Сложность вычислительных задач

М. Н. Вялый*

В прошлом номере «Математического просвещения» была опубликована статья А. А. Разборова, в которой излагались основные идеи теории сложности вычислений. Данная статья продолжает эту тему[1)]. Нас будут интересовать два вопроса: «Что такое сложная вычислительная задача?» и «Как доказать, что некоторая вычислительная задача сложна?» В полном объеме эти вопросы слишком широки, поэтому мы сосредоточим внимание на самом интересном случае: тех задачах, которые по всей видимости сложны, но при этом настолько близки к простым задачам, что доказательство их сложности в наиболее естественном понимании этого слова лежит вне возможностей современной науки. Чтобы анализировать сложность таких задач, придуманы особые способы отвечать на первый из сформулированных выше вопросов.

Под вычислительной задачей будем понимать следующую типичную ситуацию: по заданному *входу* нужно получить, если это возможно, *результат*, удовлетворяющий определённым условиям и однозначно определяемый входом. Другими словами, речь всегда будет идти о вычислении некоторой функции, быть может, частично определённой. Поэтому в дальнейшем мы будем использовать выражения «решить задачу» и «вычислить функцию» как синонимические.

Чтобы не вводить читателя в заблуждение относительно возможностей излагаемой теории, предупредим сразу об основном её ограничении. *Вычисление любой функции из конечного множества в конечное множество считается тривиальным.* Очевидно, что в таком контексте бессмысленным становится любой вопрос типа: «смогу ли я решить вот эту задачу за пару часов (недель, лет, веков)?»

Мы будем рассматривать сложность решения задачи на всем (обычно бесконечном) множестве входов. С «практической» точки зрения это означает, что нас интересует как быстро растут ресурсы (время, память и т. п.), требующиеся для решения задачи при увеличении длины входа. Более точное объяснение этой фразы даётся ниже.

*Работа выполнена при поддержке фонда РФФИ (проект №99–01–00122).

[1)]Она написана по материалам первой части книги [4] и существенно опирается на статью [7], помещённую в этом выпуске «Математического Просвещения».

1. Вычисление каких функций рассматривается

Вход и результат предполагаются конечными словами в некотором конечном алфавите (т. е. конечными последовательностями элементов некоторого конечного множества). Более того, часто рассматривается только двоичный алфавит. Такое ограничение по сути не слишком обременительно — данные любой доступной нам задачи можно сформулировать словами, и это будет текст в некотором конечном алфавите (включающем в себя, помимо букв, цифры и все иные специальные знаки, встречающиеся в тексте — знаки пунктуации, математические обозначения и проч.).

Есть некоторая тонкость, связанная с тем, что *кодировок* (способов представить объект словом в конечном алфавите) может быть несколько, а сложность задачи, очевидно, зависит от кодировки. Мы не будем подробно останавливаться на этом вопросе, в дальнейшем кодировки либо будут указываться явно, либо будут подразумеваться наиболее естественные кодировки (скажем, запись числа в позиционной системе счисления; хотя от выбора основания системы ничего не зависит, по умолчанию предполагается использование двоичной системы).

Пока мы говорили о функциях от одного аргумента. В дальнейшем будут возникать и функции от нескольких аргументов. Увеличение алфавита на 1 символ (обозначим его #) позволяет закодировать конечную последовательность слов $\alpha_1, \ldots, \alpha_n$ в алфавите A одним словом

$$\alpha_1 \# \alpha_2 \# \ldots \# \alpha_n \# \tag{1}$$

в алфавите $A \cup \{\#\}$. Будем иметь в виду это преобразование в тех случаях, когда возникает желание рассматривать функцию от нескольких аргументов как функцию от одного аргумента.

Приведем несколько примеров вычислительных задач.

Пример 1. Линейные уравнения над $\mathbb{Q}$.

Даны: матрица A размера $m \times n$, элементы которой — рациональные числа, вектор-столбец b (матрица размера $m \times 1$), также состоящий из рациональных чисел.

Спрашивается: разрешима ли в рациональных числах система

$$Ax = b? \tag{2}$$

Представим эту задачу в виде задачи вычисления некоторой функции. Для начала нужно описать кодировку входа в виде слова в некотором алфавите. В качестве алфавита возьмём

$$\{0, 1, \#, /, -\}.$$

Целые числа будем записывать в двоичной системе, используя знак «−» для обозначения отрицательных чисел. Рациональное число p/q будем

записывать словом *код*(p) / *код*(q), а последовательность чисел будем записывать, разделяя записи различных чисел знаком #.

В описанной кодировке вход задачи ЛИНЕЙНЫЕ УРАВНЕНИЯ НАД $\mathbb{Q}$ — слово *код*$(Ax = b)$, кодирующее последовательность чисел

$$m, n, a_{11}, \ldots, a_{1n}, a_{21}, \ldots, a_{2n}, \ldots, a_{m1}, \ldots, a_{mn}, b_1, \ldots, b_m,$$

где a_{ij} — элементы матрицы A, b_i — элементы b.

Функция, которую нужно вычислить, имеет вид

$$\chi(t) = \begin{cases} 1, \text{ если } t = \textit{код}(Ax = b) \text{ для некоторых } A, b \\ \quad \text{и } Ax = b \text{ имеет решения,} \\ 0, \text{ если } t = \textit{код}(Ax = b) \text{ для некоторых } A, b \\ \quad \text{и } Ax = b \text{ не имеет решений,} \\ \text{не определено в любом другом случае.} \end{cases} \tag{3}$$

Приведенный пример относится к одному из самых распространенных в математике типов вычислительных задач — проверке разрешимости системы уравнений или, более обшо, проверке заданного свойства у объекта, описание которого является входом задачи. В логике принято говорить о проверке *истинности предиката*. Так же, как и в примере 1, такой задаче сопоставляется функция, которая ставит в соответствие описанию объекта 1, если объект удовлетворяет свойству, и 0 в противном случае. Если слово не является описанием объекта, функция на таком слове не определена. Эта функция называется *характеристической функцией* множества описаний объектов, удовлетворяюших заданному свойству.

Сформулируем ешё несколько задач о разрешимости уравнений в виде задач вычисления характеристической функции некоторого предиката. Во всех упомянутых ниже задачах о разрешимости уравнений или систем уравнений подразумевается, что вычисляется характеристическая функция, аналогичная (3).

ПРИМЕР 2. ЛИНЕЙНЫЕ УРАВНЕНИЯ НАД $\mathbb{Z}^+$.

Даны: матрица A размера $m \times n$, элементы которой — целые числа, вектор-столбец b (матрица размера $m \times 1$), также состоящий из целых чисел.

Спрашивается: разрешима ли в неотрицательных целых числах система уравнений

$$Ax = b? \tag{4}$$

ПРИМЕР 3. СИСТЕМА УРАВНЕНИЙ В $\mathbb{F}_2$.

Даны: многочлены $p_1, \ldots, p_k \in \mathbb{F}_2[x_1, \ldots, x_n]$.

Спрашивается: разрешима ли в поле $\mathbb{F}_2$ система уравнений

$$p_i(x_1, \ldots, x_n) = 0, \quad i = 1, \ldots, k? \tag{5}$$

Как кодировать многочлен словом в некотором алфавите? Можно, например, указать степень многочлена d и количество переменных n, представить многочлен в виде суммы мономов

$$p(x_1, \ldots, x_n) = \sum_{\alpha_1 + \ldots + \alpha_n \leqslant d} p_{\alpha_1 \ldots \alpha_n} x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n},$$

после чего перечислить коэффициенты его мономов в некотором оговоренном заранее порядке. Другой способ состоит в том, чтобы перечислить все ненулевые мономы, указывая также их коэффициенты. Эти способы могут приводить к записям весьма разной длины: длина записи многочлена $x_1 \cdot \ldots \cdot x_n$ первым способом порядка n^n, а вторым — не больше, чем $n \log n$.

Пример 4. Уравнение над $\mathbb{Z}$.
Дан: многочлен $p \in \mathbb{Z}[x_1, \ldots, x_n]$.
Спрашивается: есть ли решения в целых числах у уравнения

$$p(x_1, \ldots, x_n) = 0? \tag{6}$$

2. Способы решения задач

Когда мы утверждаем, что задачу решить сложно (в предельном случае — невозможно), всегда остается сомнение, что учтены все мыслимые способы её решения. До конца это сомнение снять нельзя — кто может знать, что придумают люди в будущем? Но нужно, по крайней мере, стремиться к тому, чтобы учесть все известные на сегодняшний день способы вычислений.

При этом нас интересуют только такие способы, которые можно явно и однозначно описать и которые можно хотя бы в принципе реализовать. Последнее означает, что все действия, входящие в описание вычисления, должны быть *элементарными*, т. е. доступными всякому (даже самому тупому) исполнителю. Способ вычисления, удовлетворяющий таким свойствам, называется *алгоритмом*.

В 30-ые годы нашего (ещё) века была проделана основная часть работы по формальному определению понятия алгоритма. В результате появилось несколько различных определений и была доказана их эквивалентность. После чего был сформулирован знаменитый *тезис Чёрча*, который утверждает, что более общих определений, удовлетворяющих сформулированным выше условиям, не существует. Тезис Чёрча не является математическим утверждением, потому что приведенные условия не формализованы.

2.1. Исполнитель

Прежде чем давать формальное определение алгоритма, введем в обсуждение неформальный персонаж — Исполнителя. Это довольно ограниченное существо — у него конечная память (размер которой, впрочем, может быть сколь угодно велик). Доступные Исполнителю действия также весьма просты. Исполнитель умеет различать символы некоторого конечного алфавита (сколь угодно большого) и правильно их записывать. Кроме того, он умеет пользоваться таблицами. Это означает, что если выдать ему книжку, в которой приведена таблица некоторой функции из конечного множества в конечное множество, то Исполнитель может находить по аргументам функции её значение.

Помимо этого у Исполнителя есть карандаш, ластик и неограниченной (бесконечной) толщины тетрадь. Страницы тетради имеют ограниченный размер, так что Исполнитель может записать на них лишь ограниченное количество символов. На первых страницах тетради записано задание — вход той функции, которую нужно вычислить. Исполнитель может листать тетрадь, стирать символы, записывать новые. Заканчивается эта его деятельность тем, что он отдаёт тетрадь с результатом своей работы.

Инструкции, которым следует Исполнитель, собраны в отдельную книжечку. Можно считать, что они задают таблицу значений функции

$$\langle\text{состояние Исполнителя, текущая страница}\rangle \mapsto \langle\text{действия Исполнителя}\rangle,$$

где ⟨действия Исполнителя⟩ — это новое содержание текущей страницы и куда должен Исполнитель пролистать тетрадь — к началу или к концу.

Работа Исполнителя в несколько карикатурной форме отражает те реальные жизненные ситуации, когда мы складываем числа в столбик, приводим подобные члены в записи многочлена, вычисляем наибольший общий делитель двух чисел и т. п., т. е. когда мы «действуем по алгоритму».

С другой стороны, данное выше описание уже легко превратить в формальное определение, что и будет сделано в следующем подразделе. Впрочем, подробности этого формального определения не так уж важны. Единственное существенное в дальнейшем свойство состоит в том, что утверждение «Исполнитель правильно выполнил очередной шаг вычислений» можно записать достаточно короткой логической формулой, переменные которой кодируют описание состояний нашей системы (Исполнитель, тетрадь, книжка с инструкциями) до и после очередного действия Исполнителя. Это свойство хорошо согласуется с неформальным представлением об элементарном действии — если сделано какое-то простое действие, то и утверждение о том, что сделано именно это действие, должно

быть не слишком сложным (длинным). Доказательство этого свойства для используемого ниже формального определения алгоритма легко извлечь из приведенного в разд. 5 наброска доказательства теоремы Кука – Левина.

2.2. Машины Тьюринга

Приведем формальное определение алгоритма, согласованное с предыдущим неформальным обсуждением.

Машина Тьюринга (сокращённо МТ) однозначно задаётся указанием набора $(\mathcal{S}, \textvisiblespace, \mathcal{A}, \mathcal{Q}, q_0, \delta)$, где

- $\mathcal{S}$ — конечное множество, называемое *алфавитом* МТ (в предыдущем описании элементам этого множества соответствуют все возможные записи на одной странице тетради, выданной Исполнителю);
- $\mathcal{A}$ — также конечное множество, называемое *внешним алфавитом* (в предыдущем описании элементам этого множества соответствуют символы, которыми записывается задание для Исполнителя и результат его работы);
- $\textvisiblespace$ — пустой символ (или пробел), это некоторый элемент $\mathcal{S} \setminus \mathcal{A}$ (в предыдущем описании ему соответствует пустая страница в тетради Исполнителя);
- $\mathcal{Q}$ — конечное множество состояний управляющего устройства МТ (неформально — Исполнителя);
- $q_0 \in \mathcal{Q}$ — начальное состояние;
- $\delta\colon \mathcal{Q} \times \mathcal{S} \to \mathcal{Q} \times \mathcal{S} \times \{-1, 0, 1\}$ — (частичная) функция переходов МТ (неформально — книжка с инструкциями для Исполнителя).

Состояние МТ задаётся тройкой (σ, p, q), где σ — бесконечное слово в алфавите $\mathcal{S}$, т. е. произвольная последовательность $s_0, \ldots, s_n, \ldots$ элементов $\mathcal{S}$; p — неотрицательное целое число; $q \in \mathcal{Q}$. Символы слова σ будем, как это принято, представлять записанными на *ленте*, разбитой на *ячейки*, по ячейке на символ. На ленте также имеется *головка*, которая расположена над ячейкой с номером p. Наглядно это изображается так:

Положение головки				∇	
Ячейки	s_0	s_1	…	s_p	…
Номера ячеек	0	1		p	

Помимо ленты машина Тьюринга имеет *управляющее устройство*, состояние которого задаётся элементом q множества $\mathcal{Q}$.

Состояния МТ меняются дискретно. За один *такт работы* управляющее устройство выполняет следующие действия (полагаем, что МТ находится в состоянии (σ, p, q)):

а) *читает* символ, находящийся под головкой (т.е. определяет s_p);

б) *вычисляет* значение функции переходов: $\delta(q, s_p) = (q', s, \Delta p)$ (если функция переходов на паре (q, s_p) не определена, то останавливает машину Тьюринга);

в) *записывает* на ленту в ячейку p символ s, сдвигает головку на Δp и переходит в состояние q' (другими словами, новое состояние машины задаётся тройкой $((s_0, \ldots, s_{p-1}, s, s_{p+1}, \ldots), p + \Delta p, q')$);

г) если $p + \Delta p < 0$, то останавливает машину.

Работа машины Тьюринга начинается из состояния $(\alpha\text{␣}\ldots, 0, q_0)$, где за конечным словом α, состоящим из символов внешнего алфавита, (множество таких слов обозначается $\mathcal{A}^*$) следует бесконечное слово, целиком состоящее из пустых символов. Слово α будем называть *входом* МТ. В любой момент времени слово, записанное на ленте, однозначно записывается в виде $\sigma\text{␣}\ldots$, где последний символ слова σ — не пустой, а за ним идут только пустые символы. Будем называть слово σ *используемой частью ленты*.

Выполняя один такт работы за другим, машина Тьюринга порождает последовательность состояний

$$(\sigma_0, 0, q_0),\ (\sigma_1, p_1, q_1),\ (\sigma_2, p_2, q_2),\ \ldots$$

Если МТ останавливается, используемая часть ленты в достигнутом перед остановкой состоянии называется *результатом* работы МТ.

Каждая машина Тьюринга M *вычисляет* частичную функцию φ_M из $\mathcal{A}^*$ в $\mathcal{A}^*$, отображающую вход α в результат работы МТ на входе α при условии, что результат работы является словом во внешнем алфавите. Для входов, на которых машина не останавливается или результат содержит символы из $\mathcal{S} \setminus \mathcal{A}$, функция φ_M не определена. Из определения ясно, что любая МТ вычисляет ровно одну функцию (быть может, нигде не определённую).

Определение 1. *Частичная функция f из $\mathcal{A}^*$ в $\mathcal{A}^*$ называется вычислимой, если существует машина Тьюринга M, для которой $\varphi_M = f$. При этом будем говорить, что f вычислима на M.*

Не все функции вычислимы. Это ясно из сравнения мощности множества функций (континуум) и мощности множества машин Тьюринга (счётное множество). Есть и явные примеры невычислимых функций.

Один из важнейших — *проблема остановки*: дана машина Тьюринга и входное слово; нужно проверить, останавливается ли эта машина Тьюринга на данном входе.

Упражнение 1. *Определите функцию на множестве слов в конечном алфавите, отвечающую проблеме остановки машины Тьюринга.*

Подсказка: хотя машина Тьюринга реализует функцию на бесконечном множестве, она может быть описана конечным словом.

3. Сложность и сложностные классы

3.1. Сложность вычисления

Сложность вычисления (т.е. работы алгоритма на данном входном слове) определяется *ресурсами*, требуюшимися для этого вычисления. Два важнейших ресурса — *время* (количество тактов работы до остановки или, на другом используемом нами языке, — количество действий Исполнителя) и *память* (наибольший номер ячейки, над которым побывала головка МТ в процессе вычисления, или количество страниц, просмотренных Исполнителем).

Итак, ресурс, требуемый для конкретного вычисления, — это число.

3.2. Сложность алгоритма

Чтобы охарактеризовать сложность работы алгоритма (т.е. способа вычисления функции), мы должны принять во внимание работу алгоритма на всех входах. Сделаем мы это одним из наиболее распространенных способов, который называется «сложность в наихудшем случае».

Будем говорить, что машина Тьюринга M работает за время $T_M(n)$, если максимальное (по всем входам длины n) количество тактов, которое проработает M до остановки, равно $T_M(n)$. Аналогично, машина Тьюринга M работает на памяти $S_M(n)$, если наиболее удалённое от начала ленты положение головки при вычислениях на входах длины n равно $S_M(n)$.

Итак, сложность алгоритма характеризуется некоторой функцией от натурального параметра.

3.3. Сложность задачи

Но давайте вспомним, что нас интересует сложность вычислительной задачи. Решать задачу можно разными способами и есть разные формальные модели вычислений. К тому же (см. [6]), наивный подход — определять сложность задачи по сложности наилучшего алгоритма, её решающего, не работает.

Поэтому принят другой способ характеризации сложности задачи. Он состоит в том, что выделяются классы тех функций, вычисление которых возможно при задаваемых ограничениях на потребляемые ресурсы.

Наиболее важные классы получаются, если накладывать ограничения на рост времени работы и/или используемой памяти в зависимости от длины входного слова. А наиболее важное различие между эффективными и неэффективными вычислениями задаётся функциями *полиномиального роста*. Функция $f(n)$ — полиномиального роста, если для некоторой константы d при достаточно больших n выполняется неравенство $f(n) \leqslant n^d$. В этом случае будем использовать обозначение $f(n) = \mathrm{poly}(n)$.

Определение 2. *Функция f принадлежит классу* P (*и называется полиномиально вычислимой*), *если она вычислима на машине Тьюринга M, для которой $T_M(n) = \mathrm{poly}(n)$.*

Определение 3. *Функция f принадлежит классу* PSPACE (*и называется вычислимой с полиномиальной памятью*), *если она вычислима на машине Тьюринга M, для которой $S_M(n) = \mathrm{poly}(n)$.*

3.4. Почему полиномы?

Почти всякий, кто знакомится с теорией сложности вычислений, задаёт в этом месте вопрос: «почему эффективность вычислений отождествляется с полиномиальностью»? Такой выбор не представляется очевидным, и уж заведомо он не единственно возможный. Поскольку в дальнейшем изложении мы будем без специальных оговорок отождествлять полиномиальную вычислимость и эффективность, стоит сделать отступление и дать хотя бы какие-то пояснения.

Основания для такого выбора делятся на две группы: теоретические и практические. Начнём с первых.

- ▷ Полиномы удобны тем, что они замкнуты относительно композиции. Это обстоятельство будет активно эксплуатироваться в теории, излагаемой ниже.

- ▷ Вспомним, что при формулировке вычислительной задачи как задачи вычисления функции на словах есть произвол, связанный с выбором возможной кодировки входа и результата. Например, перестановку из n элементов можно задавать таблицей значений (как функцию на множестве $\{1, \ldots, n\}$) или матрицей $n \times n$. Последний способ не экономен — нужно задать $O(n^2 \log n)$ битов вместо $O(n \log n)$ битов, как в первом случае, но часто удобен при формулировке задач и алгоритмов. Хотелось бы, чтобы определение сложности задачи не зависело от таких «мелочей», как удобство формулировки.

Как ни странно, отождествление эффективного и полиномиального вычисления имеет смысл и с точки зрения практики вычислений, хотя и не всегда применимо. Напомним, что все наши рассуждения проводятся с точностью до мультипликативной константы, так что применимость рассматриваемой теории к практике нуждается в специальных обоснованиях. Тем не менее, уже к середине 60-ых годов опыт практического программирования показал, что если для полиномиальных алгоритмов есть смысл бороться за эффективную реализацию (практически возникающие задачи решаются за разумное время), то для большинства алгоритмов, не имеющих полиномиальных верхних оценок, решение практически возникающих задач занимает неприемлемо большое время и гораздо разумнее изобретать эвристические приемы решения таких задач, рассматривать приближенные постановки и т. п.

В последующие годы смысл понятия «эффективно вычислимый» уточнялся и несколько изменился. Появилось, скажем, понятие «вычислимый за реальное время». Оно более адекватно описывает ситуации, возникающие в вычислительных системах, от которых требуется по-настоящему быстрая реакция. Ни одного пользователя не устроит текстовый редактор, который вставляет в текст новый символ за время, линейно зависящее от размера текста. Данные выше определения не позволяют формально описать такие ситуации, интересовать они нас не будут. Однако читателю полезно попробовать построить формальное определение, исходя из предложенного примера.

Понятие эффективного алгоритма в последние годы стало включать в себя и вероятностные алгоритмы, работающие за полиномиальное время (о них см. ниже, раздел 8.1).

4. Как различать сложные и простые задачи?

Читатель, не потерявший основную нить рассуждений, должен в этом месте удивиться: «Как? Разве только что не было дано разъяснение этого вопроса? Простые задачи решаются за полиномиальное время, сложные — нет!»

Увы, такой простой ответ применим далеко не всегда. Конечно, есть много задач, для которых построены полиномиальные алгоритмы. Все такие задачи мы считаем простыми. Из примеров, приведенных выше, простой является задача о разрешимости системы линейных уравнений в рациональных числах (пример 1). Полиномиальный алгоритм решения этой задачи строится легко (читателя, интересующегося эффективными алгоритмами, в том числе и алгоритмами решения систем линейных уравнений, отсылаем к книгам [2], [9], [1]).

В некоторых случаях доказано отсутствие алгоритма, решающего задачу; пример 4 — один из них (это вариант десятой проблемы Гильберта; доказательство невычислимости соответствующей функции см., например, в [5]).

Но для огромного числа естественно возникающих вычислительных задач не удаётся ни построить эффективный (=полиномиальный) алгоритм, ни доказать отсутствие такого алгоритма. Именно таковы остальные примеры из раздела 1: решение линейного уравнения в неотрицательных целых числах и решение системы уравнений в поле из двух элементов. Конечно, неудача многолетних усилий по построению эффективного алгоритма для некоторой задачи уже является весомым аргументом в пользу её сложности. Но хочется иметь теорию, в которой такие задачи можно доказуемо отделить от простых задач.

Подходящая теория была построена в начале 70-ых годов и получила с тех пор широкое распространение. Основная идея состоит в том, чтобы ввести на множестве задач отношение *сводимости* («задача A не сложнее задачи B») и характеризовать сложные задачи как задачи, к которым сводятся все задачи из некоторого класса. Ниже в этом разделе описывается самый популярный вариант такой теории, основанный на классе предикатов, вычислимых за полиномиальное время *недетерминированными*[2)] машинами Тьюринга (класс NP).

4.1. Полиномиальная сводимость

Определение 4. Сводимость по Карпу. *Функция f_1 сводится к функции f_2 (обозначение $f_1 \propto f_2$), если существует такая функция $f \in$ $\in$ P, что $\forall x\ f_1(x) = f_2(f(x))$.*

Сводимость по Карпу также называют полиномиальной сводимостью, а часто — просто сводимостью. Отношение $\propto$ будем рассматривать как формальный вариант отношения «задача f_1 не сложнее задачи f_2». Действительно, если $f_2 \in$ P, то и $f_1 \in$ P: полиномиальный алгоритм для вычисления f_1 использует последовательно полиномиальные алгоритмы для f (на входе x) и для f_2 (на входе $f(x)$).

4.2. Класс NP

Прежде всего заметим, что этот класс включает в себя только характеристические функции (напомним, что задачи, соответствующие таким функциям, состоят в проверке некоторого свойства входного слова).

[2)] Мы не объясняем, что это такое; нужный нам класс NP можно определить, не прибегая к недетерминированным вычислениям.

Теперь дадим неформальное описание этого класса задач. У нас появляется новый персонаж — Советник. От Исполнителя Советник отличается двумя чертами: он *интеллектуально всемогущ* и *пристрастен* (это означает, что Советник хочет, чтобы у рассматриваемого объекта было признано наличие исследуемого свойства, безотносительно к истинному положению дел). Последняя особенность не позволяет Исполнителю слепо опираться на мнение Советника: оно всегда одинаково. Исполнитель может задавать Советнику вопросы и действовать, исходя из полученных ответов. Каждый вопрос–ответ считается одним действием. Решение о значении функции принимает Исполнитель.

Замечание 1. В литературе по теории сложности Исполнитель (несколько более сильный — снабженный монеткой для подбрасывания) именуется Артуром, а Советник — Мерлином.

Функция (характеристическая) $f(x)$ принадлежит классу NP, если существует алгоритм для пары (Исполнитель, Советник), который всегда (при любых возможных вариантах диалога между Исполнителем и Советником) заканчивается за полиномиальное от длины входа время и результат работы которого удовлетворяет следующему условию: если $f(x) = 1$, то существует такой вариант диалога между Исполнителем и Советником, что результат равен 1; если же $f(x) = 0$, то при любом варианте диалога результат равен 0.

Замечание 2. Из данного выше неформального определения ясно, что $\mathrm{P} \subseteq \mathrm{NP}$ (Исполнитель может ничего не спрашивать). Является ли это включение строгим? Довольно интенсивные, хотя и безуспешные, попытки ответить на этот вопрос продолжаются уже почти 30 лет. С. Смейл включил проблему $\mathrm{P} \stackrel{?}{\neq} \mathrm{NP}$ в число трёх важнейших математических проблем следующего столетия (две другие — гипотеза Римана и гипотеза Пуанкаре), см. [10].

Прежде чем давать формальное определение класса NP, заметим следующее. Исполнитель работает вполне определённым образом, а Советник — всеведущ. Поэтому, посмотрев на вход, Советник может сразу сообщить Исполнителю весь их диалог, а Исполнитель — убедиться, что Советник не соврал; Исполнителю на это потребуется время, полиномиально зависящее от длины диалога.

Определение 5. *Функция f (со значениями в множестве $\{0,1\}$) принадлежит классу* NP, *если она представима в форме*

$$f(x) = \exists\, y\, \big((|y| < q(|x|)) \wedge R(x,y)\big),$$

где $q(\cdot)$ — полином, $R(\cdot,\cdot) \in \mathrm{P}$, а $|\cdot|$ — длина слова.

Здесь (и всюду далее) мы отождествляем логические значения «ложь» и «истина» с 0 и 1 соответственно.

В духе предыдущего неформального обсуждения это определение нужно понимать так: y — это сообщение Советника Исполнителю, а $R(\cdot,\cdot)$ — алгоритм проверки, который осуществляет Исполнитель.

Замечание 3. Приведем ещё несколько неформальных интерпретаций класса NP. Слово y в данном выше определении можно понимать как «подсказку» Исполнителю, воспользовавшись которой он может проверить выполнение NP-свойства. Другими словами, у NP-задачи есть ответ, который, быть может, трудно найти, но проверить правильность ответа — легко. Популярно также представление о слове y как о «доказательстве наличия свойства» (подразумевается, что изучение доказательства занимает полиномиальное от его длины время).

Лемма 1. *Пусть* $f_1 \propto f_2$. *Тогда*

а) $f_1 \notin \mathrm{P} \Rightarrow f_2 \notin \mathrm{P}$; б) $f_2 \in \mathrm{NP} \Rightarrow f_1 \in \mathrm{NP}$.

Доказательство. Пункт а) фактически доказан выше.

Докажем пункт б), привлекая неформальных персонажей из предыдущего обсуждения. Советник сообщает Исполнителю $f(x)$ (длина которого ограничена некоторым полиномом h от длины x, поскольку $f \in \mathrm{P}$) и слово y, которое убеждает Исполнителя в том, что $f_2(f(x)) = 1$. Исполнитель может проверить, что ему действительно сообщено $f(x)$.

Определение 6. *Функция* $f \in \mathrm{NP}$ *называется* NP-*полной, если любая функция из* NP *к ней сводится*[3)].

Если некоторую NP-полную функцию f можно вычислять за время $T(n)$, то любую функцию g из NP можно вычислять за время $T(n^c)$, где число c зависит от g, но не от входа.

NP-полные функции существуют, например, функция задающая предикат выполнимость для булевых формул:

$$SAT(x) = \begin{cases} 1, \text{ если } \exists\, t_1, \ldots, t_k : x(t_1, \ldots, t_k) = 1, \\ 0 \text{ в противном случае.} \end{cases}$$

В первой строчке предполагается, что x есть запись логической формулы с булевыми переменными $t_1, \ldots, t_k$ и пропозициональными связками $(\neg, \vee, \wedge)$.

Теорема 1 (Кук, Левин). 1) $SAT \in \mathrm{NP}$; 2) SAT — NP-*полна.*

Следствие. *Если* $SAT \in \mathrm{P}$, *то* $\mathrm{P} = \mathrm{NP}$.

[3)] Ниже будут использоваться аналогичные понятия полноты для других сложностных классов.

Эскиз доказательства теоремы Кука – Левина будет приведен в следующем разделе.

А пока заметим, что бóльшая часть доказательств NP-полноты использует полиномиальную сводимость и следующую лемму.

Лемма 2. *Если* $SAT \propto L$ *и* $L \in \mathrm{NP}$, *то* L — NP-*полная. И вообще, если* L_1 — NP-*полная,* $L_1 \propto L_2$ *и* $L_2 \in \mathrm{NP}$, *то* L_2 — NP-*полная.*

Доказательство. Достаточно проверить транзитивность сводимости: если $L_1 \propto L_2$, $L_2 \propto L_3$, то $L_1 \propto L_3$. Она следует из того, что композиция двух полиномиально вычислимых функций полиномиально вычислима.

5. Набросок доказательства теоремы Кука – Левина

1) Это утверждение почти очевидно. Советник сообщает Исполнителю значения переменных, входяших в формулу, при которых она истинна. Исполнитель справится с проверкой истинности полученного высказывания за полиномиальное время.

2) Пусть функция из NP, которую нужно свести к SAT, имеет вид $f(x) = \exists y\,((|y| < q(|x|)) \wedge R(x,y))$.

Рассмотрим машину Тьюринга, вычисляющую $R(\cdot,\cdot)$ за полиномиальное время. Нам нужно записать в виде логической формулы условие того, что при некотором y (и заданном входе x), машина выдаёт результат 1. Для этого представим весь процесс вычисления, длящегося время $T = \mathrm{poly}(n)$ и использующего память $S = \mathrm{poly}(n)$, *таблицей вычисления* размера $T \times S$, показанной на рисунке 1.

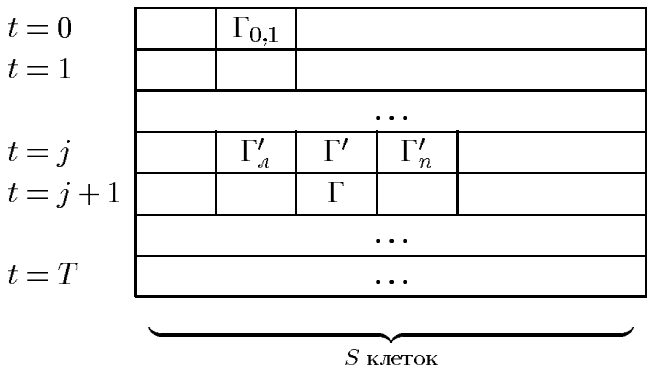


Рис. 1.

Строка с номером j таблицы задаёт состояние МТ после j тактов работы. Символы $\Gamma_{j,k}$, записанные в таблице, принадлежат алфавиту

$\mathcal{S} \times \{\varnothing \cup \mathcal{Q}\}$. Символ $\Gamma_{j,k}$ определяет пару (символ, записанный в k-й ячейке после j тактов работы; состояние управляющего устройства после j тактов работы, если головка находится над k-й ячейкой, в противном случае второй элемент пары — $\varnothing$). Для простоты также считаем, что если вычисление заканчивается при некотором входе за $T' < T$ тактов, то строки с номерами, бóльшими T', повторяют строку с номером T'.

Состояние каждой клетки таблицы можно закодировать конечным (не зависящим от n) числом булевых переменных. Имеются *локальные правила согласования*, т. е. состояние каждой клетки Г в строке ниже нулевой однозначно определяется состояниями клеток в предыдущей строке, лежащих непосредственно над данной (Γ'), левее данной ($\Gamma'_л$) и правее данной ($\Gamma'_п$). Каждое такое условие можно записать в виде логической формулы от переменных, кодирующих состояния клеток, причем размер этой формулы от n также не зависит.

Еще нам нужно записать условие успешности вычисления (результат равен 1). Для этого заметим, что без ограничения общности можно считать, что состояния клеток таблицы кодируются так, что одна из кодирующих переменных равна 1 только в том случае, когда в ячейке записана 1. Тогда значение этой переменной для кода $\Gamma_{T,0}$ и будет результатом вычисления.

Определим формулу φ_x как конъюнкцию всех формул, в которые подставлены значения переменных, кодирующих вход $x\#y$, дополненный символами $\sqcup$ до длины $|x| + 1 + q(|x|)$. Значения, соответствующие x и $\#$, — константы, поэтому переменные, от которых зависит эта формула, отвечают y и кодам внутренних ячеек таблицы. Так что можно считать, что формула φ_x зависит от y и ещё от каких-то переменных, которые мы обозначим z.

Итак, мы сопоставили слову x формулу $\varphi_x(y, z)$, которая по построению обладает следующим свойством. Если выполняется $R(x, y)$, то найдется такой набор значений $z(x, y)$, при котором $\varphi_x(y, z(x, y))$ истинна (эти значения описывают работу МТ на входе $x\#y$). А если $R(x, y)$ не выполняется, то $\varphi_x(y, z)$ всегда ложна (поскольку по сути утверждает, что вычисление на входе (x, y) даёт ответ «да»). Таким образом, при $f(x) = 1$ такая формула иногда (при некоторых значениях y) истинна, при $f(x) = 0$ — всегда ложна.

6. Примеры NP-полных задач

Обширный список NP-полных задач содержится в книге Гэри и Джонсона [3]. Как правило, их NP-полнота доказывается с помощью сведéний. Приведём несколько примеров таких доказательств.

6.1. 3-КНФ

Эта задача задаётся предикатом

3-$SAT(x) = 1 \Longrightarrow$ x есть 3-КНФ, которая истинна при некоторых значениях переменных. 3-КНФ — это конъюнкция дизъюнкций, каждая из которых содержит три литерала, а литерал — это переменная или её отрицание.

3-$SAT(x) = 0 \Longrightarrow$ x есть 3-КНФ, которая ложна при всех значениях переменных.

3-$SAT(x)$ не определена во всех остальных случаях.

3-SAT также NP-полна. Это устанавливается сведением к ней SAT.

Теорема 2. $SAT \propto$ 3-SAT.

Доказательство. Рассмотрим вычисление истинности заданной формулы $\varphi(x_1, \ldots, x_n)$, использующей связки $(\neg, \vee, \wedge)$. Результаты промежуточных вычислений обозначим $y_1, \ldots, y_s$. Для каждой переменной y_k выполнено одно из трёх равенств

$$\begin{aligned} y_k &= u \vee v, \\ y_k &= u \wedge v, \\ y_k &= \neg v, \end{aligned} \tag{7}$$

где u, v либо переменные формулы, либо уже определённые вспомогательные переменные, т. е. $u, v \in \{x_1, \ldots, x_n, y_1, \ldots, y_{k-1}\}$.

Теперь запишем 3-КНФ, равносильную исходной формуле $\varphi(\cdot)$. В эту КНФ будут входить переменные $x_1, \ldots, x_n$ и $y_1, \ldots, y_s$. Построим вначале конъюнкцию K'' условий, означающих, что каждая из вспомогательных переменных y_k имеет значение, задаваемое формулой (7). Есть три типа таких условий и каждый можно записать в виде 3-КНФ:

$$\begin{aligned} \big(y \Leftrightarrow (x_1 \vee x_2)\big) &= (x_1 \vee x_2 \vee \neg y) \wedge (\neg x_1 \vee x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee y) \wedge \\ &\quad \wedge (\neg x_1 \vee \neg x_2 \vee y), \\ \big(y \Leftrightarrow (x_1 \wedge x_2)\big) &= (x_1 \vee x_2 \vee \neg y) \wedge (\neg x_1 \vee x_2 \vee \neg y) \wedge (x_1 \vee \neg x_2 \vee \neg y) \wedge \\ &\quad \wedge (\neg x_1 \vee \neg x_2 \vee y), \\ \big(y \Leftrightarrow \neg x\big) &= (x \vee y) \wedge (\neg x \vee \neg y). \end{aligned}$$

Подставляя эти 3-КНФ в K'', получим 3-КНФ K'. Искомая 3-КНФ имеет вид $K = K' \wedge y_s$. Действительно, истинность K равносильна утверждению: все присваивания выполнены правильно и в результате получилась 1 ($y_s = 1$). Значит, если при каких-то значениях переменных x_i формула φ истинна, то 3-КНФ K выполнима, и наоборот.

6.2. Разрешимость системы уравнений в поле $\mathbb{F}_2$

Теперь докажем NP-полноту задачи из примера 3. Вспомним, что мы можем записывать многочлен двумя существенно различающимися по длине способами. Так что речь идет о двух, вообще говоря, разных вычислительных задачах. Однако они обе принадлежат классу NP. Это вполне очевидно — если Советник сообщает решение системы, то Исполнитель может проверить выполнение всех равенств при данных значениях переменных за полиномиальное время и в одном, и в другом случае.

Более того, обе эти задачи NP-полны. Для доказательства полноты сведем задачу выполнимости 3-КНФ к разрешимости системы вида (5). Запишем связки $\neg, \vee, \wedge$ в виде многочленов. Если считать, что истинностные значения — это 0 и 1, то для любого поля и любых $x, y \in \{0, 1\}$ выполнено

$$\begin{aligned} (\neg x) &= (1 - x), \\ (x \wedge y) &= xy, \\ (x \vee y) &= 1 - (1 - x)(1 - y). \end{aligned} \tag{8}$$

Поэтому выполнимость 3-КНФ $\bigwedge_{k=1}^{m} D_k$ равносильна разрешимости над полем $\mathbb{F}_2$ системы

$$A(D_k) + 1 = 0, \quad k = 1, \ldots, m, \tag{9}$$

где $A(D_k)$ обозначает многочлен, соответствующий дизъюнкции D_k при соответствии, задаваемом формулами (8). Поскольку мы использовали для сводимости многочлены степени 3, то полнота доказана для обеих кодировок: для многочленов степени 3 длины записей многочлена этими способами различаются не более чем кубически.

6.3. Целочисленное линейное программирование (ЦЛП)

Теперь разберем задачу из примера 2, т.е. разрешимость системы линейных диофантовых уравнений в неотрицательных числах.

Замечание 4. Название «линейное программирование» относится к задачам минимизации линейной функции на множестве решений системы линейных неравенств. Слово «целочисленное» указывает, что учитываются только те решения, для которых значения всех переменных — целые. С точностью до полиномиальных сводимостей задача разрешимости системы (4) эквивалентна задаче ЦЛП. Доказательство этого оставляется читателю в качестве упражнения.

Теорема 3. *Задача проверки разрешимости системы линейных уравнений в неотрицательных целых числах* NP*-полна.*

Как и в предыдущих случаях, нам необходимо доказать два утверждения: что эта задача принадлежит классу NP и что к ней сводятся все задачи из NP. Но теперь нетривиальны оба утверждения. Естественно пытаться доказывать принадлежность NP следующим образом: пусть Советник сообщит решение, а Исполнитель его проверит. Но тогда решение должно состоять из не слишком больших чисел, чтобы Исполнитель мог проверить выполнение равенств (4) за полиномиальное время.

Оказывается, что так оно и есть: из разрешимости системы (4) следует существование не слишком длинных решений. Опишем кратко идею доказательства.

Основная используемая оценка — это оценка величины определителя матрицы через длину её описания. Обозначим длину записи матрицы A (кодировка указана в примере 1) через s. Тогда

$$|\det A| \leqslant \prod_{i,j=1}^{n} (1 + |a_{ij}|) \leqslant 2^s. \tag{10}$$

Первое неравенство следует непосредственно из определения $\det A$, второе — из того, что запись числа a требует не менее $1+\log|a| \geqslant \log(1+|a|)$ битов.

Множество вещественных решений системы (4) является *полиэдром* — пересечением конечного числа полупространств. Хорошо известная теорема выпуклой геометрии утверждает, что любая точка полиэдра является выпуклой комбинацией его крайних точек и точек на его крайних лучах. Крайние точки полиэдра, задаваемого системой (4), являются решениями систем линейных уравнений вида

$$\begin{cases} Ax = b, \\ x_i = 0, \quad i \in S, \end{cases} \tag{11}$$

для каких-то подмножеств $S \subseteq \{1, \ldots, n\}$. Координаты этих точек по правилу Крамера являются отношениями миноров расширенной матрицы этой системы. В силу оценки (10) они не превосходят 2^{s+n^2+n} (в показателе написана максимально возможная длина описания минора расширенной матрицы).

Направляющие векторы крайних лучей — ненулевые (без ограничения общности, целочисленные) решения однородных систем вида

$$\begin{cases} Ax = 0, \\ x_i = 0, \quad i \in S, \end{cases} \tag{12}$$

поэтому можно считать, что для их координат выполняются те же неравенства.

Обозначим крайние точки полиэдра $x^{(1)}, \ldots, x^{(N)}$, направляющие векторы крайних лучей $y^{(1)}, \ldots, y^{(M)}$. Пусть у системы (4) есть целочисленное решение x. Тогда его можно записать в виде

$$x = \sum_{i=1}^{N} \lambda_i x^{(i)} + \sum_{j=1}^{M} \mu_j y^{(j)} = x^{(\lambda)} + \sum_{j=1}^{M} (\mu_j - \lfloor \mu_j \rfloor) y^{(j)} + y^{(\mu)}, \qquad (13)$$

где $\lambda_i, \mu_j \geqslant 0$, $\sum_{i=1}^{N} \lambda_i = 1$, все координаты вектора $x^{(\lambda)}$ ограничены 2^{s+n^2+n}, а вектор $y^{(\mu)}$ — целочисленный. Но тогда и вектор $x - y^{(\mu)}$ является целочисленным решением системы (4), а его координаты не превосходят $(M+1)2^{s+n^2+n}$. Заметим, что M заведомо не превосходит 2^n (а если вспомнить теорему Каратеодори, то $M \leqslant n$).

Итак, мы доказали, что для разрешимой в неотрицательных целых числах системы найдется и такое решение, длина записи которого $O(s^3)$ (мы грубо оценили n как s). Отсюда следует принадлежность этой задачи классу NP.

Для доказательства NP-полноты задачи разрешимости системы линейных уравнений в неотрицательных целых числах сведем к ней задачу 3-КНФ. Построим по 3-КНФ систему линейных уравнений в неотрицательных целых числах. Булевой переменной x_i из 3-КНФ сопоставим целочисленную неотрицательную переменную p_i, отрицанию переменной x_i — n_i, дизъюнкции D — переменную q_D. Каждой дизъюнкции $D = X_j \vee X_k \vee X_m$ (X_* — литералы) сопоставим также уравнение $P_j + P_k + P_m - q_D = 1$, в котором P_j, P_k, P_m — переменные, сопоставленные литералам дизъюнкции.

Искомая система содержит для всех i уравнения $p_i + n_i = 1$, а также все уравнения, сопоставленные дизъюнкциям из КНФ. Очевидно, что выполнимость 3-КНФ равносильна совместности такой системы.

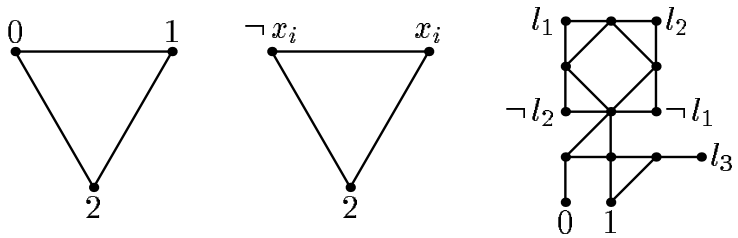
6.4. 3-раскраска

Дан граф. Спрашивается, можно ли раскрасить его вершины в три цвета так, чтобы концы каждого ребра были покрашены в разные цвета?

Будем считать, что граф задаётся матрицей смежности A. Строки и столбцы этой матрицы индексированы вершинами графа, $a_{ij} = 1$ если в графе есть ребро ij, в противном случае $a_{ij} = 0$.

Легко понять, что задача 3-раскраска принадлежит NP: Советник предъявит раскраску, а Исполнитель за линейное от длины описания графа время проверит, что она правильная.

Теорема 4. *Задача* 3-раскраска NP-*полна.*



константы переменная дизъюнкция

а) б) в)

Рис. 2.

Доказательство. Сведем к этой задаче задачу 3-КНФ. Пусть есть 3-КНФ из m дизъюнкций, в которые входят n переменных. Граф, который сопоставляется этой КНФ, имеет $7m+2n+3$ вершин. Для описания структуры графа удобно пометить часть вершин. Во-первых, пометим три вершины числами 0, 1, 2. Во-вторых, пометим каждым литералом (переменной или её отрицанием) по одной вершине. Остальные $7m$ вершин разобьём на группы по 7, каждая группа соответствует одной из дизъюнкций.

Теперь опишем рёбра этого графа. Как показано на рис. 2а, вершины 0, 1 и 2 соединены между собой рёбрами. На рис. 2б показаны ещё n треугольников в этом графе. И, наконец, рис. 2в показывает, как соединены рёбрами вершины, соответствующие каждой дизъюнкции. На этом рисунке l_1, l_2, l_3 обозначают вершины, помеченные литералами, входящими в дизъюнкцию. Заметим, что некоторые рёбра мы перечислили по несколько раз.

Очевидно, что описанное выше построение можно выполнить за полиномиальное от n и m время. Докажем его корректность.

Рассмотрим, какие правильные раскраски в 3 цвета возможны для такого графа. Без ограничения общности можно считать, что вершины 0, 1 и 2 покрашены в цвета 0, 1 и 2 (из шести раскрасок, различающихся перестановками цветов, мы выбрали одну). Тогда вершины, помеченные x_i и $\neg x_i$, покрашены в цвета 0 и 1, причём их цвета должны быть противоположны (см. рис. 2б).

Прямым перебором вариантов можно проверить, что граф, изображённый на рис. 2в, удовлетворяет следующему свойству: если красить вершины, помеченные литералами, в цвета 0 или 1, а вершины, отмеченные отрицаниями литералов, — в противоположные цвета 1 или 0, то правильная раскраска остальных вершин этого графа в 3 цвета существует (и единственна!) тогда и только тогда, когда хотя бы одно из значений литералов отлично от 0.

Поэтому правильные 3-раскраски построенного графа, для которых вершины 0, 1, 2 покрашены в цвета 0, 1, 2 соответственно, находятся во взаимно однозначном соответствии с выполняющими наборами значений переменных исходной 3-КНФ.

Заметим, что проверить раскрашиваемость графа в 2 цвета просто. Нужно покрасить одну из вершин в какой-нибудь цвет, все смежные с этой вершины в противоположный и т. д. Если ещё не покрашенные вершины не соединены ни с одной из уже покрашенных, опять красим одну из них в произвольно выбранный цвет. Мы можем столкнуться с тем, что какую-то вершину нельзя покрасить ни в один цвет. Тогда граф нельзя правильно раскрасить в 2 цвета. Доказательство корректности этого (очевидно, полиномиального) алгоритма оставляется читателю в качестве легкого упражнения.

Знаменитая гипотеза четырёх красок утверждает, что любой планарный граф (т.е. граф, который можно нарисовать на плоскости без пересечений ребер) раскрашивается в 4 цвета. Легко построить примеры планарных графов, не раскрашиваемых в 3 цвета (скажем, полный граф на 4 вершинах, см. рис. 3).

Оказывается, что проверка раскрашиваемости планарного графа в 3 цвета NP-полна. Доказательство проводится полиномиальным сведением общей задачи 3-РАСКРАСКА к задаче 3-РАСКРАСКА ПЛАНАРНОГО ГРАФА. Идея сведения очень проста, но использует нетривиальную конструкцию, изображенную на рис. 4 (мы позаимствовали эту конструкцию из книги [3], в свою очередь Гэри и Джонсон приписывают её М. Дж. Фишеру).

Итак, нам нужно по произвольному графу G построить некоторый планарный граф, 3-раскрашиваемость которого равносильна 3-раскрашиваемости исходного графа. Нарисуем граф G на плоскости, допуская пересечения ребер, но не проводя ребра через вершины. Из нарисованного с пересечениями ребер графа G изготовим планарный граф по следующему рецепту. Вместо каждой точки пересечения ребер вклеим экземпляр графа H, изображенного на рис. 4, так, чтобы вершины, помеченные x, x', оказались на одном ребре, а вершины, помеченные y, y' — на другом. С полученным (уже планарным) графом проделаем ещё одну процедуру. Выбрав ребро uv исходного графа, будем двигаться от u к v и стягивать ребра типа xx', соединяющие вершины из идущих подряд вдоль нашего пути экземпляров графа H. Планарность при этом сохраняется. Стянем также ребро, соединяющее вершину u с первой из вершин графов H на нашем пути. (Если на ребре uv не было точек пересечения с другими ребрами, то не делаем ничего.) Повторив эту процедуру для всех ребер графа G, получим искомый граф $P(G)$. Заметим, что множество вершин $P(G)$ содержит по построению множество вершин G.

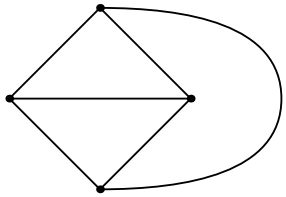


Рис. 3.

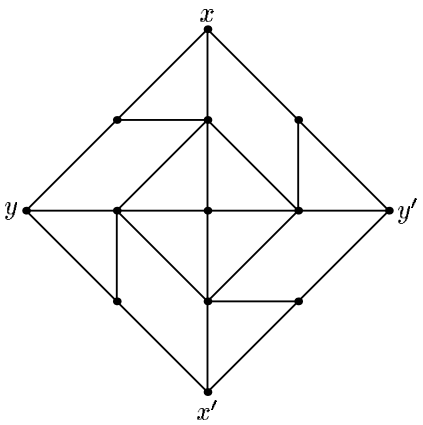


Рис. 4.

Полиномиальность такого сведения очевидна. Доказательство его корректности использует следующие два свойства графа H:

- ▷ при любой раскраске H в 3 цвета цвета вершин x и x', равно как и цвета y и y', совпадают;
- ▷ есть раскраски в три цвета, при которых вершины x и y покрашены одинаково, есть и такие, при которых они покрашены в разные цвета.

Читателю предоставляется самостоятельно проверить выполнение этих свойств и убедиться, что правильная 3-раскраска $P(G)$ порождает правильную 3-раскраску G (красим вершину в тот же цвет, в который она покрашена в $P(G)$) и наоборот (любая правильная 3-раскраска вершин G продолжается до правильной 3-раскраски $P(G)$).

7. Класс PSPACE

Как уже говорилось, в этот класс попадают те функции, которые могут быть вычислены на МТ, использующей память, ограниченную полиномом от длины входного слова. Этот класс включает в себя класс P (за время T мы сможем использовать память, не превосходящую T) и класс NP (диалог Советника с Исполнителем занимает полиномиальное время, поэтому Исполнитель, не ограниченный во времени, может перебрать по очереди все варианты записей такого диалога, используя для этого сравнительно небольшую — ограниченную полиномом от длины входа — память). Но класс PSPACE, по-видимому, гораздо шире чем NP. К сожалению, доказать это никому до сих пор не удалось.

Классу PSPACE, точнее говоря, характеристическим функциям, входящим в PSPACE, можно дать следующее неформальное описание. Представим несколько идеализированную модель судопроизводства. Слово x — описание некоторого дела, находящегося на рассмотрении суда, который должен вынести вердикт: виновен ли обвиняемый (значение характеристической функции равно 1). Судья — уже знакомый нам полиномиально ограниченный Исполнитель (он будет работать время, не превосходящее полинома от длины описания дела). Есть также Прокурор, настаивающий на виновности обвиняемого, и Адвокат, убеждающий судью в обратном. Оба они интеллектуально всемогущи и по очереди приводят свои аргументы. Правила ведения судебного разбирательства зафиксированы в Уголовно-Процессуальном Кодексе. Судья подводит итог разбирательства, основываясь на описании дела x, на состоявшейся дискуссии между Прокурором и Адвокатом и на УПК (т. е. по сути вычисляет значение некоторой полиномиально вычислимой функции, определяемой УПК, на входе, описывающем судебное разбирательство — слово x, текст первого выступления Прокурора, текст первого выступления Адвоката, текст второго выступления Прокурора и т. д.).

В таком экстравагантном контексте класс PSPACE определяется следующим образом: характеристическая функция f принадлежит PSPACE, если можно придумать такой УПК, что при $f(x) = 1$ у Прокурора есть стратегия ведения дискуссии, которая убеждает Судью в виновности обвиняемого при любых вариантах действий Адвоката; а при $f(x) = 0$ аналогичная стратегия есть у Адвоката.

Замечание 5. Обычно описанную выше модель излагают в более нейтральных терминах. Говорят об игре двух лиц, зависящей от входного слова, и существовании выигрышной стратегии у одного из игроков.

Теорема 5. *Стандартное и уголовно-процессуальное определения класса* PSPACE *равносильны.*

Доказательство. Покажем, что язык из PSPACE в смысле УПК принадлежит PSPACE в смысле стандартного определения. Пусть число выступлений сторон ограничено $p(|x|)$. Определим по индукции набор машин Тьюринга M_k для $k = 0, \ldots, p(|x|)$. Каждая M_k по заданному началу дискуссии $x, a_1, b_1, \ldots$ длины k определяет наличие убеждающей стратегии для Прокурора. Последней в этом ряду машине $M_{p(|x|)}$ нужно просто проимитировать работу Судьи и вычислить функцию УПК$(x, a_1, \ldots)$. Машина M_k перебирает все возможные варианты $(k+1)$-го выступления и консультируется с M_{k+1} по поводу окончательных результатов судебного разбирательства. Ее оценка перспектив разбирательства составляется очень просто: если текущее выступление за

Прокурором, то достаточно найти хотя бы один вариант выступления, после которого M_{k+1} гарантирует убеждающую стратегию для Прокурора. Если текущее выступление за Адвокатом, то после любого из возможных выступлений M_{k+1} должна обнаружить убеждающую стратегию для Прокурора. Машина M_0 определяет наличие убеждающей стратегии для Прокурора в самом начале процесса и для её работы нужно задействовать всю последовательность машин M_k. Но каждая из этих машин использует небольшую (полиномиально ограниченную) память, так что весь процесс потребует лишь полиномиально ограниченной памяти.

А теперь покажем обратное. Пусть есть машина M, вычисляющая некоторую характеристическую функцию f на полиномиальной памяти. Заметим прежде всего, что вычисление на памяти S бессмысленно проводить дольше, чем время $2^{O(S)}$ (все начнет повторяться после того, как мы исчерпаем все состояния нашей системы, а их не более чем $|\mathcal{A}|^S \cdot |\mathcal{Q}| \cdot S$, где $\mathcal{Q}, \mathcal{A}$ — соответственно множество состояний управляющего устройства и алфавит рассматриваемой МТ). Поэтому можно считать без ограничения общности, что время работы машины M ограничено 2^q, где $q = O(p(|x|))$.

Для простоты описания потребуем, чтобы после завершения вычисления МТ сохраняла без изменений достигнутое состояние.

Правила судебной дискуссии состоят в следующем. Прокурор утверждает, что на входном слове x машина выдаёт результат 1, а Адвокат подвергает это сомнению. В первом своём выступлении Прокурор обязан описать состояние машины M (строка, записанная на ленте, положение читающей головки, состояние управляющего устройства) после 2^{q-1} тактов работы. В ответном слове Адвокат указывает на один из промежутков: от начала до (2^{q-1})-го такта или от (2^{q-1})-го такта до конца. После этого Прокурор обязан описать состояние M в середине этого промежутка. Далее всё повторяется: Адвокат выбирает одну из половинок, Прокурор описывает состояние M в середине выбранной половинки и т. д.

Дискуссия заканчивается, когда длина промежутка становится равной 1. Судья подводит итог так: в течение процесса для обоих концов этого промежутка были описаны состояния M, если состояние правого конца получается из состояния левого конца за один такт работы M, то Судья склоняется к мнению Прокурора, в противном случае — к мнению Адвоката. Необходимые проверки займут у Судьи время, полиномиально ограниченное длиной входного слова.

Пусть $f(x) = 1$. Тогда убеждающая стратегия для Прокурора состоит в том, чтобы каждый раз сообщать истинное положение дел (напомним, что Прокурор интеллектуально всемогущ, так что он в состоянии промоделировать в уме работу машины M).

Пусть $f(x) = 0$. Тогда, что бы ни говорил Прокурор, на одном из промежутков (или на обоих) будет содержаться ошибка. Адвокат должен указывать каждый раз именно такой промежуток — это гарантирует ему успех.

В классе PSPACE сушествуют полные относительно полиномиальной сводимости задачи. Простейший вариант получается из приведенного выше доказательства.

Задача TQBF (Truth of Quantified Boolean Formula). Задаётся предикатом

$TQBF(x) \Leftrightarrow$ x есть истинная квантифицированная булева формула (QBF), т.е. формула вида

$$\mathsf{Q}_1 y_1 \ldots \mathsf{Q}_n y_n \varphi(y_1, \ldots, y_n),$$

где $y_i \in \{0, 1\}$, φ — некоторая логическая формула, а Q_i — либо $\forall$, либо $\exists$.

По определению кванторы действуют на формулы следующим естественным образом:

$$\forall x \psi(x_1, \ldots, x) \stackrel{\text{def}}{=} \psi(x_1, \ldots, 0) \wedge \psi(x_1, \ldots, 1), \tag{14}$$

$$\exists x \psi(x_1, \ldots, x) \stackrel{\text{def}}{=} \psi(x_1, \ldots, 0)) \vee \psi(x_1, \ldots, 1)). \tag{15}$$

Теорема 6. *Задача* TQBF PSPACE-*полна.*

Итог описанной выше дискуссии можно выразить в виде квантифицированной формулы, если записать условие «одно состояние МТ получается из другого за один такт» в виде логической формулы. Хотя в этой дискуссии выступления были длиннее чем в один бит, их нетрудно превратить в однобитовые, вставляя после каждого бита настоящего выступления одной из сторон фантомное выступление противоположной стороны («покашливание»). Формула φ от «покашливаний» не зависит.

8. Класс IP

Вернёмся к паре (Исполнитель, Советник). Работа этой пары позволяет решать задачи из класса NP. А изменится ли что-нибудь, если выдать Исполнителю генератор случайности (монетку для подбрасывания)? Класс функций, вычислимых парой (Исполнитель с монеткой, Советник) за полиномиальное время называется IP(более точное определение будет дано ниже). Справедлива следующая замечательная теорема.

Теорема 7. PSPACE = IP.

Подбрасывание монетки значительно улучшает эффективность советов! В остальной части этого раздела мы попробуем объяснить это неожиданное явление.

8.1. Дополнительные возможности: подбрасывание монетки и класс BPP

Для начала нужно понять, что изменится в определении алгоритма, если разрешить использование генератора случайности. Результат работы Исполнителя, использующего подбрасывания монетки, перестает быть однозначно определённым. Как тогда понимать утверждение «алгоритм (вероятностный) решает задачу»?

Естественно полагать, что вероятностный алгоритм решает задачу, если велика вероятность правильного ответа. Конечно, эта вероятность зависит от свойств генератора случайности. Мы будем считать, что генератор случайности производит последовательность независимых случайных битов (0 или 1 с равными вероятностями). Простейшая реальная модель такого генератора — подбрасывание монетки.

Если известна верхняя оценка времени работы алгоритма, то его можно преобразовать к следующему стандартному виду: сделаем достаточное количество подбрасываний монетки и запомним их результаты, составив таблицу случайных битов. После этого начнем действовать по алгоритму, заменяя подбрасывание монетки обращением к таблице случайных чисел, составленной на первом шаге. Время работы такой версии алгоритма имеет ту же (с точностью до мультипликативного множителя) верхнюю оценку. А вероятность получения ответа a для алгоритма в таком формате подсчитывается просто: это отношение количества тех таблиц случайных чисел, при использовании которых получается данный ответ a, к общему количеству таблиц, равному 2^r, где r — количество случайных битов.

Это замечание позволяет дать определение класса BPP — задач, решаемых вероятностными алгоритмами за полиномиальное время, — аналогично определению 5 класса NP.

Определение 7. *Функция f принадлежит классу* BPP, *если существуют такие полином $q(\cdot)$ и функция $R(\cdot,\cdot) \in$ P, что доля слов r длины $q(|x|)$, для которых выполнено $f(x) = R(x,r)$, больше* $2/3$.

Если в этом определении заменить число $2/3$ на любое фиксированное число, бóльшее $1/2$, класс BPP не изменится. Есть простой способ добиться вероятности, сколь угодно близкой к 1. Возьмём несколько одинаковых машин, запустим их все, а окончательным результатом будем считать мнение большинства. Если вероятность правильного ответа для

каждого экземпляра машины равна $c > 1/2$, то вероятность неправильного ответа после голосования n машин

$$\sum_{k=0}^{\lfloor n/2\rfloor}\binom{n}{k}c^k(1-c)^{n-k} < \sum_{k=0}^{\lfloor n/2\rfloor}2^n\left(c(1-c)\right)^{n/2}\left(\frac{1-c}{c}\right)^{n/2-k} <$$

$$< \lambda^n\sum_{k=0}^{\lfloor n/2\rfloor}\left(\frac{1-c}{c}\right)^{n/2-k} < \lambda^n\left(\frac{1-c}{c}\right)^{n/2-\lfloor n/2\rfloor}\frac{1}{1-(1-c)/c} =$$

$$= O(\lambda^n) \quad (16)$$

для подходящего λ ($2\sqrt{c(1-c)} < \lambda < 1$). Таким образом, голосование экспоненциально быстро уменьшает вероятность неправильного ответа.

8.2. Определение класса IP

Класс IP составляют задачи, для которых существуют «интерактивные доказательства правильности ответа». Другими словами, есть интеллектуально всемогуший и пристрастный Советник, к услугам которого разрешено прибегать полиномиально ограниченному вероятностному алгоритму, а задача принадлежит классу IP, если есть такой способ организовать работу пары (Исполнитель с монеткой, Советник), что вероятность правильного результата достаточно высока (больше 2/3).

Как и в предыдуших случаях, мы хотим привести работу пары наших персонажей к некоторому каноническому виду. Заметим, что теперь, в отличие от класса NP, Советник не знает заранее списка вопросов, которые ему будут заданы: эти вопросы определяются, помимо прочего, подбрасыванием монетки. Однако, как и в случае класса BPP, Исполнитель может составлять таблицы случайных битов и пользоваться ими.

Будем считать, что Советник видит результаты подбрасывания монетки. Поэтому со стороны Исполнителя было бы неосмотрительно составлять таблицу случайных битов на все время диалога с Советником. Однако он ничего не теряет в возможности контролировать Советника, если такая таблица составляется перед каждым вопросом. А Советник, глядя на таблицу, уже может понять, какой именно вопрос задаст ему Исполнитель, сообщить Исполнителю этот вопрос и свой ответ на него.

Поэтому работу пары (Исполнитель с монеткой, Советник) можно преобразовать (с незначительной потерей ресурсов) к следующему каноническому виду: Исполнитель составляет таблицу случайных битов, показывает её Советнику, тот говорит нечто в ответ и т. д. Количество таких раундов ограничено заранее заданным полиномом от длины входного слова. После завершения диалога с Советником, Исполнитель по записи этого диалога принимает решение. Вероятность правильного ответа

(доля диалогов, приводящих к правильному ответу) должна быть достаточно велика. Все эти неформальные соображения учтены в следующем определении, в котором f — характеристическая функция.

Определение 8. $f \in \mathrm{IP}$ *означает, что есть такие полиномиально вычислимая (характеристическая) функция V (Исполнитель), некоторая функция P (Советник) и полиномы $q(\cdot)$ (длина таблицы случайных чисел), $s(\cdot)$ (длина диалога), что $|P(u)| = \mathrm{poly}(|u|)$ (Советник даёт полиномиально ограниченные советы) и для каждого x доля тех двойных последовательностей*

$$\begin{array}{cccc} r_{11} & r_{12} & \dots & r_{1q(|x|)} \\ r_{21} & r_{22} & \dots & r_{2q(|x|)} \\ \dots & \dots & \dots & \dots \\ r_{s(|x|)1} & r_{s(|x|)2} & \dots & r_{s(|x|)q(|x|)}, \end{array}$$

для которых

$$f(x) = V(x, r_1, P(r_1), r_2, P(r_1, r_2), \dots, r_{s(|x|)}, P(r_1, r_2, \dots, r_{s(|x|)})),$$

больше $2/3$.

Мы использовали обозначение $r_k = (r_{k1}, r_{k2}, \dots, r_{kq(|x|)})$ и естественное соглашение: когда мы применяем функцию к последовательности аргументов, предполагается, что эта последовательность представлена в виде, аналогичном (1) (см. начало статьи).

Замечание 6. Работа Исполнителя в описанном выше формате похожа на работу Судьи из уголовно-процессуальной интерпретации класса PSPACE. Только теперь одна из спорящих сторон — генератор случайных чисел. Теорема 7 получает любопытную интерпретацию: анализ игры против сколь угодно умного противника с вычислительной точки зрения эквивалентен анализу (другой) игры против простейшего противника: генератора случайных чисел[4)].

8.3. IP $\subseteq$ PSPACE

Определение IP похоже на уголовно-процессуальное определение класса PSPACE. И доказательство IP $\subseteq$ PSPACE напоминает ту часть доказательства теоремы 5, в которой строится алгоритм определения выигрышной стратегии, использующий полиномиально ограниченную память.

Пусть $f(x) \in \mathrm{IP}$. Снова по индукции определяется набор машин Тьюринга M_k для $k = 0, \dots, s(|x|)$.

Каждая M_k по заданному началу диалога $x, r_1, p_1, \dots, r_k, p_k$ длины k между Исполнителем и Советником оценивает вероятность ответа 1 при

[4)] Автор благодарен А. Китаеву, обратившему его внимание на эту, известную специалистам по теории сложности, интерпретацию теоремы 7.

продолжении диалога. Последняя в этом ряду машина $M_{s(|x|)}$ имитирует работу Исполнителя, вычисляет функцию $V(x, r_1, p_1 \dots)$ и сообщает в качестве ответа её значение. Машина M_k перебирает все возможные варианты $(k+1)$-го сообщения и консультируется с M_{k+1} по поводу окончательных результатов. Если слово за Советником, то оценка M_k — максимум оценок M_{k+1} по всем возможным вариантам сообщения Советника. Если слово за Исполнителем, то оценка M_k равна среднему арифметическому оценок M_{k+1} по всем r_k (это формула полной вероятности).

Алгоритм вычисления f на полиномиальной памяти запускает M_0 и выдаёт в качестве результата 1, если оценка M_0 больше $2/3$, и 0, если оценка M_0 меньше $2/3$.

8.4. PSPACE $\subseteq$ IP

Дадим набросок доказательства, подробно это доказательство изложено в книге Сипсера [9].

Достаточно доказать, что PSPACE-полная задача TQBF принадлежит IP.

Итак, пусть нужно проверить истинность высказывания

$$T_0 = \mathsf{Q}_1 x_1 \mathsf{Q}_2 x_2 \dots \mathsf{Q}_m x_m \varphi(x_1, \dots, x_m),$$

в котором $\varphi(\cdot)$ — некоторая логическая формула, $\mathsf{Q}_i \in \{\forall, \exists\}$, кванторы действуют на формулы по правилам (14)–(15).

Оценить истинность высказывания T_0 можно следующей рекурсивной процедурой. Обозначим через $T_j(x_1, x_2, \dots, x_j)$ высказывание

$$\mathsf{Q}_{j+1} x_{j+1} \mathsf{Q}_{j+2} x_{j+2} \dots \mathsf{Q}_m x_m \varphi(x_1, \dots, x_j, x_{j+1}, \dots, x_m).$$

Тогда $T_0 = T_1(0) *_1 T_1(1)$, где $*_1$ — связка, определяемая квантором Q_1. Найдем вначале $T_1(0)$, а затем $T_1(1)$, каждый раз вызывая рекурсивно саму описываемую процедуру (для определения $T_m(x_1, \dots, x_m)$ достаточно вычислить значение формулы $\varphi(x_1, \dots, x_m)$, так что рекурсия конечная) и вычислим $T_1(0) *_1 T_1(1)$.

Эта процедура требует вычисления всех 2^m возможных значений формулы $\varphi(\cdot)$, а потому занимает экспоненциально большое по сравнению с размером входа задачи время. Наглядно это вычисление можно изобразить в виде дерева, как показано на рис. 5а. В процессе вычисления мы получаем не только значение в корне, но и значения во всех промежуточных узлах дерева, количество которых экспоненциально велико.

Чем может помочь Советник в таком вычислении? Он, конечно, может сообщить значение T_0. Впрочем, поскольку Советник пристрастен, то в любом случае он скажет, что $T_0 = 1$. Попробуем проверить это сообщение и запросим значения $T_1(0)$ и $T_1(1)$ — должно выполняться соотношение $T_0 = T_1(0) *_1 T_1(1)$. Далее выберем случайно и равновероятно число

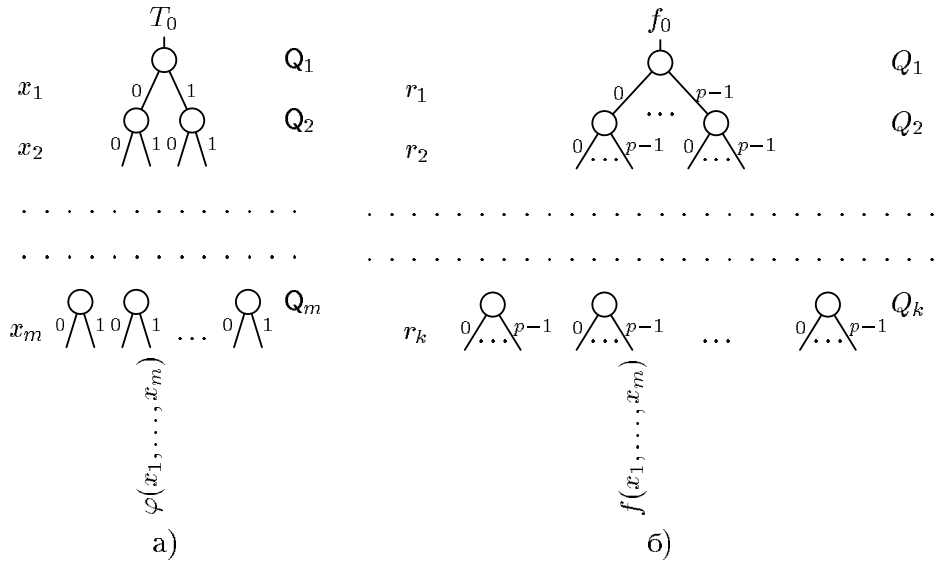


Рис. 5.

$r_1 \in \{0,1\}$, запросим у Советника значения $T_2(r_1,0)$ и $T_2(r_1,1)$, проверим $T_1(r_1) = T_2(r_1,0) *_2 T_2(r_1,1)$ и т. д. На последнем шаге уже без всякого Советника можно проверить, что

$$T_{m-1}(r_1,\ldots,r_{m-1}) = T_m(r_1,\ldots,r_{m-1},0) *_m T_m(r_1,\ldots,r_{m-1},1).$$

Вычисления будут длиться $O(m)+\Phi(n) = \mathrm{poly}(n)$ тактов, где $\Phi(n)$ — количество тактов, необходимых для вычисления значения логической формулы длины n при заданных значениях переменных. Они происходят вдоль одного из путей от корня к листьям в дереве на рис. 5а, выбираемого случайно и равновероятно.

С какой вероятностью Исполнитель принимает решение об истинности QBF? Если рассматриваемая формула истинна, то Советнику достаточно говорить истинные значения всех запрашиваемых величин, Исполнитель не увидит никакого противоречия и с вероятностью 1 признает формулу истинной. Что будет, если QBF ложна? Чтобы убедить Исполнителя в обратном, Советнику хотя бы раз придется солгать, сообщив неверное значение одной из пары запрашиваемых переменных. Эта ложь с вероятностью $1/2$ замечена не будет, что показывает непригодность описанной процедуры для надежного определения истинности QBF.

Как ни странно, есть способ исправить описанную выше процедуру. Если в исходном варианте из полного бинарного дерева перебора случайно выбирался один путь, и это ничего не дало, то в исправленной версии дерево станет шире (каждый раз будет выбор из p вариантов, где

$p = \Omega(n^4)$ — простое число) и глубже (глубина дерева была m, а в исправленной версии будет примерно m^2). Это дерево изображено на рис. 5б.

Чтобы построить такое разветвлённое дерево, будем считать истинностные значения 0 и 1 элементами поля $\mathbb{F}_p$ вычетов по модулю p. Любую логическую формулу $\varphi(x_1, x_2, \ldots, x_m)$ можно записать как многочлен $p(x_1, x_2, \ldots, x_m)$ над полем $\mathbb{F}_p$, используя выражения (8) на с. 97. Степень d этого многочлена не превосходит длины записи формулы $\varphi(x_1, x_2, \ldots, x_m)$, которая меньше n — размера входа задачи TQBF.

Кванторам сопоставим следующие операции с многочленами:

$$\forall \mapsto A_x f(x_1, \ldots, x) \stackrel{\text{def}}{=} f(x_1, \ldots, 0) f(x_1, \ldots, 1), \tag{17}$$

$$\exists \mapsto E_x f(x_1, \ldots, x) \stackrel{\text{def}}{=} 1 - (1 - f(x_1, \ldots, 0))(1 - f(x_1, \ldots, 1)). \tag{18}$$

Теперь число

$$p_0 = (Q_1)_{x_1} (Q_2)_{x_2} \ldots (Q_m)_{x_m} p(x_1, \ldots, x_m) \tag{19}$$

равняется истинностному значению формулы $\varphi(x_1, x_2, \ldots, x_m)$ (Q_i — операция, соответствующая квантору Q_i). Вычисление этого числа организовано в дерево, аналогичное изображенным на рис. 5. Степень ветвления у этого дерева равна p, а глубина — m.

Высказываниям $T_j(x_1, x_2, \ldots, x_j)$ при этом соответствуют многочлены

$$p_j(x_1, \ldots, x_j) = (Q_{j+1})_{x_{j+1}} (Q_{j+2})_{x_{j+2}} \ldots (Q_m)_{x_m} p(x_1, \ldots, x_j, x_{j+1}, \ldots, x_m).$$

Заметим, что операции A_x, E_x уменьшают число переменных на 1, но удваивают, вообще говоря, степень каждой из оставшихся переменных, так что степени p_j могут быть велики. Чтобы степени многочленов не росли чрезмерно при вычислениях, можно воспользоваться следующей операцией:

$$L_x f(x_1, \ldots, x) \stackrel{\text{def}}{=} x f(x_1, \ldots, 1) + (1 - x) f(x_1, \ldots, 0). \tag{20}$$

Многочлены f и $L_x f$ имеют одинаковое количество переменных, совпадают при $x \in \{0, 1\}$, а степень переменной x в $L_x f$ равна 1.

Итак, истинностное значение высказывания T_0 равно

$$p_0 = (Q_1)_{x_1} L_{x_1} (Q_2)_{x_2} L_{x_1} L_{x_2} \ldots (Q_m)_{x_m} L_{x_1} L_{x_2} \ldots L_{x_m} p(x_1, \ldots, x_m),$$

где Q_i определяется по входу задачи TQBF (так же, как в (19)). К многочлену $p(\cdot)$ применяются в заданном порядке $k = m + m(m+1)/2$ операций вида A_x, E_x или L_x. Пусть после применения первых $k - i$ операций получается многочлен $p_i(t_1, \ldots, t_j)$, $1 \leqslant i \leqslant k$. Поскольку операция L не меняет количества переменных в многочлене, число j не обязательно равно i.

Теперь опишем диалог между Исполнителем и Советником. Начинается он с того, что Советник сообщает значение p_0 (как и раньше, в самом начале он скажет, что $p_0 = 1$ независимо от его настоящего значения).

Затем Исполнитель спрашивает у Советника, чему равен многочлен $p_1(t)$. Получив коэффициенты этого многочлена, Исполнитель проверяет, что $p_0 = (S_1)_t p_1(t)$ (будем операцию, стоящую на i-м месте обозначать S_i). После этого Исполнитель случайно и равновероятно выбирает число $r_1 \in \mathbb{F}_p$ и применяет рекурсивно ту же процедуру для проверки равенства

$$p_1(r_1) = L_{x_1}(Q_2)_{x_2} L_{x_1} L_{x_2} \dots (Q_m)_{x_m} L_{x_1} L_{x_2} \dots L_{x_m} p(x_1, \dots, x_m).$$

Конечность рекурсии обеспечивается тем, что значение многочлена $p(x_1, \dots, x_m)$ Исполнитель вычисляет самостоятельно. Если все проверки дают положительный результат, Исполнитель выдаёт в качестве ответа 1, если хотя бы раз обнаружено неравенство — 0.

Теперь оценим вероятности ответа 1.

Если рассматриваемая QBF истинна, то Советнику опять достаточно говорить истинные значения всех запрашиваемых величин. А вот если QBF ложна, Советнику хотя бы раз придется солгать. Поскольку два многочлена степени $d = O(n)$ совпадают не более чем в d точках, то с вероятностью не меньшей $1 - d/p$ на следующем шаге будет проверяться ложное условие. По индукции получаем, что вероятность обнаружить ложь в описанном диалоге не меньше

$$\left(1 - \frac{d}{p}\right)^N,$$

где $N = O(n^2)$ — общее количество проверок. При $p = \Omega(n^4)$ эта вероятность будет близка к 1.

9. Заключение

Итак, помимо прямого способа доказывать вычислительную сложность задачи, есть косвенные — доказывать полноту задачи в подходящем сложностном классе. Мы привели два примера таких классов — NP и PSPACE. Если когда-нибудь в будущем удастся доказать, что включения $\mathrm{P} \subseteq \mathrm{NP}$ и/или $\mathrm{NP} \subseteq \mathrm{PSPACE}$ строгие, то, как следствие, для очень многих задач появятся доказательства их сложности.

Существует много других сложностных классов. Например, в уголовно-процессуальной модели вычисления можно ограничить число выступлений сторон некоторой абсолютной константой. Получится пелая *иерархия* сложностных классов, включающая на одном из нижних уровней класс NP (единственное выступление Прокурора). Популярна гипотеза, что все включения в этой иерархии строгие.

Упомянём также особую модель вычислений — квантовые вычисления (см. [4]). Рассказывать о них мы не будем, заметим лишь, что они

содержат BPP и содержатся в PSPACE. Но приведем задачу, по вычислительной силе эквивалентную квантовым вычислениям.

Пример 5 (см. [8]). Оценка квадратичной суммы. *Дана*: (0,1)-*матрица* A *размера* $n \times n$, *на диагонали которой стоят единицы* ($a_{ii} = 1$).

Спрашивается: *каков знак у суммы*

$$S = \sum_{Ax=0, x \in \{0,1\}^n} (-1)^{x^T B x} 4^{\|x\|} 3^{n-\|x\|},$$

если известно, что $|S| > 5^n$?

В приведённой формуле матрица B получается из матрицы A заменой всех элементов на главной диагонали и выше нулями, $\|x\|$ — количество единиц в записи x.

Слова «если известно» означают, что на входах, не удовлетворяющих условию $|S| > 5^n$, функция не определена, и алгоритм может работать на таких входах как угодно.

Этот пример замечательным образом иллюстрирует обсуждавшиеся нами идеи полноты и сведения. Если интересоваться только временем работы алгоритмов с точностью «до полинома», то вычислительные возможности квантовых компьютеров совпадают с возможностями вероятностных машин, работающих полиномиальное время и имеющих дополнительно доступ к устройству (оракулу), решающему задачу об оценке квадратичной суммы.

Список литературы

[1] *Кормен Т. Х., Лейсерсон Ч. Е., Райвест Р. Л.* Алгоритмы: построение и анализ / Пер. с англ. под ред. А. Шеня. М.: МЦНМО, 1999.

[2] *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.

[3] *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

[4] *Китаев А., Шень А., Вялый М.* Классические и квантовые вычисления. М.: МЦНМО, ЧеРо, 1999.

[5] *Манин Ю. И.* Вычислимое и невычислимое. М.: Советское радио, 1980.

[6] *Разборов А. А.* О сложности вычислений // Математическое Просвещение. Сер. 3, вып. 3. М.: МЦНМО:ЧеРо, 1999. С. 127–141.

[7] *Верещагин Н., Шень А.* Логические формулы и схемы // Математическое Просвещение. Сер. 3, вып. 4. М.: МЦНМО, 2000. С. 53–80.

[8] *Knill E., Laflamme R.* Quantum Computation and Quadratically Signed Weight Enumerators. `xxx.lanl.gov/quant-ph/9909094`

[9] *Sipser M.* Theory of computation. Boston, MA: PWS Publ. Co, 1997.

[10] *Smale S.* Problems for the next century // Math. Intelligencer, 1998. Vol. 20, no. 2. P. 7–15.