А. А. Разборов

Лекция 23 апреля 1998 года

Основы теории сложности вычислений

Эта лекция рассчитана на тех, кто не знаком с теорией сложности вычислений. Поэтому здесь будет рассказано только об основах этой теории и самых первых ее результатах. При этом мы постараемся передать основные идеи, которыми руководствуются исследователи в этой области науки.

Сюжетной канвой дальнейшего рассказа послужат истории, происходящие с некоторым персонажем —назовем его М (от слова «математик»). И начнем рассказ со следующей истории.

1 Предыстория

Однажды М сидел дома и пытался доказать некоторую (возможно важную, возможно нет) теорему T. Он пытался доказать эту теорему в течение недели, двух, месяца, ... , но ничего у него не получалось. В конце концов он не выдержал и задал вполне естественный вопрос:

А можно ли в принципе доказать эту теорему?

Вопрос был обращен неизвестно к кому, и, скорее всего, улетел бы в пространство, если бы мимо не проходил другой наш персонаж, которого мы обозначим буквой Л, от слова «логик».

Л услышал вопрос, зашел в комнату и объяснил, что такими вопросами математики начали интересоваться примерно с начала 20-го века. В общем виде этот вопрос входит в знаменитую «программу Гильберта», посвященную понятию математического доказательства.

Эта программа, в частности, включала три следующих пункта.

Формализация понятия доказательства. Перед тем как задавать вопросы о том, можно ли доказать то или иное утверждение, необходимо дать строгое математическое определение доказуемости.

Полнота. После формализации понятия доказательства нужно установить *полноту* построенной *формальной теории*. Это означает, что любое истинное утверждение T должно быть доказуемым в нашей формализации. В частности (в силу некоторых исторических причин), самого Гильберта преимущественно интересовал вопрос о доказуемости утверждения о непротиворечивости математики, формализованного подходящим образом.

Разрешимость. Следующей целью программы было построение такого вычислительного устройства, которое по внешнему виду теоремы T, записанной в некотором формальном языке, определяло бы, является ли эта теорема доказуемой (предполагалось, что в силу пункта 2 программы это будет эквивалентно истинности).

Хорошо известно, что первый пункт программы был успешно выполнен. В настоящее время под словом «теорема» большинство математиков (хотя не все это признают) понимает то, что можно доказать в теории множеств Цермело–Френкеля.

Хуже дело обстояло с последующими двумя пунктами. Первым ударом, потрясшим в свое время все математическое сообщество, был полученный в 30-е годы результат Курта Гёделя, утверждающий, что никакая достаточно сильная теория с задаваемым явным списком множеством аксиом не может быть полной. Более точно, если наша теория непротиворечива и в ней можно формализовать рассуждения о натуральных числах, то существует утверждение, которое нельзя ни доказать, ни опровергнуть, более того, в качестве примера можно взять интересовавшее Гильберта утверждение о непротиворечивости рассматриваемой теории.

Мы сегодня занимаемся вычислениями, поэтому для нас более интересна теорема Чёрча (1936). Он доказал, что не существует никакого алгоритма, который по утверждению автоматически проверял бы, является это утверждение доказуемым или нет.

Итак, наш математик М узнал от логика Л, что в полной общности ответить на вопрос о доказуемости математического утверждения невозможно.

Впрочем, к этому моменту у него уже начали появляться сомнения в том, что теорема T истинна, и он решил заняться поиском контрпримера. М пошел в компьютерную комнату и написал программу Π, проверяющую последовательно все слова до тех пор, пока она не найдет данные, на которых T не верна. Он запустил свою программу и стал ждать окончания ее работы.

Программа работала час, два, день, неделю, ... И снова у математика возник вопрос, на этот раз:

Остановится ли П когда-нибудь?

На этот раз математик уже знал, к кому обратиться с таким вопросом. Он нашел логика и спросил существует ли способ узнать, прекратит ли данная программа свое выполнение или же будет работать до бесконечности. И снова Л дал вполне квалифицированный ответ. По теореме, доказанной Тьюрингом в 30-е годы, не существует алгоритма, определяющего, остановится ли когда-нибудь данная программа или нет, т. е. проблема остановки неразрешима.

2 Отправная точка

После этого наш математик отправился домой. Как это иногда бывает, его ребенок, изучающий геометрию в седьмом или восьмом классе, попросил папу помочь сделать домашнее задание. В народе бытует мнение, что люди, занимающиеся математикой, умеют хорошо считать, решать квадратные уравнения и разные задачки из элементарной геометрии, что не всегда соответствует действительности.

Итак, наш математик начал решать задачу, предложенную своим чадом, и понял, что напрочь забыл геометрию, которую учил в школе. Он однако помнил, что любую задачу геометрии можно записать в координатах на языке действительных чисел.

И ему в голову пришел вопрос, а существует ли универсальный алгоритм, решающий школьные задачи по геометрии. После разговора с логиком, он знал, что если в теории выразимы натуральные числа, то она неразрешима. Интуиция подсказывала, что теория, работающая с действительными числами, которых гораздо больше, должна быть уж тем более неразрешима. Все же, для очистки совести, он решил позвонить логику и удостовериться в этом. Как ни странно, выяснилось противоположное. Классический результат Тарского, доказанный в 1948 году, утверждал существование алгоритма, проверяющего доказуемость утверждений элементарной геометрии.[1]

[1] И вообще любых утверждений о вещественных числах, использующих арифметические операции, элементарные логические связки (отрицание, конъюнкция, дизъюнкция, импликация) и кванторы по множеству всех вещественных чисел: «для всех (вещественных чисел)», «существует (вещественное число), такое что ... »; совокупность таких утверждений называется алгеброй Тарского.

Математик обрадовался. Ему уже начинало казаться, что от логики нет никакой пользы, а тут оказалось, что логика имеет весьма практичное применение в реальной жизни.

Итак, М попросил ребенка подождать, а сам отправился к торговцам программным обеспечением. Там он обнаружил два компакт-диска, посвященных решению задач по геометрии, которые назывались соответственно

Tarsky for Windows 95 и Collins for Windows 95.

Первый диск стоил 30 у.е., второй —300 у.е. Естественно, М попытался понять, чем же вызвана такая разница в цене. Никаких объяснений он не получил, поэтому он купил более дешевый диск Tarsky for Windows 95.

М пришел домой, вставил CD в компьютер и решил проверить программу на задачке из школьного учебника. Однако стала повторяться все та же история. Программа работала час, два, ... но не проявляла никакого желания выдать решение. Математик прервал выполнение программы и попросил ее доказать какую-нибудь элементарную теорему, например о том, что сумма углов в треугольнике равна 180 градусам. Результат не сильно изменился, и программа продолжала думать час, второй, сутки, другие, ... Тогда М позвонил логику и несколько повышенным тоном спросил, что же происходит. Л ответил, что это не его проблема. Тарский доказал теорему о существовании алгоритма, он совершенно уверен, что авторы этого диска запрограммировали алгоритм правильно, а что происходит дальше —не имеет никакого отношения ни к математике, ни к логике.

И это как раз то место, где начинается теория сложности вычислений. Нас интересует не просто существование алгоритмов для нашей задачи, а то, насколько они эффективны.

Конечно, рассказанная история несколько стилизована, но, в общем-то, так и происходило развитие исследований, которое привело к современному состоянию. В частности, тот этап, когда стало выкристаллизовываться понимание того, что одни алгоритмы могут быть лучше других и это существенно, пришелся на 60-е годы. Тогда стали появляться настоящие компьютеры (они назывались таким страшным словосочетанием: «электронные вычислительные машины»), например, БЭСМ (многие в аудитории и не знают, наверное, что это такое). Стало понятно, что необходимо построение некоторой математической теории.

3 Начала теории: основные понятия

Давайте пока оставим в стороне истории из жизни математика M, потом мы к нему еще вернемся не раз.

Попробуем дать несколько определений.

Первое наблюдение, которое сделал M, когда пытался разобраться в этой теории, состоит в том, что подавляющее большинство алгоритмических задач можно представить в подходящей кодировке как задачи вычисления некоторого отображения

$$f\colon \{0,1\}^* \to \{0,1\}^*$$

множества конечных двоичных слов в себя. Именно такими отображениями мы и будем заниматься.

Второй вопрос: на каких устройствах мы собираемся вычислять наше отображение? Есть огромное разнообразие и алгоритмических языков, и компьютерных архитектур, но насколько нам это важно? Именно ответ на этот вопрос отличает теорию сложности вычислений от различных смежных областей. Поэтому рассмотрим этот вопрос более подробно.

В качестве примера возьмем программу, с которой чаще всего и сталкивается большинство математиков, которые не программируют. Это TEX процессор. Выбор TEX здесь никакого особого значения не имеет, то же самое можно сказать и о других программах.

Итак, имеется некоторое отображение f, которое в данном случае представляет собой отображение

$$\boxed{\texttt{paper.tex}} \longrightarrow \boxed{\texttt{paper.dvi}},$$

преобразовывающее файл `paper.tex` в файл `paper.dvi` (оба файла мы рассматриваем как два длинных двоичных слова). Это преобразование осуществляется некоторым алгоритмом `texdvi.exe`, закодированным для выполнения на 286-м процессоре (те, кто пытался использовать TEX на 286-м процессоре, помнят, как это было).

Нас сейчас интересует время (быстродействие) работы алгоритма. Именно такую, как говорят в этой науке, *меру сложности* мы будем рассматривать сегодня чаще всего. Бывают и другие характеристики (память —вторая по значению мера сложности), но за недостатком времени мы их не сможем подробно обсудить.

Развитие техники, относящееся к рассматриваемой нами задаче, шло по двум направлениям. Во-первых, и это хорошо все знают, улучшались процессоры:

$$\text{Intel 286},\ \ \text{Intel 386},\ \ \text{Intel 486},\ \ \ldots$$

появлялись один за другим все более мощные модели, возникали разнообразные конструкторские ухищрения, ускоряющие их работу, и т. д. Во-вторых, улучшался сам алгоритм, появлялись его версии

`texdvi1.exe`, ... `texdvi10.exe`, ...

Справедлива, хотя и с некоторыми оговорками, такая формула

$$\left\langle \begin{matrix}\text{общее время}\\ \text{работы}\end{matrix} \right\rangle = \left\langle \begin{matrix}\text{число операций}\\ \text{алгоритма}\end{matrix} \right\rangle \times \left\langle \begin{matrix}\text{время на одну}\\ \text{операцию}\end{matrix} \right\rangle.$$

Улучшение обоих членов в этой простой формуле —вещи более-менее независимые. Грубо говоря, программы (software) отвечают за первые сомножитель, а процессоры (hardware) —за второй.

Что здесь происходит с математической точки зрения? Если у нас имеется какой-то алгоритм, который использует t операций, то от улучшения быстродействия нашего процессора время решения нашей задачи умножается на некоторую константу.

Поэтому, и это очень важно, в теории сложности вычислений сложилась традиция измерять время работы алгоритма с точностью до мультипликативной константы («с точностью до $O(\cdot)$», как говорят математики). Такой подход позволяет нам абстрагироваться от выбора конкретной модели вычислительного устройства, от того, сколько времени занимает выполнение одной операции, какую систему команд имеет используемая нами машина и т. п. Именно этот подход и позволяет нам построить довольно красивую математическую теорию.

Теория сложности вычислений несколько отличается от всеобъемлющей (по определению) теории марксизма-ленинизма (не все в аудитории видимо знают, что это такое, и это хорошо): если в формуле есть два члена, и за второй наша теория никакой ответственности не несет, то об этом заявляется явно. Улучшением процессоров занимаются другие люди, мы занимаемся улучшением алгоритмов с точностью до мультипликативной константы.

Продолжим построение строгой теории. Одним из непосредственных преимуществ введенного выше соглашения является то, что нас не очень заботит выбор точной модели. Как правило, от смены модели (сейчас я имею в виду уже абстрактные математические модели) улучшение или ухудшение происходят с точностью до мультипликативной константы, а мы договорились этого не замечать.

Стандартный способ определения вычислительной модели состоит в том, что для исполнения алгоритма у нас имеется некоторая машина с адресуемой памятью, ячейки которой индексируются натуральными числами; в каждой ячейке можно хранить натуральные числа; можно выполнять арифметические действия и т. п. Детали здесь мало интересны, потому что наше $O(\cdot)$-соглашение позволяет о них заботиться не слишком. Если же заменить «с точностью до $O(\cdot)$» на слегка более грубое «с точностью до полинома», то вообще все реалистические вычислительные устройства оказываются эквивалентными.

Итак, фиксируем некоторую вычислительную модель. Если есть машина M, которая вычисляет функцию f, и определены входные данные x —некоторое двоичное слово, тогда можно определить нашу основную функцию $T(M, x)$ —количество операций (тактов), которые требуются машине M для работы на входном слове x. Функция $T(M, x)$ называется *сигнализирующей функцией по времени.*

4 Теорема об ускорении

Давайте теперь изучать эту функцию $T(M, x)$. Первое, что приходит в голову: у нас имеется алгоритмическая задача f; давайте выберем самый хороший алгоритм для решения этой задачи и назовем сложностью задачи сложность этого самого хорошего алгоритма.

Оказывается, что такой интуитивно очевидный способ действий, к сожалению, неосуществим и на этот счет имеется *теорема Блюма об ускорении.* В вольной формулировке эта теорема утверждает, что как бы мы ни пытались определять понятие «наилучшая для данной задачи машина», у нас ничего не получится (по крайней мере для некоторых задач).

Теорема Блюма открывает серию теорем, лежащих в основе современной теории сложности вычислений. Все эти теоремы были доказаны в 70-е годы, например, теорема Блюма об ускорении —это 1971 год. Примерно в то же время появилась концепция NP-полноты, изложением которой и завершится наш вводный рассказ о теории сложности вычислений.

Для дальнейшего нам потребуется еще одно важное определение. Функция $T(M, x)$ устроена очень нерегулярно. Рассмотрим пример TEX процессора. Для подавляющего большинства файлов произойдет остановка в самом начале работы из-за несоответствия входному формату TEX, а на каких-то файлах это время возможно окажется бесконечным из-за зацикливания. В общем случае эту функцию изучать никакой возможности не представляется, она слишком рыхлая. Мы хотим извлечь из нее функцию натурального аргумента, т. е. получить функцию из натуральных чисел в натуральные числа, которая отражала бы поведение $T(M,x)$. Есть несколько способов подойти к этой задаче. Мы здесь ограничимся самым распространенным, который называется *сложность в наихудшем случае*. Она определяется следующей формулой

$$t_M(n) = \max_{|x|\leq n} T(M,x).$$

Из всех слов битовой длины, не превышающей n, выбираем то, на котором наша машина работает хуже всего (т. е. дольше всего). Время работы на таком слове и называем сложностью $t_M(n)$. За время $t_M(n)$ машина гарантированно закончит работу на любом входном слове длины, не превышающей n. Может, конечно, случиться, что на каких-то словах работа завершится раньше.

Мы приводим упрощенный вариант теоремы Блюма; на самом деле, вместо $\log t$ в ней можно взять любую «разумную» стремящуюся к бесконечности функцию.

Теорема 1 (Блюм, 1971). *Существует такая вычислимая*[2] *функция f, что любую машину M, вычисляющую f, можно ускорить следующим образом: существует другая машина M', также вычисляющая f и такая, что*

$$t_{M'}(n) \leq \log t_M(n)$$

для почти всех n.

Функция из теоремы Блюма —это некоторая экзотика (как следует из формулировки теоремы, время ее вычисления в наихудшем случае растет очень быстро для любой машины). Одна из «идеологических» задач в теории сложности вычислений —это строить теорию так, чтобы такие патологические явления в ней по возможности не возникали. Теорема Блюма доказывается построением с помощью техники диагонализации, и получающаяся функция не имеет никакого отношения к реальной практике вычислений или к остальной математике. Но тем не менее, поскольку мы строим математическую теорию, ничего не поделаешь —мы вынуждены признать, что выбранный подход не годится для ее построения. Нужно двигаться дальше.

[2]Вычислимая функция —это такая функция, которую можно вычислить хотя бы одним алгоритмом.

5 Сложностные классы

Итак, мы не можем надеяться на построение для каждой функции самой лучшей машины, вычисляющей эту функцию. Альтернативой является понятие *сложностного класса* —одно из центральных понятий для теории сложности вычислений.

Несколько вольная аналогия может быть здесь проведена с определениями интеграла по Риману и по Лебегу. Если мы не можем проинтегрировать по Риману, то мы меняем ось и начинаем суммировать по другой оси. В нашей ситуации, если мы не можем для функции сказать, что такое самая хорошая машина, то давайте также сменим ось. Рассмотрим множество всех машин, которые нас устраивают, и назовем класс всех функций, вычислимых такими машинами, *классом сложности*. Пожалуй, проще всего дать сразу конкретный пример, чем долго рассуждать на эту тему:

$$\mathrm{DTIME}(t(n)) := \\ = \{f \mid \exists\, M : (M \text{ вычисляет } f)\ \&\ (t_M(n) = O(t(n)))\}.$$

Это одно из центральных определений в теории сложности. Буква D означает детерминированные алгоритмы (бывают и другие), TIME означает ровно то, о чем вы подумали. Если у нас есть произвольная функция $t(n)$ от натурального аргумента, то мы образуем класс сложности, состоящий из таких функций, для которых существует вычисляющая f машина M, такая что сигнализирующая функция по времени ограничена исходной функцией $t(n)$ с точностью до мультипликативной константы. Приведенная выше теорема Блюма справедлива только для некоторых специальных функций. Но если мы хотим достичь ускорения, скажем, в 10 раз, то это можно сделать с любой функцией —например, путем увеличения количества машинных команд. Именно поэтому в правой части определения стоит O-большое.

Теперь определим один из самых важных сложностных классов

$$\mathrm{P} = \bigcup_{k \geq 0} \mathrm{DTIME}(n^k).$$

Класс P —это те функции, которые можно вычислить на наших машинах, и время их вычисления растет полиномиально с ростом длины слова. Он очень удобен и с практической, и с теоретической точки зрения. С практической точки зрения это достаточно хорошая аппроксимация (бывают исключения, о которых будет сказано ниже) класса тех функций, которые поддаются вычислению за реальное время на реальных компьютерах. С математической точки зрения этот класс бесконечно удобен тем, что он замкнут относительно суперпозиции. Как мы увидим дальше, именно это обстоятельство позволяет построить теорию вычислимости для этого класса.

Бывают аналогичные классы языков, которые можно распознать за экспоненциальное время

$$\mathrm{EXPTIME} = \bigcup_{k \geq 0} \mathrm{DTIME}\left(2^{n^k}\right),$$

также можно определить двойное экспоненциальное время

$$\mathrm{DOUBLEEXPTIME} = \bigcup_{k \geq 0} \mathrm{DTIME}\left(2^{2^{n^k}}\right)$$

и так далее.

6 Теорема об иерархии и сложность элементарной геометрии

Теперь давайте вернемся к нашему математику M. Оказывается, время работы алгоритма Тарского, с помощью которого M пытался решать задачи элементарной геометрии, не лежит ни в одном из приведенных выше классов. Для него есть такая нижняя оценка

$$t_{\mathsf{Tarski}}(n) \geq 2^{2^{\cdot^{\cdot^{2}}}} \Big\} \, \varepsilon n$$

Как вы помните, для разрешимости алгебры Тарского продавался еще и диск Collins for Windows 95.

Теорема (Коллинз). *Алгебра Тарского принадлежит классу* DOUBLEEXPTIME.

Теперь M смог понять разницу между этими CD-ROM: время работы алгоритма Коллинза неизмеримо меньше времени работы алгоритма Тарского (хотя оно все равно может быть очень велико —оценка времени работы алгоритма двойной экспонентой не гарантирует нам, что даже теорема о сумме углов треугольника будет доказана за время существования Вселенной).

Возникает вопрос: а нельзя ли еще улучшить алгоритм разрешимости для алгебры Тарского? И более общий вопрос: а нельзя ли то же самое сделать вообще с любым алгоритмом? Вдруг, например, всякая вычислимая функция лежит в классе P. Или, по крайней мере, всякая функция из DOUBLEEXPTIME лежит в P.

Другими словами, вопрос в том, есть ли у нас предмет исследования, или, возможно, теорема Блюма об ускорении применима вообще ко *всем* функциям.

Второй краеугольный камень теории сложности вычислений —это *теорема об иерархии.*

Как и в случае с теоремой об ускорении, мы приводим ее далеко не в самой общей форме.

Теорема (Хартманис (1965)). *P $\neq$ EXPTIME.*

Таким образом, не все сложностные классы совпадают, так что предмет исследования есть.

Я не могу отказать себе в удовольствии привести почти полное доказательство этой теоремы. Если вы еще помните нашего математика M, второй момент его злоключений состоял в том, что он спрашивал, остановится ли его программа *когда-нибудь.* Сейчас, наученный горьким опытом, он задал такой вопрос:

остановится ли эта программа до Нового Года?

Выясняется, что если до Нового Года осталось в точности экспоненциальное время, то эта задача и отделяет EXPTIME от P. Потому что есть очень простой алгоритм, позволяющий проверить, остановится ли программа до Нового Года, а именно, нужно просто подождать до него, и все само собой решится. Теорема об иерархии утверждает, что эту задачу нельзя решить существенно быстрее, чем описанным выше способом.

Я, конечно, слегка утрирую, есть там и кое-какие технические детали, но смысл состоит именно в этом.

И теперь наш математик оказался вполне подготовлен к восприятию следующей теоремы.

Теорема (Фишер–Рабин (1974)). *Алгебра Тарского не принадлежит к классу* P. *Время работы любого разрешающего алгоритма для алгебры Тарского не меньше, чем $2^{\varepsilon n}$, где ε —некоторая константа.*

Такая высокая *нижняя оценка* объясняет причины неудачи нашего математика с практическим решением задач в алгебре Тарского.

Наиболее сложная и, по-видимому, наиболее важная область теории вычислений, связана как раз с доказательством нижних оценок. В англоязычной литературе тот раздел теории сложности, который занимается конструированием алгоритмов, так и называется —"Theory of Algorithms", а собственно под "Complexity Theory" подразумевается как раз построение нижних оценок. Таким образом, теория сложности пытается доказать, что не существует эффективных алгоритмов.

Уже на этом примере видны две трудности, стоящие перед людьми, которые пытаются доказывать нижние оценки. Смотрите, была теорема Тарского, ее пытались улучшить, но ничего не получалось Естественно было предположить, что эта оценка оптимальна. После этого придумали алгоритм, основанный на совсем других идеях. Алгоритмов бывает много, алгоритмы бывают сложные, алгоритмы бывают разные. А ведь мы пытаемся доказать, что более эффективный алгоритм не появится никогда, и при этом пытаемся анализировать свойства алгоритмов из достаточно широкого класса.

7 Сводимость и полнота

Давайте продвинемся дальше в построении нашей теории. Понятие сложностного класса было введено вовсе не для того, чтобы экономить на формулировках теорем (в конце концов, теорему Фишера–Рабина можно сформулировать и без упоминания каких бы то ни было сложностных классов —для этого достаточно оставить в ее формулировке лишь вторую фразу). Понятие сложностного класса становится важным в тот момент, когда у нас появляется понятие *сводимости*, и это второе центральное понятие современной теории сложности вычислений. Есть несколько вариантов определения сводимости, я расскажу лишь о наиболее важном из них.

Сводимость по Карпу. Эта сводимость устроена очень просто. Для начала заметим, что мы занимаемся вычислением функций, отображающих конечные слова в конечные слова. Но во многих случаях оказывается гораздо удобнее, и это, как правило, не приводит к потере общности, рассматривать так называемые *языки*. Язык L можно рассматривать как множество слов $L \subseteq \{0,1\}^*$ или интерпретировать его как отображение вида $\varphi\colon \{0,1\}^* \to \{0,1\}$, тогда $L = \varphi^{-1}(1)$. Переход к языкам не слишком нас ограничивает: если есть произвольная функция из слов в слова, то с ней можно связать такой язык —множество пар (x, i) таких, что i-й бит $f(x)$ равен 1.

Определение. Язык L_1 сводится к языку L_2 по Карпу (обозначается это таким образом $L_1 \preceq_p L_2$), если существует такая функция f из P, что

$$\forall\, x : (x \in L_1 \equiv f(x) \in L_2)\,.$$

Сводимость можно рассматривать как использование некоторой подпрограммы, перерабатывающей исходное слово x в $f(x)$, к которому затем нужно применить алгоритм распознавания принадлежности языку L_2. В этом варианте понятия сводимости обращение к программе распознавания L_2 осуществляется ровно один раз, если же разрешено использовать в качестве подпрограммы распознавание языка L_2 сколько угодно раз, то получается другая сводимость (по Тьюрингу). Сейчас это различие нам неважно.

Отношение сводимости —это отношение предпорядка. Оно рефлексивно и транзитивно. Самое главное свойство состоит в том, что если $L_2 \in$ P, то и $L_1 \in$ P. Здесь важно, что класс полиномов замкнут относительно операции суперпозиции.

Например, EXP-сводимость не получится. Не будет замкнутости относительно такой сводимости. Замкнется все лишь на конечных башнях экспонент —классе *элементарных рекурсивных функций*.

Если говорить только о естественных классах, в которых реально есть хотя бы одна разумная функция и которые замкнуты относительно суперпозиции, то есть еще класс квазиполиномов $2^{(\log n)^{O(1)}}$ и класс квазилинейных функций $n \log^{O(1)} n$. Квазилинейными функциями сейчас начинают интенсивно интересоваться, потому что считается, что по-настоящему хороший алгоритм —это алгоритм, который работает как раз квазилинейное время.

В любом случае, класс P является центральным в теории, и мы сейчас будем рассматривать именно полиномиальную сводимость.

Легко видеть, что класс EXPTIME замкнут относительно этой сводимости. Поэтому естественно поставить вопрос, а существуют ли в этом классе самые сложные языки, т. е. такие, что любой другой язык из этого класса к ним сводится. В этом случае можно решить любую задачу из EXPTIME, используя произвольный алгоритм распознавания для такого самого сложного языка и полиномиальную сводимость. Языки из некоторого сложностного класса, к которым сводится любой язык из этого класса, называются *полными* (относительно данного класса и данного типа сводимости). А если мы опустим требование принадлежности нашему классу самого языка, получим определение *трудного* языка.

Теорема Фишера–Рабина доказывается именно так. Вместо того, чтобы доказывать напрямую, что не существует полиномиального алгоритма для алгебры Тарского, доказывается, что алгебра Тарского трудна для класса EXPTIME. Поэтому полиномиальный алгоритм для алгебры Тарского давал бы полиномиальный алгоритм и для всех остальных задач из этого класса. А мы знаем, что в EXPTIME есть задачи, которые нельзя решить за полиномиальное время («остановка программы до Нового Года»).

Такой способ рассуждений типичен для теории сложности. Мы не пытаемся действовать «в лоб», а сводим одну задачу к другой. И, естественно, ситуация наиболее хороша, когда к данной задаче сводится много других задач из данного класса.

Успех этой науки определяется тем практическим обстоятельством, что полных задач оказывается много в самых разных ситуациях и они более естественны, чем проблема остановки.

8 Всякий ли полиномиальный алгоритм хорош?

Теперь самое время вернуться к нашему бедному математику M и поговорить об исключениях из правила «класс P = класс эффективно вычислимых функций». Как-то M понадобилось для каких-то своих целей решать системы линейных неравенств:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1, \ldots, m).$$

Другими словами, ему потребовался пакет программ для задач линейного программирования. Специалисты по теории сложности вычислений перед покупкой программного обеспечения изучают литературу, и M, наученный предыдущим горьким опытом, стал следовать этому правилу. Он, конечно же, слышал про симплекс-метод, которым пользуются для решения задачи линейного программирования почти всюду, включая военную кафедру Московского университета. Но M обнаружил статью, в которой доказывается, что симплекс-метод не полиномиален. А после этого наш M обнаружил и статью Хачияна (1979), в которой был построен полиномиальный алгоритм для решения задачи линейного программирования. Поэтому, приехав на Митинский радиорынок, он попытался найти что-то вроде CD Khachiyan for Windows 95. К его удивлению, ничего похожего не было. Всё, что ему предлагали, было основано на симплекс-методе и его вариациях. Оказывается, что хотя теоретически алгоритм решения задачи линейного программирования симплекс-методом экспоненциальный, а алгоритм Хачияна полиномиальный, на практике первый работает быстрее второго. Чтобы построить пример задачи линейного программирования, на которой симплекс-метод будет работать долго, нужно очень и очень постараться, а полиномиальный алгоритм Хачияна работает примерно одинаково на любом входе (показатель экспоненты 6, что весьма плохо).

Это самое известное исключение из того правила, что полиномиальные алгоритмы хороши, а экспоненциальные —плохи. Но это исключение, на самом деле, подтверждает правило, потому что, хотя алгоритмом Хачияна никто и не пользуется для решения задач линейного программирования, выяснилось, что с помощью этого алгоритма можно решать такие задачи, к которым с симплекс-методом подступиться в принципе невозможно. Например, пусть у нас есть произвольное выпуклое тело K и задано некоторое направление. Нужно максимизировать значение соответствующей линейной формы на K. Про выпуклое тело вы ничего не знаете: оно задано с помощью черного ящика (или, как иногда говорят, *оракула*). То есть, если вы указываете точку p, то с некоторой погрешностью ε можно сказать, лежит ли точка p в теле K, а если не лежит —получить некоторую отделяющую p от K гиперплоскость. Ответ нужно получить с той же точностью. Никакой симплекс-метод здесь, понятное дело, не работает —вообще нет никаких вершин, перебором которых занимается симплекс-метод. А алгоритм Хачияна и та наука, которая из него выросла, замечательно (т. е. полиномиально) с такими задачами справляется. Роль параметра n, описывающего размер входа, здесь играет $d\cdot(\log\varepsilon^{-1})$, где d —размерность, а ε —точность вычисления.

Таких исключений известно очень мало. Второй известный пример —проверка простоты числа. Как правило, подтверждается тезис, что если у вас имеется алгоритм, который теоретически работает хорошо, и это —алгоритм для нормальной задачи, которая и в самом деле откуда-то возникла, а не сконструирована с хулиганскими намерениями, то этот алгоритм будет хорошо работать и на практике. В частности, показатель k в оценке скорости работы алгоритма n^k обычно оказывается небольшим, а когда он поначалу велик, его можно уменьшить различными ухищрениями. Для подавляющего большинства естественных задач показатель не превосходит 3.

9 Недетерминированные вычисления

По-видимому, те, кто пришел на эту лекцию, хотели узнать что-нибудь про самую известную открытую проблему в этой области $\mathrm{P} \overset{?}{\neq} \mathrm{NP}$. Что такое P, я уже немножко объяснил. Теперь давайте займемся NP.

Для этого опять вернемся к М. Пока он разбирался в теории сложности, его сын поступил в университет на первый курс и стал изучать математическую логику. Изучение математической логики, как известно, начинается с такой знаменитой науки, как исчисление высказываний. У вас имеется некоторая пропозициональная формула $\Phi(p_1, \ldots, p_n)$, она называется тавтологией, если она истинна независимо от того, что мы в нее подставим. И одно из неприятных упражнений при занятиях этой наукой состоит в том, что нужно выяснить по выписанной пропозициональной формуле, является она тавтологией или нет. Именно с таким вопросом и обратился сын нашего математика к своему папе. М, естественно, уже никуда не поехал, а стал, как и поступают всегда специалисты по теории сложности, пытаться поместить задачу в один из уже известных сложностных классов. Чтобы использовать стандартные обозначения, будем говорить о двойственной ей задаче SAT — выполнимости[3] : есть ли хотя бы один набор переменных, при котором формула истинна.

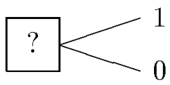
Накладывая эту задачу на нашу картину сложностных классов, М увидел первым делом, что $SAT \in \mathrm{EXPTIME}$. Алгоритм решения задачи SAT за экспоненциальное время очевиден —всего возможных наборов значений переменных 2^n, а время вычисления значения формулы при заданных значениях переменных полиномиально. Следующий шаг — классифицировать эту задачу: лежит ли она в P, или полна в EXPTIME? Этот вопрос, возникший в начале 70-х годов, до сих пор открыт. Почему у специалистов, потративших на решение этой задачи почти 30 лет, не получается построение полиномиального алгоритма, объяснить сложно (я, во всяком случае, никак это объяснять не берусь). Но тот факт, что не получается доказательство полноты этой задачи в EXPTIME, некоторому объяснению поддается. Если мы посмотрим на настоящие экспоненциальные алгоритмы, например, на алгоритм для аналога алгебры Тарского над комплексными числами, и сравним с ними то детское рассуждение, которое мы привели выше, то сразу невооруженным глазом видна разница —приведенный алгоритм для SAT использует экспоненциальное время очень слабо —только для *перебора* экспонен-

[3] SAT —от слова satisfiability —выполнимость.

циального числа возможностей, с каждой из которых он справляется за полиномиальное время.

Взрыв в теории сложности вычислений начался с того, что было сформулировано определение класса NP, как класса языков, которые распознаются переборными алгоритмами. Более научно, переборные алгоритмы называют недетерминированными алгоритмами. NP —это класс языков, которые можно распознать за недетерминированное полиномиальное время.

Теперь дадим определение этого класса. В этом определении, как и в определении класса P, будет фигурировать слово «машина», сокращенно мы будем обозначать ее НМТ («недетерминированная машина Тьюринга»); впрочем, использование слова «машина» в этом контексте может вызывать у людей, имеющих отношение к реальным машинам, некоторое чувство неудовольствия. Под недетерминированной машиной мы понимаем такую машину, которая работает как обычно, но в какой-то момент она может нарисовать в какой-то ячейке знак вопроса и после этого ее работа разделяется на две ветви 0 или 1:



(в клетке окажется записанным либо 0, либо 1). После этого машина продолжает работу. В какой-то момент она может раздвоиться еще раз. Появляется дерево вычислений. Вдоль каждой ветви дерева вычислений НМТ работает как обычное вычислительное устройство, но результат работы НМТ зависит от результатов работы вдоль всех ветвей. Определим результат работы НМТ в случае проверки принадлежности слова x к языку L. На каждой из ветвей вычисления получается один из двух возможных ответов: «да» или «нет». НМТ распознаёт язык L, если всякое слово x принадлежит L тогда и только тогда, когда *хотя бы на одной* ветке вычислений получен ответ «да».

Проиллюстрируем возможности НМТ на примере с упражнениями на проверку выполнимости пропозициональных формул. Частенько, когда студент (полиномиальная детерминированная машина Тьюринга) проваливается и не может решить задачу, преподаватель выступает в роли недетерминированной машины —он показывает (загодя приготовленный) ответ, который легко может быть проверен. Другими словами можно сказать так: НМТ «стремится» доказать утверждение $x \in L$, а в момент раздвоения она обладает неограниченными интеллектуальными возможностями и выбирает наилучший вариант. Если существует ветка вычислений, при которых ответ «да», то $x \in L$. В противном случае никакое ветвление не приведет к положительному результату (слово не принадлежит языку лишь тогда, когда у НМТ нет никакой возможности доказать обратное).

Не спрашивайте меня о физике процесса: как происходит такое раздвоение, где находится такая машина, можно ли ее посмотреть... Таких машин в реальности нет. Одним из самых значительных достижений в теории сложности за последние годы стало то, что была сформулирована квантовая модель вычислений. Квантовые компьютеры являются одним из кандидатов на роль такой машины в реальном мире. По крайней мере, существование квантовых компьютеров не вызывает принципиальных возражений у физиков, а надежда на имитацию недетерминированных машин квантовыми компьютерами пока не вызывает категорических возражений со стороны специалистов в теории сложности. Более того, абстрактные квантовые компьютеры уже умеют решать некоторые важнейшие переборные задачи, такие, как факторизация чисел[4] .

Подчеркнем еще раз, что недетерминированная машина является чисто теоретическим понятием. Удобства от введения такого понятия получаются совершенно фантастические.

Это самая первая вымышленная модель, появившаяся в теории сложности вычислений. В настоящий момент есть масса таких моделей, гораздо более сложных: интерактивные доказательства и т. п. Эти модели появляются не сами по себе, а с целью определения сложностных классов и для классификации естественных задач.

Интуитивно ясно, что понятие НМТ прекрасно приспособлено к моделированию переборных алгоритмов. Собственно, уже показано, как это делать. Если у нас есть алгоритм, который пытается перебрать некоторое количество возможностей, то наша машина может угадать, какая из возможностей является хорошей, а потом имитировать вторую часть (полиномиальную проверку конкретного варианта). В обратную сторону это несколько сложнее, но все равно достаточно просто. Если у нас есть НМТ, то она порождает дерево вычисления, и перебор нужно производить по всем возможным ветвям.

Современная теория сложности вычислений началась с результатов Кука, Карпа, Левина (который получил их независимо) в начале 70-х годов (1970–1972). Эта серия теорем состоит вот в чем.

---

[4]На последнем Международном Конгрессе (Берлин, август 1998г.) американский математик П. Шор был удостоен за эти исследования премии им. Неванлинны.

1. Задача *выполнимость* (SAT) полна для класса NP (теорема Кука). Таким образом, задача нашего математика M (можно ли придумать полиномиальный алгоритм для SAT) эквивалентна вопросу: совпадают ли классы P и NP ($\mathrm{P} \overset{?}{\neq} \mathrm{NP}$). Если полиномиального алгоритма нет, то эти классы не совпадают —их отделяет задача SAT, если же он есть, то можно эффективно решить любую задачу из класса NP. Задача выполнимости отвечает за класс NP. Грубо говоря, NP —это не что иное, как задачи, которые могут быть сведены к выполнимости пропозициональных формул.

2. Естественно, что такой результат может вызвать некоторую реакцию отторжения —не настолько уж важна задача выполнимости, чтобы строить целую теорию ее решения. Но следующим шагом была статья Карпа (1971), в которой уже была указана 21 полная задача для класса NP. Все они между собой эквивалентны.

В настоящее время список NP-полных задач, которые возникают буквально во всех областях математики, содержит тысячи задач. Везде, где возникают алгоритмы, возникают и переборные задачи. Это неудивительно, потому что бóльшая часть программистской работы и состоит в выборе варианта получше. Удивительно то, что часто, а практически —всегда (с некоторыми исключениями) —если у вас есть конкретная переборная задача, то ее можно сравнительно легко классифицировать —либо для нее есть полиномиальный алгоритм, либо она полна.

Таким образом строится теория переборных задач. Из-за того, что переборные алгоритмы возникают почти всюду, она приобрела большое значение. По мнению американского тополога Смейла, вопрос $\mathrm{P} \overset{?}{\neq} \mathrm{NP}$ будет одним из самых важных вопросов математической науки следующего столетия.

Вот и подошел к концу рассказ о том, что является, так сказать, ядром теории сложности вычислений. Не стоит понимать этот рассказ так, что вся теория сложности занимается только лишь соотношением $\mathrm{P} \overset{?}{\neq} \mathrm{NP}$. Изложенная схема исследований —которая объединяет задачи в сложностные классы и потом исследует эти задачи с помощью их сводимости друг к другу —оказалась удивительно эффективна плодотворна в самых разных ситуациях.

Перечислим коротко несколько самых важных возможностей (помимо уже упомянутых выше квантовых вычислений).

Бывает сложность алгебраическая. Если нас интересует вычисление некоторого полинома, и мы абстрагируемся не только от деталей битового вычисления, но и от того, как производятся арифметические операции, то все зависит лишь от того, с какими числами производятся арифметические операции. Тогда длина вычисления определяется количеством сложений и умножений.

Бывает сложность геометрическая —диаграммы Вороного и близкие к ним вещи, о которых я говорить не буду.

Бывает сложность булева —она отличается от того, чем мы занимались сегодня, тем, что нас интересуют функции $f\colon \{0,1\}^n \to \{0,1\}$, определенные на словах фиксированной длины.

В любом из этих разделов возникает масса постановок интереснейших задач именно в рамках этой общей идеологии.

Благодарности

Я выражаю глубокую благодарность М. Н. Вялому и М. В. Алехновичу за квалифицированную помощь в приведении данной лекции к пригодному для чтения виду и проф. J. Krajíček за ряд ценных замечаний.