a = 31 \qquad c = 59$$
$$y = 5927 \qquad b = 41 \qquad d = 27$$
$$a + b = 72 \qquad c + d = 86$$
$$u = (a + b)(c + d) = 72 \times 86 = 6192$$
$$v = ac = 31 \times 59 = 1829$$
$$w = bd = 41 \times 27 = 1107$$
$$xy = 18290000^{1}) + (6192 - 1829 - 1107) \times 100 + 1107 = 18616707.$$

[1]) число v следует сдвинуть на четыре десятичных разряда, а u-v-w – на два.

Заметим, что наш алгоритм заменял одно умножение тремя сложениями и вычитанием. Почему такая замена приводит к асимптотической эффективности можно интуитивно объяснить тем, что умножение выполнить труднее, чем сложение, и для достаточно больших n любое фиксированное число n-разрядных сложений требует меньше времени, чем n-разрядное умножение, независимо от того, какой из (известных) алгоритмов применяется. Сначала кажется, что уменьшение числа $n/2$-разрядных произведений с четырех до трех может уменьшить общее время в лучшем случае на 25%. Однако эти соотношения применяются рекурсивно для вычисления $n/2$-разрядных, $n/4$-разрядных и т. д. произведений. Эти 25%-процентные уменьшения объединяются и дают в результате значимое улучшение временной сложности. Временная сложность процедуры определяется числом и размером подзадач, в меньшей степени работой, необходимой для разбиения данной задачи на подзадачи. В заключение данной Главы приведём ряд упражнений.

УПРАЖНЕНИЯ

1. Напишите алгоритм для обращения порядка элементов в списке, докажите, что он работает правильно и оцените его сложность.

*2. *Задача об устойчивом бракосочетании.* Пусть B - множество из n юношей, а G — множество из n девушек. Каждый юноша оценивает девушек числами от 1 до n, и каждая девушка оценивает юношей числами от 1 до n. Паросочетанием называется взаимно однозначное соответствие между юношами и девушками. Паросочетание устойчиво, если для любых двух юношей b_1 и b_2 и соответствующих им в этом паросочетании девушек g_1 и g_2 выполняются следующие два условия:

1) либо b_1 оценивает g_1 выше, чем g_2, либо g_2 оценивает b_2 выше, чем b_1;

2) либо b_2 оценивает g_2 выше, чем g_1, либо g_1 оценивает b_1 выше, чем b_2.

Докажите, что устойчивое паросочетание всегда существует и напишите алгоритм для нахождения одного из таких паросочетаний.

*3. *Ханойские башни.* Имеются три стержня A, B и C и n дисков разных размеров. Вначале все диски нанизаны на стержень A в порядке убывания размеров, так что наибольший диск находится внизу. Надо так переместить диски со стержня A на стержень B, чтобы никакой больший диск ни разу не оказался над меньшим; за один шаг можно перемещать только один диск. Стержень C можно использовать для временного хранения дисков. Напишите рекурсивный алгоритм решения этой задачи. Каково время работы вашего алгоритма в терминах числа перемещений дисков?

*4. Докажите, что для решении упр. 3 необходимо и достаточно 2^n-1 перемещений.

5. Напишите алгоритм порождения всех перестановок целых чисел от 1 до n. *Указание*: множество перестановок целых чисел от 1 до n можно получить из множества перестановок целых чисел от 1 до n-1, вставляя все возможные позиции в каждой перестановке.

6. Решите следующие рекуррентные уравнения, считая, что $T(n)=1$:

$T(n)=a\cdot T(n-1)+bn$;
$T(n)=a\cdot T(n-1)+bn^c$;
$T(n)=a\cdot T(n/2)+bn\log n$;
$T(n)=a\cdot T(n/2)+bn^c$.

7. Покажите, что для одновременного нахождения наибольшего и наименьшего элементов n-элементного множества необходимо и достаточно $\lceil (3/2)n-2 \rceil$ сравнений.

8. Модифицируйте алгоритм умножения целых чисел с помощью разбиения каждого целого числа на

(а) три и

(б) четыре части. Какова сложность получаемых алгоритмов?

9. Пусть A-массив размера n, состоящий из целых чисел, данных в порядке возрастания (числа различны и не равны 0). Написать алгоритм нахождения числа i такого, что $A_i=i$ (если такое существует). Докажите, что $O(\log n)$ – наилучший возможный порядок.

10. Показать, что решением рекуррентного уравнения $X(1)=1$, $X(n)=\sum_{i=1}^{n-1} X(i)X(i-1)$ для $n>1$ служит $X(n+1)=[1/(n+1)]\,C_{2n}^{\,n}$.

Здесь $X(n)$ – число способов правильной расстановки скобок в цепочке из n символов (числа Каталана). Докажите, что $X(n) \geq 2^{n-2}$.

ГЛАВА 3. НЕДЕТЕРМИНИРОВАННЫЕ ВЫЧИСЛЕНИЯ И КЛАСС ЗАДАЧ NP

Теперь введём второй важный класс языков (задач распознавания свойств) - класс NP (см. также (1)). Прежде чем перейти к формальному определению в терминах языков и машин Тьюринга, полезно пояснить смысл понятия, лежащего в основе определения класса NP.

Рассмотрим задачу КОММИВОЯЖЕР, описание которой приведено в начале настоящей главы. В условии даны: множества городов, расстояния между ними и граница B; при этом

спрашивается, существует ли проходящий через все города маршрут длины, не превосходящей B. Полиномиальный алгоритм решения этой задачи не известен. Предположим, однако, что относительно некоторой индивидуальной задачи кто-то получил ответ «да». Если вы в этом сомневаетесь, то можно требовать «доказательства» этого утверждения - предъявления маршрута, обладающего необходимыми свойствами. Имея предъявленное решение, нетрудно проверить, является ли оно на самом деле маршрутом, и если это так, то вычислить его длину, сравнить ее с границей B и тем самым проверить соответствующее утверждение. Более того, эту «процедуру проверки» можно представить в виде алгоритма, временная сложность которого ограничена полиномом от *Length* [I].

Другой пример задачи, которая обладает этим свойством, - задача ИЗОМОРФИЗМ ПОДГРАФУ из Главы 1. Пусть дана индивидуальная задача I, в которой указаны два графа $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$. Если ответом на вопрос задачи будет «да», то этот факт можно «доказать», предъявив подмножества $V' \subseteq V_1$ и $E' \subseteq E_1$ и взаимно-однозначную функцию f: $V_2 \rightarrow V'$, которые обладают необходимыми свойствами. В этом случае верность утверждения опять-таки может быть легко установлена за полиномиальное от *Length*[I] время путем простой проверки того, что V', E' и f удовлетворяют требованиям, сформулированным в задаче.

Понятие полиномиальной проверяемости позволяет выделить задачи класса NP. Отметим, что проверяемость за полиномиальное время не влечет разрешимости за полиномиальное время. Утверждая, что за полиномиальное время можно проверить ответ «да» для задачи КОММИВОЯЖЕР, мы не учитываем время, которое может понадобиться на поиск нужного маршрута среди экспоненциального числа всех возможных маршрутов. Мы лишь утверждаем, что по любому заданному маршруту для индивидуальной задачи I можно за полиномиальное время

проверить, «доказывает» ли этот маршрут, что ответ на вопрос относительно индивидуальной задачи I есть «да».

Неформально класс NP можно определить с помощью понятия, которое называется *недетерминированным алгоритмом.* Такой алгоритм состоит из двух различных стадий - *стадии угадывания,* и *стадии проверки.* По заданной индивидуальной задаче I на первой стадии происходит просто угадывание некоторой структуры *S.* Затем I и *S* вместе подаются в качестве входа на стадию проверки, которая выполняется обычным детерминированным образом и заканчивается либо ответом «да», либо ответом «нет», либо продолжается бесконечно без остановки (позже увидим, что последние две возможности различать не обязательно).

Неформальное определение.

Недетерминированный алгоритм решает задачу распознавания П, если для любой индивидуальной задачи $I \in D_{П}$ выполнены следующие два свойства:

1. Если $I \in Y_{П}$, то существует такая структура *S,* угадывание которой для входа I приведет к тому, что стадия проверки, начиная работу на входе (I, *S*), закончится ответом «да».
2. Если то $I \notin Y_{П}$, то не существует такой структуры *S,* угадывание которой для входа I обеспечило бы окончание стадии проверки на входе (I,*S*) ответом «да».

Недетерминированный алгоритм решения задачи КОММИВОЯЖЕР можно построить, используя в качестве стадии угадывания просто выбор произвольной последовательности городов, а в качестве стадии проверки - упомянутую выше полиномиальную процедуру «проверка доказательства» для задачи КОММИВОЯЖЕР. Очевидно, для любой индивидуальной задачи I найдется такая догадка *S,* что результатом работы стадии проверки на входе (I,*S*) будет «да» в том и только в том случае, если для индивидуальной задачи I существует маршрут искомой длины. Для любой индивидуальной задачи I из КОММИВОЯЖЕР, ДЛЯ КОТОРОЙ СУЩЕСТВУЕТ маршрут нужной длины, этот

маршрут и будет такой догадкой. Если в индивидуальной задаче такого маршрута НЕ СУЩЕСТВУЕТ, то никакая догадка не пройдёт стадию проверки с положительным ответом.

Говорят, что недетерминированный алгоритм, решающий задачу распознавания П, работает в течение полиномиального времени, если найдется полином p такой, что для любого $I \in Y_{\Pi}$ найдется догадка S, приводящая на стадии детерминированной проверки на входе (I, S) к ответу «да» за время $p(Length\ [I])$ и «размер» угадываемой структуры S будет ограничен полиномом от $Length$ [I].

Таким образом, для любой индивидуальной задачи с положительным ответом, можно указать догадку, дающую положительное решение, которую можно проверить за полиномиальное время от длины задачи.

Класс NP, определяемый неформально, есть класс всех задач распознавания П, которые (при разумном кодировании) могут быть решены недетерминированными (N - nondeterministic) алгоритмами за полиномиальное (P - polinomial) время. Задача КОММИВОЯЖЕР принадлежит классу NP. Доказательство аналогичного утверждения о задаче ИЗОМОРФИЗМ ПОДГРАФУ оставляем в качестве упражнения.

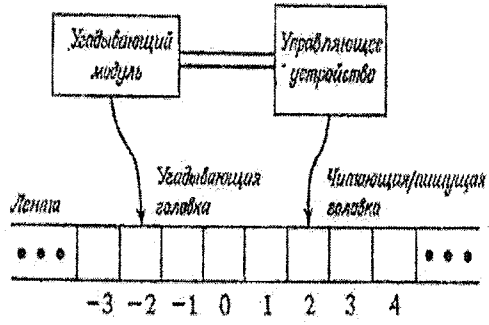
В неформальных определениях термином «решает» следует пользоваться осторожно. Основное назначение полиномиального недетерминированного алгоритма состоит в объяснении понятия «проверяемость за полиномиальное время», а не в том, чтобы служить методом решения задач распознавания свойств. При каждом входе такой алгоритм имеет не одно, а несколько возможных вычислений: по одному для каждой возможной догадки.

Имеется еще одно важное отличие «решения» задачи распознавания недетерминированным алгоритмом от решения детерминированным алгоритмом, а именно в первом случае отсутствует симметрия между ответами «да» и «нет». Если

задача: «Дано I; верно ли, что для I выполняется свойство X?» может быть решена полиномиальным детерминированным алгоритмом, то такое же утверждение справедливо и для дополнительной задачи: «Дано I, верно ли, что для I не выполняется свойство X?». Это следует из того, что детерминированный алгоритм останавливается при всех входах, поэтому достаточно поменять местами ответы «да» и «нет» (переставить состояния q_Y и q_N в ДМТ-программе).

Совершенно не очевидно, что то же самое верно для всех задач, разрешимых за полиномиальное время недетерминированными алгоритмами. Рассмотрим, например, дополнение задачи КОММИВОЯЖЕР: дано множество городов, расстояния между ними и граница B; верно ли, что *нет* маршрута, проходящего через все города и имеющего длину, не превосходящую B? Для выяснения, имеет ли поставленный вопрос ответ «да», не известен способ, который был бы короче, чем проверка всех (или почти всех) возможных маршрутов. Другими словами, не известен полиномиальный недетерминированный алгоритм решения этой дополнительной задачи. Та же самая ситуация имеет место для многих других задач из NP. Таким образом, принадлежность задачи П классу P влечет принадлежность дополнительной задачи классу P, но не известно, имеет ли место аналогичное утверждение для класса NP.

Формализуем приведенное определение в терминах языков и машин Тьюринга (см.(1)). Формальным эквивалентом недетерминированного алгоритма является программа для *недетерминированной одноленточной машины Тьюринга* (НДМТ). Модель НДМТ имеет в точности такую же структуру, как ДМТ: отличие состоит лишь в том, что НДМТ дополнена *угадывающим модулем* со своей *головкой,* которая может только *записывать* на ленту (см. рис. 3.1.). Угадывающий модуль дает информацию для записывания «догадки» и применяется только с этой целью.



3.1 Схематическое представление недетерминированной одноленточной машины Тьюринга (НДМТ).

Программа для НДМТ, или НДМТ-программа, определяется точно так же, как ДМТ-программа, при этом используются: ленточный алфавит Γ, входной алфавит Σ, пустой символ b, множество состояний Q, начальное состояние q_0, заключительные состояния q_Y и q_N, функция перехода δ:

$$(Q \setminus \{q_Y, q_N\}) \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$$

Вычисление НДМТ-программы при входе $x \in \Sigma^*$ в отличие от вычисления ДМТ-программы имеет две различные стадии. На первой стадии происходит «*угадывание*». В начальный момент времени входное слово x записывается в ячейках с номерами 1, 2, ..., $|x|$ (остальные ячейки пусты), читающая/пишущая головка «смотрит» на ячейку с номером 1, пишущая головка угадывающего модуля смотрит на ячейку с номером 1, а устройство управления пассивно. Затем угадывающий модуль начинает управлять угадывающей головкой, которая делает один шаг в каждый момент времени и либо пишет в находящейся под ней ячейке одну из букв алфавита Γ и сдвигается при этом на одну ячейку влево, либо останавливается. В последнем случае угадывающий модуль переходит в пассивное состояние, а управляющее устройство

начинает работу в состоянии q_0. Угадывающий модуль решает, продолжить ли работу, перейти ли в пассивное состояние, какую букву из Γ написать на ленте, причем делает это совершенно произвольно. Таким образом, угадывающий модуль до момента окончания своей работы может написать любое слово из Γ^* и в действительности может никогда не остановиться.

Стадия проверки начинается в тот момент, когда управляющее устройство переходит в состояние q_0. Начиная с этого момента, вычисление НДМТ-программы осуществляется в точности по тем же правилам, что и ДМТ-программы. Угадывающий модуль и его головка в вычислении больше не участвуют, выполнив свою роль, т. е. записав на ленте слово-догадку.. Конечно, слово-догадка может просматриваться читающей/пишущей головкой в процессе проверки. Вычисление заканчивается тогда, когда управляющее устройство перейдет в одно из двух заключительных состояний q_Y или q_N ; это состояние называется *принимающим*, если остановка происходит в состоянии q_Y. Все остальные вычисления, которые заканчиваются или нет, называются *непринимающими.*

Отметим, что любая НДМТ-программа М может иметь бесконечное число возможных вычислений при данном входе x, по одному для каждого слова-догадки из Γ^*. Будем говорить, что НДМТ-программа M *принимает* x, если по крайней мере одно из ее вычислений на входе x является принимающим. Язык, *распознаваемый* программой M есть язык $L_M = \{x \in \Sigma^*: M \text{ принимает } x\}$. Время, требующееся недетерминированной программе M для того, чтобы принять слово $x \in L_M$, есть минимальное число шагов, выполняемых на стадиях угадывания и проверки до момента достижения заключительного состояния q_Y, где минимум берется по всем *принимающим* вычислениям программы M на входе x.

Временная сложность НДМТ-программы M есть функция $T_M: Z^* \to Z^*$, определяемая следующим образом:

$$T_M(n) = \max\left(\{1\} \cup \left\{ m: \begin{array}{l} \textit{существует } x \in L_M, \ |x| = n, \\ \textit{такое, что время принятия } x \\ \textit{программой } M \textit{ равно } m \end{array} \right\}\right).$$

Временная сложность программы M зависит только от числа шагов, выполняемых в принимающих вычислениях. Мы полагаем $M(n)$ равным 1, если нет ни одного входа длины n, принимаемого программой M.

НДМТ-программа называется *НДМТ-программой с полиномиальным временем работы,* если найдется полином p такой, что $T_M(n) \leq p(n)$ для всех $n \geq 1$. Наконец, класс NP формально определяется так:

$$\mathrm{NP} = \left\{ L: \begin{array}{l} \textit{существует НДМТ-программа } M \\ \textit{с полиномиальным временем работы,} \\ \textit{такая, что } L_M = L \end{array} \right\}.$$

Установим взаимосвязь между этими формальными определениями и предшествующими неформальными. Единственный момент, который следует специально оговорить, состоит в следующем. Если недетерминированный алгоритм угадывает структуру S, зависящую от условия рассматриваемой задачи, то угадывающий модуль НДМТ, напротив, полностью игнорирует входное слово. Хотя *любое* слово из Γ^* может быть получено в результате работы угадывающего модуля, НДМТ-программу всегда можно сконструировать так, что стадия проверки будет начинаться с проверки того, соответствует ли догадка подходящей структуре для заданного входа. Иначе программа может сразу же перейти в заключительное состояние q_N.

Будем говорить, что задача распознавания принадлежит классу NP при схеме кодирования e, если $L[n,e] \in$ NP. Как и в случае класса P, мы будем просто говорить, что П лежит в NP.

В заключение заметим, что наши формальные определения можно было бы перефразировать для любой другой стандартной модели вычислительного устройства. Т.к. все такие модели эквивалентны с точностью до детерминированных полиномиальных вычислений, все получаемые при этом версии класса NP будут идентичны. Итак, правомерно сделанное ранее предложение отождествить формально определенный класс NP с классом всех задач распознавания, разрешимых недетерминированными алгоритмами с полиномиальным временем работы.

Перед определением третьего, самого важного класса - класса NP-полных задач, мы обсудим взаимоотношения между классами P и NP.

ВЗАИМООТНОШЕНИЕ МЕЖДУ КЛАССАМИ P И NP

Вопрос о взаимоотношении классов P и NP имеет фундаментальное значение для теории NP-полных задач. Одно соотношение, до настоящего момента явно не сформулированное, заключается в том, что $P \subseteq NP$. Всякая задача распознавания, разрешимая за полиномиальное время детерминированным алгоритмом, разрешима также за полиномиальное время недетерминированным алгоритмом. Чтобы убедиться в этом, достаточно заметить, что любой детерминированный алгоритм может быть использован в качестве стадии проверки недетерминированного алгоритма. Если $П \in P$ и A - произвольный детерминированный полиномиальный алгоритм решения П, то полиномиальный недетерминированный алгоритм для П можно получить, воспользовавшись A в качестве стадии проверки и игнорируя стадию угадывания. Таким образом, из $П \in P$ следует, что $П \in NP$.

Есть много причин считать это включением строгим. Полиномиальные недетерминированные алгоритмы

определенно оказываются более мощными, чем полиномиальные детерминированные алгоритмы и не известны общие методы их превращения в детерминированные полиномиальные алгоритмы. В действительности самый сильный из известных результатов состоит в следующем:

Теорема 3.1. *Если* $\Pi \in$ NP, *то существует такой полином p, что* П *может быть решена детерминированным алгоритмом с временной сложностью* $O(2^{p(n)})$.

Доказательство. Пусть A-полиномиальный недетерминированный алгоритм решения задачи П и $q(n)$ - полином, ограничивающий временную сложность алгоритма A (без ограничения общности можно считать, что q может быть вычислен за полиномиальное время; этого можно добиться, взяв, например, $q(n) = C_1 n^c$ при достаточно больших константах C_1 и c.)

По определению класса NP, для каждого принимаемого входа длины n найдется некоторое слово-догадка (в алфавите Γ символов ленты) длины не более $q(n)$, такое, что в алгоритме A стадия проверки дает при этом входе ответ «да» не более чем за $q(n)$ шагов. Общее число догадок, которые нужно рассмотреть, не превосходит $k^{q(n)}$, где $k = |\Gamma|$ (если слово-догадка короче $q(n)$, то его можно дополнить пустыми символами и рассматривать как слово длины $q(n)$).

Теперь можно детерминированным образом выяснить, имеет ли алгоритм A на заданном входе длины n принимающее вычисление. Для этого достаточно на каждой из $k^{q(n)}$ возможных догадок запустить детерминированную стадию проверки алгоритма A и позволить ей работать до того момента, пока она не остановится или не сделает $q(n)$ шагов. Этот моделирующий алгоритм даст ответ «да», если ему встретится слово-догадка, приводящее к принимающему вычислению длины не более $q(n)$, и ответ «нет» в противном случае. Такой алгоритм будет детерминированным алгоритмом решения задачи П. Его временная сложность

равна $q(n)\cdot k^{q(n)}$ и хотя это экспонента, но при подходящем выборе полинома p сложность не превосходит $O(2^{p(n)})$.

Процесс моделирования, предложенный в доказательстве **Теоремы 3.1** можно ускорить с помощью метода ветвей и границ или с помощью более тщательного перебора, когда избегаются, очевидно, ненужные слова-догадки. Но не известен метод, осуществляющий такое моделирование быстрее, чем за экспоненциальное время.

Способность недетерминированного алгоритма проверить за полиномиальное время экспоненциальное число возможностей может навести на мысль, что полиномиальные недетерминированные алгоритмы являются более мощным средством, чем полиномиальные детерминированные алгоритмы. Для многих частных задач класса NP, таких, как КОММИВОЯЖЕР, ИЗОМОРФИЗМ ПОДГРАФУ и многих других не найдено полиномиального детерминированного алгоритма, несмотря на упорные усилия многих известных исследователей.

Не удивляет поэтому широко распространенное мнение, что P≠NP, хотя пока доказательство этой гипотезы отсутствует.

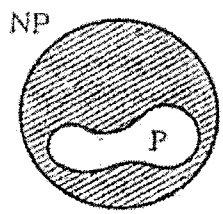


Рис. 3.2 Гипотетическая картина класса NP.

Скептик может сказать, что неудача в доказательстве гипотезы P≠NP является столь же сильным аргументом в пользу соотношения P = NP, как и неудача в поиске соответствующих полиномиальных алгоритмов в пользу

противоположного ему утверждения. Задачи всегда кажутся труднорешаемыми, пока не найдены эффективные алгоритмы их решения. Но при существующем в настоящее время уровне знаний, по-видимому, более разумно работать при соглашении P≠NP, чем пытаться доказать противоположное. На основе накопленного опыта мы будем представлять себе класс NP так, как он изображен на рис. 3.2, ожидая, что затененная область, обозначающая NP\P, не пуста.

ГЛАВА 4. ПОЛИНОМИАЛЬНАЯ СВОДИМОСТЬ И NP-ПОЛНЫЕ ЗАДАЧИ

Если P не совпадает с NP, то различие между P и NP\P очень существенно. Все задачи из P могут быть решены полиномиальными алгоритмами, а все задачи из NP\P труднорешаемы. Поэтому, если P≠NP, то для каждой конкретной задачи $\Pi \in$ NP важно знать, какая из двух возможностей реализуется.

Конечно, пока не доказано, что P≠NP, нет никакой надежды показать, что некоторая конкретная задача принадлежит классу NP\P. По этой причине цель теории NP-полных задач заключается в доказательстве более слабых результатов вида: если P≠NP, то $\Pi \in$ NP\P. Хотя доказательство таких условных результатов может показаться столь же трудным, как и безусловных, но имеются несложные методы доказательства. Такие условные результаты можно рассматривать как подтверждение труднорешаемости с той же степенью уверенности, с какой мы считаем, что класс P отличается от NP.

Основная идея условного подхода основана на понятии полиномиальной сводимости. Будем говорить, что имеет место *полиномиальная сводимость* языка $L_1 \subseteq \Sigma^*_1$ к языку $L_2 \subseteq \Sigma^*_2$, если существует функция f: $\Sigma^*_1 \rightarrow \Sigma^*_2$, удовлетворяющая, двум условиям:

1.Существует ДМТ-программа, вычисляющая f с временной сложностью, ограниченной полиномом.

2. Для любого $x \in \Sigma^*_1$, $x \in L_1$, в том и только в том случае, если $f(x) \in L_2$.

Если L_1 полиномиально сводится к L_2, то будем писать $L_1 \propto L_2$ и говорить «L_1 сводится к L_2» (опуская слово «полиномиально»). Важность понятия полиномиальная сводимость вытекает из следующей леммы:

Лемма 4.1. *Если $L_1 \propto L_2$, то из $L_2 \in P$ следует, что $L_1 \in P$. (Эквивалентное утверждение: из $L_1 \notin P$ следует, что $L_2 \notin P$).*

Доказательство. Пусть Σ_1 и Σ_2 - алфавиты языков L_1 и L_2 соответственно, функция f: $\Sigma_1 \to \Sigma_2$ осуществляет полиномиальную сводимость L_1 к L_2. Пусть M_1 - полиномиальная ДМТ-программа, вычисляющая f, и M_2 — полиномиальная ДМТ-программа, распознающая L_2. Полиномиальная ДМТ-программа, распознающая L_1, может быть получена композицией программы M_1 и M_2. Ко входу $x \in \Sigma^*_1$ вначале применяется M_1, чтобы построить $f(x) \in \Sigma^*_2$. Затем к $f(x)$ применяется программа M_2, выясняющая, верно ли, что $f(x) \in L_2$. Поскольку $x \in L_1$ тогда и только тогда, когда $f(x) \in L_2$, то это описание дает ДМТ-программу, распознающую L_1. То, что время работы этой программы ограничено полиномом, немедленно следует из полиномиальности программ M_1 и M. Точнее, если p_1 и p_2 - полиномы, ограничивающие время работы программ M_1 и M_2 соответственно, то $|f(x)| \le p_1(|x|)$, а время работы только что построенной программы, как не трудно видеть, ограничено функцией $O(p_1(|x|) + p_2(|x|))$, которая является полиномом от $|x|$.

Если Π_1 и Π_2 - задачи распознавания, а e_1 и e_2 - их схемы кодирования, то будем писать $\Pi_1 \propto \Pi_2$ (относительно заданных схем кодирования), если существует полиномиальная сводимость языка $L[\Pi_1, e_1]$ к $L[\Pi_2, e_2]$. Когда будет действовать стандартное предположение о разумности используемых схем кодирования, упоминание о конкретных схемах кодирования,

как обычно, будет опускаться. Таким образом, на уровне задач полиномиальная сводимость задачи распознавания Π_1 к задаче распознавания Π_2 означает наличие функции f: $D_{n1} \rightarrow D_{n2}$, удовлетворяющей двум условиям:

(1) f вычисляется полиномиальным алгоритмом;

(2) для всех $I \in D_n$, $I \in Y_{n1}$, тогда и только тогда, когда $f(I) \in Y_{n2}$.

Чтобы получить конкретное представление о содержательном смысле этого определения, рассмотрим пример. Пусть $G = (V,E)$ - граф с множеством вершин V и множеством ребер E. *Простым циклом* в G называется такая последовательность $< v_1, v_2, \ldots v_k >$ различных вершин из V, что $\{v_i, v_{i+1}\} \in E$, $1 \leq i < k$, и $\{v_k, v_1\} \in E$.

Гамильтоновым циклом в G называется простой цикл, содержащий все вершины графа G. Задача ГАМИЛЬТОНОВ ЦИКЛ определяется следующим образом:

ГАМИЛЬТОНОВ ЦИКЛ

УСЛОВИЕ. Задан граф $G=(V, E)$.

ВОПРОС. Верно ли, что G содержит гамильтонов цикл?

Читатель, несомненно, заметил определенную связь между этой задачей и задачей распознавания КОММИВОЯЖЕР. Покажем, что ГАМИЛЬТОНОВ ЦИКЛ (ГЦ) сводится к КОММИВОЯЖЕРУ (КМ). Для этого требуется указать функцию f, которая отображает каждую индивидуальную задачу из ГЦ в соответствующую индивидуальную задачу из КМ и удовлетворяет двум условиям, которые требуются от полиномиальной сводимости.

Функция f определяется очень просто. Пусть $G = (V, E)$, $|V|=m$ означает фиксированную индивидуальную задачу из ГЦ. Соответствующая задача из КМ строится так: множество городов C совпадает с V и для любых двух городов v_i, $v_j \in C$ расстояние $d(v_i, v_j)$ между ними полагаем равным 1, если $\{v_i, v_j\} \in E$ и 2 в противном случае. Граница B для длины искомого маршрута берется равной m. Каждой

индивидуальной задаче из ГЦ мы сопоставляем индивидуальную задачу из КМ. Каждой вершине графа сопоставляем некоторый город, т.е. число городов равно числу вершин графа m и между ними имеется взаимно однозначное соответствие. Расстояние между i-ым и j-ым городами равно 1, если они соединены ребром в исходном графе и равно 2 в противном случае.

Легко показать (на неформальном уровне), что эта функция f осуществляет сводимость и может быть вычислена за полиномиальное время. Для вычисления $m(m—1)/2$ расстояний $d(v_i,v_j)$ необходимо лишь выяснить, принадлежит ли (v_i , v_j) множеству E или нет. Поэтому первое требование полиномиальности сводимости выполнено. Для проверки второго требования необходимо показать, что G содержит гамильтонов цикл тогда и только тогда, когда в $f(G)$ имеется проходящий через все города маршрут длины, не превосходящей B. Вначале допустим, что $<v_1, v_2, ..., v_m>$ - гамильтонов цикл в G. Тогда $<v_1, v_2, ..., v_m>$ - маршрут в $f(G)$, а его длина равна m ($m = B$), так как расстояние между любыми соседними городами маршрута равно 1, поскольку оно соответствует ребру в G. Наоборот, предположим, что $(v_1, v_2.....v_m)$ - маршрут в $f(G)$, длина которого не превосходит B. Поскольку расстояние между любыми двумя городами в $f(G)$ равно либо 1, либо 2 и при вычислении длины маршрута суммируется ровно m таких расстояний, то из равенства $B = m$ следует, что расстояние между каждой парой соседних городов в маршруте равно 1. По определению $f(G)$, отсюда следует, что (v_i , v_{i+1}), $1 \le i < m$ и (v_m , v_1) являются ребрами графа G и, следовательно, $< v_1, v_2 , ... , v_m>$ - гамильтонов цикл в G.

Таким образом, мы доказали, что ГЦ $\propto$ КМ. Хотя это доказательство гораздо проще многих доказательств, которые будут рассмотрены в дальнейшем, оно содержит все существенные элементы доказательства полиномиальной

сводимости и на неформальном уровне может служить моделью для построения таких доказательств.

Значение **Леммы 4.1** выше для задач распознавания теперь может быть проиллюстрировано на примере сводимости ГЦ $\propto$ КМ. По существу, мы пришли к следующему заключению: если задача КОММИВОЯЖЕР может быть решена полиномиальным алгоритмом, то этот факт верен и для задачи ГАМИЛЬТОНОВ ЦИКЛ, а если ГЦ - труднорешаемая задача, то и КМ также труднорешаемая задача. Таким образом, **Лемма 4.1** позволяет интерпретировать сводимость $\Pi_1 \propto \Pi_2$ как утверждение, что задача Π_2 «не проще» задачи Π_1.

Отношение полиномиальной сводимости особенно удобно, поскольку оно является транзитивным. Это устанавливает следующая лемма.

Лемма 4.2. *Если* $L_1 \propto L_2$ *и* $L_2 \propto L_3$, *то* $L_1 \propto L_3$.

Доказательство. Пусть Σ_1, Σ_2, Σ_3 - алфавиты языков L_1, L_2 и L_3 соответственно, функция f_1: $\Sigma^*_1 \rightarrow \Sigma^*_2$ реализует полиномиальную сводимость L_1 к L_2, а f_2: $\Sigma^*_2 \rightarrow \Sigma^*_3$ - полиномиальную сводимость L_2 к L_3. Тогда функция f: $\Sigma^*_1 \rightarrow \Sigma^*_3$, которая для всех $x \in \Sigma^*$ определяется соотношением $f(x) = f_2(f_1(x))$, реализует искомую сводимость языка L_1 к языку L_3. В самом деле, $f(x) \in L_3$ тогда и только тогда, когда $x \in L_1$, а вычислимость f за полиномиальное время получается с помощью рассуждений, аналогичных рассуждениям, использованным при доказательстве **Леммы 4.1**.

Теперь можно сказать, что языки L_1 и L_2 (соответственно задачи распознавания Π_1 и Π_2) *полиномиально эквивалентны,* если они сводятся друг к другу, т. е. $L_1 \propto L_2$ и $L_2 \propto L_1$ (имеет место сводимость $\Pi_1 \propto \Pi_2$ и $\Pi_2 \propto \Pi_1$). **Лемма 4.2** утверждает, что это отношение является отношением эквивалентности, а также, что отношение $\propto$ определяет частичное упорядочение возникающих классов эквивалентности языков (задач распознавания). На самом деле класс P - это наименьший

относительно этого частичного порядка класс эквивалентности и с вычислительной точки зрения его можно рассматривать как класс самых легких языков (задач распознавания). Класс NP-полных языков (задач) дает нам другой класс эквивалентности, который характеризуется тем, что он содержит самые трудные языки (задачи распознавания) из NP.

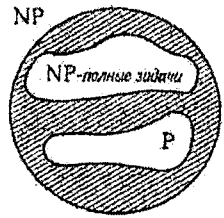


Рис. 4.1 Уточнённая картина класса NP.

Язык L называется NP-*полным,* если $L \in$ NP и любой другой язык $L' \in$ NP сводится к языку L. Говоря неформально, задача распознавания П называется NP-полной, если $П \in$ NP и любая другая задача распознавания $П \in$ NP сводится к П. Таким образом, **Лемма 4.1** позволяет отождествить NP-полные задачи с самыми трудными задачами из NP. Если хотя бы одна NP-полная задача может быть решена за полиномиальное время, то и все задачи из NP также могут быть решены за полиномиальное время. Если хотя бы одна задача из NP труднорешаема, то и все NP-полные задачи труднорешаемы. Следовательно, любая NP-полная задача П обладает свойством, которое сформулировано в начале настоящего раздела: если $P \neq NP$, то $П \in NP \backslash P$. Точнее, $П \in P$ тогда и только тогда, когда $P = NP$.

В предположении $P \neq NP$ можно дать более точную картину класса NP, см. рисунок выше.(рис. 4.1). Заметим, что NP не просто разделяется на две области: класс P и класс NP-полных задач. Если P отличен от NP, то должны существовать

задачи из NP, неразрешимые за полиномиальное время и не являющиеся NP-полными.

Мы будем интересоваться в основном NP-полными задачами. Хотя в начале и говорилось, что имеются простые методы доказательства NP-полноты задач, требования, описанные нами, как будто свидетельствуют об обратном. Нужно доказать, что любая задача из NP сводится к некоторому кандидату на NP-полную задачу. Совершенно неясно, как это можно было бы доказать. Не очевидно даже, что существует хотя бы одна NP-полная задача.

Следующая лемма, которая немедленно следует из определений и транзитивности отношения $\propto$ показывает, что вопрос сильно упростился бы, если бы была известна по крайней мере одна NP-полная задача.

Лемма 4.3. *Если* L_1 *и* L_2 *принадлежат классу* NP, L_1 - это *NP-полный язык и* $L_1 \propto L_2$, *то* L_2 *также NP-полный язык.*

Доказательство. Так как $L_2 \in$ NP, то достаточно показать, что для всякого $L' \in$ NP L' сводится к L_2. Рассмотрим любой язык $L' \in$ NP. Так как L_1 - NP-полный-язык, то L' сводится к L_1. В силу транзитивности отношения $\propto$, из сводимости $L_1 \propto L_2$ следует $L' \propto L_2$.

На уровне задач распознавания **Лемма 4.3** указывает простой путь доказательства NP-полноты новой задачи П, если известна хотя бы одна NP-полная задача. Для того чтобы доказать NP-полноту задачи П, достаточно показать, что:

1. $\Pi \in$ NP.
2. Какая-то одна известная NP-полная задача П' сводится к П.

Но прежде чем можно будет воспользоваться этим методом доказательства, необходимо найти некоторую исходную NP-полную задачу. Такую задачу дает нам фундаментальная теорема Кука, которую мы сформулируем и докажем в следующем разделе.

ТЕОРЕМА КУКА

Честь быть первой NP-полной задачей выпала на долю задачи распознавания из булевской логики, задачи, которую обычно называют ВЫПОЛНИМОСТЬ (сокращенно ВЫП). Термины, использующиеся для ее описания, определяются следующим образом.

Пусть $U = \{u_1, u_2, \ldots, u_m\}$ — множество булевских *переменных*. Под *набором значений истинности* на множестве U будем понимать функцию $t: U \to \{T, F\}$. Если $t(u) = T$, то будем говорить, что u принимает значение «истина» относительно t; если $t(u) = F$, то будем говорить, что u принимает значение «ложь». Если u - переменная из U, то u и $\bar{u}$ назовем *литералами* из U. Литерал u принимает значение «истина» относительно t в том и только в том случае, если переменная u принимает значение «истина» относительно t; литерал $\bar{u}$ принимает значение истина в том и только в том случае, если переменная u принимает значение «ложь».

Дизъюнкцией над U назовем множество литералов над U, например $\{u_1, \bar{u}_3, u_8\}$. Она представляет дизъюнкцию этих литералов и называется *выполненной при* некотором наборе значений истинности тогда и только тогда, когда при рассматриваемом наборе значений истинности хотя бы один из ее членов принимает значение «истина». В нашем примере дизъюнкция будет выполнена относительно t, если одновременно *не окажется*, что $t(u_1)=F$, $t(\bar{u}_3)=T$, $t(u_8)=F$. Набор C дизъюнкций над U .называется *выполнимым* в том и только, в том случае, если найдется некоторый набор значений истинности на множестве U такой, что одновременно выполнены все дизъюнкции из C. Такой набор значений истинности называется *выполняющим набором значений истинности* для C. Задача. ВЫПОЛНИМОСТЬ формулируется следующим образом:

ВЫПОЛНИМОСТЬ

УСЛОВИЕ. Заданы множество переменных U и набор C дизъюнкций над U.

ВОПРОС. Существует ли выполняющий набор значений истинности для C?

Например, пусть $U = \{u_1, u_2\}$ и C= $\{\{ u_1, \bar{u}_2\}, \{ \bar{u}_1, u_2\}\}$. Это индивидуальная задача из ВЫП, ответ на которую есть «да». Выполняющее задание значений истинности определяется так: $t(u_1) = t(u_2) = T$. С другой стороны, заменив C на $C' = \{\{ u_1, u_2\},\{u_1, u_2\},\{\bar{u}_2\}\}$, получим пример индивидуальной задачи, ответ на которую есть «нет», поскольку С невыполнима. Теперь можно сформулировать фундаментальную теорему Кука.

Теорема 4.4 (теорема Кука). *Задача ВЫПОЛНИМОСТЬ есть NP-полная задача*

Доказательство. Легко видеть, что ВЫП лежит в классе NP. Недетерминированному алгоритму для ее решения достаточно указать набор значений истинности на исходном множестве переменных и осуществить проверку того, что этот набор значений выполняет все дизъюнкции из исходного набора C. Все это легко сделать (недетерминированным образом) за полиномиальное время. Таким образом, первое требование, которому должны удовлетворять NP-полные задачи, выполнено.

Для того чтобы проверить выполнение второго требования, вернемся к уровню языков, т. е. к представлению ВЫП языком $L_{вып} = L$[ВЫП, e] для некоторой разумной схемы кодирования e. Необходимо показать, что для всех языков $L \in$ NP имеет место соотношение $L \propto L_{вып}$. Разные языки из NP могут сильно отличаться; число этих языков бесконечно, поэтому невозможно указать отдельное сведение для каждого из них. Однако каждый язык из NP может быть представлен в стандартном виде, а именно НДМТ-программой, работающей полиномиальное время и распознающей этот язык.

Такое представление позволяет иметь дело с общей НДМТ-программой, время работы которой полиномиально, и

получить общую сводимость языка, распознаваемого этой программой, к $L_{вып}$. Если эту общую сводимость специализировать для конкретной НДМТ-программы M, распознающей L_M, то получится искомое полиномиальное сведение L_M к $L_{вып}$. Таким образом, по существу, для всех $L \in$ NP будет представлено общее доказательство того, что $L \propto L_{вып}$.

Вначале рассмотрим произвольную НДМТ-программу M с полиномиальным временем работы, которая имеет компоненты Γ, Σ, b, Q, q_0, q_y, q_N, δ и распознает язык $L = L_M$. Кроме того, пусть $p(n)$ - полином с целыми коэффициентами, ограничивающий сверху временную сложность $T_M(n)$. Не умаляя общности, можно считать, что $p(n) \geq n$ для всех $n \in Z^+$. Функция f_L, реализующая общую сводимость, будет описана в терминах M, Γ, Σ, b, Q, q_0, q_Y, q_N, δ и p.

Удобно рассматривать f_L как отображение из множества слов в алфавите Σ в индивидуальные задачи из ВЫП, а не в слова в алфавите Σ, которые кодируют задачи из ВЫП, так как детали, связанные с кодированием, могут быть легко восполнены. Таким образом, функция f_L будет обладать тем свойством, что для всех $x \in \Sigma^*$ принадлежность $x \in L$ имеет место в том и только в том случае, если для набора дизъюнкций $f_L(x)$ имеется выполняющий набор значений истинности. Ключом к построению функции f_L является использование некоторого набора дизъюнкций для записи утверждения, что x принимается НДМТ-программой M, т. е. утверждение $x \in L$.

Если входное слово $x \in \Sigma^*$ принимается программой M, то для x существует принимающее вычисление программы M, такое, что число шагов на стадии проверки и число символов в слове-догадке ограничены величиной $p(n)$, где $n=|x|$. В таком вычислении будут участвовать лишь ячейки с номерами от $-p(n)$ до $p(n)+1$. Это следует из того, что читающая/пишущая головка начинает работу в ячейке с номером 1 и на каждом шаге сдвигается не более чем на 1 ячейку. Проверяющее

вычисление полностью определяется заданием в каждый момент времени содержания ячеек с указанными номерами, внутреннего состояния и положения читающей/пишущей головки. Далее, поскольку в проверяющем вычислении имеется не более $p(n)$ шагов, то необходимо учесть не более $p(n)+1$ моментов времени. Это позволяет полностью описать такое вычисление, используя полиномиально ограниченное число булевских переменных и некоторый набор значений истинности на их множестве.

Множество переменных U, участвующих в описании функции f_L, предназначено именно для указанных целей. Перенумеруем элементы множеств Q и Γ следующим образом: Q: q_0, $q_1 = q_Y$, $q_2 = q_N$, q_3, ..., q_r, где $r=|Q|$; Γ: $s_0 = b$, s_1, s_2, ..., s_v, где $v=|\Gamma|-1$. В дальнейших рассуждениях будут участвовать три типа переменных, каждому из которых придается определенный смысл согласно рис. 4.2. При этом фраза «в момент времени i» служит сокращением точного выражения «после выполнения i-го шага, проверяющего вычисления».

Вычисление программы M очевидным образом индуцирует на множестве этих переменных набор значений истинности, если принять соглашение, что при окончании вычисления раньше момента времени $p(n)$ конфигурация остается неизменной во все моменты времени после остановки, т. е. сохраняются заключительное состояние, положение головки и запись на ленте.

В нулевой момент времени запись на ленте состоит из входного слова x, расположенного в ячейках с номерами от 1 до n, и слова-догадки w в ячейках с номерами от -1 до $|w|$, при этом все остальные ячейки пусты. С другой стороны, произвольный набор значений истинности этих переменных

Переменная	Пределы изменения индексов	Придаваемый смысл
$Q[i, k]$	$0 \leqslant i \leqslant p(n)$ $0 \leqslant k \leqslant r$	В момент времени i программа M находится в состоянии q_k
$H[i, j]$	$0 \leqslant i \leqslant p(n)$ $-p(n) \leqslant j \leqslant p(n)+1$	В момент времени i читающая/пишущая головка просматривает ячейку с номером j
$S[i, j, k]$	$0 \leqslant i \leqslant p(n)$ $-p(n) \leqslant j \leqslant p(n)+1$ $0 \leqslant k \leqslant v$	В момент времени i в j-й ячейке записан символ s_k

Рис. 4.2 Переменные набора дизъюнкций $f_L(x)$ и придаваемый им смысл.

не обязательно соответствует какому-то вычислению, а тем более принимающему. При произвольном наборе значений истинности переменных в некоторой ячейке могли бы быть одновременно записаны несколько различных символов, машина могла бы находиться одновременно в нескольких различных состояниях, а читающая/пишущая головка могла бы одновременно просматривать любое подмножество ячеек с номерами от $-p(n)$ до $p(n)+1$. Описание функции f_L осуществляется посредством построения такого набора дизъюнкций, содержащего перечисленные переменные, что набор значений истинности будет *выполняющим* тогда и только тогда, когда этот набор значений истинности индуцируется некоторым принимающим вычислением на входе x, причем стадия проверки этого вычисления выполняется не более чем за $p(n)$ шагов и слово-догадка имеет длину не более $p(n)$. Таким образом, мы получим импликации: $x \in L \Leftrightarrow$ на входе x существует принимающее вычисление программы $M \Leftrightarrow$ на входе x существует принимающее вычисление программы M, число шагов которого не превосходит $p(n)$, а слово-догадка w

имеет длину, равную $p(n)$ $\Leftrightarrow$ существует выполняющее задание значений, истинности для набора дизъюнкций задачи $f_L(x)$.

Это будет означать, что для f_L выполнено одно из двух условий, требующихся в определении полиномиальной сводимости. Другое условие, заключающееся в том, что f_L должна быть вычислима за полиномиальное время, можно будет легко проверить после того, как описание f_L будет закончено.

Дизъюнкции индивидуальной задачи $f_L(x)$ можно подразделить на 6 групп, каждая из которых будет налагать ограничение определенного типа на любой выполняющий набор значений истинности. Эти группы показаны на рис. 4.3.

Легко видеть, что если все 6 групп дизъюнкций действительно осуществляют поставленные перед ними цели, то выполняющий набор значений истинности обязан соответствовать нужному принимающему вычислению на входе x. Единственное, что остается - это указать способ построения групп дизъюнкций, осуществляющих эти цели.

Группа G_1 состоит, из следующих дизъюнкций:

$$\{Q[i, 0], Q[i, 1], \ldots, Q[i, r]\}, \quad 0 \leqslant i \leqslant p(n);$$

$$\{\overline{Q[i, j]}, \overline{Q[i, j']}\}, \quad 0 \leqslant i \leqslant p(n), \quad 0 \leqslant j < j' < r.$$

Первые $p(n)+1$ из этих дизъюнкций могут быть выполнены одновременно в том и только в том случае, если в каждый момент времени i программа M находится *по крайней мере* в одном состоянии. Остальные $(p(n)+1)(r+1)(r/2)$ дизъюнкций могут быть выполнены одновременно в том и только в том случае, если ни в один момент времени i программа M не находится *более чем в одном* состоянии. Таким образом, G_1 выполняет свое назначение.

Группа дизъюнкций	Налагаемое ограничение
G_1	В любой момент времени t программа M находится ровно в одном состоянии.
G_2	В любой момент времени t читающая/пишущая головка просматривает ровно одну ячейку.
G_3	В любой момент времени t каждая ячейка содержит ровно один символ из Г.
G_4	В момент времени 0 вычисление находится в исходной конфигурации стадии проверки при входе x.
G_5	Не позднее чем через $p(n)$ шагов M переходит в состояние q_Y и, следовательно, принимает x.
G_6	Для любого момента времени t, $0 \leqslant t \leqslant p(n)$, конфигурация программы M в момент времени $t+1$ получается из конфигурации в момент времени t одноразовым применением функции перехода δ.

Рис. 4.3 Группы дизъюнкций для $f_L(x)$ и ограничения, накладываемые ими на выполняющий набор истинностных значений

Группы G_2 и G_3 строятся аналогично, а группы G_4 и G_5 очень просты, каждая из них включает дизъюнкции, состоящие только из одного литерала. На рис.4.4 дано полное описание первых пяти групп дизъюнкций. Отметим, что как число дизъюнкций в этих группах, так и максимальное число литералов, входящих в каждую дизъюнкцию, ограничены некоторым полиномом от n (так как r и v - константы, которые определяются по M и, значит, по языку L).

Несколько сложнее выглядит описание последней группы дизъюнкций G_6, которая гарантирует, что каждая следующая конфигурация машины получается из предыдущей в результате применения одной команды программы M. Эта группа состоит из двух подгрупп дизъюнкций.

Группа дизъюнкций	Дизъюнкции, входящие в группу
G_1	$\{Q[i, 0], Q[i, 1], \ldots, Q[i, r]\}, \quad 0 \leqslant i \leqslant p(n),$ $\{\overline{Q[i, j]}, \overline{Q[i, j']}\}, \quad 0 \leqslant i \leqslant p(n), \quad 0 \leqslant j < j' \leqslant r,$
G_2	$\{H[i, -p(n)], H[i, -p(n)+1], \ldots, H[i, p(n)+1]\}, 0 \leqslant i \leqslant p(n);$ $\{\overline{H[i, j]}, \overline{H[i, j']}\}, \quad 0 \leqslant i \leqslant p(n), \quad -p(n) \leqslant j < j' \leqslant p(n)+1$
G_3	$\{S[i, j, 0], S[i, j, 1], \ldots, S[i, j, v]\}, \quad 0 \leqslant i \leqslant p(n),$ $-p(n) \leqslant j \leqslant p(n)+1,$ $\{\overline{S[i, j, k]}, \overline{S[i, j, k']}\}, \quad 0 \leqslant i \leqslant p(n), \quad -p(n) \leqslant j \leqslant p(n)+1,$ $0 \leqslant k < k' \leqslant v,$
G_4	$\{Q[0, 0]\}, \{H[0, 1]\}, \{S[0, 0, 0]\},$ $\{S[0, 1, k_1]\}, \{S[0, 2, k_2]\}, \ldots, \{S[0, n, k_n]\},$ $\{S[0, n+1, 0]\}, \{S[0, n+2, 0]\}, \ldots, \{S[0, p(n)+1, 0]\},$ где $x = s_{k_1} s_{k_2} \ldots s_{k_n}$
G_5	$\{Q[p(n), 1]\}.$

Рис. 4.4 Первые 5 групп дизъюнкций для $f_L(x)$

Первая подгруппа гарантирует, что если читающая/пишущая головка в момент времени i не просматривает ячейку с номером j, то символ в ячейке с номером j от момента времени i к моменту времени $i+1$ не изменится. Дизъюнкции этой подгруппы имеют вид:

$$\{S[i, j, l], H[i, j], S[i+1, j, l]\}, \quad 0 \leqslant i \leqslant p(n), \quad -p(n) \leqslant j \leqslant p(n)+1, \quad 0 \leqslant l \leqslant v.$$

Зафиксируем произвольно момент времени i , ячейку с номером j и символ s_l. Если в момент времени i читающая/пишущая головка не просматривает ячейку с номером j и в этой ячейке в момент времени i записан символ j, а в момент времени $i+1$ его там уже нет, то указанная выше

дизъюнкция не будет выполнена (в противном случае она будет выполнена). Таким образом, $2(p(n)+1)^2(v+1)$ дизъюнкций этой группы выполняют свое назначение.

Вторая подгруппа дизъюнкций из группы G_6 гарантирует, что перестройка одной машинной конфигурации в следующую происходит согласно функции перехода δ программы M. Для каждой четверки

$$(i,\ j,\ k,\ l),\ 0 \leqslant i < p(n),\ -p(n) \leqslant j \leqslant p(n)+1,$$
$$0 \leqslant k \leqslant r \quad \text{и} \quad 0 \leqslant l \leqslant v,$$

эта подгруппа содержит следующие три дизъюнкции:

$$\{\overline{H[i,\ j]},\ \overline{Q[i,\ k]},\ \overline{S[i,\ j,\ l]},\ H[i+1,\ j+\Delta]\},$$
$$\{\overline{H[i,\ j]},\ \overline{Q[i,\ k]},\ \overline{S[i,\ j,\ l]},\ Q[i+1,\ k']\},$$
$$\{\overline{H[i,\ j]},\ \overline{Q[i,\ k]},\ \overline{S[i,\ j,\ l]},\ S[i+1,\ j,\ l']\},$$

где значения Δ, k' и l' определены так:

если $q_k \in Q \setminus \{q_Y,\ q_N\}$, то $\delta(q_k,\ s_l) = (q_{k'},\ s_{l'},\ \Delta)$,
а если $q_k \in \{q_Y,\ q_N\}$, то $\Delta = 0$, $k' = k$ и $l' = l$.

Нетрудно понять, что эти $6p(n)(p(n)+1)(r+1)(v+1)$ дизъюнкций дают необходимое ограничение на выполняющий набор значении истинности.

Таким образом, мы показали, как построить группы дизъюнкций G_1–G_6 которые выполняют свое назначение. Если $x \in L$, то у программы M на входе x есть принимающее вычисление длины не более $p(n)$, и это вычисление дает при заданной интерпретации переменных набор значений истинности, который выполнит все дизъюнкции из $C = G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \cup G_6$. И наоборот, набор дизъюнкций C построен так, что любой выполняющий набор значений

истинности для C обязан соответствовать некоторому принимающему вычислению программы M на входе x. Отсюда следует, что для $f_L(x)$ имеется выполняющий набор значений истинности тогда и только тогда, когда $x \in L$.

Теперь осталось показать, что для любого фиксированного языка L индивидуальная задача $f_L(x)$ может быть построена за время, ограниченное полиномом от $n=|x|$. Если язык L задан, то можно выбрать некоторую НДМТ-программу M, которая распознает L за время, ограниченное полиномом p (нет необходимости за полиномиальное время находить саму недетерминированную программу, так как необходимо лишь показать, что отображение f_L *существует).* Если имеются определенная НДМТ-программа M и многочлен p, то построение множества переменных U и набора дизъюнкций C требует чуть больше работы, чем заполнение всех позиций в стандартной (хотя и довольно сложной) формуле. Полиномиальная ограниченность этого вычисления будет ясна, если мы покажем, что *Length* $[f_L(x)]$ ограничена сверху полиномом от n, где *Length* [I] характеризует длину слова, кодирующего индивидуальную задачу I при некоторой разумной схеме кодирования.

В качестве «разумной» функции длины для задачи ВЫП можно, например, взять $|U|\cdot|C|$. Ни одна из дизъюнкций не может содержать более $2\cdot|U|$ литералов (т. к. общее число литералов равно $2\cdot|U|$), а число символов, необходимое для описания каждого индивидуального литерала, не более $log|U|$, но этой величиной можно пренебречь, так как она дает полиномиальный вклад в общую длину задачи. Поскольку r и v фиксированы заранее, то они могут увеличить $|U|$ и $|C|$ в постоянное число раз, поэтому имеем, что мощности $|U|=O(p^2(n))$ и $|C|=O(p^2(n))$. Следовательно, $Length[f_L(x)]=|U|\cdot|C|=O(p^4(n))$ и, значит, ограничена полиномом от n, что и требовалось доказать.

Таким образом, преобразование f_L может быть вычислено алгоритмом, имеющим полиномиальную временную сложность (однако конкретная полиномиальная граница для сложности будет зависеть от L, а также от выбора M и p), из чего можно заключить, что для всех $L \in$ NP функция f_L вычислима за полиномиальное время и отображает L в ВЫП (точнее, отображает L в $L_{вып}$). Отсюда вытекает утверждение теоремы, что ВЫП - NP-полная задача.

ГЛАВА 5. КЛАСС NP-ПОЛНЫХ ЗАДАЧ

Если бы все доказательства NP-полноты были так же сложны, как доказательство NP-полноты задачи ВЫПОЛНИМОСТЬ, то очень сомнительно, что список NP-полных задач смог бы стать столь обширным, как в настоящее время. Однако, как указывалось выше, если уже известна одна NP-полная задача, то процедура доказательства NP-полноты других задач значительно упрощается. Для доказательства NP-полноты задачи $П \in$NP достаточно показать, что какая-нибудь из известных NP-полных задач П' может быть сведена к П. Таким образом, в дальнейшем процесс доказательства NP-полноты задачи распознавания П будет состоять из следующих четырех шагов:

(1) доказательства того, что П лежит в NP;

(2) выбора известной NP-полной задачи П';

(3) построения функции f, сводящей задачу П' к задаче П, и

(4) доказательства того, что функция f осуществляет полиномиальное сведение.

Цель настоящей главы состоит не только в том, чтобы ознакомить читателей с конечными результатами этого процесса (с окончательными доказательствами NP-полноты), но также и в том, чтобы подготовить их к самостоятельному поиску таких доказательств.

Ниже приводятся 6 задач, которыми чаще других пользуются при доказательстве результатов об NP-полноте в каче-

стве известных NP-полных задач и доказывается NP-полнота этих шести задач. Затем приводится описание трех общих подходов к установлению сводимости задач и дается их иллюстрация посредством доказательства NP-полноты большого числа различных задач. В заключение приводится несколько результатов об NP-полноте, которые предлагается доказать читателю.

ШЕСТЬ ОСНОВНЫХ NP-ПОЛНЫХ ЗАДАЧ

Когда практику встречается задача П, NP-полноту которой требуется доказать, его преимущество заключается в том, что он имеет богатый выбор путей такого доказательства. Вполне может случиться, что в прошлом он уже доказывал или видел доказательство NP-полноты похожей задачи П'. Это может привести к мысли попытаться построить доказательство NP-полноты задачи П, копируя доказательство NP-полноты задачи П', или просто сводя задачу П' к задаче П. Во многих случаях это может привести к достаточно простому доказательству NP-полноты задачи П.

Но очень часто не удается найти NP-полную задачу, похожую на П. В таких случаях практику может быть не ясно, какая из сотен известных NP-полных задач лучше всего подходит в качестве основы искомого доказательства. Однако предыдущий опыт может оказаться полезным для ограничения области выбора некоторым ядром из базовых задач, использовавшихся ранее. Хотя теоретически любую из известных NP-полных задач можно наравне с другими выбрать для доказательства NP-полноты новой задачи, на практике оказывается, что некоторые задачи подходят для этой цели гораздо лучше других. Следующие шесть задач входят в число тех, которые используются наиболее часто и могут служить основным ядром списка известных NP-полных задач.

3-ВЫПОЛНИМОСТЬ (3-ВЫП)

УСЛОВИЕ. Дан набор $C=\{c_1, c_2, \ldots, c_m\}$ дизъюнкций на конечном множестве переменных U, таких, что $|c_i| = 3$, $1 \le i \le m$.

ВОПРОС. Существует ли на U набор значений истинности, при котором выполняются все дизъюнкции из C?

ТРЕХМЕРНОЕ СОЧЕТАНИЕ (3-С)

УСЛОВИЕ. Дано множество $M \subseteq W \times X \times Y$, где W, X и Y — непересекающиеся множества, содержащие одинаковое число элементов q.

ВОПРОС. Верно ли, что M содержит трехмерное *сочетание*, т. е. подмножество $M' \subseteq M$, такое, что $|M'|=q$ и никакие два разных элемента M' не имеют ни одной равной координаты?

ВЕРШИННОЕ ПОКРЫТИЕ (ВП)

УСЛОВИЕ. Дан граф $G=(V,E)$ и положительное целое число K такое, что $K \le |V|$.

ВОПРОС. Имеется ли в графе G *вершинное покрытие* не более чем из K элементов, т. е. такое подмножество $V' \subseteq V$, что $|V'| \le K$ и для каждого ребра $\{u, v\} \in E$ хотя бы одна из вершин u или v принадлежит V'?

Т.е можно ли найти подмножество вершин (не более K) такое, что для каждого ребра из исходного графа хотя бы одна из двух вершин была бы в выбранном подмножеств?

КЛИКА

УСЛОВИЕ. Дан граф $G=(V,E)$ и положительное целое число $J \le |V|$.

ВОПРОС. Верно ли, что G содержит некоторую *клику* мощности не менее J, т. е. такое подмножество $V' \subseteq V$, что $|V'| \ge J$ и любые две вершины из V' соединены ребром из E?

Т.е. содержит ли граф связный подграф, включающий $\geq J$ вершин?

ГАМИЛЬТОНОВ ЦИКЛ (ГЦ)

УСЛОВИЕ. Дан граф $G=(V,E)$.

ВОПРОС. Верно ли, что G содержит гамильтонов цикл, т. е. такую последовательность $\langle v_1, v_2, \ldots, v_n \rangle$ вершин графа G, что $n=|V|$, $\{v_n, v_1\} \in E$ и $\{v_i, v_{i-1}\} \in E$ для всех i, $1 \leq i \leq n$.

Т.е. существует ли цикл, проходящий через все вершины графа ровно по одному разу и возвращающийся в начальную точку?

РАЗБИЕНИЕ

УСЛОВИЕ. Заданы конечное множество A и «вес» $s(a) \in Z^+$ для каждого $a \in A$.

ВОПРОС. Существует ли подмножество $A' \subseteq A$, такое, что

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)?$$

Можно ли разбить множество A на два подмножества так, чтобы их суммарные «веса» были равны?

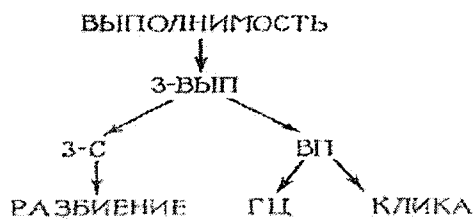


Рис.5.1.Диаграмма последовательности сведенияз адач, используемых используемых для доказательства NP-полноты шести основных задач

Одна из причин популярности этих шести задач заключается в том, что все они содержались в исходном

списке из 21 NP-полной задачи, приведенном в одной из работ Карпа.

Начнем демонстрацию методов доказательства NP-полноты с доказательства NP-полноты этих шести задач, указывая в подходящих случаях такие их варианты, NP-полнота которых более или менее непосредственно следует из NP-полноты этих шести задач.

Первая сводимость будет получена с помощью задачи ВЫПОЛНИМОСТЬ, поскольку пока это единственная задача NP-полноту, которой мы установили. Однако по мере доказательства NP-полноты этих шести задач список NP-полных задач будет расширяться и доказательство NP-полноты некоторой задачи может быть получено с использованием любой из задач, NP-полнота которой установлена раньше, чем для задачи П. На рис. 5.1 указана последовательность сведения задач, которая применяется в доказательстве NP-полноты шести основных задач, причем если стрелка ведет от одной задачи к другой, то первая сводится ко второй.

3-ВЫПОЛНИМОСТЬ

Задача 3-ВЫПОЛНИМОСТЬ есть просто ограниченный вариант задачи ВЫПОЛНИМОСТЬ, в котором каждая индивидуальная задача имеет ровно три литерала в каждой дизъюнкции. Из-за своей простой структуры эта задача наиболее часто применяется для доказательства результатов об NP-полноте.

Теорема 5.1. *Задача 3-ВЫПОЛНИМОСТЬ является* NP-*полной.*

Доказательство. Нетрудно видеть, что 3-ВЫП$\in$ NP. Это следует из того, что недетерминированному алгоритму необходимо угадать лишь набор значений истинности переменных задачи и проверить за полиномиальное время, будут ли при таком наборе значений истинности выполняться все заданные трехлитеральные дизъюнкции.

Сведем задачу ВЫП к задаче 3-ВЫП. Пусть $U = \{u_1, u_2, \ldots, u_n\}$-множество переменных и $C=\{c_1,c_2, \ldots ,c_m\}$ - произвольный набор дизъюнкций, определяющий произвольную индивидуальную задачу из ВЫП. Построим набор C' трехлитеральных дизъюнкций на некотором множестве переменных U', такой, что C' выполним тогда и только тогда, когда выполним C.

Набор C' будет строиться путем замены каждой отдельной дизъюнкции $c_j \in C$ «эквивалентным» набором трехлитеральных дизъюнкций C_j' на множестве U исходных переменных и множестве U'_j некоторых дополнительных переменных, причем переменные из U'_j будут использоваться только в дизъюнкциях из C'_j. Другими словами,

$$U' = U \cup \left(\bigcup_{j=1}^{m} U'_j \right) \quad \text{и} \quad C' = \bigcup_{j=1}^{m} C'_j.$$

Таким образом, нужно показать, как, исходя из c_j, можно построить C'_j и U'_j.

Пусть c_j задается множеством $\{z_1,z_2,\ldots,z_k\}$, где z_i — литералы на множестве U. Способ образования C'_j и U'_j зависит от значения k.

Случай 1. $k=1$. Тогда $U'_j = \{y_j^1, y_j^2\}$,

$$C'_j = \{\{z_1, y_j^1, y_j^2\}, \{z_1, y_j^1, \bar{y}_j^2\}, \{z_1, \bar{y}_j^1, y_j^2\}, \{z_1, \bar{y}_j^1, \bar{y}_j^2\}\}.$$

Случай 2. $k=2$. Тогда $U'_j = \{y_j^1\}$,

$$C'_j = \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \bar{y}_j^1\}\}.$$

Случай 3. $k=3$. Тогда $U'_j = \varnothing$, $C'_j = \{\{c_j\}\}$.

Случай 4. $k>3$. Тогда $U'_j = \{y_j^i : 1 \leqslant i \leqslant k-3\}$,

$$C'_j = \{\{z_1, z_2, y_j^1\}\} \cup \{\{\bar{y}_j^i, z_{i+2}, y_j^{i+1}\} : 1 \leqslant i \leqslant k-4\} \cup$$
$$\cup \{\{\bar{y}_j^{k-3}, z_{k-1}, z_k\}\}.$$

Для доказательства того, что здесь в самом деле имеет место сводимость, необходимо показать, что набор

дизъюнкций C' выполним тогда и только тогда, когда выполним набор дизъюнкций C. Предположим вначале, что t: $U \rightarrow \{T,F\}$ есть набор значений истинности, выполняющий C. Покажем, что t может быть продолжен до набора значений истинности t': $U' \rightarrow \{T,F\}$, который выполняет C'. Поскольку переменные в множестве $U' \backslash U$ делятся на группы U'_j и так как переменные в каждой группе U'_j входят только в дизъюнкции, принадлежащие C'_j, то достаточно показать, как t может быть продолжен на каждое множество U'_j отдельно, и в каждом случае нужно проверить, что выполняются все дизъюнкции в соответствующем множестве C'_j.

Это можно сделать следующим образом. Если U'_j построены, как в случае 1 или 2, то дизъюнкции из C'_j уже выполнены с помощью t, поэтому t может быть продолжен на U'_j произвольно, например, если положить $t'(y)$=T для всех $y \in U'_j$. Если U'_j построено, как в случае 3, то U'_j пусто и единственная дизъюнкция в C'_j уже выполнена с помощью t. Остается только случай 4, который соответствует дизъюнкции $\{z_1, z_2, \ldots, z_k\}$ из C, причем $k > 3$. Поскольку t - выполняющий набор значений истинности для C, то найдется такое целое l, что z_l при t принимает значение *истина*. Если l=1 или 2, то полагаем $t(y^j_l) = F$, $1 \le i \le k-3$. Если $l = k-1$ или k, то полагаем $t'(y^j_l) = F$, $1 \le i \le k-3$. В противном случае полагаем $t(y^j_l) = F$ при $1 \le i \le l-2$. Легко проверить, что такой набор значений истинности обеспечит выполнение всех дизъюнкций в C'_j, поэтому t' выполняет все дизъюнкции из C'. Обратно, если t' - некоторый, выполняющий набор значений истинности для C', то легко проверить, что ограничение t' на переменные из U должно быть выполняющим набором значений истинности для C. Таким образом, C' выполним тогда и только тогда, когда выполним C.

Для того чтобы убедиться, что эта сводимость полиномиальная, достаточно заметить, что число трехлитеральных дизъюнкций в C' ограничено полиномом от

тп. Следовательно, размер индивидуальной задачи 3-ВЫП ограничен сверху некоторым полиномом от размера соответствующей задачи ВЫП, а так как все детали построения сводимости очевидны, то читателю будет нетрудно проверить, что эта сводимость является полиномиальной.

Ограниченная структура задачи 3-ВЫП делает ее гораздо более полезной при доказательстве результатов об NP-полноте по сравнению с задачей ВЫП. Любое доказательство, основанное на задаче ВЫП (кроме только что приведенного), может быть легко перестроено в доказательство, основанное на 3-ВЫП, даже без изменения функции, осуществляющей сводимость. На самом деле дополнительное условие о том, что все дизъюнкции имеют одинаковую длину, часто может упростить сводимость, которую нужно построить, и тем самым облегчить ее отыскание. Более того, очень малая длина дизъюнкций позволяет использовать сводимости, которые вообще не могли бы быть построены для индивидуальных задач с дизъюнкциями большей длины. Это наводит на мысль, что было бы еще лучше, если бы удалось показать, что аналогичная задача 2-ВЫПОЛНИМОСТЬ (каждая дизъюнкция содержит ровно два литерала) является NP-полной. Однако 2-ВЫП может быть решена, например, методом «резолюций» за время, ограниченное полиномом от произведения числа дизъюнкций и числа переменных заданной индивидуальной задачи и, следовательно, принадлежит классу P (для других доказательств полиномиальности 2-ВЫП см. (4), стр. 57-58 или (5), Теорема 5.2, стр.30-32; также, для доказательства NP-полноты задачи 3-ВЫП см. (4), стр. 55-56 или (5), стр. 32-33).

ТРЁХМЕРНОЕ СОЧЕТАНИЕ

Задача ТРЕХМЕРНОЕ СОЧЕТАНИЕ (3-С) обобщает следующую известную задачу о «бракосочетании»: имеется n холостых мужчин и n незамужних женщин, а также список

всех пар мужчин и женщин, согласных вступить в брак друг с другом; можно ли осуществить n бракосочетаний так, чтобы каждый получил бы приемлемого супруга (или супругу), причем никто не должен вступать в брак дважды? Аналогичным образом в задаче ТРЕХМЕРНОЕ СОЧЕТАНИЕ множества W, X и Y соответствуют *трем* различным «полам», а каждая тройка из M отвечает «*трехмерному сочетанию*», *приемлемому* для всех трех участников. В то время как задача 3-С является NP-полной, обычная задача о бракосочетании может быть решена за полиномиальное время.

Теорема 5.2. *Задача ТРЕХМЕРНОЕ СОЧЕТАНИЕ является NP -полной.*

Доказательство. Легко видеть, что 3-С$\in$ NP. Это следует из того, что недетерминированному алгоритму нужно только угадать подмножество в M, состоящее из $q=|W|=|X|=|Y|$ троек, и за полиномиальное время проверить, что любые две из угаданных троек отличаются по всем координатам.

Сведем задачу 3-ВЫП к задаче 3-С. Пусть $U = \{u_1, u_2, \ldots, u_n\}$ - множество переменных и $C=\{c_1, c_2, \ldots, c_m\}$ - множество дизъюнкций произвольной индивидуальной задачи из 3-ВЫП. Нужно построить непересекающиеся множества W, X, Y и $M \subseteq W \times X \times Y$, такие, что $|W|=|X|=|Y|$ и M содержит паросочетание в том и только в том случае, когда C выполнима.

Искомое множество упорядоченных троек M будет разбито на три различных класса в соответствии с их функциональным назначением: «набор значений истинности и развертка», «проверка выполнения» и «достройка».

Тройки, входящие в первый класс, разбиваются на группы (компоненты), каждая из которых соответствует одной переменной $u\in U$, а ее строение зависит от общего числа m дизъюнкций в C. Строение этой компоненты для случая m=4 показано на рис. 5.2. В общем случае каждая такая компонента, соответствующая переменной u_i , включает «внутренние» элементы $a_i[j]\in X$ и $b_i[j]\in Y$, $1\le j\le m$, которые не

будут входить ни в какую тропку вне этой компоненты, и «внешние» элементы $u_i[j]$, $\bar{u}_i[j] \in W$, $1 \in j \leq m$, которые будут входить в другие тройки. Образующие эту компоненту тройки могут быть подразделены на два множества:

$$T_i^t = \{(\bar{u}_i[j],\ a_i[j],\ b_i[j]):\ 1 \leqslant j \leqslant m\};$$

$$T_i^f = \{(u_i[j],\ a_i[j+1],\ b_i[j]):\ 1 \leqslant j < m\} \bigcup \bigcup \{(u_i[m],\ a_i[1],\ b_i[m])\}.$$

Так как внутренние элементы $\{a_i[j],\ b_i[j];\ 1 \leqslant j \leqslant m\}$ могут входить только в тройки, принадлежащие множеству $T_i = T_i^t \cup T_i^f$, то легко видеть, что любое трехмерное сочетание M' должно иметь ровно m троек из T_i, а именно либо все тройки из T_i^t , либо все тройки из T_i^f. Следовательно, можно считать, что компонента из T_i приводит к тому, что трехмерное сочетание индуцирует присвоение переменной u_i значения «истина» или «ложь». Таким образом, трёхмерное сочетание $M' \subseteq M$ определяет набор значений истинности на множестве U, при этом переменная u_i принимает значение «истина» тогда и только тогда, когда $M' \cap T_i = T_i^t$

Каждая компонента множества M, предназначенная для проверки выполнимости, соответствует одной дизъюнкции $c_j \in C$. Такая компонента содержит только два «внутренних» элемента $s_1[j] \in X$ и $s_2[j] \in Y$, а также «внешние» элементы из множества $\{u_i[j],\ \bar{u}_i:\ 1 \leq j \leq n\}$, выбираемые в зависимости от того, какие, литералы содержатся в дизъюнкции c_j.

Множество троек, образующих эту компоненту, определяются следующим образом:

$$C_j = \{(u_i[j],\ s_1[j],\ s_2[j]):\ \bar{u}_i \in c_j\} \bigcup \{(\bar{u}_i[j],\ s_1[j],\ s_2[j]):\ \bar{u}_i \in c_j\}.$$

Итак, любое трехмерное сочетание $M' \subseteq M$ должно содержать ровно одну тройку из C_j . Но это условие может быть выполнено только в том случае, если для некоторого

литерала $u_i \in c_j$ (или $\bar{u}_i \in c_j$) элемент $u_i[j]$ (соответственно $\bar{u}_i[j]$)

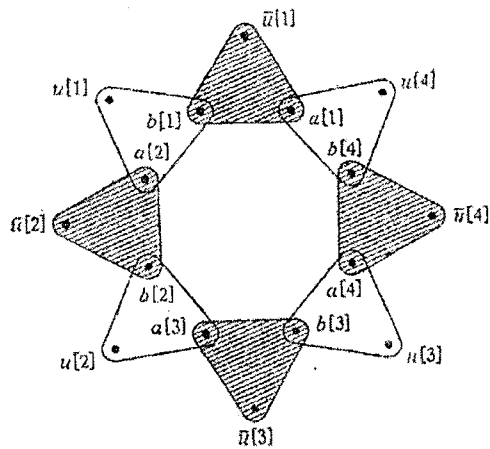


Рис. 5.2. Компонента T_i, задающая значения истинности при m=4 (нижние индексы переменных для простоты опущены). Должно быть выбрано либо множество T_i^t (за штрихованное множество), либо T_i^f (незаштрихованное множество), при этом остаются НЕПОКРЫТЫМИ (т.е. не объединёнными в тройки) либо все $u_i[j]$, либо все $\bar{u}_i[j]$ соответственно.

не войдет ни в одну из троек множества $T_i \cap M'$, а это будет, иметь место тогда и только тогда; когда набор значений истинности, определяемый множеством M', выполняет дизъюнкцию c_j.

Построение нужной индивидуальной задачи из 3-С будет закончено после описания одной большой компоненты «достройки» G. Эта компонента включает внешние элементы $g_1[k] \in X$, $g_2[k] \in Y$, $1 \le k \le m(n-1)$, а также внешние элементы вида $u_i[j]$ (или $\bar{u}_i[j]$ из множества W. Компонента G состоит из следующего множества троек:

$$G = \{(u_i[j],\ g_1[k],\ g_2[k]), (\bar{u}_i[j],\ g_1[k],\ g_2[k]):$$
$$1 \leqslant k \leqslant m(n-1),\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m\}.$$

Таким образом, каждая пара $g_1[k]$, $g_2[k]$ должна войти в трехмерное сочетание только с одним из элементов $u_i[j]$ или $\bar{u}_i[j]$, не попавших в тройки из $M' \setminus G$. Имеется ровно $m(n-1)$ таких «непокрытых» внешних элементов, и структура компоненты G обеспечивает, что они всегда могут быть покрыты с помощью выбора соответствующего подмножества $M' \cap G$. Следовательно, компонента G гарантирует, что, если для некоторого подмножества из $M \setminus G$ выполняются все ограничения, налагаемые компонентами набора значений истинности, тогда это подмножество может быть дополнено до трехмерного сочетания в M.

Резюмируя сказанное, положим:

$$W = \{u_i[j],\ \bar{u}_i[j]:\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m\},$$
$$X = A \bigcup S_1 \bigcup G_1$$

где

далее, полагаем:

$$A = \{a_i[j]:\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m\},$$
$$S_1 = \{s_1[j]:\ 1 \leqslant j \leqslant m\},$$
$$G_1 = \{g_1[j]:\ 1 \leqslant j \leqslant m(n-1)\};$$

$$Y = B \bigcup S_2 \bigcup G_2,$$

где

$$B = \{b_i[j]:\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m\},$$
$$S_2 = \{s_2[j]:\ 1 \leqslant j \leqslant m\},$$
$$G_2 = \{g_2[j]:\ 1 \leqslant j \leqslant m(n-1)\},$$

и

$$M = \left(\bigcup_{i=1}^{n} T_i\right) \cup \left(\bigcup_{j=1}^{m} C_j\right) \cup G.$$

Заметим, что каждая тройка из M есть элемент множества $W \times X \times Y$, что и требовалось. Кроме того, поскольку множество M содержит только $2mn + 3m + 2m^2n(n-1)$ троек и множество M в терминах индивидуальной задачи из 3-ВЫП определяется явным образом, легко видеть, что M может быть построено за полиномиальное время. Из комментариев, которыми сопровождалось описание множества M, немедленно следует, что оно не может содержать трехмерного сочетания, если набор C невыполним. Теперь необходимо доказать, что существование выполняющего набора значений истинности для набора дизъюнкций C влечет за собой существование в M трехмерного сочетания.

Пусть $t: U \to \{T, F\}$ - произвольный выполняющий набор значений истинности для C. Построим трехмерное сочетание $M' \subseteq M$ следующим образом. Для каждой дизъюнкции $c_j \in C$ рассмотрим некоторый литерал $z_j \in \{u_i, \bar{u}_i \mid 1 \leqslant i \leqslant n\} \cap c_j$, принимающий значение «истина» при отображении t (такое z_j должно существовать, поскольку t выполняет c_j). Затем положим

$$M' = \left(\bigcup_{t(u_i)=T} T_i^1\right) \cup \left(\bigcup_{t(u_i)=F} T_i^1\right) \cup \left(\bigcup_{j=1}^{m} \{(z_j[l], s_1[l], s_2[l])\}\right) \cup G',$$

где G' — соответствующее подмножество множества G, выбранное так, что оно содержит все $g_1[k]$, $g_2[k]$ и оставшиеся $u_i[j]$ и $\bar{u}_i[j]$. Нетрудно проверить, что такое множество G' всегда можно выбрать и что полученное в результате множество W будет трехмерным сочетанием.

Вместо задачи 3-С для доказательства результатов об NP-полноте часто пользуются следующей, более общей и более наглядной, версией этой задачи.

ТОЧНОЕ ПОКРЫТИЕ 3-МНОЖЕСТВАМИ (ТП-3)

УСЛОВИЕ. Задано конечное множество X, такое, что $|X|=3q$, и семейство C трехэлементных подмножеств множества X.

ВОПРОС. Верно ли, что семейство C содержит *точное покрытие* множества X, т. е. такое подсемейство $C' \subseteq C$, что каждый элемент из X содержится ровно в одном элементе из C'?

Заметим, что любую индивидуальную задачу из 3-С можно рассматривать как индивидуальную задачу из ТП-3, считая, что M состоит из неупорядоченных троек элементов множества $W \cup X \cup Y$. При этом трехмерные сочетания для индивидуальной задачи из 3-С взаимно однозначно соответствуют точным покрытиям соответствующей индивидуальной задачи из ТП-3. Таким образом, задача 3-С есть ограниченная версия задачи ТП-3 и NP-полнота ТП-3 следует из естественного сведения к ней задачи 3-С.

ВЕРШИННОЕ ПОКРЫТИЕ И КЛИКА

Несмотря на то, что задачи ВЕРШИННОЕ ПОКРЫТИЕ и КЛИКА при доказательстве результатов об NP-полноте используются независимо, в действительности они представляют собой всего лишь разные способы рассмотрения одной и той же задачи. Для того чтобы это понять, удобно рассмотреть их вместе с третьей задачей, которая называется НЕЗАВИСИМОЕ МНОЖЕСТВО.

Независимым множеством в графе $G = (V, E)$ называется такое подмножество V' из V, что для любых $u, v \in V'$, ребро $\{u, v\}$ *не* принадлежит E. В задаче НЕЗАВИСИМОЕ МНОЖЕСТВО для заданного графа $G = (V, E)$ и положительного целого числа $J \leq |V|$ спрашивается, верно ли, что G содержит независимое множество V' такое, что $|V'| \geq J$. Легко убедиться, что независимые множества, клики и вершинные покрытия связаны между собой следующим образом.

Лемма 5.3. *Для любого графа $G(V,E)$ и подмножества $V' \subseteq V$ следующие утверждения эквивалентны:*

(а) V' есть вершинное покрытие G;

(б) $V \setminus V'$ есть независимое множество в G;

(в) $V \setminus V'$ есть клика в дополнении CG графа G, где $CG=(V,CE)$,

$CE=\{(u,v)$: $u,v \in V$ и $(u,v) \notin E\}$

Таким образом, эти три задачи можно рассматривать в некотором смысле как различные версии одной и той же задачи. Более того, взаимосвязи между задачами, указанные в **лемме 5.3**, делают тривиальным вопрос о сводимости между любыми двумя из них.

Сведем, например, задачу ВЕРШИННОЕ ПОКРЫТИЕ к задаче КЛИКА. Пусть $G = (V,E)$ и $K \leq |V|$ - произвольная индивидуальная задача из ВП. Соответствующая индивидуальная задача из КЛИКИ получается, если взять граф G и целое число $J = |V| - K$.

Отсюда следует, что NP-полнота всех трех задач следует из NP-полноты любой из них. Докажем, что задача ВЕРШИННОЕ ПОКРЫТИЕ NP-полна.

Теорема 5.4. *Задача ВЕРШИННОЕ ПОКРЫТИЕ NP-полна.*

Доказательство. Легко видеть, что задача ВП$\in$NP. Это следует из того, что недетерминированному алгоритму достаточно угадать подмножество вершин и за полиномиальное время проверить, что это подмножество содержит хотя бы один конец любого ребра и имеет соответствующее число элементов. Сведем задачу 3-ВЫП к задаче ВП. Пусть $U = \{u_1, u_2, \ldots, u_n\}$ и $C=\{c_1, c_2, \ldots, c_m\}$ определяют произвольную индивидуальную задачу из 3-ВЫП. Укажем граф $G = (V, E)$ и положительное целое число $K \leq |V|$ такие, что G имеет вершинное покрытие с числом элементов не более K тогда и только тогда, когда выполним набор дизъюнкций C.

Как и в предыдущем доказательстве, наша конструкция будет составлена из нескольких компонент. В данном случае будут присутствовать только компоненты набора значений истинности и проверки выполнимости, дополненные некоторыми вспомогательными ребрами, связывающими различные компоненты. Для каждой переменной $u_i \in U$ имеется компонента $T_i=(V_i, E_i)$ набора значений истинности (здесь $V_i=\{u_i, \bar{u}_i\}$, $E_i=\{\{u_i, \bar{u}_i\}\}$), т. е. T_i — это пара вершин, соединенных одним ребром. Заметим, что любое вершинное покрытие должно покрыть ребро из E_i, поэтому оно должно содержать по крайней мере одну из вершин u_i или $\bar{u}_i$. Для каждой дизъюнкции $c_j \in C$ имеется компонента проверки выполнимости $S_j=(V'_j, E'_j)$, состоящая из трех вершин и трех соединяющих их ребер, образующих треугольник:

$$V'_j = \{a_1[j],\ a_2[j],\ a_3[j]\},$$
$$E'_j = \{\{a_1[j],\ a_2[j]\}, \{a_1[j],\ a_3[j]\},\ \{a_2[j],\ a_3[j]\}\}.$$

Отметим, что любое вершинное покрытие должно содержать хотя бы две вершины из V'_j, чтобы покрыть все три ребра из E'_j. Связывающие ребра - это единственная часть конструкции, зависящая от того, какие литералы входят в дизъюнкции. Их лучше всего рассматривать с точки зрения компонент проверки выполнимости. Для каждой дизъюнкции $c_j \in C$ обозначим через x_j, y_j, z_j три входящих в нее литерала. Тогда связывающие ребра, исходящие из S_j, задаются следующим образом:

$$E''_j = \{\{a_1[j],\ x_j\}, \{a_2[j],\ y_j\}, \{a_3[j],\ z_j\}\}.$$

Построение индивидуальной задачи из ВП будет закончено, если положить $k = n+2m$ и $G=(V,E)$, где

$$V = \left(\bigcup_{i=1}^{n} V_i\right) \cup \left(\bigcup_{j=1}^{m} V'_j\right);$$
$$E = \left(\bigcup_{i=1}^{n} E_i\right) \cup \left(\bigcup_{j=1}^{m} E'_j\right) \cup \left(\bigcup_{j=1}^{m} E''_j\right).$$

На рис. 5.3 приведен граф, соответствующий индивидуальной задаче из 3-ВЫП, когда U $\{u_1, u_2, u_3, u_4\}$ и $C=\{\{u_1, \bar{u}_3, \bar{u}_4\},\{\bar{u}_1, u_2, \bar{u}_4\}\}$. Нетрудно видеть, что эту конструкцию можно осуществить за полиномиальное время. Поэтому достаточно показать, что C выполнима тогда и только тогда, когда у G есть вершинное покрытие с числом элементов не более K.

Предположим вначале, что $V' \subseteq V$ - вершинное покрытие G и $|V'| \le K$. В силу сделанных выше замечаний, V' должно содержать по крайней мере одну вершину каждого T_i и хотя бы две вершины каждого S_j. В совокупности это дает по крайней мере $n+2m=K$ вершин. Поэтому V' должно содержать ровно одну вершину каждого T_i и ровно две вершины каждого S_j. Таким образом, можно использовать пересечение V с компонентами наборов значений истинности, чтобы получить отображение

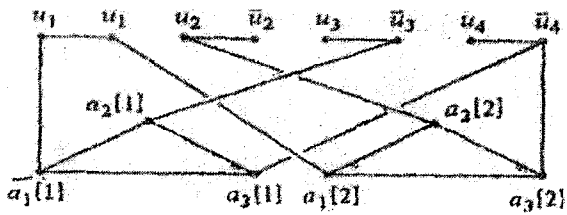


Рис. 5.3 Индивидуальная задача из ВЕРШИННОЕ ПОКРЫТИЕ, к которой приводит индивидуальная задача из 3-ВЫП при $U=\{u_1, u_2, u_3, u_4\}$, $C=\{\{u_1, \bar{u}_3, \bar{u}_4\},\{\bar{u}_1, u_2, \bar{u}_4\}\}$. Здесь $K=n+2m=8$.

t: $U \to \{T, F\}$. Для этого положим $t(u_i)=T$, если $u_i \in V'$, и $t(u_i)=F$, если $\bar{u}_i \in V$. Для доказательства того, что этот набор значений истинности выполняет каждую дизъюнкцию $c_j=C$, рассмотрим три ребра подграфа E''_j. Только два из них могут быть инцидентны вершинам из $V'_j \cap V'$, поэтому одно из них должно быть инцидентно вершине некоторого подграфа V_i,

принадлежащей также V'. Отсюда следует, что соответствующий литерал u_i (или $\bar{u}_i$) дизъюнкции c_j принимает значение «истина» при рассматриваемом наборе значений истинности t, и, следовательно, t выполняет дизъюнкцию c_j. Поскольку это утверждение имеет место для каждой дизъюнкции $c_j \in C$, то отсюда следует, что t - выполняющий набор значений истинности для C.

Обратно, предположим, что t: $U \rightarrow \{T,F\}$ - выполняющий набор значений истинности для C. Тогда соответствующее вершинное покрытие V' содержит одну вершину из каждого T_i и две вершины из каждого S_i. При этом вершиной из T_i принадлежащей V', будет u_i (если $t(u_i)$=T) или $\bar{u}_i$ (если $t(u_i)$=F). Поскольку t выполняет каждую дизъюнкцию c_j, то это обеспечивает покрытие по крайней мере одного из трех ребер, принадлежащих множеству E''_j. Следовательно, осталось включить в V' концы оставшихся двух ребер из E''_j принадлежащиё подграфу S_j (которые могут быть либо инцидентны, либо не инцидентны вершинам из компонент наборов значений истинности) и при этом получится искомое вершинное покрытие.

ГАМИЛЬТОНОВ ЦИКЛ

Ранее мы видели, что к задаче ГАМИЛЬТОНОВ ЦИКЛ (ГЦ) может быть сведена задача распознавания КОММИВОЯЖЕР, поэтому NP-полнота последней задачи следует из NP-полноты задачи ГЦ. После завершения доказательства NP-полноты задачи ГЦ будет указано несколько вариантов задачи ГЦ, NP-полнота которых более или менее непосредственно следует из аналогичного факта для задачи ГЦ.

В дальнейшем для удобства будем пользоваться следующим соглашением: если $<v_1, v_2, \ldots, v_n>$ - некоторый гамильтонов цикл, то ребра $\{v_i, v_{i-1}\}$, $1 \le i \le n$ и $\{v_n, v_1\}$ будут называться ребрами в этом цикле. Приводимая ниже сводимость получена комбинацией двух сводимостей.

Теорема 5.5. *Задача ГАМИЛЬТОНОВ ЦИКЛ является* NP-*полной.*

Доказательство. Легко видеть, что ГЦ∈NP. Это вытекает из того, что недетерминированному алгоритму нужно угадать только последовательность вершин и за полиномиальное время проверить, что все вершины этой последовательности соединены ребрами, принадлежащими данному графу.

Сведем задачу ВЕРШИННОЕ ПОКРЫТИЕ к задаче ГЦ. Пусть индивидуальная задача из ВП задана графом $G = (V, E)$ и положительным целым числом $K \le |V|$. Построим граф $G' = (V', E')$, такой, что в G' есть гамильтонов цикл тогда и только тогда, когда в G' есть вершинное покрытие, состоящее не более чем из K элементов.

В данном случае наша конструкция опять будет состоять из нескольких компонент, соединенных линиями связи. Во-первых, граф G' имеет K «выбирающих» вершин $a_1, a_2 \ldots a_k$, которые будут использованы для выбора K вершин графа G. Во-вторых, для каждого ребра из E граф G' содержит компоненту, «проверяющую покрытие», которая будет использована для того, чтобы обеспечить присутствие одного из концов каждого ребра среди К выбираемых вершин. Компонента, соответствующая ребру $e=\{u, v\}\in E$, показана на рис. 5.4. Она имеет 12 вершин

$$V'_e = \{(u, e, i), (v, e, i): 1 \leqslant i \leqslant 6\}$$ и 14 ребер

$$E'_e = \{\{(u, e, i), (u, e, i+1)\}, \{(v, e, i), (v, e, i+1)\}: 1 \leqslant i \leqslant 5\}$$
$$\cup \{\{(u, e, 3), (v, e, 1)\}, \{(v, e, 3), (u, e, 1)\}\}$$
$$\cup \{\{(u, e, 6), (v, e, 4)\}, \{(v, e, 6), (u, e, 4)\}\}.$$

Когда конструкция будет закончена, то дополнительные ребра, ведущие к этой проверяющей покрытие компоненте, будут инцидентны только вершинам $(u, e, 1)$, $(v, e, 1)$, $(u, e, 6)$ и $(v, e, 6)$.

Отсюда следует (убедитесь), что любой гамильтонов цикл в графе G' должен пересекать ребра из E'_e по одной из трех конфигураций, изображенных на рис. 5.5. Например, если цикл «подходит» к рассматриваемой компоненте в вершине

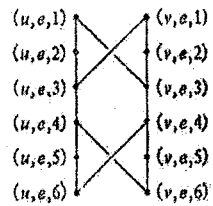


Рис. 5.4 Компонента для проверки покрытия, соответствующая ребру $e=\{u,v\}$ в сведении задачи ВЕРШИННОЕ ПОКРЫТИЕ к задаче ГАМИЛЬТОНОВ ЦИКЛ.

$(u, e, 1)$, то он «выходит» обязательно в вершине $(u, e, 6)$ и проходит либо через все 12 вершин этой компоненты, либо только через 6 вершин (u, e, i), $1 \leq i \leq 6$.

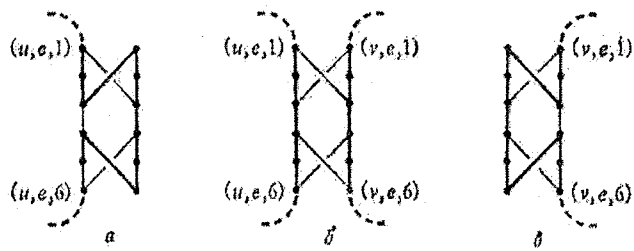
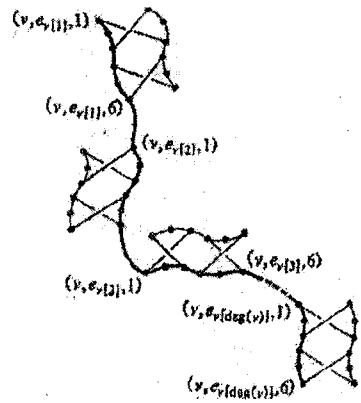


Рис. 5.5 Три возможные конфигурации пересечения гамильтонова цикла с компонентой для проверки покрытия ребра $e=\{u,v\}$. Конфигурации соответствуют следующим случаям: (а) u принадлежит, а v не принадлежит покрытию; (б) u и v принадлежат покрытию; (в) v принад лежит, а u не принадлежит покрытию.

Дополнительные ребра в окончательной конструкции служат либо для соединения пар компонент проверки

покрытия, либо для соединения компоненты проверки, покрытия с выбирающей вершиной. Для каждой вершины $v \in V$ упорядочим произвольным образом ребра, инцидентные v: $e_{v[1]}, e_{v[2]}, \ldots, e_{v[\deg(v)]}$, где deg ($v$) - степень вершины v в графе G. Все компоненты проверки покрытия, соответствующие ребрам, инцидентным v, соединяются вместе следующими связующими ребрами:

$$E'_v = \{\{(v, e_{v[l]}, 6), (v, e_{v[l+1]}, 1)\}: 1 \leqslant l \leqslant \deg(v)\}.$$



5.6 Путь, соединяющий все компоненты проверки покрытия,которые соответствуют рёбрам из E, инцидентным вершине c.

Как показано на рис. 5.6, такое соединение образует единственный путь в C, включающий в точности те вершины (x, y, z), для которых $x = v$.

Последняя группа связующих ребер в G' соединяет первую и последнюю вершины этого пути с каждой из выбирающих вершин $a_1, a_2, \ldots, a_K$. Эти ребра имеют следующий вид:

$$E'' = \{\{a_i, (v, e_{v[1]}, 1)\}, \{a_i, (v, e_{v[\deg(v)]}, 6)\}: 1 \leqslant i \leqslant K, v \in V\}.$$

Окончательно граф $G' = (V', E')$ имеет вид

$$V' = \{a_i : 1 \leqslant i \leqslant K\} \cup \left(\bigcup_{e \in E} V'_e\right)$$

$$E' = \left(\bigcup_{e \in E} E'_e\right) \cup \left(\bigcup_{v \in V} E'_v\right) \cup E''.$$

Нетрудно видеть, что граф G' можно построить за полиномиальное время исходя из G и K.

Утверждается, что G' имеет гамильтонов цикл тогда и только тогда, когда G имеет вершинное покрытие, состоящее не более чем из K вершин. Предположим, что $\langle v_1, v_2, \ldots, v_n \rangle$ - гамильтонов цикл в графе G' ($n = |V'|$). Рассмотрим любую часть этого цикла, которая начинается и кончается в вершинах множества $\{a_1, a_2, \ldots, a_k\}$ и не содержит ни одной вершины этого множества в качестве промежуточной. В силу ранее упомянутых ограничений на конфигурацию, по которой гамильтонов цикл может проходить через компоненту проверки покрытия, рассматриваемая часть цикла должна проходить только через компоненты проверки покрытия, соответствующие ребрам из E, инцидентным некоторой общей вершине $v \in V$. Каждая компонента проверки покрытия пересекается одним из способов (а), (б) или (в), указанных на рис. 5.5, причем она не проходит через вершины ни из какой другой компоненты проверки покрытия.

Таким образом, K вершин множества $\{a_1, a_2, \ldots, a_k\}$ делят рассматриваемый гамильтонов цикл на K путей, соответствующих некоторым вершинам $v \in V$. Поскольку гамильтонов цикл должен проходить через все вершины каждой компоненты проверки покрытия, причем вершины из компоненты проверки покрытия ребра $e \in E$ могут принадлежать только пути, соответствующему одной из вершин, инцидентных e, то любое ребро в E должно быть инцидентно по крайней мере одной из K выбранных вершин. Следовательно, это множество, состоящее из K вершин, образует искомое вершинное покрытие графа G.

Обратно, предположим, что $V^* \subseteq V$ - вершинное покрытие графа G и $|V^*| \leq K$. Можно считать, что $|V^*| = K$, поскольку если к V^* добавить несколько вершин из V, то V^* останется вершинным покрытием. Обозначим элементы V^* через $v_1, v_2, \ldots, v_k$. Рассмотрим, какие ребра нужно включить в гамильтонов цикл графа G. Из компоненты проверки покрытия, представляющей ребро $e = \{u, v\} \in E$, выберем ребра, изображенные на рис. 3.5 (а), (б) или (в), в зависимости от того, равно ли $\{u,v\} \cap V^*$ соответственно $\{u\}$, $\{u, v\}$ или $\{v\}$. Поскольку V^* - вершинное покрытие графа G, то одна из этих возможностей обязательно должна быть реализована. Затем возьмем все ребра в E'_{v_i}, $1 \leq i \leq K$. Наконец, выберем ребра

$$\{a_i, (v_i, e_{v_i}[1], 1)\}, \quad 1 \leqslant i \leqslant K,$$

$$\{a_{i+1}, (v_i, e_{v_i}[\deg(v_i)], 6)\}, \quad 1 \leqslant i < K,$$

и

$$\{a_1, (v_K, e_{v_K}[\deg(v_K)], 6)\}.$$

Проверьте, что выбранное множество ребер в действительности соответствует гамильтонову циклу в G'.

Рассмотрим несколько вариантов задачи ГАМИЛЬТОНОВ ЦИКЛ. Задача ГАМИЛЬТОНОВ ПУТЬ формулируется так же, как задача ГЦ, но без условия, что первая и последняя вершины в последовательности должны быть соединены ребром. Задача ГАМИЛЬТОНОВ ПУТЬ МЕЖДУ ДВУМЯ ВЕРШИНАМИ отличается от задачи ГАМИЛЬТОНОВ ПУТЬ только тем, что в условии этой задачи фиксируются две вершины u и v и спрашивается, верно ли, что G содержит гамильтонов путь из u в v. NP-полнота обеих этих задач может быть доказана с помощью простой модификации сводимости, использованной для доказательства NP-полноты задачи ГЦ. Модифицируем граф G, который получается в конце рассмотренной выше конструкции следующим образом: добавим три новые вершины a_0, a_{k+1} и a_{k+2}, два новых ребра

$\{a_0, a_1\}$ и $\{a_{k-1}, a_{k-2}\}$, а также заменим каждое ребро вида $\{ a_1, (v, e_{v[\deg(v)]},6)\}$ на ребро $\{ a_{k-1}, (v, e_{v[\deg(v)]},6)\}$. В последней модификации задачи ГЦ выделенными вершинами являются a_0 и a_{k-1}.

Все три перечисленные выше задачи остаются NP-полными и в том случае, если неориентированный граф G заменить ориентированным, а неориентированный гамильтонов цикл или путь заменить на ориентированный. Вспомним, что ориентированный граф $G = (V, A)$ состоит из множества вершин V и множества A *упорядоченных* пар вершин, называемых *дугами.* Гамильтоновым путем в ориентированном графе G = (V, A) называется такая последовательность $< v_1, v_2, \ldots , v_n>$ $(n = |V|)$ вершин графа G, что

$<v_i,v_{i+1}>\in A$, $1 \le i \le n$. Гамильтонов путь называется гамильтоновым циклом, если $<v_n,v_1>\in A$. Каждая из упомянутых выше задач на неориентированных графах может быть сведена к задаче на ориентированном графе заменой ребра $\{u,y\}$ неориентированного графа парой дуг (u,v) и (y,u). Задачи на неориентированных графах являются частными случаями соответствующих задач на ориентированных графах.

РАЗБИЕНИЕ

В настоящем разделе будет рассмотрена последняя из шести основных NP-полных задач - РАЗБИЕНИЕ. Она особенно полезна при доказательстве NP-полноты задач, условие которых содержит такие числовые параметры, как длины, веса, стоимости, пропускные способности и т. д.

Теорема 5.6. *Задача РАЗБИЕНИЕ является NP-полной.*

Доказательство. Легко видеть, что РАЗБИЕНИЕ$\in$NP. В самом деле, недетерминированный алгоритм должен угадать только подмножество A' множества A и за полиномиальное время проверить, что сумма размеров элементов из A' такая же, как сумма размеров элементов из $A\backslash A'$.

Сведем задачу 3-С к задаче РАЗБИЕНИЕ. Пусть множества W, X, Y при $|W| = |X| = |Y| = q$ и подмножество $M \subseteq W \times X \times Y$ образуют произвольную индивидуальную задачу из 3-С. Обозначим элементы этих множеств так:

$$W = \{w_1, w_2, \ldots, w_q\},$$
$$X = \{x_1, x_2, \ldots, x_q\},$$
$$Y = \{y_1, y_2, \ldots, y_q\}$$

и

$$M = \{m_1, m_2, \ldots, m_k\},$$

где $k = |M|$. Требуется построить множество A и размеры $s(a) \in Z^+$ всех элементов $a \in A$ так, чтобы A содержало подмножество A', для которого справедливо

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$

, в том и только в том случае, если M содержит трехмерное сочетание.

Множество A будет содержать $k+2$ элемента и строиться в два шага. Первыми k элементами множества A будут $\{a_i: 1 \le i \le k\}$, где a_i ассоциируется с тройкой $m_i \in M$. Размер $s(a_i)$ элемента a_i определяется с помощью двоичного представления числа $s(a_i)$. Слово из нулей и единиц, соответствующее $s(a_i)$, разбивается на $3q$ зон по $p = \lceil \log_2(k+1) \rceil$ бит в каждой. Каждая из этих зон помечается некоторым элементом из $W \cup X \cup Y$ так, как указано на рис. 3.7. Двоичное представление числа $s(a_i)$ зависит от соответствующей тройки $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)}) \in M$, где f, g и h - функции, задающие индексы первой, второй и третьей компонент каждой тройки m_i. В числе $s(a_i)$ правые концы зон, соответствующих числам $w_{f(i)}$, $x_{g(i)}$ и $y_{h(i)}$ помечены 1, остальные знаки числа $s(a_i)$ - нули. Другими словами, $s(a_i) = 2^{p(3q-f(i))} + 2^{p(2q-g(i))} + 2^{p(q-h(i))}$.

Так как двоичная запись каждого из чисел $s(a_i)$ имеет длину не более $3pq$, то ясно, что $s(a_i)$ можно построить по

заданной индивидуальной задаче из 3-С за полиномиальное время.

На данной стадии конструкции важно отметить, что если просуммировать содержимое одной зоны всех элементов множества $\{a_i: 1 \le i \le k\}$, то результат всегда не будет превосходить 2^p-1. Следовательно, при суммировании $\Sigma\{s(a): a \in A'\}$ по некоторому подмножеству $A' \subseteq \{a_i: 1 \le i \le k\}$ никогда не придется переносить единицы из одной зоны в соседнюю.

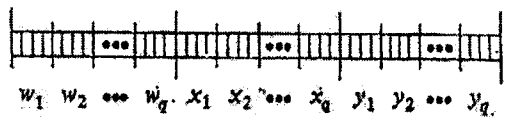


Рис.5.7 Двоичная запись числа $s(a)$ с помеченными $3q$ «зонами» с $p = \lceil log_2(k+1) \rceil$ двоичными знаками в каждой, которая используется в сведении 3-С к РАЗБИЕНИЕ.

Отсюда следует, что если положить $B = \sum_{i=0}^{3q-1} 2^{p_i}$ (B - число, в двоичной записи которого в правом конце каждой зоны стоит 1), то для любого подмножества $A' \subseteq \{a_i: 1 \le i \le k\}$ соотношение

$$\sum_{a \in A'} s(a) = B$$

выполняется тогда и только тогда, когда $M' = \{m_i: a_i \in A'\}$ - трехмерное сочетание для M.

На последней стадии конструкции строятся оставшиеся два элемента из A. Они определяются элементами b_1 и b_2, имеющими следующие размеры:

$$s(b_1) = 2\left(\sum_{i=1}^{k} s(a_i)\right) - B,$$

$$s(b_2) = \left(\sum_{i=1}^{k} s(a_i)\right) + B^{1)}.$$

Двоичные записи этих величин имеют длину не более $3pq+1$ и могут быть построены за время, зависящее полиномиальным образом от размера индивидуальной задачи из 3-С.

Теперь предположим, что имеется подмножество $A' \subseteq A$, такое, что

$\sum_{a \in A'} s(a) = B$. Тогда каждая из этих сумм должна быть равна $2\sum_{i=1}^{k} s(a_i)$, причем одно из множеств A' или $A \backslash A'$ содержит b_1 и не содержит b_2. Отсюда следует, что остальные элементы этого множества принадлежат множеству $\{a_i: 1 \le i \le k\}$. Сумма их размеров равна B и, в силу сделанного замечания, они образуют подмножество, соответствующее некоторому трехмерному сочетанию M' в M. Обратно, если задано трехмерное сочетание $M' \subseteq M$, то множество $\{b_1\} \cup \{a_i: m_i \in M\}$ образует искомое множество A' для заданной индивидуальной задачи из задачи ПАРОСОЧЕТАНИЕ. Таким образом, задача 3-С $\propto$ к задаче РАЗБИЕНИЕ и теорема **5.6** доказана.

ГЛАВА 6. МЕТОДЫ ДОКАЗАТЕЛЬСТВА NP-ПОЛНОТЫ

Методы, используемые при доказательстве результатов об NP-полноте, меняются в столь же широких пределах, как и сами NP-полные задачи, поэтому нет возможности проиллюстрировать их все. Однако имеется несколько общих приемов доказательства, которые часто встречаются и могут подсказать путь к доказательству NP-полноты новой задачи. Они называются: (а) сужение задачи, (б) локальная замена и (в) построение компоненты.

В настоящем разделе, в основном на примерах, мы объясним, что понимается под каждым из этих приемов доказательства. Было бы ошибкой пытаться дать их явное определение. Многие доказательства допускают

интерпретации, позволяющие причислить их к любой из этих трех категорий. Некоторые доказательства так сильно зависят от специфики описания задачи, что их не удаётся классифицировать естественным образом в столь ограниченном множестве категорий. Так что читателю не следует рассматривать нижеизложенное как попытку классификации всех доказательств NP-полноты. Наша цель - проиллюстрировать некоторые подходы к доказательству NP-полноты, которые с интуитивной точки зрения являются конструктивными и привлекательными.

В дальнейшем мы опускаем во всех доказательствах проверку того, что рассматриваемая задача принадлежит NP. Относительно каждой из рассматриваемых ниже задач легко установить разрешимость за полиномиальное время недетерминированным алгоритмом.

СУЖЕНИЕ ЗАДАЧИ

Из трех рассматриваемых приемов доказательства метод сужения - самый простой и, вероятно, наиболее часто встречается.

Доказательство методом сужения NP-полноты фиксированной задачи П$\in$NP заключается в установлении того, что задача П включает в качестве частного случая известную NP-полную задачу П'. Суть в том, чтобы указать дополнительные ограничения, которые требуется наложить на индивидуальные задачи из П, чтобы получившаяся в результате сужения задача была бы эквивалентна П'. Не требуется, чтобы возникающая в результате сужения задача была точной копией известной NP-полной задачи. Нужно, чтобы между задачами имелось очевидное взаимнооднозначное соответствие, сохраняющее ответы «да» или «нет».

Несколько примеров доказательств этого типа уже встречалось. NP-полнота задачи ТОЧНОЕ ПОКРЫТИЕ 3-МНОЖЕСТВАМИ была доказана путем рассмотрения среди

ее индивидуальных задач только тех, в которых трехэлементные множества содержат по одному элементу из множеств W, X и Y (где W, X, Y - равномощные непересекающиеся множества). Такие задачи идентичны задаче 3-С. Ранее NP-полнота задачи ОРИЕНТИРОВАННЫЙ ГАМИЛЬТОНОВ ЦИКЛ была доказана благодаря тому, что среди ее индивидуальных задач мы рассматривали только те, в которых ориентированный граф вместе с каждой дугой (u, v) содержал противоположно ориентированную дугу (v, u), т. е. индивидуальные задачи, в совокупности образующие задачу, идентичную задаче ГАМИЛЬТОНОВ ЦИКЛ.

Метод сужения можно рассматривать как иной взгляд на задачу, а не как стандартный способ доказательства NP-полноты. Вместо того чтобы пытаться свести одну из известных NP-полных задач к заданной, мы концентрируем внимание на последней и пытаемся отбросить несущественныё ее детали с тем, чтобы получилась известная NP-полная задача. Ниже приведено несколько примеров задач, NP-полнота которых доказывается методом сужения.

(1) МИНИМАЛЬНОЕ ПОКРЫТИЕ

УСЛОВИЕ. Задано семейство C подмножеств множества S и положительное целое число K.

ВОПРОС. Содержит ли C *покрытие* множества S размера не более K, т. е. найдется ли подмножество $C' \subseteq C$ такое, что $|C'| \leqslant K$ и $\bigcup_{c \in C'} c = S$?

Доказательство. Задача превращается в ТП-3, если ограничиться только теми индивидуальными задачами, у которых $|C| = 3$ для всех множеств $c \in C$ и $K = |S|/3$.

(2) МНОЖЕСТВО ПРЕДСТАВИТЕЛЕЙ

УСЛОВИЕ. Задано семейство C подмножеств множества S и положительное целое число K.

БОПРОС. Содержит ли S множество *представителей* для C мощности, не превосходящей K; т. е. существует ли подмножество $S' \subseteq S$, такое, что $|S'| \leq K$ и S' содержит по крайней мере один элемент из каждого множества семейства C?

Доказательство. Задача превращается в задачу ВП, если рассмотреть только те индивидуальные задачи, у которых $|c| = 2$ для всех $c \in C$.

(3) ИЗОМОРФИЗМ ПОДГРАФУ

УСЛОВИЕ. Заданы два графа $G = (V_1, E_1)$ и $H = (V_2, E_2)$.

ВОПРОС. Содержит ли граф G подграф, *изоморфный* H? Другими словами, существуют ли такие подмножества $V \subseteq V_1$, $E \subseteq E_1$ и взаимно-однозначная функция f: $V_2 \rightarrow V$, что $|V| = |V_2|$, $|E| = |E_2|$ тогда и только тогда, когда $\{f(u), f(v)\} \in E$?

Доказательство. Задача превращается в задачу КЛИКА, если рассматривать только те индивидуальные задачи, для которых H - полный граф, т. е. те индивидуальные задачи, в которых E_2 содержит все возможные ребра, соединяющие вершины V_2.

(4) ОСТОВНОЕ ДЕРЕВО ОГРАНИЧЕННОЙ СТЕПЕНИ

УСЛОВИЕ. Заданы граф $G=(V,E)$ и положительное целое число K, $K \leq |V| - 1$.

ВОПРОС. Существует ли в G *остовное дерево*, в котором всё вершины имеют степень не более K, т. е. такое подмножество $E \subseteq E'$, что $|E'|=|V|-1$, граф $G'=(V,E')$ связен и ни одна вершина в V не принадлежит более чем K ребрам из E'?

Доказательство. Если рассмотреть только те индивидуальные задачи, в которых K=2, то задача превращается в задачу ГАМИЛЬТОНОВ ПУТЬ.

(5)МИНИМАЛЬНЫЙ ЭКВИВАЛЕНТНЫЙ ОРИЕНТИРОВАННЫЙ ГРАФ

УСЛОВИЕ. Заданы ориентированный граф $G = (V,A)$ и положительное целое число $K \leq |A|$.

ВОПРОС. Существует ли ориентированный граф $G'=(V, A')$, такой, что $A' \subseteq A$, $|A'| \leq K$ и для каждой пары вершин u и v из V в G' имеется ориентированный путь от u к v тогда и только тогда, когда в G имеется ориентированный путь из u к v.

Доказательство. Если рассматривать только индивидуальные задачи, для которых граф G сильно связен, т. е. содержит ориентированный путь от любой вершины к любой другой вершине v и $|K| = |V|$, то задача превращается в задачу ОРИЕНТИРОВАННЫЙ ГАМИЛЬТОНОВ ЦИКЛ. Отметим, что при таком сужении фактически возникает задача ОРИЕНТИРОВАННЫЙ ГАМИЛЬТОНОВ ЦИКЛ В СИЛЬНО СВЯЗНОМ ОРИЕНТИРОВАННОМ ГРАФЕ, однако NP-полнота этой задачи немедленно следует из приведенных выше конструкций для задач ГЦ и ОРИЕНТИРОВАННЫЙ ГЦ.

(6) РЮКЗАК

УСЛОВИЕ. Задано конечное множество U, «размеры» $s(u) \in Z^+$, «стоимости» $v(u) \in Z^+$ для всех $u \in U$, общее ограничение B на размеры, $B \in Z^+$, и значение стоимости $K \in Z^+$.

ВОПРОС. Существует ли подмножество $U' \subseteq U$, такое, что

$$\sum_{u \in U'} s(u) \leqslant B \quad \text{и} \quad \sum_{u \in U'} v(u) \geqslant K?$$

Доказательство. Если ограничиться рассмотрением только таких индивидуальных задач, в которых для всех $u \in U$, $s(u)=v(u)$ и $B = K = \frac{1}{2}\Sigma_{u \in U} s(u)$, то задача превращается в задачу РАЗБИЕНИЕ.

(7)РАСПИСАНИЕ ДЛЯ МУЛЬТИПРОЦЕССОРНОЙ СИСТЕМЫ

УСЛОВИЕ. Задано конечное множество A заданий», «длительности» $l(a) \in Z^+$ для всех $a \in A$, число $m \in Z^+$ «процессоров» и «директивный срок» $D \in Z^+$.

ВОПРОС. Существует ли разбиение $A = A_1 \cup A_2 \cup, \ldots, \cup A_m$ множества A на m непересекающихся множеств, такое, что

$$\max \left\{ \sum_{a \in A_i} l(a): \ 1 \leqslant i \leqslant m \right\} \leqslant D?$$

Доказательство. Если рассматривать только те индивидуальные задачи, для которых $m = 2$ и

$$D = 1/2 \sum_{a \in A} l(a),$$

то задача превращается в задачу РАЗБИЕНИЕ.

В заключение заметим, что из трех обсуждаемых подходов к установлению NP-полноты метод сужения в наибольшей степени выигрывает от наличия обширного списка NP-полных задач (сверх основных шести задач и их вариантов). Многие возникающие на практике задачи представляют собой просто более сложные версии задач из списка NP-полных задач в приложении, и способность усмотреть этот факт во многих случаях позволяет быстро доказать NP-полноту методом сужения.

ЛОКАЛЬНАЯ ЗАМЕНА

Сводимости, возникающие при доказательстве методом локальной замены, достаточно нетривиальны, чтобы их всегда можно было представить в стандартном виде, однако они остаются относительно несложными. Этот метод состоит в том, что выбирается характерное свойство известной NP-полной задачи, с помощью него образуется семейство *основных модулей,* а соответствующие индивидуальные задачи заданной задачи получаются путем единообразной замены каждого основного модуля некоторой другой структурой. Сводимость задачи ВЫП к задаче 3-ВЫП относится к этому типу.

В ней основными модулями задачи ВЫП были дизъюнкции, и каждая дизъюнкция заменялась набором дизъюнкций согласно общему правилу. Важным моментом здесь является то, что каждая замена приводила только к локальному изменению структуры. Эти замены, по существу, не зависели одна от другой, за исключением тех случаев, когда они отражали не подвергавшиеся изменению части исходной индивидуальной задачи.

Конкретизируем эти общие рассуждения примерами. Сначала
рассмотрим задачу распознавания, которая соответствует задаче минимизации числа умножений, необходимых для вычисления заданного набора произведений элементарных термов. При этом предполагается, что операция умножения ассоциативна и коммутативна.

ВЫЧИСЛЕНИЕ СЕМЕЙСТВА

УСЛОВИЕ. Задано семейство C подмножеств конечного множества A и положительное целое число J.

ВОПРОС. Существует ли такая последовательность

$$\langle z_1 = x_1 \bigcup y_1,\ z_2 = x_2 \bigcup y_2,\ \ldots,\ z_l = x_l \bigcup y_l \rangle,$$

содержащая не более чем J операций типа объединение, что для каждого подмножества $c \in C$ найдется номер i, $1 \le i \le j$, такой, что z_i совпадает с c? (здесь x_i и y_i - это либо множества вида $\{a\}$ для некоторого $a \in A$, либо это множество вида z_k для некоторого $k<i$, причем x_i и y_i, $1 \le i \le j$, не пересекаются).

Теорема 6.1. *Задача ВЫЧИСЛЕНИЕ СЕМЕЙСТВА NP-полна.*

Доказательство. Сведем задачу ВЕРШИННОЕ ПОКРЫТИЕ к задаче ВЫЧИСЛЕНИЕ СЕМЕЙСТВА. Пусть граф $G=(V,E)$ и положительное целое число $K \le |V|$, определяют условие произвольной индивидуальной задачи из ВП.

В качестве основных модулей ВП возьмем ребра графа G. Пусть a_0 новый элемент, не принадлежащий V. Операция локальной замены замещает каждое ребро $\{u,v\}\in E$ подмножеством $\{a_0,u,v\}\in C$. В результате возникает индивидуальная задача из задачи ВЫЧИСЛЕНИЕ СЕМЕЙСТВА, условие которой полностью определяется следующими данными:

$$A = V \bigcup \{a_0\},$$

$$C = \{\{a_0,\ u,\ v\}:\ \{u,\ v\} \in E\},$$

$$J = K + |E|.$$

Легко видеть, что эту индивидуальную задачу можно построить за полиномиальное время. Утверждается, что в G есть вершинное покрытие не более чем из K элементов тогда и только тогда, когда для семейства C существует последовательность длины не более J, обладающая необходимыми свойствами. Вначале предположим, что V - вершинное покрытие в G, содержащее не более K вершин. Если к покрытию V добавить новые вершины, то оно по-прежнему, будет оставаться покрытием, поэтому без ограничения общности можно полагать, что $|V| = K$. Обозначим вершины покрытия V через $v_1, v_2, \ldots, v_K$, а ребра из множества E через $e_1, e_2, \ldots, e_m$, где $m=|E|$. Поскольку V - вершинное покрытие, то каждое ребро e_j инцидентно по крайней мере одной вершине из V. Поэтому каждое ребро e_j может быть представлено в виде $e_j=\{u_j, v_{r[j]}\}$, где $r[j]$ - целое число, заключенное между 1 и K. Нетрудно видеть, что приведенная ниже последовательность, состоящая из $K+|E| = J$ операций объединения, обладает всеми необходимыми свойствами:

$$\langle z_1 = \{a_0\} \bigcup \{v_1\},\ z_2 = \{a_0\} \bigcup \{v_2\},\ \ldots,\ z_K = \{a_0\} \bigcup \{v_K\},$$

$$z_{K+1} = \{u_1\} \bigcup z_{r[1]},\ z_{K+2} = \{u_2\} \bigcup z_{r[2]},\ \ldots,\ z_J = \{u_m\} \bigcup z_{r[m]}\rangle.$$

Обратно, предположим, что $S = \langle z_1=x_1\cup y_1, \ldots, z_j=x_j\cup y_j\rangle$ - искомая последовательность для индивидуальной

задачи из ВЫЧИСЛЕНИЕ СЕМЕЙСТВА, содержащая не более J операций объединения. Предположим далее, что последовательность S -кратчайшая из всех последовательностей для рассматриваемой индивидуальной задачи и что среди всех таких кратчайших последовательностей S содержит минимальное число операции вида $z_i=\{u\}\cup\{v\}$, u, $v\in V$. Первое утверждение состоит в том, что в S вообще нет операций этого вида. Действительно, предположим, что операция $z_i=\{u\}\cup\{v\}$, u, $v\in V$, содержится в S. Поскольку $\{u, v\}$ не принадлежит C, а S имеет минимальную возможную длину, то с необходимостью имеем $\{u, v\}\in E$ и операция $\{a_0, u, v\} = \{a_0\}\cup z_i$ (или $z_i\cup\{a_0\}$) должна содержаться в S. Но поскольку $\{u, v\}$ - подмножество только одного элемента из C, то z_i не может участвовать ни в какой другой операции рассматриваемой последовательности S. Отсюда следует, что пару операций

$$z_i = \{u\} \bigcup \{v\} \text{ и } \{a_0, u, v\} = \{a_0\} \bigcup z_i$$

можно заменить другой парой операций

$$z_i = \{a_0\} \bigcup \{u\} \text{ и } \{a_0, u, v\} = \{v\} \bigcup z_i,$$

при этом длина всей последовательности S не увеличится, а общее число операций вида $\{u, v\}$, u, $v\in V$ уменьшится, что противоречит выбору S. Следовательно, S состоит только из операций, имеющих вид $z_i=\{a_0\}\cup\{u\}$, $u\in V$, или $\{a_0,u,v\}=\{v\}\cup z_i$, $\{u,v\}\in E$ (пренебрегаем порядком операндов). Так как $|C| = |E|$ и любой элемент из C содержит три элемента, то S должно содержать в точности $|E|$ операций второго вида и в точности $J-|E|\le K$ операций первого вида. Поэтому множество

$$V' = \{u \in V: z_i = \{a_0\} \bigcup \{u\} \text{ есть операция из } S\}$$

содержится самое большое K вершин из V. Исходя из способа построения C нетрудно проверить, что V' должно быть вершинным покрытием в G.

Приведем другой пример полиномиальной сводимости, которая получается из задачи ТОЧНОЕ ПОКРЫТИЕ ТРЕХ-

ЭЛЕМЕНТНЫМИ МНОЖЕСТВАМИ методом локальной замены.

РАЗБИЕНИЕ НА ТРЕУГОЛЬНИКИ

УСЛОВИЕ. Задан граф $G=(V, E)$, число вершин которого $|V|$ делится на три, т. е. $|V|=3q$ для некоторого целого числа q.

ВОПРОС. Существует ли такое разбиение V на q непересекающихся подмножеств $V_1, V_2, \ldots, V_q$ (содержащих по 3 элемента), что для каждого $V_i = \{v_{i[1]}, v_{i[2]}, v_{i[3]}\}$ три ребра $\{v_{i[1]}, v_{i[2]}\}$, $\{v_{i[1]}, v_{i[3]}\}$ и $\{v_{i[2]}, v_{i[3]}\}$ принадлежат E?

Теорема 6.2. *Задача РАЗБИЕНИЕ НА ТРЕУГОЛЬНИКИ NP-полна.*

Доказательство. Сведем задачу ТОЧНОЕ ПОКРЫТИЕ ТРЕХЭЛЕМЕНТНЫМИ множествами к задаче РАЗБИЕНИЕ НА ТРЕУГОЛЬНИКИ. Пусть множество X ($|X|=3q$) и семейство C его трехэлементных подмножеств образуют условие произвольной индивидуальной задачи из ТП-3.

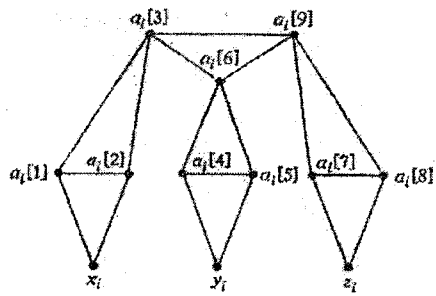


Рис. 6.1 Элемент, используемый для локальной замены подмножества $c_i=(x_i, y_i, z_i)\in C$ при построении сведения задачи ТП-3 к задаче РАЗБИЕНИЕ НА ТРЕУГОЛЬНИКИ.

Построим граф $G=(V,E)$, такой, что $|V| = 3q'$ и для него существует искомое разбиение тогда и только тогда, когда C содержит точное покрытие. Основными модулями рассматриваемой индивидуальной задачи из ТП-3 будут трехэлементные подмножества из C. Операция локальной

замены превращает каждое такое подмножество $C_i = \{x_i, y_i, z_i\} \in C$ в набор E_i, состоящий из 18 ребер, который изображен на рис. 6.1. Таким образом, $G=(V,E)$ определяется так:

$$V = X \cup \bigcup_{i=1}^{|C|} \{a_i[j]: 1 \leqslant j \leqslant 9\},$$

$$E = \bigcup_{i=1}^{|C|} E_i.$$

Заметим, что только вершины из множества X могут быть инцидентны более чем одному множеству E_i. Отметим также, что

$$|V| = |X| + 9|C| = 3q + 9|C|,$$

так что $q'=q+3|C|$. Нетрудно видеть, что данную индивидуальную задачу РАЗБИЕНИЕ НА ТРЕУГОЛЬНИКИ можно построить, отправляясь от рассматриваемой индивидуальной задачи из ТП-3, за полиномиальное время.

Если $c_1, c_2, \ldots, c_q$ - трехэлементные подмножества из C, которые образуют точное покрытие X, то соответствующее разбиение $V = V_1 \cup V_2 \cup, \ldots, \cup V_q$ множества V определяется следующим образом. Если $c_i=\{x_i,y_i,z_i\}$ есть элемент точного покрытия, то из множества E_i выбираются следующие подмножества:

$$\{a_i[1], a_i[2], x_i\}, \{a_i[4], a_i[5], y_i\},$$
$$\{a_i[7], a_i[8], z_i\}, \{a_i[3], a_i[6], a_i[9]\},$$

а если c_i не является элементом точного покрытия, то из E_i выбираются подмножества

$$\{a_i[1], a_i[2], a_i[3]\}, \{a_i[4], a_i[5], a_i[6]\},$$
$$\{a_i[7], a_i[8], a_i[9]\}.$$

Такой выбор подмножеств обеспечивает, что каждый элемент из X содержится ровно в одном трехвершинном подмножестве рассматриваемого разбиения.

Обратно, если $V = V_1 \cup V_2 \cup$, ... ,$\cup V_q$ - произвольное разбиение графа G на треугольники, то соответствующее точное покрытие получается, если выбирать те $c_i \in C$, для которых $\{a_i[3],a_i[6],a_i[9]\}=V_i$ при некотором i, $1 \leq i \leq q'$. Непосредственную проверку того, что два построенных разбиения обладают всеми нужными свойствами, оставляем читателю.

Оба только что приведенных примера представляют собой доказательство методом локальной замены «в чистом виде». Структура рассматриваемой индивидуальной задачи однозначно определялась структурой исходной задачи (NP-полнота которой установлена) и операцией локальной замены. Часто оказывается полезным дополнить рассматриваемую индивидуальную задачу вспомогательными элементами, которые играют роль «ограничителя», налагающего дополнительные ограничения на способы получения ответа «да».

Если рассматриваемая задача имеет вид «задана индивидуальная задача I; верно ли, что существует некоторый объект X_i с требуемым свойством?», то роль ограничителя в задаче I - сузить множество допустимых объектов X_i так, чтобы они отражали все варианты, возможные в исходной индивидуальной задаче. В то же время часть индивидуальной задачи I, получающаяся операцией локальной замены из исходной задачи, дает различные варианты этих ответов и обеспечивает выполнение условий, требующихся от этих ответов. Элементы b_1 и b_2 из доказательства NP-полноты задачи РАЗБИЕНИЕ как раз и играют роль подобного ограничителя. Приведем еще два примера на доказательство методом локальной замены с использованием ограничителей. Сначала рассмотрим следующую задачу о составлении расписания.

УПОРЯДОЧЕНИЕ ВНУТРИ ИНТЕРВАЛОВ

УСЛОВИЕ. Задано конечное множество «заданий» T и для каждого $t \in T$ заданы целое число $r(t) \geq 0$ - время готовности, «директивный срок» $d(t) \in Z^+$ и «длительность» $l(t) \in Z^+$.

ВОПРОС. Существует ли для T допустимое расписание, т. е. такая функция $\sigma: T \to Z^+$, что для всех $t \in T$ имеет место:

(1) $\sigma(t) \geq r(t)$,

(2) $\sigma(t)+l(t) \leq d(t)$,

(3) если $t' \in T \setminus \{t\}$, то либо $\sigma(t')+l(t') \leq \sigma\{t)$, либо $\sigma(t') \geq \sigma(t)+l(t)$?

(другими словами, задание t выполняется с момента времени $\sigma(t)$ до момента $\sigma(t)+l(t)$; оно не может быть начато ранее момента $r(t)$, должно быть закопчено не позднее $d(t)$ и временной интервал выполнения задания t не может перекрываться с интервалом выполнения другого задания t'.)

Теорема 6.3. *Задача УПОРЯДОЧЕНИЕ ВНУТРИ ИНТЕРВАЛОВ NP-полна.*

Доказательство. Сведем к этой задаче задачу РАЗБИЕНИЕ. Пусть конечное множество A и размеры $s(a)$ всех элементов из A составляют условие индивидуальной задачи РАЗБИЕНИЕ и пусть также

$$B = \sum_{a \in A} s(a).$$

Возьмем в качестве базисных модулей исходной индивидуальной задачи РАЗБИЕНИЕ отдельные элементы $a \in A$. Операция локальной замены превращает a в задание t_a, такое, что $r(t_a)=0$, $d(t_a)=B+l$ и $l(t_a)=s(a)$. В качестве «ограничителя» возьмем еще одно задание $\bar{t}$ такое, что $r(\bar{t}) = \lceil B/2 \rceil$, $d(\bar{t}) = \lceil (B+1)/2 \rceil$ и $l(\bar{t})=1$. Ясно, что эта индивидуальная задача может быть построена по исходной индивидуальной задаче РАЗБИЕНИЕ за полиномиальное время.

Ограничения, которые налагает ограничитель на допустимые расписания, имеют двоякий характер. Во-первых, ограничитель не позволяет построить допустимое расписание,

если B нечетно (в этом случае для рассматриваемой задачи РАЗБИЕНИЕ искомое разбиение на два подмножества не существует) по той причине, что если B нечетно, то $r(\vec{t}) = d(\vec{t})$ и $\vec{t}$ вообще не может быть включено в расписание. Поэтому с настоящего момента будем предполагать, что B четно. В этом случае на передний план выступает второе ограничение. Так как B четно, то $r(\vec{t})=B/2$ и $d(\vec{t})=r(\vec{t})+1$, поэтому в любом допустимом расписании $\sigma(\vec{t})$ должно быть равно $B/2$. Отсюда время, которое доступно для выполнения остальных заданий, делится на два отдельных блока, каждый из которых, как показано на рис. 6.2,

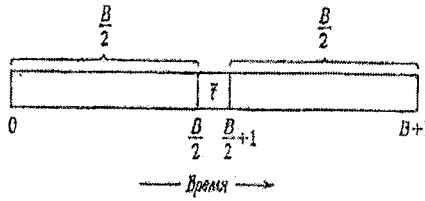


Рис. 6.2 Расписание, получаемое с помощью «ограничителя» при сведении задачи РАЗБИЕНИЕ к задаче УПОРЯДОЧЕНИЕ ВНУТРИ ИНТЕРВАЛОВ.

имеет длину $B/2$. Таким образом, задача составления расписания превращается в задачу выбора подмножества заданий, которые будут выполняться раньше задания $\vec{t}$ и подмножества заданий, которые будут выполняться после $\vec{t}$. Поскольку общее время, имеющееся в обоих блоках, равно общей длительности B оставшихся заданий, то отсюда следует, что каждый из этих блоков должен быть полностью заполнен. Это возможно тогда и только тогда, когда существует подмножество $A' \subseteq A$ такое, что

$$\sum_{a \in A'} s(a) = B/2 = \sum_{a \in A \setminus A'} s(a).$$

Таким образом, для рассматриваемой индивидуальной задачи РАЗБИЕНИЕ

искомое подмножество A' существует тогда и только тогда, когда для соответствующей индивидуальной задачи УПОРЯДОЧЕНИЕ ВНУТРИ ИНТЕРВАЛОВ существует допустимое расписание.

В качестве заключительного примера использования ограничителя в методе локальной замены рассмотрим следующую задачу, возникающую в диагностическом тестировании.

МИНИМАЛЬНЫЙ НАБОР ТЕСТОВ

УСЛОВИЕ. Задано конечное множество A «возможных диагнозов», набор C подмножеств множества A, представляющий бинарные «тесты» и положительное целое число $J \le |C|$.

ВОПРОС. Существует ли поднабор $C' \subseteq C$, $|C'| \le J$, такой, что для любой пары a_i, a_j возможных диагнозов из A имеется некоторый тест $c \in C'$, для которого $|\{a_i,a_j\} \cap c| = 1$ (т. е. тест c, различающий a_i и a_j)?

Теорема 6.4. *Задача МИНИМАЛЬНЫЙ НАБОР ТЕСТОВ NP-полна.*

Доказательство. Сведем к этой задаче задачу 3-С. Пусть множества W, X, Y такие, что $|W|=|X|=|Y|=q$ и подмножество $M \subseteq W \times X \times Y$ составляют условие индивидуальной задачи из 3-С.

В качестве основного модуля задачи 3-С возьмем упорядоченные тройки из множества M. Операция локальной замены вместо каждой тройки $m = (w, x, y) \in M$ образует подмножество $\{w,x,y\} \in C$. Роль ограничителей играют три дополнительных элемента w_0, x_0 и y_0, не принадлежащие $W \cup X \cup Y$, и два дополнительных теста $W \cup \{w_0\}$ и $X \cup \{x_0\}$. Окончательно индивидуальная задача МИНИМАЛЬНЫЙ НАБОР ТЕСТОВ определяется следующим образом:

$$A = W \cup X \cup Y \cup \{w_0, x_0, y_0\},$$
$$C = \{\{w, x, y\}: (w, x, y) \in M\} \cup \{W \cup \{w_0\}, X \cup \{x_0\}\},$$
$$J = q + 2.$$

Легко видеть, что эта индивидуальная задача может быть построена по исходной задаче 3-С за полиномиальное время. Ограничитель налагает определенные ограничения на вид возможных компонент (в данном случае на набор тестов C). Во-первых, набор C должен содержать как $W \cup \{w_0\}$, так и $X \cup \{x_0\}$ по той причине, что только эти тесты отличают y_0 от w_0 и x_0. Далее, поскольку w_0, x_0, y_0 не принадлежат больше ни одному тесту из C, то каждый элемент множества $W \cup X \cup Y$ должен отличаться от одного элемента w_0, x_0 или y_0 ввиду того, что он входит в некоторый дополнительный тест $c \in C' \setminus \{W \cup \{w_0\}, X \cup \{x_0\}\}$. При этом может быть использовано самое большее $J-2=q$ дополнительных тестов. Поскольку каждый из оставшихся тестов содержит ровно по одному элементу из множеств W, X и Y, и, кроме того, множества W, X и Y попарно не пересекаются и каждое из них содержит ровно q элементов, то отсюда следует, что любые q дополнительных тестов из C' должны соответствовать q тройкам из M, образующим трехмерное сочетание. Обратно, если задано любое трехмерное сочетание из M, то соответствующие q тестов из C можно использовать для завершения построения оставшихся тестов в нужном количестве. Таким образом, M содержит трехмерное сочетание тогда и только тогда, когда в множестве C существует искомый набор тестов.

Хотя в обоих рассмотренных примерах ограничители очень просты, нужно отдавать себе отчет в том, что это не всегда так. Ограничитель особо сложной конструкции используется в доказательстве NP-полноты задачи ОРИЕНТИРОВАННЫЙ ГАМИЛЬТОНОВ ПУТЬ В ПЛОСКОМ ГРАФЕ. Бывают и другие довольно сложные ограничители.

ПОСТРОЕНИЕ КОМПОНЕНТ

Последним и наиболее сложным из рассматриваемых нами методов доказательств NP-полноты является метод построения компонент. Доказательства NP-полноты задач ТРЕХМЕРНОЕ СОЧЕТАНИЕ, ВЕРШИННОЕ ПОКРЫТИЕ и ГАМИЛЬТОНОВ ЦИКЛ, приведенные ранее, представляют собой типичные примеры доказательств этого типа.

Основная идея таких доказательств заключается в том, чтобы с помощью составных частей рассматриваемой задачи сконструировать некоторые «компоненты», соединяя которые можно «реализовать» индивидуальные задачи известной NP-полной задачи. В трех перечисленных примерах имеются компоненты двух основных типов. Одну из них можно рассматривать как компоненту, «делающую выбор» (например, выбирающую вершины, выбирающую значение истинности переменных), а другую - как компоненту, «проверяющую свойства» (например, проверяющую, что каждое ребро покрыто или что каждая дизъюнкция выполнена).

В рассматриваемой индивидуальной задаче эти компоненты связаны так, что выбранные значения передаются компонентам, проверяющим условия, и последние проверяют, удовлетворяют ли сделанные выборы значений необходимым условиям. Взаимодействие компонент осуществляется с помощью прямых связей (таких, например, как ребра, соединяющие компоненты набора значений истинности с компонентами проверки выполнения, условий в сводимости задачи 3-ВЫП к задаче ВП, а также с помощью глобальных ограничений (таких, например, как общая граница K в сводимости задачи 3-ВЫП к задаче ВП, которая наряду со структурой компонент обеспечивает, чтобы каждая компонента наборов значений истинности содержала в точности одну вершину из покрытия и каждая компонента

проверки свойств содержала ровно две вершины этого покрытия).

Вообще говоря, любое доказательство можно считать основанным на методе построения компонент, если конструируемая в нем индивидуальная задача представляет собой набор компонент, каждая из которых выполняет определенные функции, формулируемые в терминах исходной задачи. Общая сводимость, примененная ранее для доказательства теоремы Кука, является хорошим примером доказательства этого типа. При этом каждая из шести групп дизъюнкций представляет собой один из типов компонент.

Поскольку доказательства методом построения компонент часто оказываются довольно длинными и примеры таких доказательств уже рассматривались, то в настоящем разделе мы приведем еще только один пример. Этот заключительный пример, отличающийся от стандартных, иллюстрирует подход, который оказался полезным для сведения задачи КЛИКА к другим задачам. Здесь доказывается NP-полнота задачи теории расписаний, близкой по формулировке к задаче УПОРЯДОЧИВАНИЕ ВНУТРИ ИНТЕРВАЛОВ, рассмотренной в предыдущем подразделе.

УПОРЯДОЧЕНИЕ С МИНИМАЛЬНЫМ ЗАПАЗДЫВАЕМ.

УСЛОВИЕ. Заданы множество T «заданий», каждое из которых имеет длительность 1 и «директивный» срок $d(t) \in Z$, частичное упорядочение $<$ на T и неотрицательное целое число $K \leq T$.

ВОПРОС. Существует ли «расписание» $\sigma : T \to \{0,1,\ldots, |T|-1\}$ такое, что:

(1) $\sigma(t) \neq \sigma(t')$ при $t \neq t'$;

(2) $\sigma(t) < \sigma(t')$ при $t < t'$;

3) $|\{t \in T: \sigma(t)+1 > d(t)\}| \leq K$?

Теорема 6.5. *Задача УПОРЯДОЧЕНИЕ С МИНИМАЛЬНЫМ ЗАПАЗДЫВАНИЕМ NP-полна*

Доказательство. Пусть граф $G = (V, E)$ и положительное целое число J, $J \le |V|$, образуют произвольную индивидуальную задачу КЛИКА. Соответствующая индивидуальная задача из задачи УПОРЯДОЧЕНИЕ С МИНИМАЛЬНЫМ ЗАПАЗДЫВАНИЕМ имеет множество заданий $T = V \cup E$, $K = |E| - J(J-1)/2$. Частичное упорядочение заданий определяется следующим образом:

$t < \cdot t' \Leftrightarrow t \in V$, $t' \in E$ и вершина t инцидентна ребру t';

$$d(t) = \begin{cases} J(J+1)/2, & \text{если } t \in E; \\ |V| + |E| & \text{если } t \in V. \end{cases}$$

Таким образом, «компонента», соответствующая каждой вершине, состоит из единственного задания с директивным сроком $|V| + |E|$, а «компонента», соответствующая каждому ребру, состоит из единственного задания с директивным сроком $J(J+1)/2$. Благодаря наличию частичного порядка на множестве T в искомом расписании задание, соответствующее ребру, следует за заданиями, соответствующими его концам, и поэтому задания, для которых имеется опасность запаздывания (т. е. завершения уже после директивного срока), обязательно соответствуют ребрам. Искомое расписание удобно представлять себе схематически, как показано на рис. 6.3. Часть расписания до директивного срока заданий, соответствующих ребрам, можно рассматривать как «компоненту выбора клики».

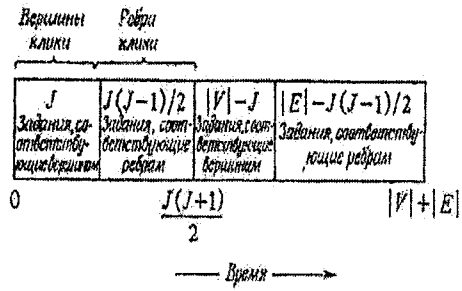


Рис. 6.3 Диаграмма искомого расписания для индивидуальной задачи УПОРЯДОЧЕНИЕ С МИНИМАЛЬНЫМ ЗАПАЗДЫВАНИЕМ, соответствующая КЛИКЕ размера J.

До этого срока может быть выполнено не более $J(J+1)/2$ заданий. Для того чтобы число запоздавших заданий не превосходило заданной величины K, по крайней мере $J(J-1)/2$ из «ранних» заданий должны соответствовать ребрам. Однако если задание, соответствующее ребру, выполняется ранее своего директивного срока, то и задания, соответствующие его концевым вершинам, также должны быть закопчены раньше этого срока. Минимально возможное число вершин, которые могут быть инцидентны $J(J-1)/2$ различным ребрам, равно J (причем это может быть тогда и только тогда, когда эти ребра образуют полный граф на этих J вершинах). Отсюда следует, что среди «ранних» заданий должно быть но крайней мере J заданий, соответствующих вершинам. Однако до директивного срока заданий, соответствующих ребрам, может быть выполнено не более

$$J(J+1)/2 - J(J-1)/2 = J$$

заданий, которые отвечают вершинам. Следовательно, в любом таком расписании до директивного срока заданий, соответствующих ребрам, должно закончиться ровно J заданий, отвечающих вершинам, и ровно $J(J-1)/2$ заданий, соответствующих ребрам, которые в совокупности соответствуют клике на J вершинах в графе G.

Обратно, если граф G содержит полный подграф на J вершинах, то искомое расписание можно построить способом, указанным на рис. 6.3.

УПРАЖНЕНИЯ

Приведём двенадцать NP-полных задач, доказательства NP-полноты которых предоставляются читателю. Ни одна из этих задач не требует сложного доказательства, поэтому мы советуем попытаться найти их все. В этих доказательствах в качестве известных NP-полных задач достаточно использовать только задачи, упоминавшиеся ранее. Для облегчения решения задачи сгруппированы по методам доказательства. Однако читатель может не обращать внимания на эти указания, если другой путь доказательства покажется ему более перспективным.

Метод сужения

1. САМЫЙ ДЛИННЫЙ ПУТЬ
УСЛОВИЕ. Задачи граф $G = (V, E)$ и положительное целое число $K \leqslant |V|$.
ВОПРОС. Имеется ли в G простой путь (т. е. путь, не проходящий дважды ни через одну вершину), состоящий не менее чем из K ребер?

2. УПАКОВКА МНОЖЕСТВ
УСЛОВИЕ. Заданы семейство C конечных множеств и положительное целое число K, $K \leqslant |C|$.
ВОПРОС. Верно ли, что в C имеется K непересекающихся множеств?

3. РАЗБИЕНИЕ НА ГАМИЛЬТОНОВЫ ПОДГРАФЫ
УСЛОВИЕ. Заданы граф $G=(V, E)$ и положительное целое число K, $K \leqslant |V|$.
ВОПРОС. Можно ли множество вершин графа G разбить на $k \leqslant K$ непересекающихся подмножеств $V_1, V_2, \ldots, V_k$ так, чтобы для всех i $(1 \leqslant i \leqslant k)$ каждый подграф, индуцированный под множеством V_i, содержал гамильтонов цикл?

4. НАИБОЛЬШИЙ ОБЩИЙ ПОДГРАФ
УСЛОВИЕ. Заданы графы $G_1=(V_1, E_1)$, $G_2=(V_2, E_2)$ и положительное целое число K.
ВОПРОС. Существуют ли такие подмножества $E_1' \subseteq E_1$ и $E_2' \subseteq E_2$, что $|E_1'|=|E_2'| \geqslant K$, а подграфы $G_1'=(V_1, E_1')$ и $G_2'=(V_2, E_2')$ изоморфны?

5. МИНИМУМ СУММЫ КВАДРАТОВ
УСЛОВИЕ. Заданы конечное множество A, «размер» $s(a) \in Z^+$ для каждого $a \in A$ и положительные целые числа K и J.
ВОПРОС. Могут ли элементы из A быть разбиты на K непересекающихся множеств $A_1, A_2, \ldots, A_K$ так, что $\sum_{i=1}^{K}\left(\sum_{a \in A_i} s(a)\right)^2 \leqslant J$?

Метод локальной замены

6. МНОЖЕСТВО ВЕРШИН, РАЗРЕЗАЮЩИХ КОНТУРЫ
УСЛОВИЕ. Заданы ориентированный граф $G=(V, A)$ и положительное целое число K, $K \leqslant |V|$.
ВОПРОС. Существует ли такое подмножество $V' \subseteq V$, что $|V'| \leqslant K$ и любой ориентированный цикл в G содержит по крайней мере одно ребро из V'?

7. ТОЧНОЕ ПОКРЫТИЕ ЧЕТЫРЕХЭЛЕМЕНТНЫМИ МНОЖЕСТВАМИ
УСЛОВИЕ. Задано конечное множество X, $|X|=4q$, целое число q и семейство C четырехэлементных подмножеств множества X.
ВОПРОС. Существует ли подсемейство $C' \subseteq C$, такое, что каждый элемент из X принадлежит в точности одному элементу из C'?

8. ДОМИНИРУЮЩЕЕ МНОЖЕСТВО
УСЛОВИЕ. Заданы граф $G = (V, E)$ и целое число K, $K \leqslant |V|$.
ВОПРОС. Существует ли такое подмножество $V' \subseteq V$, что $|V'| \leqslant K$ и каждая вершина $v \in V \setminus V'$ соединена ребром по крайней мере с одной вершиной из V'?

9. ДЕРЕВО ШТЕЙНЕРА
УСЛОВИЕ. Заданы граф $G = (V, E)$, подмножество $R \subseteq V$ и положительное целое число $K \geqslant |V| - 1$.
ВОПРОС. Существует ли поддерево в G, содержащее все вершины из R и имеющее не более K ребер?

10. НЕЭКВИВАЛЕНТНОСТЬ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ, НЕ СОДЕРЖАЩИХ ЗВЕЗДОЧЕК
УСЛОВИЕ. Заданы два не содержащие звездочек регулярных выражения E_1 и E_2 в конечном алфавите Σ. Такое выражение определяется следующим образом: (1) любой символ σ алфавита Σ есть не содержащее звездочек регулярное выражение, (2) если e_1 и e_2 — два не содержащие звездочек регулярных выражения, то и слова e_1e_2 и $(e_1 \vee e_2)$ также не содержащие звездочек регулярные выражения.
ВОПРОС. Верно ли, что E_1 и E_2 представляют различные языки в алфавите Σ? (Язык, представляемый символом $\sigma \in \Sigma$, есть $\{\sigma\}$, а если e_1 и e_2 представляют соответственно языки L_1 и L_2, то e_1e_2 представляет язык $\{xy: x \in L_1, y \in L_2\}$, а $(e_1 \vee e_2)$ представляет язык $L_1 \cup L_2$.)

Метод построения компонент

11. РАСЩЕПЛЕНИЕ МНОЖЕСТВА
УСЛОВИЕ. Задано семейство C подмножеств конечного множества S.
ВОПРОС. Существует ли разбиение S на две части S_1 и S_2, такое, что ни одно подмножество из C не содержится ни в S_1 ни в S_2?
Указание. Использовать 3-ВЫП.

12. РАЗБИЕНИЕ НА ПУТИ ДЛИНЫ 2
УСЛОВИЕ. Задан граф $G = (V, E)$, где $|V| = 3q$ для некоторого целого положительного числа q.

ВОПРОС. Существует ли разбиение V на q непересекающихся подмножеств $V_1, V_2, \ldots, V_q$, по три вершины в каждом, такое, что для всех $V_i = \{v_{i[1]}, v_{i[2]}, v_{i[3]}\}$ по крайней мере два из трех ребер $\{v_{i[1]}, v_{i[2]}\}$, $\{v_{i[1]}, v_{i[3]}\}$, $\{v_{i[2]}, v_{i[3]}\}$ принадлежат E?
Указание. Использовать 3-С.

АППЕНДИКС: ПРИМЕНЕНИЕ ТЕОРИИ NP-ПОЛНОТЫ ДЛЯ АНАЛИЗА ЗАДАЧ

Перейдем к использованию теории NP-полноты для анализа задач. Из рассуждений, приведенных ранее, следует, что, встретившись с новой задачей, естественно сначала задать вопрос: «Можно ли рассматриваемую задачу решить полиномиальным алгоритмом?» Если ответ на этот вопрос оказывается положительным, то с точки зрения теории NP-полноты ничего большего о задаче сказать нельзя. Дальнейшие усилия мы можем сконцентрировать на поиске как можно более эффективных полиномиальных алгоритмов. Но если для решения задачи не известно полиномиального алгоритма, естественно возникает второй вопрос: «Верно ли, что рассматриваемая задача NP-полна?»

Чтобы поставленный вопрос имел смысл, предположим, что рассматриваемая задача сформулирована как задача распознавания и что она принадлежит классу NP. В некоторых случаях легко устанавливается полиномиальная разрешимость интересующей нас задачи, в других же оказывается очевидной ее NP-полнота. Если задача оказалась NP-полной, то это является сильным аргументом в пользу того, что ее нельзя решить за полиномиальное время. В большинстве случаев, однако, найти ответ на каждый из поставленных вопросов довольно трудно. Обычно совсем не очевидно, что задача разрешима за полиномиальное время или что она NP-полна, а требуется некоторая работа для выяснения, какая из этих двух возможностей в действительности реализуется (в том случае, конечно, если реализуется хотя бы одна из них). Напомним, что ранее говорилось о том, что если $P \neq NP$, то в классе NP существуют задачи, которые неразрешимы за полиномиальное время и не NP-полны). Как в этом случае выяснить сложность рассматриваемой задачи?

Если интуитивно мы склоняемся в пользу одной из возможностей, то очень соблазнительно сконцентрировать усилия в этом направлении. Однако в подобных вопросах интуиция оказывается особенно обманчивой, ибо зачастую

задачи, разрешимые за полиномиальное время, очень незначительно отличаются от NP-полных задач. Например, мы видели, что задачи 3-ВЫПОЛНИМОСТЬ (3-ВЫП) и ТРЕХМЕРНОЕ СОЧЕТАНИЕ (3-С) NP-полны, а близкие задачи 2-ВЫПОЛНИМОСТЬ и ПАРОСОЧЕТАНИЕ разрешимы за полиномиальное время. На рис. 6.4 указано несколько других пар задач, одна из которых принадлежит классу P, а другая NP-полна.

Наша интуиция основана на опыте работы с близкими задачами. По этой причине мы серьезно рискуем сбиться с правильного пути, если позволим себе, доверяясь интуиции, сконцентрироваться на исследовании только одной возможности.

Таким образом, при анализе задач лучше пользоваться двусторонним подходом. Пытаясь с одной стороны доказать NP-полноту задачи, одновременно целесообразно искать полиномиальный алгоритм ее решения. Конечно, в каждый данный момент мы выбираем направление исследований исходя из того, что представляется нам перспективнее, но нужно быть готовым сменить направление исследований. В действительности, по мере того как мы будем переходить от одного направления к другому и обратно, оба направления, взаимодействуя друг с другом, сливаются в единое целое. Неудачи в доказательстве NP-полноты задачи могут породить идею алгоритма ее решения, а неудачи в построении алгоритма могут указать путь к доказательству NP-полноты задачи. Любые частичные результаты, полученные в процессе подобных исследований, особенно результаты о «нормальной форме» решений, могут оказаться в равной степени полезными как для доказательства NP-полноты задачи, так и для построения эффективного алгоритма ее решения.

Само собой разумеется, что успешное применение на практике такого двустороннего подхода требует от исследователя мастерства как в отыскании доказательств NP-полноты, так и в построении полиномиальных алгоритмов. О

методах доказательства NP-полноты было достаточно много сказано выше. Мы сосредоточим внимание на следующем вопросе: если доказана NP-полнота некоторой задачи (что бывает очень часто), то как использовать рассмотренный выше двусторонний подход для более глубокого анализа этой задачи?

Сначала обсудим вопрос, каким образом можно глубже прозондировать сложность NP-полной задачи, изучая ее подзадачи и пытаясь разграничить полиномиально разрешимые и NP-полные подзадачи. Затем рассмотрим подзадачи специального вида, часто заслуживающие внимания в тех случаях, когда в исходной задаче существенную роль играют целые числа. Такой подход приводит к понятиям «алгоритм с псевдополиномиальным временем работы» и «сильная NP-полнота», а также к дополнению списка основных NP-полных задач седьмой задачей. Обсудим и вопрос использования подзадач при изучении влияния на сложность задачи ее индивидуальных параметров (а не только общей «длины входа»).

P	NP-полные
КРАТЧАЙШИЙ ПУТЬ МЕЖДУ ДВУМЯ ВЕРШИНАМИ УСЛОВИЕ. Заданы: граф $G = (V, E)$, длины $l(e) \in \mathbf{Z}^+$ для всех $e \in E$, выделенные вершины $a, b \in V$ и положительное целое число B. ВОПРОС. Существует ли в G простой путь из a в b, имеющий длину не более B?	**САМЫЙ ДЛИННЫЙ ПУТЬ МЕЖДУ ДВУМЯ ВЕРШИНАМИ** УСЛОВИЕ. Заданы: граф $G = (V, E)$, длины $l(e) \in \mathbf{Z}^+$ для всех $e \in E$, выделенные вершины $a, b \in V$ и положительное целое число B. ВОПРОС. Существует ли в G простой путь из a в b, имеющий длину не менее B?
РЕБЕРНОЕ ПОКРЫТИЕ УСЛОВИЕ. Заданы граф $G = (V, E)$ и положительное целое число K. ВОПРОС. Существует ли подмножество $E' \subseteq E$, такое, что $\lvert E' \rvert \leqslant K$ и для каждой вершины $v \in V$ имеется такое ребро $e \in E'$, что $v \in e$?	**ВЕРШИННОЕ ПОКРЫТИЕ** УСЛОВИЕ. Заданы граф $G = (V, E)$ и положительное целое число K. ВОПРОС. Существует ли подмножество $V' \subseteq V$, такое, что $\lvert V' \rvert \leqslant K$ и для любого ребра $e \in E$ имеется такая вершина $v \in V'$, что $v \in e$?
ТРАНЗИТИВНАЯ РЕДУКЦИЯ УСЛОВИЕ. Заданы ориентированный граф $G = (V, A)$ и положительное целое число K. ВОПРОС. Существует ли подмножество $A' \subseteq V \times V$, такое, что $\lvert A' \rvert \leqslant K$ и для всех $u, v \in V$ граф $G' = (V, A')$ содержит путь из u в v тогда и только тогда, когда граф G содержит такой путь?	**МИНИМАЛЬНЫЙ ЭКВИВАЛЕНТНЫЙ ОРИЕНТИРОВАННЫЙ ГРАФ** УСЛОВИЕ. Заданы ориентированный граф $G = (V, A)$ и положительное целое число K. ВОПРОС. Существует ли подмножество $A' \subseteq A$, такое, что $\lvert A' \rvert \leqslant K$ и для всех $u, v \in V$ граф $G' = (V, A')$, содержит путь из u в v тогда и только тогда, когда граф G содержит такой путь?
РАСПИСАНИЕ ДЛЯ ПРЯМОГО ДЕРЕВА ЗАДАНИЙ УСЛОВИЕ. Заданы: множество T заданий с единичной длительностью, директивный срок $d(t) \in \mathbf{Z}^+$ для каждой $t \in T$, частичный порядок $\prec$ на множестве T, относительно которого каждое задание имеет не более одного непосредственно следующего за ним, положительное целое число m. ВОПРОС. Можно ли составить для множества T расписание на m процессорах, на каждом из которых задания выполняются согласно заданному частичному порядку, так что все директивные сроки окажутся выполненными?	**РАСПИСАНИЕ ДЛЯ ОБРАТНОГО ДЕРЕВА ЗАДАНИЙ** УСЛОВИЕ. Заданы: множество T заданий с единичной длительностью, директивный срок $d(t) \in \mathbf{Z}^+$ для каждого $t \in T$, частичный порядок $\prec$ на множестве T, относительно которого каждое задание имеет не более одного непосредственного предшественника, положительное целое число m. ВОПРОС. Можно ли составить для множества T расписание на m процессорах, на каждом из которых задания выполняются согласно заданному частичному порядку, так что все директивные сроки окажутся выполненными?

Рис. 6.4 Пары близких задач, одна из которых принадлежит классу P, а другая NP-полна.

АНАЛИЗ ПОДЗАДАЧ

Предположим, нам удалось доказать, что интересующая нас задача NP-полна. При этом мы получаем полные ответы на оба вопроса, с которых начинался анализ вопросов. Однако это лишь первый шаг на пути решения задачи. Задача, которой мы занимаемся, часто получается как идеализация менее привлекательной прикладной задачи и некоторые детали, опущенные в процессе идеализации, могут изменить задачу так, что она станет разрешимой за полиномиальное время. Даже если это не так, у задачи могут иметься некоторые важные частные подзадачи, которые могут быть решены за полиномиальное время. Возможно, что индивидуальные задачи, плохо поддающиеся решению, встречаются относительно редко и имеют легко выявляемые особенности, позволяющие заблаговременно опознать такие задачи. Эти возможности могут быть изучены с помощью анализа подзадач интересующей нас задачи.

Согласно принятому нами описанию массовых задач распознавания, каждая из них состоит из двух частей: множества D всех индивидуальных задач и множества Y индивидуальных задач из D с ответом «да». *Подзадачей* задачи $\Pi = (D,Y)$ называется такая задача $\Pi'=(D', Y')$, что $D' \subseteq D$ и $Y'=Y \cap D'$, т.е. Π' есть подзадача задачи Π, если в ней сформулирован тот же вопрос, что и в Π, но только на подмножестве множества индивидуальных задач из Π.

Таким образом, подзадачи возникают всякий раз, когда на множество индивидуальных задач налагаются дополнительные ограничения. В задачах из теории графов, например, можно ограничиваться индивидуальными задачами, в которых графы планарны, или двудольны, или ациклични, или обладают некоторыми из этих свойств одновременно. В задачах, содержащих множества, можно ограничиться индивидуальными задачами, в которых мощность множеств не превосходит заданной величины, или задачами, в которых

все элементы содержатся не более чем в ограниченном числе множеств. Можно потребовать, чтобы любые величины, сопоставляемые элементам множества, принадлежали некоторому ограниченному множеству допустимых значений. Из всех этих возможных задач для детального анализа обычно выбираются те, которые имеют известные приложения или в каком-то смысле являются «естественными» подзадачами, возникновение которых можно ожидать в приложениях.

Совершенно ясно, что, если даже задача П является NP-полной, ее подзадачи могут быть как NP-полными, так и полиномиально разрешимыми. Конечно, если задача П принадлежит классу P, то любая ее подзадача, индивидуальные задачи которой распознаваемы за полиномиальное время (а мы будем рассматривать только подзадачи, обладающие последним свойством),

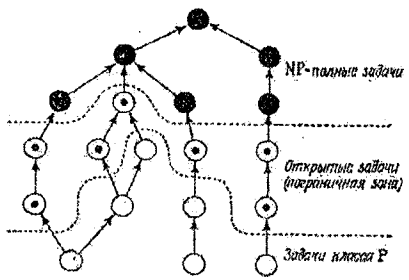


Рис. 6.5 Одна из возможных ситуаций состояния знаний о подзадачах NP-полной задачи П. Задачи изображены кружками. Тёмные кружки соответствуют NP-полным задачам, светлые – задачам класса P, а кружки с точкой – «открытым» задачам. Стрелка от задачи $П_1$ к задача $П_2$ указывает на то, что $П_1$ есть подзадача $П_2$.

также должна лежать в P. Мы уже отмечали отличающиеся в этом отношении две подзадачи ВЫПОЛНИМОСТЬ: а именно, задачи 3-ВЫПОЛНИМОСТЬ и 2-ВЫПОЛНИМОСТЬ. В предположении, что P≠NP можно считать, что подзадачи

любой NP-полной задачи лежат по разные стороны воображаемой границы, разделяющей полиномиально разрешимые и труднорешаемые задачи. При анализе задач наша цель состоит в выяснении, какие подзадачи по какую сторону от этой границы лежат.

В действительности лучше будет в каждый отдельный момент времени представлять себе пограничную зону между подзадачами, разрешимость которых за полиномиальное время уже известна, и подзадачами, для которых установлена NP-полнота. Подобная пограничная зона состоит из подзадач, NP-полнота которых остается под вопросом. На рис. 6.5 схематически изображено возможное состояние знаний о наборе подзадач задачи П.

Если удаётся определить, что некоторая открытая задача лежит в P или NP-полна, то тем самым сужается пограничная зона и увеличивается изученная часть класса NP. Конечно, если задача неразрешима за полиномиальное время, нам никогда не удастся полностью исчерпать пограничную зону, кроме, возможно, случая, когда представляет интерес только ограниченное число подзадач. Даже когда мы ограничиваемся фиксированным, конечным набором подзадач, некоторые из них могут оказаться принадлежащими к группе промежуточных задач, которые ни NP-полны, ни принадлежат классу P (как мы уже отмечали, если P≠NP, то такие задачи должны существовать). Тем не менее, когда мы выясняем место одной из задач, оставшейся открытой, можно считать, что мы приближаемся к воображаемой пограничной линии.

Чтобы конкретизировать сказанное выше, рассмотрим задачу составления расписания выполнения заданий, имеющих равную длительность и подчиненных отношению предшествования (эта задача сама по себе представляет частный случай некоторых более общих задач теории расписаний).

РАСПИСАНИЕ С ОТНОШЕНИЕМ ПРЕДШЕСТВОВАНИЯ

УСЛОВИЕ. Задано множество T «заданий» (предполагается, что все они имеют длительность 1) и частичный порядок $<$ на множестве T, число «процессоров» m и общий директивный срок $D \in Z^+$.

ВОПРОС. Существует ли «расписание» $\sigma: T \to \{0,1,\ldots,D\}$, что для каждого $i \in \{0,1,\ldots,D : |\{t \in T: \sigma(t)=i\}| \leq m$ и если $t < t'$, то $\sigma(t) < \sigma(t')\}$?

В качестве возможного приложения этой задачи рассмотрим такую ситуацию. Предположим, что вы ассистент кафедры прикладной математики МГУПС (МИИТа) и только что получили задание помочь поступившим первокурсникам составить план своих занятий на все время обучения. Студенты представляют вам список всех курсов, которые они намереваются прослушать, число m - максимальное число курсов, которые они могут слушать одновременно, и D - число семестров обучения. Университет достаточно большой, так что каждый курс лекций читается каждый семестр и никакие два курса не пересекаются по времени. Однако некоторые курсы являются вспомогательными для других и поэтому должны быть прослушаны раньше последних ($t<t'$ означает, что t является вспомогательным для t'). Предположим, что вы решили написать программу для ЭВМ, которая, пользуясь этой информацией, составляла бы для каждого студента соответствующее расписание.

Было доказано, что задача РАСПИСАНИЕ С ОТНОШЕНИЕМ ПРЕДШЕСТВОВАНИЯ NP-полна, поэтому мало вероятно, что вам удастся построить полиномиальный алгоритм составления расписания в общем случае. Тем не менее, возможно, найдутся некоторые естественные ограничения, приемлемые для большинства студентов и облегчающие решение задачи. Поскольку работоспособность

студентов ограничена, то вполне вероятно, что m можно ограничить сверху некоторым числом, например 6. Вспомогательные курсы лекций также могут удовлетворять дополнительным ограничениям. Многие из студентов могли выбрать столь разнообразную программу обучения, что ни один из выбранных ими курсов лекций не потребует вспомогательных курсов (в этом случае условие порядка отсутствует).

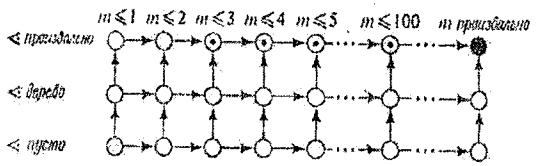


Рис. 6.6 Современное состояние знаний о семействе подзадач задачи РАСПИСАНИЕ С ОТНОШЕНИЕМ ПРЕДШЕСТВОВАНИЯ (в обозначениях рис. 6.5)

Иногда может оказаться, что для каждого данного курса лекций требуется только один явный вспомогательный курс, а остальные курсы, вспомогательные для данного, являются вспомогательными также и для явного вспомогательного. В этом случае частичный порядок представляет собой древовидную структуру. Возможны и другие ограничения, но остановимся только на перечисленных. Возникает семейство подзадач, представленное на рис.6.6. На этом же рисунке приведены известные сведения о сложности этих подзадач. Заметим, что все возможности, указанные на рис. 6.5, включая пограничную зону, встречаются на рис.6.6. В случае, когда имеется подобная однородная иерархия подзадач, очень часто ту же самую информацию можно задать более кратко, указав «минимальную» NP-полную и «максимальную» разрешимую за полиномиальное время подзадачу. Если задано семейство C подзадач некоторой NP-полной задачи, то задача $П \in C$ является (в настоящее время) *минимальной* NP-полной

подзадачей, если известна NP-полнота задачи П и у нее нет подзадачи П', которая была бы NP-полна и принадлежала семейству *C*. Задача П∈*C* является (в настоящее время) *максимальной* полиномиально разрешимой подзадачей, если известна ее принадлежность классу P и в семействе *C* нет задачи П', которая была бы разрешима за полиномиальное время и содержала бы П в качестве подзадачи. Аналогичным образом можно определить минимальную и максимальную открытые подзадачи семейства *C*.

Для класса подзадач задачи РАСПИСАНИЕ С ОТНОШЕНИЕМ ПРЕДШЕСТВОВАНИЯ, представленного на рис. 6.6, две подзадачи, определяемые ограничениями «< - произвольно, $m \leq 2$» и «< - дерево, m – произвольно», являлись на 1982г. максимальными подзадачами, разрешимыми за полиномиальное время. Сама общая задача являлась на 1982 г. *минимальной* NP-полной подзадачей. Минимальная открытая подзадача определяется ограничениями «< - произвольно, $m \leq 3$», а максимальной открытой подзадачи не существует, поскольку все задачи с ограничением вида «< - произвольно, $m \leq J$» открыты при всех целых $J \geq 3$.

Вполне естественным представляется изучение пограничной области задачи с помощью общей схемы двустороннего подхода, описанной выше. По очереди рассматриваются подзадачи, представляющиеся наиболее вероятными кандидатами на принадлежность классу P, и подзадачи, кажущиеся NP-полными. Используя в качестве отправных точек подзадачи, разрешимые за полиномиальное время, и подзадачи, NP-полнота которых следует непосредственно из NP-полноты общей задачи, мы постепенно увеличиваем множество индивидуальных задач первой подзадачи и уменьшаем множество индивидуальных задач второй. В отличие от случая, когда изучается одна отдельная задача с переходами от построения алгоритма к доказательству NP-полноты, при анализе подзадач у нас

появляется возможность изменять также и сами задачи, от более ограничительных к менее ограничительным.

Техника, используемая для доказательства NP-полноты подзадач, по существу аналогична описанной ранее технике доказательства NP-полноты изолированных задач. Однако имеется весьма важное различие. Когда мы пытаемся доказать NP-полноту подзадачи, нам уже известно доказательство NP-полноты некоторой обобщенной версии этой подзадачи. Это дает нам хорошего кандидата на известную NP-полную задачу (чем можно воспользоваться в искомом доказательстве), а также доказательство NP-полноты, которое можно было бы модифицировать, чтобы получить соответствующее доказательство для рассматриваемой подзадачи. Хотя это и не всегда облегчит достижение нашей цели, но, по крайней мере, в отличие от доказательства NP-полноты изолированной задачи, нам будет от чего оттолкнуться.

ЛИТЕРАТУРА

1.Гэри М., Джонсон Д. Вычислительные машины и трудно-решаемые задачи. М., Мир, 1982.

2 .Кузюрин Н.Н.Лекции по курсу «Сложность комбинаторных алгоритмов». Препринт, июнь 2002.

3.Крупский В.Н. Введение в сложность вычислений. М., ФАКТОРИАЛ ПРЕСС, 2006, 128 С.

4. Учебное пособие по курсу «Сложность алгоритмов». МГУ имени М.В.Ломоносова, факультет ВМК, М., 2002.

5.Сапоженко А.А. Некоторые вопросы сложности алгоритмов. МГУ имени М.В.Ломоносова, факультет ВМК, М., 2001.

6. Нечаев В.И. Элементы криптографии. Основы защиты информации. Москва, Высшая школа, 1999.

ОГЛАВЛЕНИЕ

Св. план 2010 г., поз. 185

Хаханян Валерий Христофорович

Элементы теории сложности
алгоритмов и вычислений

Учебное пособие

Подписано в печать - 15.04.10. Формат 60×84/16

Усл. - печ. л. - 9,25. Заказ № 252. Тираж 100 экз.

127994 Москва, ул.Образцова, д. 9, стр. 9
Типография МИИТа.