Косовская Татьяна Матвеевна

О КЛАССАХ СЛОЖНОСТИ АЛГОРИТМОВ

В предыдущем номере журнала [1] было дано понятие временно́й сложности алгоритма как количества шагов его работы в зависимости от длины записи исходных данных и определены два класса предикатов. Класс **P** – класс предикатов, проверка истинности которых на детерминированной машине Тьюринга может быть осуществлена с количеством шагов, не превосходящим некоторого полинома от длины записи исходных данных. Класс **NP** – класс предикатов, проверка истинности которых на недетерминированной машине Тьюринга может быть осуществлена с количеством шагов, не превосходящим некоторого полинома от длины записи исходных данных. Было показано, что $\mathbf{P} \subseteq \mathbf{NP}$ и сформулирована.

Теорема. Если задача принадлежит классу **NP**, то существует решающий ее алгоритм, реализованный на детерминированной машине Тьюринга, заканчивающей работу с числом шагов, не превосходящим $2^{p(n)}$, где $p(n)$ – некоторый полином от длины записи исходных данных n.

То есть в настоящее время для задач из класса **NP** можно гарантировать только экспоненциальное время ее решения. Одной из самых сложных и актуальных проблем математики в настоящее время признана проблема совпадения или несовпадения классов **P** и **NP**.

Напомним, что если функция временной сложности ограничена многочленом от длины записи исходных данных, то говорят, что алгоритм имеет полиномиальную сложность. Если же функция временной сложности ограничена степенной функцией (или функцией вида $a^{f(n)}$, где $f(n) \geq n^k$, $a > 1$, $k > 0$) от длины записи исходных данных, то алгоритм имеет экспоненциальную сложность.

NP-ПОЛНЫЕ ЗАДАЧИ

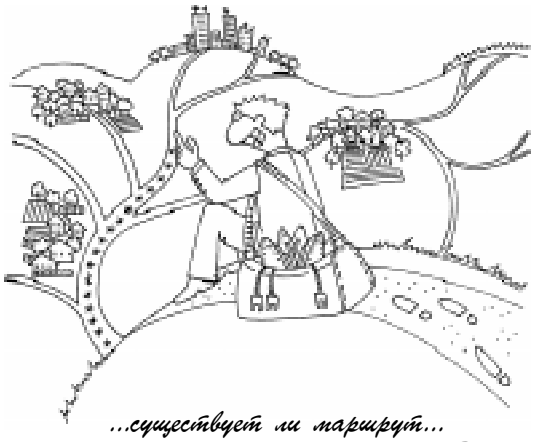
Для решения вопроса о совпадении или различии классов **P** и **NP** было введено понятие полиномиальной сводимости задач. Строгое определение можно посмотреть, например, в [2]. По сути дела задача P_1 полиномиально сводится к задаче P_2, если любые исходные данные задачи P_1 можно быстро (за полином шагов от длины их записи) переформулировать в исходные данные для P_2, причем обе задачи на этих парах исходных данных имеют (или не имеют) решения одновременно. Примерами таких задач являются, например, задачи «Коммивояжер» и «Гамильтонов цикл». Ниже через Z_+ будет обозначаться множество целых положительных чисел.

Коммивояжер

Дано: $C = \{c_1, ..., c_n\}$ – множество городов, $d(c_i, c_j) \in Z_+$ – расстояния между городами, $B \in Z_+$.

Вопрос: существует ли маршрут, проходящий через все города из C, длина которого не превосходит B?

То есть существует ли последовательность городов $c_{i1}, ..., c_{in}$, элементов из C, такая что $S_{j=1}^{n-1} d(c_{ij}, c_{i(j+1)}) + d(c_{in}, c_{i1}) \leq B$?



...существует ли маршрут...
длина которого не превосходит B?

Гамильтонов цикл (ГЦ)

Дано: граф $G = (V, E)$.

Вопрос: существует ли в G гамильтонов цикл?

То есть существует ли последовательность различных вершин графа v_{i1}, ..., v_{in}, такая что n – количество вершин в графе и каждая пара соседних вершин этой последовательности (а также первая и последняя) соединены ребром?

Задача «ГЦ» полиномиально сводится к задаче «Коммивояжер» следующим образом. Пусть заданы исходные данные для задачи «ГЦ» – множество вершин графа V и множество его ребер E. Исходные данные для задачи «Коммивояжер» будут следующими. В качестве множества городов C берем множество V. Расстояниям присваиваем значения 1 или 2 по следующему правилу: если $\{v_i, v_j) \in E$, то $d(v_i, v_j) = 1$; в противном случае $d(v_i, v_j) = 2$. В качестве числа B берем количество вершин в графе.

Можно убедиться, что по всяким исходным данным для задачи «ГЦ» полиномиально быстро строятся исходные данные для задачи «Коммивояжер». Время требуется только на выписывание матрицы расстояний, а это квадрат от количества вершин графа. При этом на этих парах исходных данных задачи имеют (или не имеют) решение одновременно, так как маршрут для коммивояжера длины не более чем n может проходить только по ребрам графа.

Кроме того, если для решения задачи «Коммивояжер» имеется быстрый (полиномиальный) алгоритм, то и для решения задачи «ГЦ» также имеется быстрый (полиномиальный) алгоритм, время работы которого больше времени работы исходного алгоритма только на время записи новых исходных данных. К сожалению, полиномиальные алгоритмы для решения этих задач в настоящее время неизвестны, несмотря на практическую полезность задачи «Коммивояжер» и разнообразие методов, разработанных для ее решения.

Задача называется **NP**-полной, если она принадлежит классу **NP** и любая задача из класса **NP** полиномиально сводится к ней.

Надеюсь, что из рассмотренного примера должно быть ясно, что

– если для какой-либо **NP**-полной задачи доказано существование полиномиального алгоритма, то тем самым доказано, что **P = NP**;

– если же хоть для одной **NP**-полной задачи будет доказано, что не существует решающего ее полиномиального алгоритма (то есть любой алгоритм, решающий ее, работает за время, большее любого полинома от длины записи исходных данных), то будет доказано, что **P ≠ NP**.

В настоящее время доказана **NP** -полнота тысяч задач распознавания. Обширный список, содержащий более трехсот **NP**-полных задач, приведен в [2]. Эти задачи относятся к самым различным областям математики и информатики: теория графов и построение сетей, множества и разбиения, хранение и поиск данных, теория расписаний, математическое программирование, алгебра и теория чисел, логика, теория автоматов и языков, оптимизация программ, игры и головоломки.

До сих пор ни для одной **NP**-полной задачи не найден полиномиальный алгоритм. Среди математиков имеется достаточное основание предполагать, что таких алгоритмов и не существует. Но что же делать, если на практике требуется решать **NP**-полную задачу для многих различных исходных данных? Неужели ждать завершения работы программы сотни лет?

К счастью, «Бог изощрен, но не злонамерен». Для различных исходных данных одной и той же длины время решения задачи различно. Лет тридцать назад некоторые математики «развлекались» тем, что для алгоритма, решающего **NP**-полную задачу, находили такие исходные данные, на которых число шагов работы алгоритма действительно экспоненциально. Это была не очень простая задача. При решении огромного количества практических задач их исходные данные таковы, что время работы алгоритма над этими исходными данными не столь велико, как могло бы быть. Но это далеко не для всех задач.

АНАЛИЗ ПОДЗАДАЧ

Подзадачей данной задачи называется такая задача, которая получается из исходной наложением некоторых ограничений на ее исходные данные. Например, подзадачей простейшей задачи наличия вещественных корней квадратного трехчлена с коэффициентами a, b, c $\exists x$ ($ax^2 + bx + c = 0$) является задача наличия вещественных корней квадратного трехчлена с коэффициентами a, b, c при условии, что $ac < 0$ $\exists x$ ($ax^2 + bx + c = 0$ & $ac < 0$). Исходная задача хоть и не сложна, но требует некоторых вычислений. Ее подзадача же тривиальна, так как такие квадратные трехчлены всегда имеют вещественные корни.

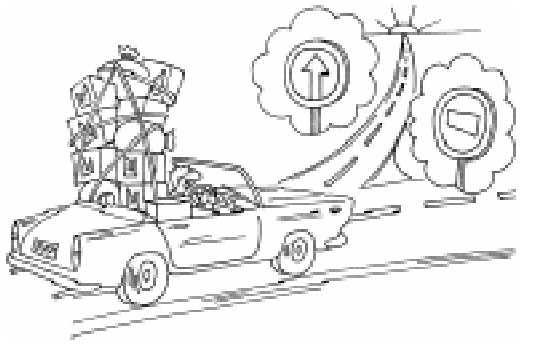
Анализ подзадач **NP**-полной задачи имеет важное практическое значение при ответе на вопрос, будет ли решающая поставленную задачу программа работать приемлемое время или нет.

Приведем несколько примеров **NP**-полных задач и их подзадач.

Выполнимость (Вып)

Дано: $U = \{u_1, ..., u_n\}$ – множество пропозициональных (булевских) переменных, $C = \{c_1, ..., c_m\}$ – множество предложений (простых дизъюнкций, то есть дизъюнкций переменных и/или их отрицаний) над U.

Вопрос: Существует ли набор значений переменных, выполняющий каждое предложение из C? (То есть такой набор значений переменных, на котором каждое предложение из C истинно?)



...«Бог изощрен, но не злонамерен»...

3-Выполнимость (3-Вып)

Дано: $U = \{u_1, ..., u_n\}$ – множество пропозициональных (булевских) переменных, $C = \{c_1, ..., c_m\}$ – множество предложений (простых дизъюнкций) над U, каждое из которых содержит ровно три переменные.

Вопрос: Существует ли набор значений переменных, выполняющий каждое предложение из C?

2-Выполнимость (2-Вып)

Дано: $U = \{u_1, ..., u_n\}$ – множество пропозициональных (булевских) переменных, $C = \{c_1, ..., c_m\}$ – множество предложений (простых дизъюнкций) над U, каждое из которых содержит ровно две переменные.

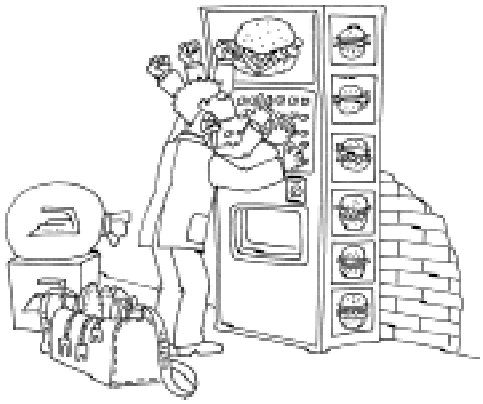
Вопрос: Существует ли набор значений переменных, выполняющий каждое предложение из C?

1-Выполнимость (1-Вып)

Дано: $U = \{u_1, ..., u_n\}$ – множество пропозициональных (булевских) переменных, $C = \{c_1, ..., c_m\}$ – множество предложений (простых конъюнкций) над U, каждое из которых содержит ровно одну переменную.

Вопрос: Существует ли набор значений переменных, выполняющий каждое предложение из C?

Первые две задачи **NP**-полны, хотя задача «3-Вып» и является подзадачей задачи «Вып» и полиномиальные алгоритмы их решения неизвестны. Задачи «2-Вып» и «1-Вып» тоже являются подзадачами задачи «Вып»,



Существует ли набор значений переменных, выполняющий каждое предложение из C?

но для «2-Вып» известен полиномиальный алгоритм, а задача «1-Вып» легко решается непосредственным просмотром множества *C*.

Эти примеры могут показаться слишком абстрактными и не имеющими практического применения (хотя это и не так). Практическая полезность следующей **NP**-полной задачи, по-видимому, видна всем.

Расписание для мультипроцессорной системы

Дано: 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 $A = A_1 \cup A_2 \cup ... \cup A_m$ множества A на m непересекающихся множеств, такое что $\max_{1 \le i \le m} \Sigma_{a \in A_i} l(a) \le D$?

...можно ли распределить все задания ... так, чтобы к директивному сроку все задания были бы выполнены?

То есть можно ли распределить все задания на m процессоров так, чтобы к директивному сроку все задания были бы выполнены?

Задача **NP**-полна, и полиномиальный алгоритм ее решения неизвестен. Но если требуется распределение на 2 процессора, на 3 процессора или даже на 100 процессоров (все это подзадачи исходной задачи), то такие подзадачи могут быть решены за время, не превосходящее некоторого полинома от длины записи исходных данных (число процессоров m в этих подзадачах исходными данными не являются). Конечно же, для каждого конкретного фиксированного количества процессоров m эти полиномы будут иметь различную степень, причем с ростом количества процессоров эта степень будет расти. Стоит задуматься о том, какая функция временной сложности – 2^n (экспоненциальная сложность) или n^{100} (полиномиальная сложность) более приемлема для ваших исходных данных, если длина их записи n не превосходит 10, или 100, или 10 000.

Таким образом, первая рекомендация к тому, что же делать, если требуется многократно с исходными данными большой длины решать задачу, про которую вам кажется, что она **NP**-полна, заключается в следующем: проверьте, действительно ли исходные данные, которые будут подвергаться обработке, – это исходные данные для известной вам **NP**-полной задачи, а не для какой-то ее подзадачи.

ПСЕВДОПОЛИНОМИАЛЬНЫЕ ЗАДАЧИ

В некоторых случаях в качестве исходных данных для задачи выступают числа. Если временная сложность решения такой задачи не превосходит полинома от этих чисел (а не от длины их записи) и длины записи остальных исходных данных, то такая задача называется псевдополиномиальной. Среди псевдополиномиальных задач есть и **NP**-полные. Примером такой псевдополиномиальнойи **NP**-полной задачи является задача «Разбиение».

Разбиение

Дано: Конечное множество A, для каждого $a \in A$ его «вес» $s(a) \in Z_+$.

Вопрос: Существует ли подмножество $A' \subseteq A$ такое, что

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

То есть можно ли разбить множество A на два подмножества одинакового веса?

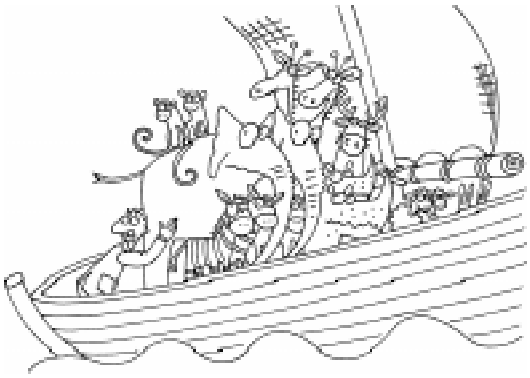
Для этой задачи известен решающий ее алгоритм, который использует не более чем nB операций сложения (здесь B – сумма весов всех элементов из A, n – количество элементов в A). То есть временная сложность этого алгоритма ограничена полиномом от длины записи множества A (число n) и суммы числовых параметров задачи (число B).

Почему же эта задача псевдополиномиальна, а не полиномиальна? Вспомним, что временная сложность задачи – это функция от длины записи исходных данных. А всякое целое положительное число N может быть записано как $N \approx 2^{\log N} \approx 2^{|N|}$. Здесь $|N|$ – длина записи числа N в двоичной системе счисления, запись логарифмической функции без указания основания подразумевает, что ее основание равно двум. То есть перед нами на самом деле экспоненциальный алгоритм.

Но у псевдополиномиальных задач есть одно преимущество – числовые параметры обычно не имеют очень большой длины записи. Например, трудно себе представить, чтобы вес многотонныхглыб измерялся в граммах. В крайнем случае, это будут килограммы.

Вот и вторая рекомендация к тому, что делать, если требуется многократно решать задачу, которая **NP**-полна. Проверьте, не является ли она псевдополиномиальной. И если длина записи числовых параметров, которые придется обрабатывать, не слишком велика, то смело пишите программу, предназначенную для многократного использования с различными исходными данными.

Ну а для **NP**-полных задач, в которых нечисловые параметры могут принимать самые различные значения, обычно используют приближенные методы их решения,



...можно ли разбить множество A на два подмножества одинакового веса?

алгоритмы для которых иногда оказываются полиномиальными.

НЕКОТОРЫЕ ДРУГИЕ КЛАССЫ СЛОЖНОСТИ

До сих пор мы все время говорили о временной сложности алгоритмов, то есть о такой функции от длины записи исходных данных, что число шагов работы алгоритма (число шагов машины Тьюринга) с исходными данными заданной длины не превосходит значения этой функции. Однако достаточно часто встает вопрос о памяти, необходимой для решения той или иной задачи.

Под ёмкостной (или зональной) сложностью алгоритма A обычно понимают функцию, зависящую от длины записи исходных данных и выражающую верхнюю оценку памяти, необходимой для работы алгоритма.

В названиях классов сложности для различения временной или зональной сложности используют обозначения **TIME** и **SPACE** соответственно. При этом в случае, когда не может возникнуть неоднозначного прочтения, слово **TIME** может быть опущено.

Для обозначения того, какая модель машины Тьюринга использована в определении класса сложности – классическая (детерминированная) или недетерминированная, в начале названия класса ставят букву **D** или **N** соответственно, при этом буква **D**, как правило, опускается.

...достаточно часто встает вопрос о памяти, необходимой для решения той или иной задачи.

В зависимости от того, какой функцией ограничены время или память вычислений, используют следующие обозначения:
LOG – *logarithmical* – логарифмический;
LIN – *linear* – линейный;
P – *polynomial* – полиномиальный;
EXP – *exponential* – экспоненциальный.

Так, например, полным названием класса **P** будет **D-P-TIME** (детерминированное полиномиальное время).

Если известна временная сложность алгоритма $T(n)$, то всегда можно (достаточно грубо) оценить его зональную сложность $S(n)$ и наоборот. Несложно доказать следующие неравенства. $S(n) \le T(n)$ и $T(n) \le 2^{C \cdot S(n)}$, где C – некоторая константа.

Между классами сложности имеются некоторые включения, причем для ряда классов проблема об их совпадении или несовпадении остается открытой и является сложной математической задачей. Ниже приведем некоторые включения, при этом знак $\subset$ будет использоваться для строгого включения (то есть классы не совпадают), а для открытых проблем, то есть когда доказано нестрогое включение $\subseteq$, но неизвестно, совпадают классы или нет, будет использовано обозначение ? (см. схему 1).

NP-ТРУДНЫЕ И P-SPACE-ТРУДНЫЕ ЗАДАЧИ

Перечисленные выше классы сложности определены для задач распознавания (см. [1]), то есть для задач, имеющих в качестве ответа значения ДА или НЕТ. Для задач поиска, то есть для задач, в которых не только требуется ответить, существует ли описанный объект, но и найти его, перед именем класса иногда ставят букву **F**, чтобы подчеркнуть, что это класс функций.

Понятно, что на практике, как правило, интересуются не только, например, тем, можно ли составить расписание с тем или иным свойством, но и самим таким расписанием. Интуитивно ясно, что время, затраченное на поиск решения задачи, не может быть меньше времени, затраченного на проверку, существует ли это решение (в крайнем случае, они будут совпадать).

Если в формулировке **NP**-полной задачи вместо слов «существует ли» поставить слово «найти», то мы получим формулировку **NP**-трудной задачи. Заметим, что говорить про **NP**-трудную задачу, что она **NP**-полна, неправильно, хотя все окружающие прекрасно поймут, что на самом деле вы должны были бы сказать.

Важным классом сложности, которому принадлежит огромное количество задач распознавания, является класс **P-SPACE**, поскольку задачи из этого класса могут быть решены с использованием не слишком боль-

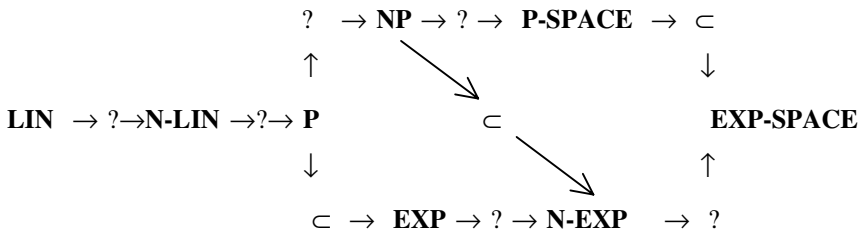


Схема 1.

шого объема памяти. Проблема совпадения или несовпадения классов **NP** и **P-SPACE** является столь же сложной, как и проблема совпадения или несовпадения классов **P** и **NP**.

Для решения этой проблемы было введено понятие **P-SPACE**-полной задачи, во многом аналогичное понятию **NP**-полной задачи. Так же, как и для задач из класса **NP**, если в формулировке **P-SPACE**-полной задачи вместо слов «существует ли» поставить слово «найти», то мы получим формулировку **P-SPACE**-трудной задачи.

В заключение хочу сказать, что так же, как не обязательно **NP**-полная или **NP**-трудная задачи непригодны для практического многократного решения при различных исходных данных, так и не все задачи с полиномиальными алгоритмами их решения могут быть решены на компьютере за приемлемое время.

При исследовании генома человека было необходимо попарно сравнить несколько сотен записей, длина которых составляла приблизительно 3 Мбайт. Точный алгоритм решения задачи имел квадратичную сложность, то есть n^2 элементарных шагов, где n – длина одной записи.

Для поставленной задачи длина одной записи в исходных данных составила 3 Мбайт $\approx 3 \cdot 10^6$ байт. При этом время сравнения одной пары записей составило $(3 \cdot 10^6)^2 \cdot 10^{-6}$ сек. $= 9 \cdot 10^6$ сек. $= 15 \cdot 10^4$ мин. $= 25 \cdot 10^2$ час. $\approx 10^2$ дней.

Исследователь воспользовался приближенным алгоритмом решения задачи.

Литература

1. *Косовская Т.М.* О временнóй сложности решения задач распознавания // Компьютерные инструменты в образовании, 2005, № 3. С. 61–68.

2. *М.Гэри, Д.Джонсон.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.

Косовская Татьяна Матвеевна, кандидат физико-математических наук, доцент кафедры математики Государственного Морского Технического Университета.