Косовская Татьяна Матвеевна

О ВРЕМЕННОЙ СЛОЖНОСТИ РЕШЕНИЯ ЗАДАЧ РАСПОЗНАВАНИЯ

ПРЕДВАРИТЕЛЬНЫЕ ПРИМЕРЫ

Уже давно математиками было замечено, что при решении одних и тех же математических задач (особенно связанных с вычислениями) с одними и теми же исходными данными затрачивается различное время, в зависимости от способа их решения. Особенно актуально это стало с развитием вычислительной техники и многократным использованием одной и той же программы для различных исходных данных. Наверняка многие слышали, как говорят о том, что один алгоритм (программа) эффективнее другого алгоритма (программы). Такое нестрогое понятие, как эффективность, в настоящее время математически строго формализовано таким понятием, как временна́я сложность алгоритма или задачи.

Заметим, что в теории сложности алгоритмов основные определения, как правило, даются в терминах машины Тьюринга, так как шаги ее работы предельно элементарны, и можно считать, что каждый шаг выполняется за одно и то же время. Для знакомства с машинами Тьюринга можно прочитать, например, статью в предыдущем номере журнала [1], тем более что в этом разделе будем использовать примеры, приведенные в ней.

Ниже посредством $|A|$ будем обозначать длину записи слова A. Вместо $log_2\ x$ будем писать $log\ x$, то есть опускать основание двоичного логарифма.

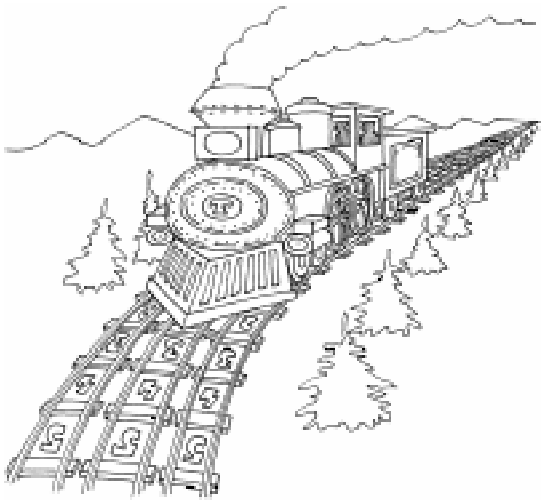
Сначала приведем примеры некоторых алгоритмов (в частности, решающих одну и ту же, с точки зрения классической математики, задачу), затрачивающих различное число элементарных шагов.

Пример 1.

Подсчитать число шагов работы машины Тьюринга, вычисляющей сумму двух положительных целых чисел, заданных в унарной системе счисления. (С точки зрения математики решается задача вычисления $x + y$.)

За x шагов из начальной конфигурации $q_1$1...1 * 1...1 (где количество единиц в первом слагаемом равно x, а количество единиц во втором слагаемом равно y) можно перейти в конфигурацию 1...1 q_1 * 1...1 , где

...в теории сложности алгоритмов основные определения... даются в терминах машины Тьюринга, так как шаги ее работы предельно элементарны



Подсчитать число шагов работы трехленточной машины Тьюринга...

количество единиц в первом слагаемом равно $x - 1$, и заменить знак * на 1. Двигаясь по ленте влево, за x шагов придем в заключительную конфигурацию $q_0 1...1$ (где количество единиц равно $x + y$). Всего для решения поставленной задачи потребовалось $2x$ шагов.

Пример 2.

Подсчитать число шагов работы машины Тьюринга, вычисляющей сумму двух чисел, заданных в двоичной системе счисления. (С точки зрения математики решается все та же задача вычисления $x + y$.)

Пусть задана начальная конфигурация машины Тьюринга $q_1\ X*Y$, где X и Y – двоичные записи натуральных чисел. При этом $|X| =]\log x[$, $|Y| =]\log y[$ (В таблице 1 дано описание работы машины Тьюринга).

Всего для решения поставленной задачи потребовалось $2 \cdot \max(|X|, |Y|) \cdot (|X| + |Y| + 7) + 2|X| + 3$ шагов. Это количество шагов не превосходит $4 \cdot \max^2(|X|, |Y|) + 16 \max(|X|, |Y|) + 3$.

Пример 3.

Подсчитать число шагов работы трехленточной машины Тьюринга, вычисляющей сумму двух чисел, заданных в двоичной системе счисления. (С точки зрения математики решается все та же задача вычисления $x + y$.)

Пусть задана начальная конфигурация машины Тьюринга $q_1 \begin{matrix} X \\ Y \\ * \end{matrix}$, где X и Y – двоичные записи натуральных чисел. Требуется, чтобы заключительной конфигурацией была $q_0 \begin{matrix} X \\ Y \\ Z \end{matrix}$, где Z – двоичная запись суммы.

«Выравнивание» слов $|X|$ и $|Y|$ по правому концу требует $\max(|X|, |Y|) + 1$ шаг. Сложение в столбик требует еще столько же шагов. Всего для решения поставленной задачи потребовалось $2 \cdot \max(|X|, |Y|) + 2$ шагов.

Таблица 1.

1. «Добежим вправо» до разделяющего аргументы пустого символа *.	На это потребуется $\|X\|$ шагов.
2. «Добежим вправо» до последней непомеченной цифры числа Y. Запомним эту цифру и пометим ее.	На это потребуется $\|Y\| + 2$ шага.
3. «Добежим влево» до последней непомеченной цифры числа X, запомним эту цифру и пометим ее.	На это потребуется $\|Y\| + 2$ шага.
4. «Добежим влево» до первой цифры числа X и отступим на одну клетку влево. «Добежим влево» до первой цифры числа, полученного сложением просмотренных частей X и Y. Отступив на одну клетку влево, запишем сумму запомненных цифр.	На это потребуется $\|X\| + 2$ шага.
5. Повторяем п.п. 1–4 до тех пор, пока все цифры X и Y не окажутся помеченными.	П.п. 1–4 выполняются $\max(\|X\|, \|Y\|)$ раз.
6. Проверяем, что все цифры записей помечены и стираем их. Возвращаем головку машины Тьюринга в начало записи результата.	На это потребуется не более чем $2(\|X + Y\| + \|X\|) + 3$ шага, то есть $2(\max(\|X\|, \|Y\|) + \|X\|) + 3$.

Сравнение оценок сложности, полученных в примерах.

На первый взгляд, может показаться, что в первом из двух примеров и алгоритм попроще, и число шагов поменьше. Однако следует вспомнить, что $|X| \approx \log x$, $|Y| \approx \log y$. Если требуется сложить числа, приблизительно равные тысяче (то есть длины их двоичных записей приблизительно равны 10), то в первом примере мы имеем около 2000 шагов, а во втором примере около 560 шагов.

В третьем примере использована другая модель точного понятия алгоритма – трехленточная машина Тьюринга. В этом примере при сложении чисел приблизительно равных тысяче потребуется всего 22 шага.

Попробуйте подсчитать самостоятельно число шагов машин Тьюринга из этих примеров, если числа приблизительно равны 10 000, 100 000, 1 000 000 и так далее.

Можно привести примеры алгоритмов, которые с данными одной и той же длины будут выполнять различное число шагов. Например, если требуется найти какой-нибудь целый корень многочлена с целыми коэффициентами (исходные данные – коэффициенты этого многочлена), то достаточно проверить все целые числа, являющиеся делителями свободного члена. При этом вам может «повезти», и первый же делитель окажется корнем, а возможно, вам придется проверять второй, третий, ..., последний делитель свободного члена.

Хотелось бы, чтобы из этих примеров стало ясно, что для оценки числа шагов решения задачи важен ЯЗЫК, на котором решается задача, то есть

– точное понятие алгоритма, выбранное для решения задачи,

– вид записи исходных данных.

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ. ПОЛИНОМ ИЛИ ЭКСПОНЕНТА?

Временем работы алгоритма A над данными P будем называть число шагов работы этого алгоритма над этими данными.

Верхней оценкой времени работы алгоритма A над данными длины n назовем время, за которое алгоритм A заведомо закончит работу над любыми данными длины n.

Нижней оценкой времени работы алгоритма A над данными длины n назовем время, за которое алгоритм A не может закончить работу ни над какими данными длины n.

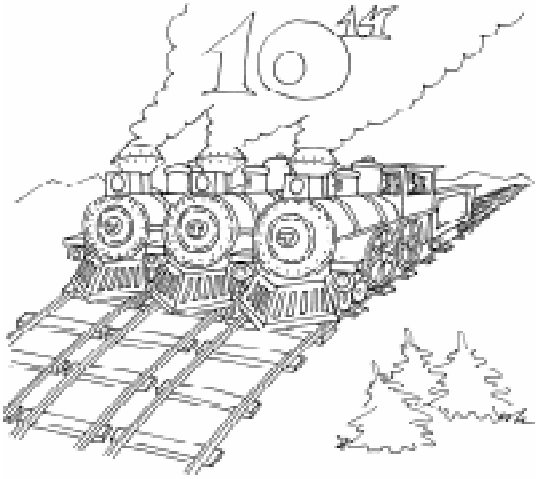
Под временно́й сложностью алгоритма A обычно понимают функцию, зависящую от длины записи исходных данных и выражающую верхнюю оценку времени работы алгоритма.

Какие же функции являются «хорошими» для временной сложности? Некоторый ответ на этот вопрос дает таблица 2, взятая из [2]. Желающие могут проверить правильность таблицы на калькуляторе или написав простенькую программу.

Пусть быстродействие компьютера составляет 10^6 операций в секунду. В левом столбце таблицы указаны временные сложности алгоритмов. В следующих колон-

Таблица 2.

	Длина записи исходных данных n					
Функция временной сложности	10	20	30	40	50	60
n	10^{-5} сек.	$2 \cdot 10^{-5}$ сек.	$3 \cdot 10^{-5}$ сек.	$4 \cdot 10^{-5}$ сек.	$5 \cdot 10^{-5}$ сек.	$6 \cdot 10^{-5}$ сек.
n^2	10^{-4} сек.	$4 \cdot 10^{-4}$ сек.	$9 \cdot 10^{-4}$ сек.	$16 \cdot 10^{-4}$ сек.	$25 \cdot 10^{-4}$ сек.	$36 \cdot 10^{-4}$ сек.
n^3	10^{-3} сек.	$8 \cdot 10^{-3}$ сек.	$27 \cdot 10^{-3}$ сек.	$64 \cdot 10^{-3}$ сек.	$125 \cdot 10^{-3}$ сек.	$216 \cdot 10^{-3}$ сек.
n^5	0.1 сек.	3.2 сек.	24.3 сек.	1.7 мин.	5.2 мин.	13.0 мин.
2^n	10^{-3} сек.	1 сек.	17.9 мин.	12.7 дн.	35.7 лет	366 век.
3^n	$59 \cdot 10^{-3}$ сек.	58 мин.	6.5 лет	3855 век.	$2 \cdot 10^8$ век.	$1.3 \cdot 10^{13}$ век.



...сообщения о кажущемся ... быстродействии, достигаемом за счет параллельного выполнения некоторых операций.

ках приведено время, которое потребуется алгоритму с указанной сложностью для обработки исходных данных, длина которых задана в соответствующей колонке.

Следует заметить, что длина записи исходных данных в реальных программах зачастую бывает не только десятки, но и сотни, и тысячи символов.

Люди, насмотревшиеся рекламы о чудесных супербыстрых компьютерах, могут возразить, что 10^6 операций в секунду – это устаревшие компьютеры. Для них приведем еще одну табличку из [2]. В левой колонке по-прежнему указаны временные сложности алгоритмов. Во второй колонке – размер исходных данных, обрабатываемых алгоритмом за один час при быстродействии 10^6 операций в секунду, а в следующих – размеры исходных данных, обрабатываемых алгоритмом за один час при увеличении быстродействия (таблица 3).

Следует заметить, что если теория относительности верна, то быстродействие выше, чем 10^8 невозможно. Появляющиеся сообщения о компьютерах с быстродействием 10^{10}, 10^{14} и т. п. – это сообщения о кажущемся пользователю быстродействии, достигаемом за счет параллельного выполнения некоторых операций. Но для тех, кто в это не верит, здесь приведена последняя колонка. Желающие могут продолжить таблицу для увеличивающегося быстродействия компьютеров.

Зная свойства степени произведения и логарифма произведения, изучаемые в школе, легко понять, что при степенной функции сложности размер обрабатываемых за один час исходных данных возрастает в разы, а при показательной – на единицы. Учитывая обе приведенные таблицы можно сделать вывод, что для практической многократной обработки данных большой длины (часто такое происходит при большой размерности задачи) алгоритмы с показательной функцией сложности не пригодны.

Обычно говорят, что если функция временной сложности ограничена многочленом от длины записи исходных данных, то 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. (Заметим, что слова многочлен и полином как название функции одной переменной являются синонимами.)

Надеюсь, теперь понятно, что если длина записи исходных данных велика, то

Таблица 3.

	Быстродействие компьютера (операций в секунду)		
Функция временной сложности	10^6	10^8	10^{10}
n	N_1	$100\ N_1$	$1000\ N_1$
n^2	N_2	$10\ N_2$	$100\ N_2$
n^3	N_3	$\sqrt[3]{100}\ N_3 \approx 4{,}64\ N_3$	$\sqrt[3]{10000}\ N_3 \approx 21{,}54\ N_3$
n^5	N_4	$\sqrt[5]{100}\ N_4 \approx 2{,}51\ N_4$	$\sqrt[5]{10000}\ N_4 \approx 6{,}31\ N_4$
2^n	N_5	$log100 + N_5 \approx 6{,}64 + N_5$	$log10000 + N_5 \approx 13{,}29 + N_5$
3^n	N_6	$log_3100 + N_6 \approx 4{,}19 + N_6$	$log_310000 + N_6 \approx 8{,}38 + N_6$

практическое применение имеют полиномиальные алгоритмы. При реализации экспоненциальных же алгоритмов будет казаться, что компьютер «завис», хотя на самом деле он усердно работает над вашей программой и (если не сломается) через несколько лет (или сотен лет) закончит свою работу.

Только что мы говорили об алгоритмах, реализованных на компьютерах, об их быстродействии. А зачем же машины Тьюринга?

Представьте себе, что вам требуется подсчитать время (количество тактов работы реального компьютера) выполнения простенькой последовательности операторов.

x := 2;

y := x^1.5;

z := sin (x+y);

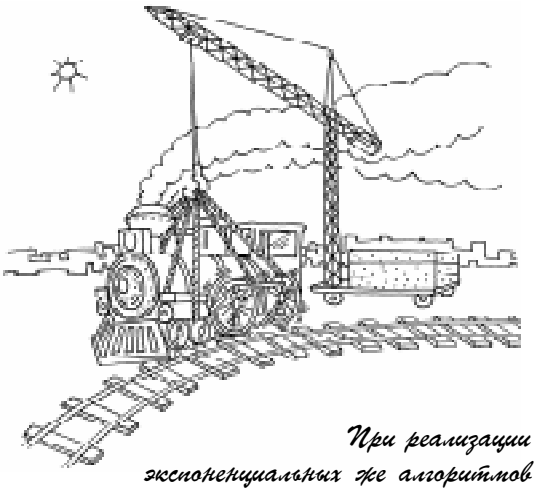
Сколько здесь операций? Три? Больше? То, что для вычисления синуса вызывается специальная программа, знают практически все. Количество элементарных операций для вычисления третьего оператора нужно подсчитать особо. А вот то, что для вычисления выражения вида a^b оно зачастую представляется в виде $e^{\ln a \cdot b}$, а для вычисления логарифма и экспоненты используются их разложения в ряд, понимают не все. (Именно поэтому обычно на занятиях по программированию обычно учат писать не x^2, а x*x.)

Лично я не берусь оценить время работы этой последовательности операторов, не зная устройства используемого транслятора.

У машины Тьюринга все шаги предельно элементарны, и вполне можно считать, что время выполнения каждого из них абсолютно одинаково. Кроме того, подсчет числа шагов работы машины Тьюринга позволяет оценить время работы реального компьютера, работающего по программе, реализующей тот же алгоритм, что и эта машина Тьюринга.

В подтверждение этого приведем еще одну табличку из [2].

Сначала определим символику, общепринятую в математике и широко используемую в теории сложности алгоритмов. Говорят, что $f(n) = O(g(n))$, если существует такая константа C, что при всех n верно неравенство $f(n) \le C \cdot g(n)$.



При реализации экспоненциальных же алгоритмов будет казаться, что компьютер «завис»...

Заметим, что при всех n если $k > m$, то $n^k \ge n^m$. Поэтому тот факт, что $f(n) = O(n^k)$, означает, что $f(n)$ не превосходит некоторого многочлена k-ой степени.

Так как $\log_a n = \log_2 n / \log_2 a = C \log_2 n$, ($C = (\log_2 a)^{-1}$), то при использовании O-символики основание у логарифма писать не обязательно и вполне можно считать (как это принято в теории сложности), что используемый логарифм является логарифмом по основанию 2. Кроме того, отметим, что логарифмическая функция растет намного медленнее, чем любая положительная степень натурального аргумента.

Предположим, что у нас имеется два точных понятия алгоритма M_1 и M_2. Некоторый алгоритм может быть реализован на модели M_1 с функцией временной сложности $T(n)$. Какую функцию сложности можно гарантировать при реализации этого алгоритма с помощью модели M_2? Ответ на этот вопрос для одноленточной машины Тьюринга, многоленточной машины Тьюринга и машин с прямым доступом (наиболее близкими к реальным компьютерам) дает таблица 4.

Что же означает эта таблица?

Пусть, например, некоторый алгоритм может быть реализован на машине с прямым доступом, число шагов работы которой не превосходит некоторого многочле-

Таблица 4.

	Моделирующая машина		
Моделируемая машина	МТ1	МТk	МПД
1-ленточная машина Тьюринга МТ1	–	$O\,(T(n))$	$O\,(T(n)\cdot\log T(n))$
k-ленточная машина Тьюринга МТk	$O\,(T^{\,2}(n))$	–	–
Машина с прямым доступом МПД	$O\,(T^{\,3}(n))$	$O\,(T^{\,2}(n)$	$O\,(T(n)\cdot\log T(n))$

на $p(n)$, зависящего от длины записи исходных данных n. Тогда этот же алгоритм может быть реализован на одноленточной машине Тьюринга, число шагов работы которой не превосходит $C\cdot p^{\,3}(n)$ для некоторой константы C, то есть его реализация с помощью машины Тьюринга тоже полиномиальна.

Если же будет доказано, что алгоритм не может быть реализован на машине с прямым доступом, число шагов работы которой не превосходит некоторого многочлена $p(n)$, зависящего от длины записи исходных данных n, то и на одноленточной машине Тьюринга он не может быть реализован за полиномиальное время.

Обычно говорят, что свойство алгоритма быть полиномиальным (или экспоненциальным) не изменяется в зависимости от того, какая *разумная* модель выбрана для его реализации. Весь вопрос заключается в том, *разумна* или *неразумна* выбранная для его реализации модель. *Разумность* выбранной модели обычно обосновывают доказательством оценок, аналогичных приведенным в таблице.

Алгоритм принадлежит классу FP, если существует реализующая его машина Тьюринга, заканчивающая свою работу за число шагов, не превосходящее некоторого полинома

Для программиста наиболее естественной моделью для реализации алгоритма является некоторый язык программирования. Эта модель может служить в качестве *разумной* модели при условии, что, например, при вычислении арифметических выражений используются только арифметические операции +, –, *, /, а при использовании других операций они не считаются за один шаг работы. При использовании процедур (особенно рекурсивных) подсчет числа шагов работы алгоритма (количества выполненных компьютером элементарных операций) зачастую сопряжен с некоторыми трудностями.

Чтобы избежать различного толкования полиномиальности алгоритма в зависимости от выбранной модели его реализации, основные определения в теории сложности алгоритмов даются в терминах одноленточной машины Тьюринга.

КЛАСС P ПОЛИНОМИАЛЬНЫХ АЛГОРИТМОВ

Определение. Алгоритм принадлежит классу **FP**, если существует реализующая его машина Тьюринга, заканчивающая свою работу за число шагов, не превосходящее некоторого полинома (многочлена) от длины записи исходных данных.

Будем говорить, что задача принадлежит классу **FP**, если существует решающий ее алгоритм из класса **FP**.

К классу **P** относят все предикаты, имеющие полиномиальную временную сложность, то есть задачи, имеющие в качестве ответа только два значения «Да» и «Нет» (задачи распознавания).

ЗАДАЧИ РАСПОЗНАВАНИЯ

Среди предикатов важное место занимают так называемые задачи распознавания, то есть задачи, сформулированные в виде «Существуют ли такие объекты Y, для которых верно утверждение $P(X, Y)$». В формализованном виде они записываются $\exists Y\, P(X, Y)$. В отличие от задач поиска, здесь не требуется найти эти объекты, а только указать, имеются ли они. Для наглядности приведем примеры задач поиска и задач распознавания (таблица 5).

Очевидно, что первая задача в обеих формулировках при известных исходных данных a и b может быть решена за полином шагов от длины записи чисел a и b. Остальные задачи могут быть решены полным перебором всех возможных претендентов на решение, так как этих претендентов конечное число. Но можно ли это сделать за полином шагов?

КЛАСС NP НЕДЕТЕРМИНИРОВАННЫХ ПОЛИНОМИАЛЬНЫХ АЛГОРИТМОВ

Определение. Алгоритм принадлежит классу **NP**, если существует реализующая его недетерминированная машина Тьюринга, заканчивающая свою работу за число шагов, не превосходящее некоторого полинома (многочлена) от длины записи исходных данных.

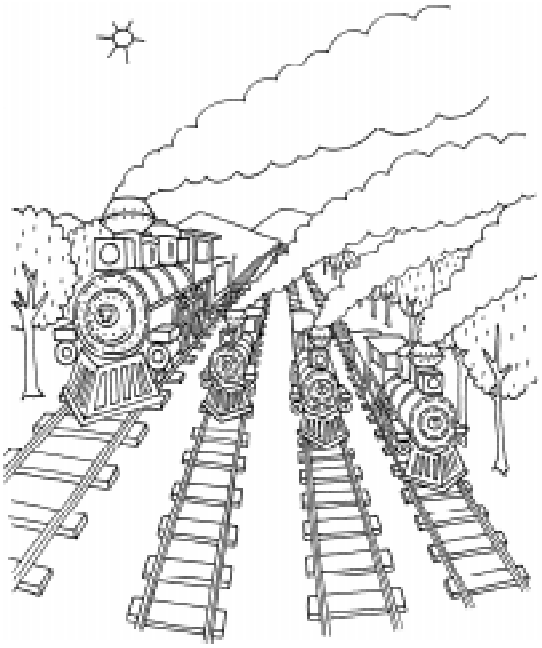
Будем говорить, что задача распознавания принадлежит классу **NP**, если существует решающий ее алгоритм из класса **NP**. Обычно это задачи вида $\exists Y\, Z(X, Y)$.

Очевидно, что $\mathbf{P} \subseteq \mathbf{NP}$, так как классическая машина Тьюринга (ее обычно называют детерминированной) является частным случаем недетерминированной (просто в ее программе не использованы несогласованные команды). Нерешенный до сих пор вопрос заключается в том, строгое это включение или не строгое, то есть $\mathbf{P} = \mathbf{NP}$ или $\mathbf{P} \neq \mathbf{NP}$?

Как правило, решение задачи распознавания разбивают на два этапа: первый этап – этап угадывания, состоящий в том, что машина Тьюринга, работая в недетерминированном режиме, размножает ленту и на каждой из них порождает претендента на решение задачи. Второй этап – этап проверки – заключается в проверке, удовлетворя-

Таблица 5.

	Задачи поиска	Задачи распознавания
1.	Решить уравнение $ax + b = 0$ для заданных рациональных чисел a и b.	По заданным рациональным числам a и b проверить, имеет ли уравнение $ax + b = 0$ решение?
2.	При каких наборах значений переменных заданная формула истинна?	Истинна ли заданная формула хоть при каких-нибудь наборах значений переменных?
3.	Найти гамильтонов цикл в заданном графе (то есть цикл, проходящий через все вершины графа ровно по одному разу).	Является ли граф гамильтоновым (то есть имеет ли он гамильтонов цикл)?
4.	Разбить объекты с заданными весами на две группы одинакового веса.	Можно ли разбить объекты с заданными весами на две группы одинакового веса?
5.	Составить расписание уроков, удовлетворяющее заданным со стороны министерства, учителей и учеников требованиям.	Можно ли составить расписание уроков, удовлетворяющее заданным со стороны министерства, учителей и учеников требованиям?
6.	Из заданного множества объектов, имеющих заданные объемы и стоимости, выбрать подмножество наибольшей стоимости, объем которого не превосходит заданного числа.	Можно ли из заданного множества объектов, имеющих заданные объемы и стоимости, выбрать подмножество, объем которого не превосходит заданного числа, а общая стоимость больше другого заданного числа?



...машина Тьюринга... размножает ленту и на каждой из них порождает претендента на решение задачи.

ет ли претендент, записанный на этом экземпляре ленты, требуемому условию. Второй этап реализуется в детерминированном режиме.

Почти сразу после определения класса **NP** была доказана достаточно очевидная теорема.

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.

Существенно более сильный результат до сих пор не известен. В доказательстве теоремы используется метод решения переборных задач разбиением их решения на только что описанные два этапа. А что будет, если задачи решать как-нибудь по-другому? Вдруг вместо $2^{p(n)}$ можно получить полином и, как следствие, **P = NP**? Или все-таки от экспоненты никуда не деться и **P ≠ NP**? В настоящий момент это одна из наиболее трудных и имеющих большое прикладное значение в области программирования задач математики. За решение этой задачи (и еще пяти математических задач) в США назначена премия миллион долларов.

Как это часто бывает со сложными, но просто формулируемыми математическими задачами (вспомните, например, великую теорему Ферма) периодически появляются сообщения о том, что эта задача решена. Как правило, ошибки этих доказательств кроются в двух моментах:

– рассматривается унарная запись числа – параметра задачи,

– для реализации алгоритма выбирается неразумная модель, в которой за один шаг работы считают выполнение достаточно сложной операции, требующей, вообще говоря, экспоненциального числа шагов работы машины Тьюринга.

Литература

1. *Косовская Т.М.* Машины Тьюринга. // Компьютерные инструменты в образовании, 2005 (№ 3), с. 52–61.

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.