Теория сложности вычислений, CS-клуб, 2021 (ЧЕРНОВИК краткого конспекта*лекций)

Д.М. Ицыксон †

3 декабря 2021 г.

Содержание

*Этот текст содержит массу опечаток, ошибок и неточностей. Просьба сообщать о найденных недостатках по электронной почте, в теме письма напишите слово КОНСПЕКТ. Все найденные недостатки будут исправляться в новых версиях конспекта.

†ПОМИ РАН E-mail: dmitrits@pdmi.ras.ru.

1 Универсальные машины Тьюринга

Машина Тьюринга является математическим определением понятия алгоритм. Такое определение нужно, чтобы можно было математически строго рассуждать об алгоритмах, о их существовании и сложности. До 30-х годов 20-го века не существовало математического определения понятия алгоритм. Конечно, в качестве определения алгоритма можно взять программы на любом современном языке программирования, в которых есть возможность доступа к неограниченной памяти. Но такое определение было бы очень сложно, хочется иметь простое определение, с которым было бы легко оперировать.

Существует огромное количество моделей вычислений (определений алгоритма): λ-исчисления, машины Поста, машины Тюринга, нормальные алгоритмы Маркова, PAM-машины и пр. Все они задают одно и то же понятие, понятие вычислимости. Мы изучаем машины Тьюринга, поскольку они чаще всего используются в теории сложности вычислений.

1.1 Одноленточная машина Тьюринга

Пусть Σ некоторый алфавит (конечное множество символов), мы считаем, что $\triangleright, \_ \in \Sigma$.

Машина Тьюринга состоит из бесконечной в одну сторону ленты, разделенной на ячейки. В самой левой ячейке написан символ $\triangleright$. После этого символа на ленте написано входное слово, после которого следует бесконечное числом символов $\_$. В каждый момент времени машина Тьюринга находится в каком-то состоянии из конечного множества Q. В множестве Q выделены два специальных состояния $q_0 \in Q$ — начальное состояние и $q_f \in Q$ — конечное состояние.

Головка машины указывает на одну из ячеек ленты. Изначально машина Тьюринга находится в состоянии q_0, а головка указывает на первый символ входного слова. Работа машины Тьюринга определяется правилами перехода, которые задаются отображением $\delta : \Sigma \times Q \to \Sigma \times Q \times \{\rightarrow, \leftarrow, \bullet\}$.

Если машина Тьюринга находится в состоянии q и головка указывает на символ c, а $\delta(q, c) = (q', c', \rightarrow)$, то один шаг машины Тьюринга состоит в том, чтобы заменить символ под головкой на c', перейти в состояние q' и сдвинуть головку вправо (символы $\rightarrow, \leftarrow, \bullet$ говорят о том, сдвинуть ли головку вправо, влево или оставить на месте).

Машина Тьюринга выполняет шаги один за одним, пока не попадет в финальное состояние q_f. В этот момент машина Тьюринга останавливается и результатом ее работы будет являться содержимое ленты. Если же машина Тьюринга не попадает в финальное состояние, то в этом случае мы считаем, что она не останавливается.

Сейчас мы описали машину Тьюринга, которая вычисляет функцию (возможно, что не всюду определенную, если машина не остановится). Очень часто нам будет важно, что машина Тьюринга проверяет принадлежность входного слова какому-либо языку, то есть ее ответ либо “да”, либо “нет”. В таком случае удобно разрешить иметь два финальных состояния q_{no} для отвержения и q_{yes} для принятия.

Пример 1.1. Приведем пример проверки на машине Тьюринга, является ли бинарное слово палиндромом (слово является палиндромом, если оно одинаково читается слева на право и справа налево).

- $(q_0, \_) \mapsto (q_{yes}, \_, \bullet)$
- $(q_0, \frac{0}{1}) \mapsto (\frac{q_2}{q_3}, \triangleright, \rightarrow)$
- $(q_2, \frac{0}{1}) \mapsto (q_2, \frac{0}{1}, \rightarrow)$
- $(q_3, \frac{0}{1}) \mapsto (q_2, \frac{0}{1}, \rightarrow)$
- $(\frac{q_2}{q_3}, \_) \mapsto (\frac{q_4}{q_5}, \_, \leftarrow)$
- $(q_4, 1) \mapsto (q_{no}, 1, \bullet)$
- $(q_5, 0) \mapsto (q_{no}, 0, \bullet)$
- $(q_4, 0) \mapsto (q_7, \_, \leftarrow)$
- $(q_5, 1) \mapsto (q_7, \_, \leftarrow)$
- $(q_7, \frac{0}{1}) \mapsto (q_7, \frac{0}{1}, \leftarrow)$
- $(q_7, \triangleright) \mapsto (q_0, \triangleright, \rightarrow)$

Сложностью машины Тьюринга по времени на данном входе называется число шагов, за которое машина приходит в финальное состояние. Временной сложностью машины Тьюринга называется функция $T : \mathbb{N} \to \mathbb{N}$, которая по n выдает максимум сложности по времени для входов длины n.

Сложностью машины Тьюринга по памяти на данном входе называется номер максимальной ячейки, в которой машина Тьюринга успела побывать до того, как машина пришла в финальное состояние. Емкостной сложностью (или сложностью по памяти, или зоной) машины Тьюринга называется функция $S : \mathbb{N} \to \mathbb{N}$, которая по n выдает максимум сложности по памяти для входов длины n.

В примере с палиндромом сложность по времени $O(n^2)$, а сложность по памяти $O(n)$. Можно ли проверить, является ли слово палиндромом быстрее, чем за $\Omega(n^2)$? Оказывается, что на одноленточной машине нельзя, позже мы докажем это.

1.2 Многоленточные машины Тьюринга

k-ленточная машина Тьюринга содержит k лент, на кажжой из них своя головка. Функция перехода теперь действует так: $\delta : Q \times \Sigma^k \to Q \times \Sigma^k \times \{\rightarrow, \leftarrow, \bullet\}^k$. Считается, что вход машины Тьюринга записан на первой ленте, результат работы тоже пишется на первую ленту.

Универсальная машина Тьюринга — такая машина, которая на входе (M, x) исполняет машину M на входе x.

1.3 Моделирование k-ленточной машины Тьюринга на одноленточной

Опишем универсальную одноленточную машину Тьюринга для семейства k-ленточных машин Тьюринга, которые используют один и тот же алфавит Σ, k и $|\Sigma|$ мы будем считать константами. Универсальная машина Тьюринга будет использовать другой алфавит, в частности в одном символе можно закодировать k символов Σ (по одному на каждой из лент) вместе со специальной пометкой, которая означает, есть ли в текущем символе моделируемой машины головка. В начале ленты универсальной машины записано описание моделируемой машины Тьюринга и текущее состояние, затем после специального разделителя идут символы, первый из которых кодирует первые символы на каждой из лент, второй кодирует второй символ на каждой из лент, когда на всех лентах начинаются пробелы, то и у универсальной машины будут идти пробелы. Универсальная машина в своем состоянии помнит символ под головкой на каждой из лент моделируемой машины. Чтобы сделать шаг, она смотрит в описании машины, чтобы узнать, что нужно сделать, для этого находит там инструкцию, которая соответствует состоянию машины и символам под головками. Соответствующим образом модифицирует состояние моделируемой машины, и в своем состоянии запоминает, что нужно сделать на лентах, после этого за два прохода туда-сюда выполняет эти действия. Если инструкция гласит сдвинуться вправо, то это делается при проходе слева направо, если инструкция гласит сдвинуться влево, то это делается во время прохода справа налево. Если моделируемая машина работает T шагов, то моделирующая будет работать $O(T^2)$, так как на каждом шаге нужно константное число раз пройтись по всей ленте от начала до конца. На изменение состояние и изучение машины Тьюринга тратится константное число шагов (которое, конечно, можно зависить от размера описания моделируемой машины).

Можно показать, что принципиально улучшить эту оценку нельзя. А именно мы покажем, что язык палиндромов на одноленточной машине не распознается быстрее, чем за $\Omega(n^2)$, а на двухленточной его можно распознать за $O(n)$.

Теорема 1.1. На одноленточной машине Тьюринга, проверить является ли строка палиндромом быстрее, чем за $\Omega(n^2)$, нельзя.

Доказательство. Мы будем применять принцип несжимаемости. Состоит он в следующем простом утверждении. Если функция $f : \{0,1\}^n \to \{0,1\}^*$ инъективна, то найдется такая строка $x \in \{0,1\}^n$, что $|f(x)| \geq n$. Объясняется это тем, что бинарных строк длины меньших, чем n, $2^n - 1$, что на один меньше, чем строк длины n.

Для каждой перегородки между двумя клетками на ленте машины Тьюринга можно записать последовательность состояний, в которых эта перегородка пересекалась при работе машины на данном входе.

Пусть есть некоторая машина Тьюринга M, которая проверяет входную строку на палиндром. Пусть $T(x)$ — время ее работы на входе x. Можно считать, что $T(x) \leq c|x|^2$. Рассмотрим поведение машины M на строчках вида $x0^n x^R$, где $x \in \{0,1\}^n$, а x^R — это перевернутая строка x. Рассмотрим перегородки между клетками с номерами

$n+1, n+2, \ldots, 2n$. Как минимум одну из них машина Тьюринга на входе x пересекает не более, чем $T(x)/n$ раз. Пусть это перегородка с номером $n+i$. Пусть на входе $x0^n x^R$ перегородка с номером $n+i$ пересекалась в состояниях $q_1, q_2, \ldots, q_m$, где $m \leq T(x)/n$.

Заметим, что строка x длины n однозначно определяется по числу i и последовательности состояний $q_1, q_2, \ldots, q_m$. Пусть существует другая такая строка $y \in \{0,1\}^n$, что машина Тьюринга на входе $y0^i$ в первый раз пересечет перегородку в состоянии q_1, если потом запустить машину на пересечении назад в состоянии q_2, то она опять пересечет перегородку в состоянии q_3 и т.д. Тогда такая машина примет и слово $y0^n x^R$, которое не является палиндромом.

Теперь мы готовы построить инъективную функцию, к которой мы применим принцип несжимаемости $f : \{0,1\}^n \to \{0,1\}^*$.

$f(x) = (i, m, q_1, q_2, \ldots, q_m)$. Нужно пояснить, как мы это кодируем в бинарном алфавите. Состояния можно кодировать с помощью константного числа символов. Чтобы кодировать запятые можно удваивать каждый бит в строчках, а запятую заменять на 01. Более того существует алгоритм, который по n, i, $q_1, q_2, \ldots, q_m$ вычисляет x. Это значит, что $|f(x)| \leq c_1 \log n + c_2 T(x)/n + c_3$. Из принципа несжимаемости существует такой $x \in \{0,1\}^n$, что $|f(x)| \geq n$, при таком x выполняется $T(x) \geq c_4 n^2$, что и требовалось доказать. □

1.4 Моделирование k-ленточной машины Тьюринга на k-ленточной

Моделировать k-ленточную машину на k-ленточной можно с линейным замедлением. Каждая лента моделируемой машины располагается на своей ленте моделируемой машины. Чтобы не бегать каждый раз в начало первой ленты смотреть на описание машины, описание машины можно возить с собой рядом с головкой машины на первой ленте. Так как описание машины занимает константное число ячеек, то время работы от моделирования увеличится лишь в константу раз.

1.5 Моделирование k-ленточной машины Тьюринга на 2-ленточной

Оказывается k-ленточную машину, работающую $T(n)$ шагов можно промоделировать за $O(T(n) \log T(n))$ шагов. Сначала покажем, что моделирование машины можно свести к моделированию стека. Каждая лента машины Тьюринга разбивается на два стека: символы слева от головки и символы справа от головки. Каждая операция с лентой соответствует нескольким операциям со стеками. k лент представляются в виде $2k$ стеков, все стеки мы будем хранить на одной ленте, к примеру, можно разделить клетки ленты между стеками через $2k$ (альтернативно можно укрупнить алфавит так, чтобы в одном символе умещалось $2k$ символов). Мы покажем, что мы сможем выполнить $T(n)$ операций с одним стеком за $O(T(n) \log T(n))$ шагов, причем после каждой операции вторая лента будет пустая и обе головки будут расположены в начале ленты. Тогда

на моделирование работы $2k$ стеков уйдет тоже $O(T(n)\log T(n))$ шагов, так как k — константа.

Прежде, чем описывать, как устроен стек, решим другую задачу. Предположим, мы хотим хранить счетчик с операциями ± 1. Причем хочется, чтобы среднее число замен цифр счетчика на одну операцию было бы константой. Если счетчик хранить просто в бинарном виде, то если N раз подряд с числом вида $2^n - 1$ совершать пару действий $+1, -1$, то число меняющихся разрядов будет не меньше Nn. Поэтому будем использовать ленивую систему, в которой цифры будут $0, 1$ и 2. Представление в такой системе счисления будет неоднозначным, к примеру $1 \cdot 2^1 = 2 \cdot 2^0$. Пусть у нас есть число $K = \sum_{i=0}^{n} a_i 2^i$, где $a_i \in \{0, 1, 2\}$. Покажем, как к нему прибавлять единицу. Мы находим наименьшее такое k, что $a_k < 2$, число $K + 1$ будет представлено в виде $\sum_{i=0}^{k-1} 2^i + (a_k + 1) \cdot 2^k + \sum_{k<i\le n} a_i 2^i$. Аналогично, чтобы отнять единицу, мы найдем наименьшее такое k, что $a_k > 0$, число $K - 1$ будет представлено в виде $\sum_{i=0}^{k-1} 2^i + (a_k - 1) \cdot 2^k + \sum_{k<i\le n} a_i 2^i$. Заметим, что если операция прибавления или вычитания единицы затронула k-й разряд, то в следующий раз k-й разряд будет затронут как минимум через 2^k операций, так как после операций, в которой участвовал k-й разряд все предыдущие разряды равны единице. Итого, если мы делали T операций ± 1, то k-й разряд затрагивался не больше, чем $\lceil \frac{T}{2^k} \rceil$ раз. Значит, всего затронутых разрядов не больше $\sum_{k\ge 0} \lceil \frac{T}{2^k} \rceil < 2T \sum_{k\ge 0} \frac{1}{2^k} = 4T$, т.е. не больше, чем 4 разряда на одну операцию.

Теперь опишем, как мы будем хранить стек. Стек мы разобьем на блоки с номерами $0, 1, 2, \ldots$. Блок с номером k состоит из двух зон размера 2^k. Каждый блок может быть пуст, либо полностью заполнены обе зоны, либо заполнен наполовину, это соответствует цифрам $0, 1$ и 2 соответственно. Содержимое стеков получится, если выписать подряд содержимое блоков с нулевого, причем сначала выписывается первая зона, затем вторая. Мы будем поддерживать инвариант, что наполненность блоков $(0, 1, 2)$ задает число в ленивой двоичной системе счисления. Чтобы добавить элемент в стек, нужно найти блок с минимальным номером, который заполнен не полностью, если требуется, то сдвинуть элементы так, чтобы свободной была первая зона, поместить в первую зону половину элементов из предыдущих блоков (сохраняя порядок), остальные элементы раскидать так, чтобы каждый блок был заполнен ровно наполовину. Аналогично можно определить операцию удаления элемента из стека.

Операции добавления и удаления элемента реализуются с помощью второй ленты, на которую копируются все элементы из блоков, которые просматриваются. Операции удаления и добавления элементов, которые затрагивают k блоков можно реализовать за $O(2^k)$ операций. Между двумя операциями, которые затрагивают k-й блок проходит не меньше 2^k операций. Таким образом, чтобы выполнить T операций со стеком, нам понадобится $C \sum_{k=0}^{\ell} 2^k \frac{T}{2^k}$ операций, где $\ell = O(\log T(n))$ — число блоков, которые можно успеть затронуть за T шагов. Таким образом общее число операций оценивается $O(T(n)\log T(n))$.

2 Классы P и P/poly

Основной моделью вычислений для нас будет являться многоленточная машина Тьюринга. Количество лент может быть произвольным.

Пусть $f : \mathbb{N} \to \mathbb{N}$. Класс DTime$[f(n)]$ состоит из языков L, для которых существует многоленточная машина Тьюринга M, распознающая этот язык, что для любого входа x эта машина остановится на входе x за не более, чем $O(f(|x|))$ шагов. Класс $\mathrm{P} = \cup_{c>0}\mathrm{DTime}[\mathrm{n}^c]$ — это множество языков, принадлежность которым можно проверить за полиномиальное время.

Напомним, что булева схема — это такое вычислительное устройство, которое представляет собой ориентированный граф без циклов, в этом графе есть n вершин, которые мы называем входами схемы, в эти вершины ребра не входят, будем считать, что входы помечены $x_1, x_2, \ldots, x_n$. Каждая из вершин, которая не является входом, в которую входит k ребер помечена конкретной булевой функцией $\{0,1\}^k \to \{0,1\}$, одна из вершин схемы называется выходом. На вход схемы подаются 0/1 переменные, вершины схемы сортируются так, чтобы ребра вели из вершин с меньшими номерами в вершины с большими номерами (для ориентированных графов без циклов это всегда можно сделать), после этого одним проходом по списку вершин в каждой вершине можно посчитать значение с помощью функции, которая задана в этой вершине. Результатом схемы является значение, посчитанное в выходе схемы. Размером схемы мы будем называть количество вершин в графе, не считая входы. В ближайшее время мы будем рассматривать только схемы, которые использую бинарные $\wedge$, $\vee$ и унарный $\neg$ в качестве функций в вершинах.

Мы рассмотрим новую модель вычислений — семейство булевых схем. Пусть $\{C_n\}_{n=1}^{\infty}$ — это семейство булевых схем, схема C_n имеет n входов. Мы говорим, что семейство схем $\{C_n\}_{n=1}^{\infty}$ распознает язык L, если для любого x выполняется $C_{|x|}(x) = L(x)$, где $L(x)$ — характеристическая функция языка L.

SIZE$[T(n)]$ — это класс языков, которые распознаются семейством схем $\{C_n\}_{n=1}^{\infty}$, где C_n имеет размер не больше $T(n)$.

$\mathrm{P/poly} = \bigcup_{c>0} \mathrm{SIZE}[n^c]$.

Лемма 2.1. Существует такой алгоритм A, который получает на вход числа T, n и строку M, что 1) этот алгоритм работает $\mathrm{poly}(n + T + |M|)$ шагов; 2) если детерминированная машина Тьюринга M на всех входах длины n работает не больше T шагов, и на всех входах длины n машина M выдает ответ из множества $\{0,1\}$, то алгоритма A выдает схему C, которая имеет n входов, и для всех $x \in \{0,1\}^n$ ответ $M(x)$ совпадает с $C(x)$.

Доказательство. Можно считать, что машина Тьюринга M одноленточная, так как иначе можно построить эквивалентную ей одноленточную машину с квадратичным замедлением. Рассмотрим квадрат $T \times T$, i-я строчка которого соответствует содержимому первых T ячеек работы машины M на входе x длины n. Кроме того в каждую клетку поместим индикатор, и несколько битов, которые будут кодировать текущее состояние, есть ли в этой клетке головка есть. Мы считаем, что если машина остановилась, то на следующем шаге состояние ленты и состояние не меняются. Итого, в каждой клетке

записано константное (зависящее от машины M) число битов. Построим схему, которая будет содержать вершины, в которых вычисляется все содержимое этого квадрата. На верхней строчке квадрата мы не знаем только битов строки x — это будут входы схемы, остальные биты константные. Значения в каждом из квадратиков i-й строки при $i \geq 2$ могут быть однозначно определены из значения трех соседних квадратиков $(i-1)$-ой строчки, следовательно эти значения могут быть вычислены схемой константного размера. Осталось добавить схему для вычисления ответа, которая выдаст 1, если состояние последней строки q_{yes} и 0 иначе. □

Из доказательства леммы 2.1 видно, что не очень важно, что машина выдает ответы из одного бита, даже несколько битов выхода можно промоделировать схемой.

Теорема 2.1. $\mathrm{P} \subseteq \mathrm{P/poly}$

Доказательство. Следует из леммы 2.1. □

Можно показать, что это включение строгое, так как, например, P/poly содержит неразрешимые языки. Пусть H неразрешимый язык, тогда язык $U_H = \{1^n \mid n \in H\}$ является неразрешимым, так как H к нему m-сводится. С другой стороны, язык U_H может быть распознан семейством схем полиномиального размера: для каждой длины входа нужно выбрать схему, которая отвергает все входы, кроме, возможно, входа из всех единиц.

3 Эффективные системы доказательств и класс NP

Определение 3.1. Системой доказательств для языка L называется алгоритм Π, который останавливается на всех входах и обладает следующими свойствами:

- (Полнота) Для любого $x \in L$ существует строка w, что $\Pi(x, w) = 1$;
- (Корректность) Для любого $x \notin L$ для любой строки w выполняется $\Pi(x, w) = 0$.

Система доказательств Π для языка L называется эффективной, если существует такой полином q, что время работы алгоритма $\Pi(x, w)$ ограничено $q(|x| + |w|)$.

Упражнение 3.1. Покажите, что если для языка существует система доказательств, то существует и эффективная система доказательств.

В дальнейшем мы будем считать, что системы доказательств эффективны.

Определение 3.2. Язык L лежит в классе NP, если существует такая эффективная система доказательств Π для L и полином q, что для любого $x \in L$ существует строка y длины не больше, чем $q(|x|)$, что $\Pi(x, y) = 1$.

Строку y будем называть сертификатом принадлежности $x \in L$, если $\Pi(x, y) = 1$. Иными словами, язык содержится в классе NP, если для каждой строки из L ее принадлежность L можно коротко сертифицировать и сертификат можно проверить за полиномиальное время.

Примеры языков из NP:

- Язык SAT, который состоит из выполнимых пропозициональных формул. Сертификатом является выполняющий набор.
- Множество графов, в которых есть гамильтонов путь (путь, проходящий по каждой вершине ровно 1 раз). Сертификатом является перестановка вершин, для проверки сертификата надо проверить, что это перестановка (все вершины разные, их нужное количество) и что соседние две вершины соединены ребром.
- Язык, состоящий из пар (G, k), где G — это граф, в котором есть полный подграф (клика) на k вершинах. Сертификатом является множество из k вершин графа, проверка сертификата состоит из проверки того, что эти вершины образуют клику.
- Язык составных натуральных чисел. Сертификатом для числа N является его разложение на множители $N = A \times B$, где A, B — натуральные числа большие 1.
- Язык простых чисел. Это нетривиальный пример, ниже мы покажем, как можно коротко сертифицировать простоту числа. На самом деле известно, что этот язык лежит в классе P.

Лемма 3.1 (Критерий простоты Пратта)**.** Число n является простым тогда и только тогда, когда найдется такое число a, что

1. $a^{n-1} \equiv 1 \bmod n$;
2. Для каждого простого делителя q числа $n-1$ выполняется $a^{\frac{n-1}{q}} \not\equiv 1 \bmod n$.

Доказательство. Пусть n является простым, тогда в качестве a можно выбрать первообразный корень по модулю n, т.е. такое число, что все степени $a, a^2, \ldots, a^{n-1}$ дают разные остатки при делении на n. a^{n-1} дает остаток 1, например, по малой теореме Ферма.

Пусть найдется такое число a, но при этом n — составное. В таком случае число $\varphi(n) = |\{1 \le k \le n \mid k \text{ и } n \text{ взаимно просты}\}|$ строго меньше, чем $n-1$. По теореме Эйлера $a^{\varphi(n)} = 1$, пусть d — минимальное такое натуральное число, что $a^d \equiv 1 \bmod n$. Известно, что $d \le \varphi(n) < n-1$. Покажем, что $n-1$ делится на d. Разделим $n-1$ на d с остатком: $(n-1) = bd + r$, где $0 \le r < d$. $a^{n-1} = (a^d)^b a^r \equiv a^r \bmod n$, поскольку d было минимальным числом, то $r = 0$. Пусть q — это простой делитель числа $\frac{n-1}{d}$, тогда $\frac{n-1}{q}$ делится на d, следовательно $a^{\frac{n-1}{q}} \equiv 1 \bmod n$. □

Теперь покажем, как с помощью этой леммы коротко сертифицировать простоту числа n. Доказательством простоты числа n будет являться число a, что $a^{n-1} \equiv 1 \bmod n$ и разложение числа $n-1 = p_1 p_2 \dots p_k$ на простые (возможно совпадающие) множители. Чтобы проверить это доказательство нужно будет проверить, что $a^{n-1} \equiv 1 \bmod n$ и, что $a^{\frac{n-1}{p_i}} \not\equiv 1 \bmod n$ для всех i, кроме того нужно проверить, что $p_1 p_2 \dots p_k = n-1$ и сертифицировать простоту чисел $p_1, p_2, \dots, p_k$. Нарисуем дерево, в корне дерева будет стоять число n, сыновья корня помечены простыми делителями числа $n-1$. И для каждой вершины дерева выполняется тоже самое: если вершина помечена числом q, то сыновья помечены всеми простыми делителями числа $q-1$, в листах дерева находятся простые числа, меньшие 10. Нетрудно видеть, что глубина дерева не больше, чем $\log n$, поскольку число в потомке как минимум в два раза меньше. Кроме того для каждой вершины выполняется, что число в ней строго больше, чем произведение чисел в сыновьях этой вершины, следовательно произведение чисел во всех вершинах, которые находятся на одинаковом расстоянии до корня, строго меньше, чем n. Значит, на каждом уровне находится не больше, чем $\log n$ вершин. Таким образом размер дерева $O(\log^2 n)$, в каждой вершине находится число размера $O(\log n)$ и все проверки можно выполнить за $poly(\log n)$ время.

В определении класса NP есть асимметрия в принадлежности и непринадлежности. К примеру, мы не знаем, лежит ли в NP язык UNSAT, состоящий из невыполнимых пропозициональных формул. Проблема заключается в том, что непонятно, как коротко сертифицировать то, что у формулы нет выполняющего набора.

Нетрудно понять, что $\mathrm{P} \subseteq \mathrm{NP}$, является ли это включение строгим — это открытый вопрос.

4 Недетерминированные машины Тьюринга и класс NP

Недетерминированная машина Тьюринга (НМТ) отличается от детерминированной тем, что вместо одной функции перехода, есть две функции перехода δ_0 и δ_1. Для одноленточной машины $\delta_0, \delta_1 : Q \times \Sigma \to Q \times \Sigma \times \{\rightarrow, \leftarrow, \bullet\}$. Когда мы говорим о недетерминированной машине Тьюринга, то задача, которую мы решаем — это задача о принадлежности к языку. То есть недетерминированная машина Тьюринга может лишь только принять входное слово, отвергнуть входное слово и не остановиться. Мы говорим, что НМТ принимает входное слово, если существует последовательность корректных переходов (т.е. каждый переход согласован с δ_0 или с δ_1), следуя которым машина из начальной конфигурации приходит в конфигурацию в состоянии q_{yes}. Мы говорим, что НМТ отвергает входное слово, если любая последовательность корректных переходов (т.е. согласованных с δ), приходит в конфигурацию в состоянии q_{no}. В противном случае НМТ не останавливается.

Заметим, что ответы q_{yes} и q_{no} не симметричны. Сложностью по времени НМТ на данном входе называется максимальное число шагов из начальной конфигурации

до конфигурации в состоянии q_{no} или q_{yes} (или ∞, если машина не останавливается). Аналогично детерминированному случаю определяется сложность по времени НМТ и сложность по памяти.

Заметим, что детерминированная машина — это частный случай недетерминированной, когда $\delta_0 = \delta_1$. Можно определнить класс NTime[f(n)] как множество языков, которые распознаются недетерминированной машиной Тьюринга за время $O(f(n))$, где n длина входа. Класс NP можно эквивалентно определить как $\cup_{c>0}\mathrm{NTime}[\mathrm{n}^{\mathrm{c}}]$, чуть позже мы покажем, что от этого класс NP не поменяется.

4.1 Машина Тьюринга с подсказкой

.

Можно определить недетерминированную машину Тьюринга и как детерминированную машину с подсказкой. У такой машины есть дополнительная лента-лента подсказки. Мы будем говорить, что машина Тьюринга принимает слово, если

- Она останавливается на данном входе при всех содержимых ленты-подсказки;
- Существует такая подсказка, что машина останавливается в состоянии q_{yes}.

Время работы такой машины измеряется как максимум по времени работы для всех возможных подсказок.

Теорема 4.1. Следующие условия эквивалентны

1. $L \in \mathrm{NP}$;
2. L распознается машиной с подсказкой за полиномиальное время.
3. L распознается недетерминированной машиной Тьюринга за полиномиальное время.

Доказательство. Пусть $L \in \mathrm{NP}$, тогда существует такая система доказательств Π и полином q, что $x \in L$ тогда и только тогда, когда $\exists y, |y| \leq q(|x|)$, что $\Pi(x, y) = 1$. Машина с подсказкой выпишет рядом со входом подсказку через запятую и будет эмулировать машину, вычисляющую Π, за полиномиальное время.

Пусть L распознается машиной с подсказкой за время, ограниченное полиномом $q(n)$. Тогда недетерминированная машина с помощью недетерминированных выборов сгенерирует строку длины $q(n)$, которую потом будет использовать как подсказку при моделировании машины с подсказкой.

Пусть L распознается недетерминированной машиной Тьюринга M за время $p(n)$. Тогда определим $\Pi(x, y)$ так: $\Pi(x, y)$ равняется единице, если строка y кодирует последовательность переходов (каждый раз применяем либо δ_0, либо δ_1), которые приводят в принимающее состояние. $\square$

4.2 Сведения, NP-полнота, теорема Кука-Левина

Определение 4.1. Будем говорить, что язык A сводится по Карпу (или полиномиально сводится) к языку B, если существует полиномиальный по времени алгоритм f, что для всех x выполняется $x \in A$ тогда и только тогда, когда $f(x) \in B$. Обозначение: $A \leq_p B$.

Предложение 4.1.

- Если $A \leq_p B$ и $B \leq_p C$, то $A \leq_p C$.
- Если $A \leq_p B$ и $B \in \mathrm{P}$, то и $A \in \mathrm{P}$;

Язык называется NP-трудным, если к нему сводится все языки из класса NP. Язык называется NP-полным, если он NP-трудный и принадлежит классу NP.

Наша ближайшая цель доказать теорему Кука-Левина об NP-полноте задачи SAT.

Построим пример NP-полного языка. Для этого немного модифицируем задачу об остановке алгоритма. Рассмотрим задачу об ограниченной остановке. $BH = \{(s, x, 1^t) \mid \exists y$, что <s>(x,y) принимает за не более, чем t шагов$\}$. Здесь и далее 1^t обозначает строчку из t единиц.

Предложение 4.2. $BH \in \mathrm{NP}$.

Доказательство. Для тройки $(s, x, 1^t) \in L$ подсказкой будет строка y, при которой $<s>(x, y)$ принимает за не более, чем t шагов. Можно считать, что длина такой строки y не превосходит t (или в константу раз больше, в зависимости от рассматриваемой модели вычисления), так как за t шагов алгоритм не успеет прочитать большее число символов y. Проверить такой сертификат можно моделированием, время проверки будет полиномиально от t, следовательно полиномиально от длины входа. □

Теорема 4.2. Для любого языка $L \in \mathrm{NP}$ выполняется $L \leq_p BH$.

Доказательство. Пусть Π система доказательств для языка L, которая работает $q(|x|+|w|)$ шагов, и размер доказательств ограничен полином p. Тогда $x \mapsto (\sharp\Pi, x, 1^{q(|x|+p(|x|))})$ является сведением языка L к языку BH. Действительно, если $x \in L$, то найдется строка y длиной не более, чем $p(x)$, что $\Pi(x, y) = 1$.

В обратную сторону, если $(\sharp\Pi, x, 1^{q(|x|+p(|x|))}) \in BH$, то $\Pi(x, y) = 1$, следовательно $x \in L$. Из того, что $\Pi(x, y) = 1$ следует, что $|y| \leq p(|x|)$. □

Язык $Circuit - SAT$ состоит из выполнимых схем, т.е. таких схем, что можно подать на входы такие значения, чтобы результат схемы был равен 1.

Теорема 4.3. Задача $Circuit - SAT$ является NP-полной.

Доказательство. Сведем задачу BH к $Circuit - SAT$. Напомним, что $BH = \{(M, x, 1^t) \mid \exists y$, что <M>(x,y) принимает за не более, чем t шагов$\}$. Пусть C — это результат работы алгоритма $A(t, t, M)$ из леммы 2.1. Пусть схема C_x получается из C, если вместо первых ее $|x|$ входов подставить x. Тогда $(M, x, 1^t) \in BH$ тогда и только тогда, когда $C_x \in Circuit - SAT$. □

$3SAT$ — это множество выполнимых формул в КНФ, в которых в каждый дизъюнкт входит ровно три литерала.

Следствие 4.1. *SAT и $3SAT$ являются NP-полными*

Доказательство. $3SAT$ сводится к SAT тождественным сведением. Поэтому достаточно доказать, что $Circuit - SAT$ сводится к $3SAT$.

Рассмотрим схему C от переменных $x_1, x_2, \ldots, x_n$, заведем также переменные для остальных вершин схемы и напишем такие условия:

- t, если t — выход схемы;
- $z = a \wedge b$, если в вершине, помеченной переменной z вычисляется конъюнкция значений в вершинах, помеченных a и b;
- $z = a \vee b$, если в вершине, помеченной переменной z вычисляется дизъюнкция значений в вершинах, помеченных a и b;
- $z = \neg b$, если в вершине, помеченной переменной z вычисляется отрицание значения в вершине, помеченной b.

Все эти условия мы соединим конъюнкцией. Каждое из этих условий зависит от трех переменных, т.е. может быть записано в виде формулы в 3-КНФ константного размера. □

Теорема 4.4. Задача о независимом множестве является NP-полной.

Доказательство. Напомним, что мы доказываем NP-полноту языка, который состоит из пар (G, k), где G — граф, в котором есть независимое множество размера k. Принадлежность этого языка классу NP очевидна, сертификатом является само независимое множество размера k.

Сведем к этой задаче задачу $3SAT$. Построим по 3КНФ формуле, в которой m дизъюнктов, граф на $O(m)$ вершинах. Каждому дизъюнкту формулы соотвествует 7 вершин: по одной для каждого набора, который выполняет этот дизъюнкт (если в дизъюнкте меньше, чем три переменных, то таких наборов будет меньше). Соединим две вершины ребром, если наборы, им соответсвующие, противоречат друг другу. Формула выполнима тогда и только тогда, когда в построенном графе есть независимое множество размера m. □

5 NP-задачи поиска

5.1 Сведение задач поиска к задачам распознавания

Будем называть предикат Q полиномиально ограниченным, если существует такой полином p, что из $Q(x, y) = 1$ следует, что $|y| \leq p(|x|)$. Каждый язык из класса NP задается

некоторым полиномиально ограниченным и полиномиально проверяемым предикатом Q, таким что $x \in L$ тогда и только тогда, когда $\exists y Q(x, y)$.

Пусть Q — полиномиально ограниченный полиномиально проверяемый двуместный предикат. Задачей поиска, заданной предикатом Q является следующая массовая вычислительная задача: по x найти y, что выполняется $Q(x, y) = 1$. Множество задач поиска, задаваемых полиномиально ограниченными и полиномиально проверяемыми предикатами мы будем обозначать $\widetilde{\mathrm{NP}}$.

Мы говорим, что задача поиска, заданная предикатом Q решается алгоритмом A, если для любого x, если существует такой y, что $Q(x, y) = 1$, то $Q(x, A(x)) = 1$. Множество задач поиска, для которых существуют полиномиальные по времени алгоритмы, их решающие, мы будем обозначать $\widetilde{\mathrm{P}}$.

Нетрудно понять, что если задача поиска, соответствующая предикату Q лежит в $\widetilde{\mathrm{P}}$, то соответствующий язык лежит в P. Оказывается, что невозможна ситуация, когда все языки простые, а задачи поиска сложные.

Определение 5.1. Пусть A — некоторый язык. Машиной Тьюринга с оракулом A отличается от обычной машины Тьюринга тем, что у нее есть специальная оракульная лента для запросов к оракулу и три специальных состояния q', q'_{yes}, q'_{no}. Если машина пришла в состояние q', то если слово, записанное на оракульной ленте лежит в языке A, то машина переходит в состояние q'_{yes}, а если не лежит, то в состояние q'_{no}. При подсчете времени работы машины Тьюринга запрос к оракулу занимает всего один шаг вне зависимости от языка A.

Определение 5.2. Вычислительная задача (проверки принадлежности языку L или NP-задача поиска Q) сводится по Куку (иногда говорят сводится по Тьюрингу за полиномиальное время) к языку A, если она может быть решена с помощью полиномиальной по времени машины Тьюринга с оракулом A. Обозначение: $L \leq_{pT} A$ ($Q \leq_{pT} A$).

Предложение 5.1.

- Если $L_1 \leq_p L_2$, то и $L_1 \leq_{pT} L_2$.
- $\leq_{pT}$ — транзитивное отношение, даже если цепочка начинается на задаче поиска.
- Если $L \leq_{pT} L'$ и $L' \in \mathrm{P}$, то $L \in \mathrm{P}$; аналогичное выполняется и для задач поиска: если $Q \leq_{pT} L'$ и $L' \in \mathrm{P}$, то $Q \in \widetilde{\mathrm{P}}$;

Теорема 5.1. Каждая задача поиска из $\widetilde{\mathrm{NP}}$ сводится по Куку к некоторому языку из NP.

Доказательство. Введем на множестве всех строк нестрогий лексикографический порядок $\preceq$. Сначала строки упорядочиваются по длине, а среди строк данной длины порядок алфавитный. Пусть Q полиномиально ограниченный (полиномом p) полиномиально проверяемый предикат. Тогда задачу поиска, задаваемую предикатом Q можно свести к такому языку $L = \{(x, a) \mid |a| \leq p(|x|), \exists y \preceq a : Q(x, y) = 1\}$. Подсказку y можно найти за полиномиальное время, обращаясь к языку L с помощью двоичного поиска. □

Следствие 5.1. Если $P = NP$, то $\widetilde{\mathrm{P}} = \widetilde{\mathrm{NP}}$.

Следствие 5.2. Если NP-полный язык L задается предикатом Q, то задача поиска Q сводится по Куку к L.

Доказательство. Задача поиска Q сводится по Куку к NP-языку, который сводится по Карпу (а следовательно и по Куку) к NP-полному языку L. □

Замечание 5.1. Для задачи SAT сведение задачи поиска к задаче распознавания можно было бы сделать так: подставить значение переменной $x := 1$, если получилась выполнимая формула, то продолжать для формулы, в которой уже одна переменная подставлена. Если получилась невыполнимая формула, подставить $x := 0$. И т.д.

6 О соотношении между классами P и NP

Мы знаем, что класс NP содержит класс P. В классе NP есть полные задачи. А есть ли в NP что-нибудь еще кроме полных задач и полиномиально разрешимых языков? Оказывается, что в случае P $\neq$ NP ответ на этот вопрос положительный.

Теорема 6.1 (Ладнер)**.** Если P $\neq$ NP, то существует язык из NP, который не является NP-полным и который не принадлежит классу P.

Доказательство. Пусть каждому натуральному числу i соответствует машина Тьюринга M_i, описание которой — это битово представление числа i. Каждая машина Тьюринга в этой нумерации встречается бесконечное число раз, так как например, к описанию машины можно добавлять его битовое представление.

Мы определим язык SAT_H — это будет такая ослабленная версия языка SAT, что она перестанет быть NP-полной, но все же не будет принадлежать P, если P $\neq$ NP. Формально $SAT_H = \{\varphi 01^{n^{H(n)}} \mid \varphi \in SAT, |\varphi| = n\}$, где $H(n)$ — функция, определяемая следующим образом: значение $H(n)$ будет определяться через значения $H(m)$ при $1 \leq m \leq \log n$. $H(n)$ — это наименьше такое число $i < \lfloor \log\log n \rfloor$, что для всех $x \in \{0,1\}^*$ для которых $|x| \leq \log n$ машина M_i выдает $SAT_H(x)$ за $i|x|^i$ шагов. Если такого i не нашлось, то $H(n) = \lfloor \log\log n \rfloor$.

Покажем, что $H(n)$ вычислимо за полиномиальное от n время. Более того за полиномиальное от n время можно посчитать все значения от $H(1)$ до $H(n)$. Пусть все значения от $H(1)$ до $H(n-1)$ посчитаны, оценим время, необходимое для того, чтобы посчитать $H(n)$. Для этого нужно $O(\log\log n)$ раз на не более, чем $2n-1$ строчках (строк длины не более $\log n$ не больше, чем $2n-1$) запустить машину на $\log\log n(\log n)^{\log\log n} = o(n)$ шагов. Для каждого из этих запусков нужно будет решить задачу о принадлежности языку SAT. Это можно сделать за время $O(n)$. Итого нужно сделать $O(n^3)$ шагов. Чтобы посчитать всю таблицу хватит $O(n^4)$ операций. Из этого следует, что $SAT_H \in$ NP.

Лемма 6.1. 1)Если $SAT_H \in$ P, то $H(n) = O(1)$. 2)Если $SAT_H \notin$ P, то $H(n) \to \infty$.

Доказательство. 1) Пусть машина M решает SAT_H за время cn^c. Существует такое число i, что M_i задает машину M и $i > c$. Тогда для $n > 2^{2^i}$ выполняется $H(n) \leq i$.

2) Нетрудно видеть, что $H(n)$ не убывает. Пусть $H(n)$ с какого-то места стабилизируется и равна i. Тогда M_i решает SAT_H за время in^i. □

Если $SAT_H \in$ P, то $H(n) \leq c$, тогда SAT сводится к SAT_H, следовательно P = NP.

Если SAT_H — это NP-полный язык, тогда существует сведение f языка SAT к языку SAT_H. Пусть для больших длин входов n функция f вычисляется за не более, чем n^c шагов. Поскольку $SAT_H \notin$ P, то $H(n) > 3c$ при достаточно больших n. Тогда при достаточно больших n функция f отображает формулу φ длины n в строчку $\psi 01^{k^{H(k)}}$, где $k = |\psi|$. Поскольку при больших n должно выполняться $|f(\varphi)| < n^c$, то $k \leq n^{1/3}$. Итого для некоторой константы n_0 при $n > n_0$ функция $f(\varphi)$ выдает либо строку вида $\psi 01^{k^{H(k)}}$, либо синтаксически некорректную строку. Во втором случае φ невыполнима, а в первом выполнимость φ эквивалентна выполнимости ψ, которая существенно короче. За логарифмическое число шагов мы сведем выполнимость формулы φ к выполнимости формулы константного размера, последняя задача решается за константное время. Значит, если такое сведение f существует, то $SAT \in$ P, или P = NP. □

6.1 Релятивизация

DTime$[f(n)]$ — это множество языков, которые можно распознать на многоленточной машине Тьюринга за время $O(f(n))$.

Тогда P $= \cup_{c>0}$DTime[n^c].

NTime$[f(n)]$ — это множество языков, которые можно распознать на многоленточной недетерминированной машине Тьюринга за время $O(f(n))$.

Тогда NP $= \cup_{c>0}$NTime[n^c].

Мы уже определили вычисления с оракулами, аналогично можно определить и классы с оракулами.

DTime$^A[f(n)]$ — это множество языков, которые можно распознать на многоленточной детерминированной машине Тьюринга с оракулом A за время $O(f(n))$.

NTime$^A[f(n)]$ — это множество языков, которые можно распознать на многоленточной недетерминированной машине Тьюринга с оракулом A за время $O(f(n))$.

P$^A = \cup_{c>0}$DTime$^A[n^c]$.

NP$^A = \cup_{c>0}$NTime$^A[n^c]$.

Определим EXP $= \cup_{c>0}$DTime$[2^{n^c}]$ — класс языков, которые можно распознать за экспоненциальное время на детерминированной машине Тьюринга.

В классе EXP есть полная задача. Устроена она стандартным образом $EXPComp = \{(M, x, t) \mid <M>(x) = 1$ за не более, чем t шагов$\}$.

Предложение 6.1. *Язык $EXPComp$ является* EXP*-полным.*

Предложение 6.2. P^{EXPComp} = NP^{EXPComp} = EXP.

Доказательство. Поскольку $EXPComp$ является EXP-полным, то EXP $\subseteq$ P^{EXPComp}. Так как детерминированная машина Тьюринга является частным случаем недетерминированной, то P$^{EXPComp} \subseteq$ NP^{EXPComp}. Осталось показать, что NP^{EXPComp} содержится в EXP. Это следует из того, что у полиномиальной по времени недетерминированной машины есть экспоненциальное число ветвей и $EXPComp \in$ EXP. □

Теорема 6.2 (Бэйкер, Гилл, Соловэй)**.** Существует такой язык B, что $\mathrm{P}^B \neq \mathrm{NP}^B$.

Доказательство. Пусть $B \subseteq \{0,1\}^*$ — некоторый язык. Определим язык $U_B = \{1^n \mid \exists x \in \{0,1\}^n \cap B\}$.

Нетрудно видеть, что $U_B \in \mathrm{NP}^B$. Действительно, подсказкой для строчки 1^n будет строка $x \in \{0,1\}^n \cap B$.

Наша цель построить такой язык B, что $U_B \notin \mathrm{P}^B$. Пусть M_i — это оракульная машина Тьюринга, i-я при перечислении всех машин с будильником $\frac{2^n}{10}$. В этой нумерации, как это обычно предполагается, каждая машина встречается бесконечное число раз. Будем определять язык B так, чтобы машина M_k с оракулом B неправильно решала язык U_B на входе 1^k. Изначально язык B пустой Мы последовательно запускаем $M_k(1^k)$ для $k = 1, 2, \ldots$, каждый раз машина обращается к текущей версии языка B. После того, как k-я машина отработала, мы немного модифицируем язык B. А именно, если ответ $M_k(1^k) = 1$, то мы менять язык B не будем, отныне $B \cap \{0,1\}^k$ всегда пусто. Если же $M_k(1^k) = 0$, то мы добавим в язык B какую-нибудь строчку длины k. Это мы можем сделать только для такой строчки, для которой ни одна из машин $M_1(1), \ldots, M_k(1^k)$ еще не запрашивали оракула. Такая строчка обязательно найдется, поскольку всего запросов было сделано не больше, чем $\frac{1}{10}\sum_{i=1}^{n} 2^i < 2^n$. □

7 Иерархия по времени

Теорема об иерархии по времени гласит, что за большее время можно решить строго большее число задач.

Рассмотрим следующий пример.

Пример 7.1. $DTime[n^2] \subsetneq DTime[n^3]$.

Доказательство. Пусть U — это универсальная двухленточная машина Тьюринга, которая моделирует многоленточные машины Тьюринга с логарифмическим замедлением. На входе $U(M, x)$ машина интерпретирует машину, которая задана строкой M на входе x. Рассмотрим язык $L = \{x \mid U(x, x)$ не принимает за $|x|^{2.5}$ шагов$\}$. Покажем, что $L \in DTime[n^3]$. Чтобы определить, лежит ли x в L, нужно просто дать вход (x, x) машине U и промоделировать ее работу $|x|^{2.5}$ шагов (для этого можно на отдельной ленте сначала вычислить число $|x|^{2.5}$ и вычитать единицу из этого числа каждый шаг машины Тьюринга U, можно проверить, что на это хватит $O(|x|^{2.5})$ шагов). Итого, на моделирование уйдет $O(|x|^{2,5})$ шагов.

Пусть $L \in DTime[n^2]$. Пусть M — это такая многоленточная машина, Тьюринга, которая решает язык L за время не большее, чем $C_1 n^2$. Если машина M на входе x работает T шагов, то машина $U(M, x)$ работает не более $C_2 T \log T$ шагов, где константа C_2 зависит от числа лент и размера алфавита машины M. Поскольку запись машины Тьюринга можно удлинять, добавляя к ней новые недостижимые состояния, можно считать, что при $n \geq |M|$ выполняется неравенство $C_1 C_2 n^2 \log(C_1 n^2) < n^{2.5}$. Тогда получается, что машина M не распознает язык L, так как M дает неправильный ответ на входе M. □

Определение 7.1. Функция $f : \mathbb{N} \to \mathbb{N}$ называется конструктивной по времени, если по n можно вычислить $f(n)$ за $O(f(n))$ шагов на многоленточной Машине Тьюринга.

Аналогично рассмотренному примеру доказывается следующая теорема:

Теорема 7.1. Пусть $h(n)$ — конструктивная по времени функция. Пусть известно, что $f(n)\log f(n) = o(h(n))$ и $h(n) = O(g(n))$. Тогда $\mathrm{DTime}[f(n)] \subsetneq \mathrm{DTime}[g(n)]$.

В частности, из этой теоремы следует, что $\mathrm{P} \subseteq \mathrm{DTime}[2^n] \subsetneq \mathrm{DTime}[2^{2n}] \subseteq \mathrm{EXP}$.

Прежде, чем перейти к теореме об иерархии для недетерминированных вычислений нужно обсудить замедление при моделировании. Оказывается, что моделирование на недетерминированной машине Тьюринга можно произвести с константным замедлением, если время работы моделируемой машины можно с точностью до константы оценить конструктивной по времени функцией. Для этого достаточно недетерминированно угадать принимающий путь моделируемой машины. А именно для каждого шага мы угадываем, какое именно правило применяется, текущее состояние и символы под головками машины. Чтобы проверить такую подсказку, нужно проверить,

- что вычисление заканчивается в принимающем состоянии;
- что описания соседних шагов согласуется с выбранными правилами перехода машины Тьюринга;
- для каждой ленты в отдельности проверить, что символы под головками будут ровно такими, если верить, что для остальных лент все правильно.

Длина подсказки $O(T(n))$, где $T(n)$ — время моделируемой машины, для проверки требуется константное число проходов по подсказке, итого время на моделирование — это $T(n)$.

Пример 7.2. $NTime[n^2] \subsetneq NTime[n^3]$.

Для начала поймем, что конструкция из предыдущего примера не пройдет, если под $<x>$ понимать недетерминированную машину, задаваемую строкой x. Точно так же показывается, что язык $L = \{x \mid <x>(x)$ не принимает за $|x|^{2.5}$ шагов$\}$ не лежит в $\mathrm{NTime}[n^2]$, но непонятно, лежит ли этот язык в $\mathrm{NTime}[n^3]$, так как непонятно, как эффективно обращать результат недетерминированной машины Тьюринга. Отметим, что мы заведомо можем обратить работу недетерминированной машины неэффективно, для этого достаточно перебрать все возможные ветви.

Доказательство. Доказательство, как и в предыдущем случае, основывается на диагонализации. Однако диагонализация будет отложенной (или ленивой). Для машины с номером i у нас будет целый промежуток длин входов от n_i до n_i^*, при этом $n_1 = 1$, $n_i^* = 2^{n_i^{2.5}}$ и $n_i^* + 1 = n_{i+1}$.

Зададим язык L на входах из промежутка от n_i до n_i^*. Язык L будет унарным, т.е. все строки, лежащие в L, будут состоять только из нулей. Для всех $n_i \le n \le$

$n_i^* - 1$ определим значение $L(0^n)$ совпадающим с результатом запуска на не более, чем $(n+1)^{2.5}$ шагов алгоритма $<i>(0^{n+1})$ (незаконченные ветви мы считаем отвергающими). Значение $L(0^{n_i^*})$ определим равным противоположному ответу $<i>(0^{n_i})$, запущенного на $(n_i)^{2.5}$ шагов.

Нетрудно понять, что $L \in \mathrm{NTime}[\mathrm{n}^3]$. Моделирование $(n+1)^{2.5}$ шагов недетерминированной машины можно осуществить за $O(n^3)$ недетерминированных шагов. А обратить ответ машины, работающей $(n_i + 1)^{2.5}$ шагов можно за время $2^{(n_i)^{2.5}}$, что меньше, чем $(n_i^*)^3$.

Покажем, что L не лежит в $\mathrm{NTime}[\mathrm{n}^2]$. Пусть $L \in \mathrm{NTime}[\mathrm{n}^2]$, пусть m — это такой номер недетерминированной машины, решающей язык L за время Cn^2, что при $n > \log m$ выполняется неравенство $Cn^2 < n^{2.5}$. Поскольку $<m>$ решает L, то $<m>(0^{n_m}) = L(0^{n_m}) = <m>(0^{n_m+1}) = L(0^{n_m+1}) = <m>(0^{n_m+2}) = \cdots = <m>(0^{n_m^*}) = L(0^{n_m^*})$, но по построению $L(0^{n_m^*}) = 1 - <m>(0^{n_m})$, противоречие. □

Этот пример обобщается до такой теоремы.

Теорема 7.2. Пусть $h(n)$ — конструктивная по времени функция, $f(n) = o(h(n))$ и $h(n+1) = o(g(n))$. Тогда $\mathrm{NTime}[f(n)] \subsetneq \mathrm{NTime}[g(n)]$.

В частности, из этой теоремы следует, что $\mathrm{NP} \subseteq \mathrm{NTime}[2^n] \subsetneq \mathrm{NTime}[2^{2n}] \subseteq \mathrm{NEXP} = \cup_{\mathrm{c}>0}\mathrm{NTime}[2^{\mathrm{n}^{\mathrm{c}}}]$.

8 Вычисления с ограничением по памяти

Когда речь идет о вычислениях с ограниченной памятью, то мы считаем, что у машины Тьюринга есть специальная входная лента, доступная только для чтения, при этом головка на входной ленте не может выйти правее первого пробела. А память меряется, как число ячеек, на которых побывала головка машины Тьюринга только на рабочих лентах. Если машина Тьюринга проверяет принадлежность языку, то у нее есть два конечных q_{yes} и q_{no}, а если машина вычисляет функцию, то у нее есть выходная лента, по которой головка может двигаться только слева направо. Память на выходной ленте также не измеряется.

$\mathrm{DSpace}[f(n)]$ — это множество языков, которые распознаются на многоленточной машине Тьюринга с использованием $O(f(n))$ памяти, где n — это длина входа.

$\mathrm{NSpace}[f(n)]$ — это множество языков, которые распознаются на многоленточной недетерминированной машине Тьюринга с использованием $O(f(n))$ памяти, где n — это длина входа. Память, использованная недетерминированной машиной, — это максимальное число посещенных ячеек на рабочих лентах по всем недетерминированным выборам машины.

Основные классы: $\mathrm{PSPACE} = \cup_{\mathrm{c}>0}\mathrm{DSpace}[\mathrm{n}^{\mathrm{c}}]$, $\mathrm{NPSPACE} = \cup_{\mathrm{c}>0}\mathrm{NSpace}[\mathrm{n}^{\mathrm{c}}]$, $\mathrm{L} = \mathrm{DSPACE}[\log n]$, $\mathrm{NL} = \mathrm{NSPACE}[\log n]$.

Предложение 8.1. Для любой функции $s(n)$ выполняется $\mathrm{DTime}[s(n)] \subseteq \mathrm{DSpace}[s(n)] \subseteq \mathrm{NSpace}[s(n)]$

Доказательство. Первое включение следует из того, что за $O(s(n))$ шагов машина успеет посетить не больше, чем $O(s(n))$ ячеек. Второе включение следует из того, что детерминированная машина является частным случаем недетерминированной. □

Определение 8.1. Функция $s : \mathbb{N} \to \mathbb{N}$ называется конструктивной по памяти, если по n можно вычислить $1^{s(n)}$ с использованием $O(s(n))$ памяти на многоленточной Машине Тьюринга.

Теорема 8.1. Если $s(n)$ — конструктивная по памяти и $s(n) \geq \log(n)$, то $\mathrm{NSpace}[s(n)] \subseteq \mathrm{DTime}[2^{O(s(n))}]$.

Доказательство. Пусть язык L решается недетерминированной машиной Тьюринга M, которая использует $O(s(n))$ памяти. Рассмотрим работу машины M на входе x. Конфигурация машины — это положение головок на всех лентах, содержимое рабочих лент (содержимое входной ленты не входит в конфигурацию) и текущее состояние. У машины M на входе x длины n может быть не более, чем $2^{cs(n)}$ различных конфигураций (тут используется, что $s(n) \geq \log n$, т.е. чтобы закодировать положение головки на входной ленте нам хватит $s(n)$ битов). Можно рассмотреть ориентированный граф конфигураций, из конфигурации K_1 есть ребро в конфигурацию K_2, если машина Тьюринга, получая на вход x, за один шаг может из конфигурации K_1 попасть в конфигурацию K_2. Машина M принимает x, если существует путь из начальной конфигурации в принимающую. Принимающих конфигураций может быть несколько, но без ограничения общности можно считать, что машина, перед тем, как закончить работу стирает все с ленты. Для задачи достижимости в графе есть полиномиальные алгоритмы, например, поиск в глубину. Т.е. детерминированно можно смоделировать работу машины M за время $2^{O(s(n))}$.

Может возникнуть вопрос, где мы использовали, что функция $s(n)$ конструктивна по памяти. На самом деле, для того, чтобы сгенерировать одну конфигурацию мы использовали, что $1^{s(n)}$ можно вычислить за время $2^{O(s(n))}$, это следует из того, что $1^{s(n)}$ вычислимо с использованием $O(s(n))$ памяти. □

Теорема 8.2 (Савич)**.** Если $s(n)$ конструктивна по памяти и $s(n) \geq \log n$, то $\mathrm{NSpace}[s(n)] \subseteq \mathrm{DSpace}[(s(n))^2]$.

Доказательство. Рассмотрим язык $L \in \mathrm{NSpace}[s(n)]$, пусть он решается недетерминированной машиной тьюринга , использующей $O(s(n))$ памяти. Как и в предыдущей теореме мы рассмотрим граф конфигураций машины M. Описание каждой конфигурации использует $O(s(n))$ битов, поэтому, мы можем перебирать все конфигурации, используя $O(s(n))$ памяти.

Докажем сначала промежуточное утверждение, а именно, покажем, что задачу достижимости в графе, в котором n вершин можно решить с использованием $O(\log^2(n))$ памяти. Затем мы применим этот результат к графу конфигураций машины Тьюринга. При этом удобно считать, что принимающая конфигурация ровно одна: для этого можно потребовать, что машина, попав в принимающее состояние, сотрет содержимое всех рабочих лент и приведет все головки в начало всех лент и только после этого примет

вход. Введем предикат $s(u, v, i)$, который истинен, если из вершины u существует путь в вершину v длины не более 2^i. Чтобы выяснить, можно ли попасть из начальной вершины a в конечную b достаточно вычислить предикат $s(a, b, n)$. Покажем, как вычислять предикат s. Наш алгоритм будет рекурсивным: чтобы вычислить $s(u, v, i)$ мы будем перебирать все z и делать два рекурсивных запуска: $s(u, z, i-1)$ и $s(z, v, i-1)$. Если найдется такая вершина z, что оба рекурсивных запуска выдают 1, то ответ будет 1, иначе 0. Покажем, как правильно распределять память, чтобы всего было использовано $(\log n)^2$ памяти. В начале ленты всегда написаны аргументы, с которыми вызывался предикат s. Когда алгоритм перебирает z, то он переиспользует память, для двух рекурсивных вызовов память тоже переиспользуется. Глубина рекурсии $O(\log n)$, каждый уровень рекурсии добавляет к используемой памяти $O(\log n)$ битов. Итого, используемая память $O((\log n)^2)$. □

Следствие 8.1. PSPACE = NPSPACE.

Итого, мы имеем, что L ⊆ NL ⊆ P ⊆ NP ⊆ NPSPACE = PSPACE.

Определим кванторную пропозициональную формулу: она имеет вид $Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, x_2, \dots, x_n)$, где φ — это пропозициональная формула от переменных $x_1, x_2, \dots, x_n$, а $Q_i \in \{\forall, \exists\}$ — кванторы. Переменные x_i принимают значения $\{0, 1\}$, истинность формулы определяется естественным образом. Обозначим $TQBF$ — это множество истинных квантовых пропозициональных формул.

Теорема 8.3. Язык $TQBF$ является PSPACE-полным.

Доказательство. Сначала мы докажем, что $TQBF \in$ PSPACE. Рассмотрим следующий рекурсивный алгоритм: пусть на вход подана формула вида $Q_1 x_1 \phi$. Сделаем рекурсивный вызов на формуле $\phi[x_1 = 0]$, затем (если значение формулы еще не определено) делаем рекурсивный вызов $\phi[x_1 = 1]$. Если на вход подана бесквантовая формула, то ее значение можно посчитать с использованием линейной памяти. Количество используемой памяти $O(n|\psi|)$, где ψ — входная формула, а n — число ее переменных.

Пусть $L \in$ PSPACE решается машиной Тьюринга M. Для каждого x можно рассмотреть граф конфигураций этой машины, каждая конфигурация описывается полиномиальным числом битов. Аналогично теореме Савича рассмотрим предикат $s(u, v, i)$, который истинен, если из вершины u существует путь в вершину v длины не более 2^i. Мы знаем рекуррентное определение предиката $s(u, v, i) = \exists z(s(u, z, i-1) \wedge s(z, v, i-1))$. Нас интересует, есть ли путь из начальной конфигурации в принимающую, т.е. значение предиката $s(K_0, K_{accept}, \log N)$, где $N = 2^{poly(|x|)}$ — общее число конфигураций. Но если развернуть это рекуррентное определение, то размер формулы будет экспоненциальным от $|x|$, так как s вызывается два раза. С помощью логического трюка можно переписать тоже самое рекуррентное определение так, чтобы s в описании встречалось бы всего один раз. Это делается вот так: $s(u, v, i) = \exists z \forall a, b(((a = u \wedge b = z) \vee (a = z \wedge b = v)) \to s(a, b, i-1))$. После разворачивания рекурсии получится формула полиномиального размера, которая использует предикат $s(u, v, 0)$, который означает, можно ли за один шаг из конфигурации u попасть в конфигурацию v, что, очевидно, может быть

записано формулой (например, аналогично доказательству теоремы Кука-Левина об NP-полноте SAT). Переменные в этой формуле описывают конфигурации, т.е. каждая такая переменная на самом деле задается полиномиальным числом пропозициональных переменных. Таким образом, мы описали сведение, которое по x выдает кванторную пропозициональную формулу. □

Пусть X — это класс языков. $\text{co}-\text{X} = \{\text{L} \mid \overline{\text{L}} \in \text{X}\}$, где $\overline{L}$ — это дополнение языка L.

Сейчас мы займемся изучением класса NL. Для этого нам понадобятся специального вида сведения, а именно логарифмические по памяти.

Определение 8.2. Мы говорим, что язык L сводится с логарифмической памятью к языку L', если существует такая вычислимая с логарифмической памятью функция f, что $x \in L \iff f(x) \in L'$. Обозначение: $L \leq_l L'$.

Предложение 8.2. 1) Если $L \leq_l L'$, то и $L \leq_p L'$; 2) Если $L_1 \leq_l L_2$ и $L_2 \leq_l L_3$, то $L_1 \leq L_3$; 3) Если $L_1 \leq_l L_2$ и $L_2 \in \text{L}$, то $L_1 \in \text{L}$.

Доказательство. Первое свойство следует из того, что если функция вычислима с использованием логарифмической памяти, то она вычислима за полиномиальное время. Оставшиеся два свойства следуют из следующей леммы. □

Лемма 8.1. Если функции f и g вычислимы с использованием логарифмической памяти, то и их композиция $f(g)$ тоже.

Доказательство. Алгоритм вычисления $f(g(x))$ такой: запустим алгоритм вычисляющий функцию f, как только ему понадобиться i-й бит входа, запустим алгоритм, вычисляющий функцию $g(x)$, и отсчитаем i-й бит на выходе, другие биты выхода хранить не будем. □

Предложение 8.3. Язык $L \in \text{NSpace}[s(n)]$ тогда и только тогда, когда существует такая машина Тьюринга M с выделенной лентой-подсказкой, головка по которой может двигаться только слева на право, M использует $O(s(n))$ памяти (лента-подсказка и вход не считается) и останавливается при любой подсказке и любом входе, что $x \in L$ тогда и только тогда, когда существует $y \in \Sigma^*$, что $M(x, y) = 1$.

Доказательство. Если $L \in \text{NSpace}[s(n)]$, то машина M будет просто моделировать недетерминированную машину, распознающую L, а подсказка будет говорить, какое правило надо применять на данном шаге.

Пусть L принимается машиной M с подсказкой. Время работы такой машины не превосходит $2^{O(s(n))}$, иначе бы машина работала бесконечно при какой-то подсказке. Тогда недетерминированная машина будет моделировать M, в случае необходимости прочитать очередной бит подсказки, этот бит будет порождаться с помощью недетерминированного выбора. □

Определим язык $DPATH = \{G, s, t \mid$ в ориентированном графе G есть путь между s и $t\}$.

Теорема 8.4. Язык $DPATH$ является полным в классе NL относительно логарифмических по памяти сведений.

Доказательство. Покажем, что $DPATH$ лежит в NL, подсказкой будет код пути в графе. Проверка происходит так: считываются две вершины, проверяется, что между ними есть ребро. Кроме того нужно проверить, что первая и последняя вершины в этом пути — это s и t.

Произвольный язык $A \in \text{NL}$ сводится к $DPATH$ так: по x строим граф конфигураций машины, стартовую конфигурацию и принимающую конфигурацию. □

Теорема 8.5. $\overline{DPATH} \in \text{NL}$, где $\overline{DPATH} = \{G, s, t \mid$ в ориентированном графе G нет пути между s и $t\}$.

Доказательство. Чтобы доказать теорему, покажем, как можно сертифицировать отсутствие пути между вершинами s и t. Напомним, что проверять такой сертификат нужно за один проход и с использованием логарифмической памяти. Обозначим через S_i множество вершин в графе, которые находятся на расстоянии не больше i от s. $S_0 = \{s\}$. Нам нужно узнать, верно ли, что $t \in S_n$, где n — это число вершин графа.

- Сертифицировать, что $v \in S_i$ просто: предъявляем список вершин на пути от s до v, проверить надо, что соседние соединены ребром и, что последняя вершина совпадает с v.

- Можно сертифицировать, что $v \notin S_i$, если известно, что $|S_i| = k$: предъявляем все вершины S_i в возрастающем порядке вместе с сертификатами их принадлежности множеству S_i. Проверить такой сертификат можно так: проверяем все сертификаты принадлежности, кроме того проверяем, что соседние вершины правильно упорядочены, и подсчитываем, что их количество равняется k. Кроме того проверяем, что v не встречается среди перечисленных вершин.

- Можно сертифицировать, что $v \notin S_i$, если известно, что $|S_{i-1}| = k$: предъявляем все вершины S_{i-1} в возрастающем порядке вместе с сертификатами их принадлежности множеству S_{i-1}. Проверить такой сертификат можно так: проверяем все сертификаты принадлежности, кроме того проверяем, что соседние вершины правильно упорядочены, и подсчитываем, что их количество равняется k. Кроме того проверяем, что v не встречается среди перечисленных вершин и нет ребра из перечисленных вершин в v.

- Можно сертифицировать размер S_i, если известен размер S_{i-1}. Для этого для каждой вершины (в порядке возрастания) предъявляется либо сертификат их принадлежности S_i, либо сертификат непринадлежности. Проверить надо, что вершины идут все в порядке возрастания, что сертификаты верные, что число принадлежащих ровно столько, сколько нужно.

- Можно сертифицировать размер S_n. Размер S_1 нам известен, он равен 1, размер S_{i+1} сертифицируется через размер S_i.

- Можно сертифицировать, что $t \notin S_n$. Сертифицируем размер S_n, а затем сертифицируем $t \notin S_n$ при известном $|S_n|$.

□

Следствие 8.2. Если $s(n)$ — конструктивная по времени и $s(n) \geq \log n$, то $\mathrm{NSpace}[s(n)] = \mathrm{co} - \mathrm{NSpace}[s(n)]$.

Доказательство. Рассмотрим язык $L \in \mathrm{NSpace}[s(n)]$, покажем, что дополнение этого языка тоже лежит в $\mathrm{NSpace}[s(n)]$. Для каждого x, $x \notin L$ тогда и только тогда, когда в графе конфигураций машины Тьюринга нет пути из начальной конфигурации в принимающую. В графе конфигураций $2^{O(s(n))}$ вершин, проверку непринадлежности по теореме можно выполнить используя $O(s(n))$ памяти на недетерминированной машине. Сам граф хранить вовсе необязательно, так как по конфигурации легко вычислить следующую. □

Следствие 8.3. $\mathrm{NL} = \mathrm{co} - \mathrm{NL}$

9 Полиномиальная иерархия

Вспомним одно из определений класса NP. А именно, класс NP состоит из языков L, для которых существует полиномиальная по времени машина Тьюринга M и полином q, что выполняется

$$x \in L \iff \exists y \in \{0,1\}^{q(|x|)} : M(x,y) = 1.$$

Перейдем в этом определении к отрицанию:

$$x \notin L \iff \forall y \in \{0,1\}^{q(|x|)} : M(x,y) = 0.$$

Теперь можно дать определение для класса coNP, он состоит из языков L, для которых существует полиномиальная по времени машина Тьюринга M' и полином q, что выполняется

$$x \in L \iff \forall y \in \{0,1\}^{q(|x|)} : M'(x,y) = 1.$$

Тут машина M' — это машина, которая соответствовала языку $\overline{L}$, ответ которой инвертирован.

Попытаемся обобщить эти два определения, разрешив использовать разные кванторы.

Определение 9.1. Для каждого $i \geq 0$ мы определим класс языков Σ_i^P и Π_i^P. $\Sigma_0^P = \Pi_0^P = \mathrm{P}$. Язык L лежит в классе Σ_{i+1}^P, если существует такой полином q и язык $B \in \Pi_i^P$, что для каждого x выполняется $x \in L$ тогда и только тогда, когда $\exists y_1 \in \{0,1\}^{q(|x|)}(x, y_1) \in B$. Аналогично язык L лежит в классе Π_{i+1}^P, если существует такой полином q и язык $B \in \Sigma_i^P$, что для каждого x выполняется $x \in L$ тогда и только тогда, когда $\forall y_1 \in \{0,1\}^{q(|x|)}(x, y_1) \in B$.

Это рекурсивное определение можно развернуть. Язык L лежит в классе Σ_k^P, если существует полиномиальная по времени машина M и полином q, что

$$x \in L \iff \exists y_1 \in \{0,1\}^{q(|x|)} \forall y_2 \in \{0,1\}^{q(|x|)} \exists y_3 \dots Q_k y_k \in \{0,1\}^{q(|x|)} : M(x, y_1, y_2, \dots, y_k) = 1.$$

$\Pi_k^P = co\Sigma_k^P$ — дополнение класса Σ_k^P. Π_k^P характеризуется условием

$$x \in L \iff \forall y_1 \in \{0,1\}^{q(|x|)} \exists y_2 \in \{0,1\}^{q(|x|)} \forall y_3 \dots Q_k y_k \in \{0,1\}^{q(|x|)} : M(x, y_1, y_2, \dots, y_k) = 1.$$

Объединение всех этих классов называется полиномиальной иерархией $\mathrm{PH} = \bigcup_i \Sigma_i^P$, а классы Σ_i^P и Π_i^P называется уровнями полиномиальной иерархии.

Свойства полиномиальной иерархии:

1. $\mathrm{NP} = \Sigma_1^{\mathrm{P}}, \mathrm{coNP} = \Pi_1^{\mathrm{P}}$;
2. $\Sigma_i^P \cup \Pi_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$.

   Это свойство доказывается добавлением фиктивных кванторов, аналогично тому, как это доказывалось для арифметической иерархии.
3. $\mathrm{PH} = \bigcup_{\mathrm{i}} \Pi_{\mathrm{i}}^{\mathrm{P}}$

   Следует из предыдущего свойства.
4. Если при некотором $i \geq 1$ выполняется $\Sigma_i^P = \Pi_i^P$, то $\mathrm{PH} = \Sigma_i^P$.

   Докажем по индукции по $k \geq i$, что $\Sigma_k^P = \Pi_k^P = \Sigma_i^P$. База индукции $k = i$. Индукционный переход. Рассмотрим язык L из Σ_{k+1}^P, по определению, существует язык $L' \in \Pi_k^P$ и полином q, что $x \in L \iff \exists y_1 \in \{0,1\}^{q(|x|)}(x, y_1) \in L'$. По индукционному предположению $\Pi_k^P = \Sigma_k^P = \Sigma_i^P$, следовательно $L' \in \Sigma_i P$. Сдедовательно существует такая полиномиальная машина M и полином p, что $z \in L'$ тогда и только тогда, когда $\exists u_1 \in \{0,1\}^{p(|z|)} \forall u_2 \in \{0,1\}^{p(|z|)} \dots Q_i u_i \in \{0,1\}^{p(|z|)} M(z, u_1, \dots, u_i)$. Тогда $x \in L \iff \exists y_1 \in \{0,1\}^{q(|x|)} \exists u_1 \in \{0,1\}^{p(|(x,y_1)|)} \forall u_2 \in \{0,1\}^{p(|(x,y_1)|)} \dots Q_i u_i \in \{0,1\}^{p(|(x,y_1)|)} M(x, y_1, u_1, \dots, u_i)$, следовательно $L \in \Sigma_i^P$. Аналогично разбирается случай $L \in \Pi_i^P$.
5. Если $\mathrm{P} = \mathrm{NP}$, то $\mathrm{PH} = \mathrm{P}$.

   Если $\mathrm{P} = \mathrm{NP}$, то $\mathrm{P} = \mathrm{NP} = \mathrm{coNP} = \Sigma_1^{\mathrm{P}} = \Pi_1^{\mathrm{P}}$, по предыдущему свойству $\mathrm{PH} = \Sigma_1^{\mathrm{P}} = \mathrm{P}$.
6. Σ_i^P и Π_i^P замкнуты относительно сведений по Карпу.

   Доказывается аналогично замкнутости уровней арифметической иерархии относительно m-сведений.
7. Если в PH есть полный язык относительно сведений по Карпу, то $\mathrm{PH} = \Sigma_{\mathrm{k}}$ для некоторого k.

   Пусть L — полный язык в PH, пусть $L \in \Sigma_k$, тогда по предыдущему свойству $\mathrm{PH} \subseteq \Sigma_{\mathrm{k}}$.

Определим язык $\Sigma_i SAT$, который состоит из истинных кванторных пропозициональных формул вида $\exists u_1 \forall u_2 \exists u_3 \dots Q_i u_i \varphi(u_1, u_2, \dots, u_i)$, где u_i — это вектор пропозициональных переменных, а φ — это пропозициональная формула. Аналогично можно определить язык $\Pi_i SAT$.

Теорема 9.1. Язык $\Sigma_i SAT$ является полным в классе Σ_i^P, а $\Pi_i SAT$ полный в классе Π_i^P.

Доказательство. Покажем, что $\Sigma_i SAT$ лежит в Σ_i^P. Для каждой φ, $\varphi \in \Sigma_i SAT$ тогда и только тогда, когда $\exists y_1 \in \{0,1\}^{|\varphi|} \forall y_2 \in \{0,1\}^{|\varphi|} \dots Q_i y_i P(\varphi, y_1, y_2, \dots, y_i)$, где предикат $P(\varphi, y_1, y_2, \dots, y_i)$ равен нулю, если φ не является формулой, в которой i чередований кванторов, начиная с $\exists$, в противном случае равен значению формулы φ, если для каждого $j = 1 \dots i$ в j-й блок ее переменных подставить значения из начала вектора y_j. Очевидно, что предикат P проверяется за полиномиальное время.

Рассмотрим язык $L \in \Sigma_i^P$.

$$x \in L \iff \exists y_1 \in \{0,1\}^{q(|x|)} \forall y_2 \in \{0,1\}^{q(|x|)} \exists y_3 \dots Q_i y_i \in \{0,1\}^{q(|x|)} : M(x, y_1, y_2, \dots, y_i) = 1.$$

По лемме 2.1 за полиномиальное от $|x|$ время можно построить схему C, которая на входах длины $|(x, y_1, \dots, y_i)|$ будет совпадать с машиной M. Из доказательства следствия 4.1 мы знаем, что по схеме $C(z)$ можно за полиномиальное от $|C|$ время построить формулу $\phi(t, z)$, что для всех z значение $C(z)$ совпадает со значением $\exists t \phi(t, z)$ (новые переменные t соответсвуют всем вершинам схемы, отличным от входа).

Зафиксируем длину входа $|x| = n$. Значение $\exists y_1 \in \{0,1\}^{q(|x|)} \forall y_2 \in \{0,1\}^{q(|x|)} \exists y_3 \dots Q_i y_i \in \{0,1\}^{q(|x|)} : M(x, y_1, y_2, \dots, y_i) = 1$ совпадает со значением $\exists y_1 \in \{0,1\}^{q(|x|)} \forall y_2 \in \{0,1\}^{q(|x|)} \exists y_3 \dots Q_i y_i \in \{0,1\}^{q(|x|)} C(x, y_1, y_2, \dots, y_i)$. Рассмотрим два случая в зависимости от четности i. Если i нечетно, то $Q_i = \exists$, в этом случае значение последнего выражения совпадает с $\exists y_1 \in \{0,1\}^{q(|x|)} \forall y_2 \in \{0,1\}^{q(|x|)} \exists y_3 \dots \exists y_i \in \{0,1\}^{q(|x|)} \exists t \phi(t, x, y_1, y_2, \dots, y_i)$, а это Σ_i-формула. Если i четно, то описанный выше метод получит Σ_{i+1} формулу. Поэтому мы рассмотрим схему $C' = \neg C$ и представим $C'(z)$ в виде $\exists t \psi(t, z)$ для пропозициональной формулы ψ. Тогда значение $C(z)$ совпадает со значением $\forall t \neg \psi(t, z)$ и мы можем поступить аналогично предыдущему случаю для такого представления схемы C. □

9.1 Оракульное определение полиномиальной иерархии

Теорема 9.2. При $i \geq 2$ выполняется $\Sigma_i^P = \mathrm{NP}^{\Sigma_{i-1} SAT}$.

Доказательство. Докажем включение $\Sigma_i^P \subseteq \mathrm{NP}^{\Sigma_{i-1} SAT}$. Рассмотрим язык $L \in \Sigma_i^P$, по определению существует такой полином q и язык $L' \in \Pi_{i-1}^P$, что $x \in L \iff \exists y \in \{0,1\}^{q(|x|)} (x, y) \in L'$. L' сводится к $\Pi_{i-1} SAT$. Заметим, что отрицание Π_k формул — это Σ_k формула, значит с оракулом $\Sigma_{i-1} SAT$ можно проверять принадлежность и языку $\Pi_{i-1} SAT$. Недетерминированный полиномиальный по времени алгоритм с оракулом $\Sigma_{i-1} SAT$ такой: угадать y, свести (x, y) к $\Pi_{i-1} SAT$ и узнать ответ у оракула.

Докажем обратное включение. Пусть $L \in \mathrm{NP}^{\Sigma_{i-1}SAT}$ и L решается оракульной недетерминированной машиной M за время $p(n)$. Строкой z длины $2p(n)$ можно закодировать последовательность детерминированных выборов принимающего вычисления машины M вместе с ответами оракула (каждому шагу соответствует два бита: первый показывает, запрос ли это к оракулу или недетерминированный выбор, второй шаг соответствует либо ответу оракула, либо недетерминированному выбору). Чтобы проверить, что строка z, действительно, задает принимающее вычисление машины, нужно проверить:

- По входу x длины n и строке z длины $2p(n)$ нужно проверить, что если машина M работает на входе x и использует переходы в соответствии со строкой z, то обращения к оракулу происходят тоже в соответствии со строкой z. И при указанных в строке z ответах оракула, машина M приходит в принимающее состояние. Это можно проверить за полиномиальное время, пусть это проверяется предикатом R.

- Нужно проверить, что все ответы оракула, действительно, правильные. Это делается различным образом в случае ответов 1 и в случае ответов 0.

  - В случае ответа 1, мы пишем такое выражение $\exists y_1 \in \{0,1\}^{p(n)^2} \forall y_2 \in \{0,1\}^{p(n)^2} \dots Q_{i-1} y_{i-1} \in \{0,1\}^{p(n)^2} T(x, z, y_1, \dots, y_{i-1}) = 1$, где предикат T вычисляет (моделированием) все формулы $\phi_1, \phi_2, \dots, \phi_k$, которые соответствуют запросам оракула с ответом 1. Всего запросов $k \leq p(n)$. Все строчки y_j разобъем на $p(n)$ частей по $p(n)$ битов, j-я часть будет использоваться для ϕ_j. Считаем значения всех формул ϕ_j так: значения из начала j-го блока y_1 подставляем первому блоку переменных ϕ_j, второму блоку значения из начала j-й части y_2 и т.д. Значение предиката равняется 1, если значения всех формул равняется 1, иначе значение предиката 0. которое принимает формула ϕ. Если так получилось, что строка ϕ_j не задает формулу, то T выдает 0.
  - В случае ответа 0, мы пишем такое выражение $\forall f_1 \in \{0,1\}^{p(n)^2} \exists f_2 \in \{0,1\}^{p(n)^2} \dots Q_{i-1} f_{i-1} \in \{0,1\}^{p(n)^2} T'(x, z, f_1, \dots, f_{i-1}) = 1$, где предикат T' отличается от T тем, что он выдает 1 только в том случае, когда значения всех формул равны нулю.

Теперь мы готовы описать Σ_i-характеризацию языка L. $x \in L$ тогда и только тогда, когда $\exists z(R(x, z) \wedge (\exists y_1 \forall y_2 \dots T(x, z, y_1, \dots, y_{i-1} = 1) \wedge \forall f_1 \exists f_2 \dots T'(x, z, f_1, \dots, f_{i-1}))$. Последнее условие доказывает, что $L \in \Sigma_i^P$.

□

Если X — это класс языков, то можно определить $\mathrm{NP}^{\mathrm{X}} = \cup_{L \in \mathrm{X}} \mathrm{NP}^L$. Если C — полный язык в X, то $\mathrm{NP}^{\mathrm{X}} = \mathrm{NP}^C$.

Поскольку $\Sigma_{i-1}SAT$ — это полный язык в Σ_{i-1}^P, то часто предыдущую теорему формулируют как $\Sigma_i^P = \mathrm{NP}^{\Sigma_{i-1}^P}$, в частности $\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}}$, $\Sigma_3^P = \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}}$ и т.д.

10 Схемная сложность

Теорема 10.1 (Карп-Липтон)**.** Если NP $\subseteq$ P/poly, то PH $= \Sigma_2^{\mathrm{P}}$.

Доказательство. Поскольку NP $\subseteq$ P/poly, то существует полином q и семейство формул C_n размера $q(n)$, которые решают язык SAT. Более того, существует семейство схем полиномиального размера, которое вычисляет выполняющий набор для каждой формулы. Такое семейство можно построить из полиномиального алгоритма, который с оракулом SAT находит выполняющий набор. Этот алгоритм надо переделать в схему и все обращения к оракулу заменить на схемы, решающие SAT. Пусть размер этих схем не превосходит полинома $r(n)$.

Покажем, что при предположении NP $\subseteq$ P/poly, полный язык в классе Π_2^P лежит в Σ_2^P. Полный язык в классе Π_2^P состоит из истинных формул вида $\forall x \in \{0,1\}^n \exists y \in \{0,1\}^n \varphi(x,y)$, где φ — пропозициональная формула, которая, возможно использует не все биты x и y.

Истинность формулы $\psi = \forall x \in \{0,1\}^n \exists y \in \{0,1\}^n \varphi(x,y)$ равносильно тому, что $\exists C_1, C_2, \dots C_{|\varphi|} \in \{0,1\}^{r(|\psi|)} \forall a \in \{0,1\}^{|\psi|} \varphi(a, C_{\varphi_a}(\varphi_a)) = 1$, где φ_a — это формула φ, в которую вместо x подставили значения из вектора a. Если ψ истинна, то в качестве C_i нужно взять семейство схем, которые ищут выполняющий набор. А если ψ ложно, то найдется такое a, что выполняющего набора у φ_a нет, значит и никакая схема его не найдет. □

Теорема 10.2. Существует булева функция $\{0,1\}^n \to \{0,1\}$, которая не вычислима ни одной схемой размера $2^n/(10n)$.

Доказательство. Схему размера $T(n)$ можно задать с помощью не более, чем $4T(n)\log T(n)$ битов. Действительно, каждая вершина схемы кодируется своим номером размера $\log T(n)$ битов, операцией и номерами вершин, из которых идут ребра в эту вершину.

Итого, схем размера не более, чем $2^n/(10n)$ менее, чем $2^{2^n/2}$, а всего различных булевых функций хотя бы 2^{2^n}. □

Теорема 10.3 (Каннан)**.** Для всех k выполняется PH $\not\subseteq$ SIZE$[n^k]$.

Доказательство. Из предыдущей теоремы следует, что булеву функцию схемной сложности больше, чем n^k можно найти среди функций $\{0,1\}^{(k+1)\log n} \to \{0,1\}$. Другими словами существует булева функция, зависящая только от первых $(k+1)\log n$ переменных, схемная сложность которой больше, чем n^k. Такую функцию можно задать в виде таблицы истинности размера $O(n^{k+1})$.

Зададим язык L, который на входах каждой длины будет решаться самой первой функцией, зависящей только от первых $(k+1)\log n$ переменных, чья схемная сложность строго больше n^k. $x \in L$ тогда и только тогда, когда $\forall f(\forall C : |C| \le n^k \exists y \in \{0,1\}^n C(x) \ne f(x) \wedge \forall g((g \le f) \to \exists C |C| \le n^k \forall y \in \{0,1\}^n C(y) = g(y)) \to f(x) = 1$. В этой формуле f и g выбираются среди функций из $\{0,1\}^n \to \{0,1\}$, которые зависят от первых $(k+1)\log n$

битов, представляются эти функции с помощью таблицы истинности для первых $(k+1)\log n$ переменных, сравниваются функции лексикографически.

Таким образом, мы получили, что $L \in$ PH. Можно определить и точный уровень полиномиальной иерархии, если вынести кванторы вперед. □

Следствие 10.1. Для всех k выполняется $\Sigma_2^P \cap \Pi_2^P \not\subseteq \mathrm{SIZE}[n^k]$.

Доказательство. Пусть $\Sigma_2^P \cap \Pi_2^P \subseteq \mathrm{SIZE}[n^k]$, тогда $\mathrm{NP} \subseteq \Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}} \subseteq \mathrm{P/poly}$. Тогда по теореме Карпа-Липтона $\mathrm{PH} = \Sigma_2^{\mathrm{P}} = \Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}} \subseteq \mathrm{SIZE}[\mathrm{n}^{\mathrm{k}}]$, что противоречет теореме Каннана. □

Можно определить понятие машины Тьюринга с неравномерной подсказкой. Имеется машина Тьюринга M и последовательность строк α_n. Когда длина входа равняется n, то машина может использовать строку α_n, которая дается ей на отдельной ленты. Если изменить строку α_n, то машина может вести себя совсем иначе. Говорят, что язык L лежит в классе $\mathrm{P}/a(n)$, если существует такая машина Тьюринга M и последовательность строк α_n, что при всех n выполняется неравенство $|\alpha_n| \leq a(n)$, что машина M, используя подсказки α_n принимает язык L за полиномиальное время.

Предложение 10.1. $\mathrm{P/poly} = \bigcup_{\mathrm{k}} \mathrm{P}/n^k$.

Доказательство. Пусть $L \in \mathrm{P/poly}$, тогда пусть в качестве подсказки α_n будет подаваться описание схемы C_n, решающей язык L, а машина будет подставлять в схему вход. Обратно, если язык решается машиной с неравномерной подсказкой полиномиальной длины, то по лемме 2.1 можно построить схему, которая будет давать тот же самый ответ, входами этой схемы будет вход машины и подсказка, вместо подсказки мы подставим значение этой подсказки. □

11 Вероятностные классы сложности

Вероятностную машину Тьюринга можно определить двумя эквивалентными способами. В первом определении вероятностная машины Тьюринга — это детерминированная машина, в которой есть специальная лента, по которой головка может двигаться только слева направо, как только головка машины передвигается в новую ячейку, в этой ячейке появляется бит, распределенный равномерно и независимо от предыдущих. Во втором определении вероятностная машина Тьюринга, как и недетерминированная, имеет две функции перехода δ_0 и δ_1. Каждый раз с вероятностью $\frac{1}{2}$ применяется δ_0 и с вероятностью $\frac{1}{2}$ применяется δ_1. Под временем работы вероятностной машины Тьюринга мы понимаем ее время работы в наихудшем случае, т.е. максимум по всем возможным последовательностям случайных битов.

Класс BPP (класс языков, решаемых за полиномиальное время вероятностными алгоритмами с ограниченной ошибкой) состоит из языков L, для которых существует вероятностная полиномиальная по времени машина M, что при всех x выполняется $\Pr[M(x) = L(x)] \geq \frac{2}{3}$. Это условие можно расписать подробнее: если $x \in L$, то $\Pr[M(x) =$

$1] \geq \frac{2}{3}$, а если $x \notin L$, то $\Pr[M(x) = 0] \geq \frac{2}{3}$. Таким образом алгоритм может ошибаться, как при $x \in L$, так и при $x \notin L$.

Класс RP состоит из языков L, для которых существует вероятностная полиномиальная по времени машина M, что при $x \in L$ выполняется $\Pr[M(x) = 1] \geq \frac{1}{2}$, а если $x \notin L$, то $\Pr[M(x) = 0] = 1$. Т.е. алгоритм может ошибаться только на элементах языка, на элементах не из языка он не ошибается.

Нетрудно видеть, что $\mathrm{P} \subseteq \mathrm{RP} \subseteq \mathrm{BPP}$. Для доказательства включения $\mathrm{RP} \subseteq \mathrm{BPP}$ нужно в RP-алгоритме понизить ошибку, для этого нужно повторить его два раза и выдать максимум из двух его ответов. Также выполняется включение $\mathrm{RP} \subseteq \mathrm{NP}$, поскольку случайные биты RP-алгоритма могут служить NP-подсказкой.

Лемма 11.1 (Шварц-Зиппель)**.** Если многочлен $p(x_1, x_2, \ldots, x_\ell)$ над конечным полем $\mathbb{F}$ ненулевой степени $\leq d$, тогда

$$\Pr_{a_1,\ldots,a_l \leftarrow F}[p(a_1, a_2, \ldots, a_\ell) \neq 0] \geq 1 - \frac{d}{|\mathbb{F}|}.$$

Доказательство. Доказательство по индукции по l. База $l = 1$ является известным утверждением. Разложим многочлен по степеням x_1: $p(x_1, \ldots, x_\ell) = \sum_{i=0}^{d} x_1^i p_i(x_2, \ldots, x_\ell)$. Пусть k наибольшее число, что $p_k \neq 0$. Поскольку степень многочлена p не превосходит d, то $\deg p_k \leq d - k$. По индукционному предположению для многочена p_k

$$\Pr_{a_2,\ldots,a_l \leftarrow F}[p_k(a_2, \ldots, a_\ell) \neq 0] \geq 1 - \frac{d-k}{|\mathbb{F}|}.$$

В случае, если $p_k(a_2, \ldots, a_\ell) \neq 0$, то $p(x_1, a_2, \ldots, a_k)$ имеет не более k корней. Тогда $\Pr[p(a_1 \ldots a_l) \neq 0] \geq (1 - \frac{k}{|\mathbb{F}|})(1 - \frac{d-k}{|\mathbb{F}|}) \geq 1 - \frac{d}{|\mathbb{F}|}$ □

Следствие 11.1. Язык ненулевых многочленов, записанных в виде формулы или схемы, содержится в классе RP.

Доказательство. Достаточно вычислять значения этих многочленов в случайной точке поля. □

Предложение 11.1 (Оценки Чернова в аддитивной форме)**.** Для независимых и одинаково распределенных случайных величин $X_1, X_2, \ldots, X_n$, таких что $X_i \in \{0, 1\}$ и $E[X_i] = \mu$ при всех $1 \leq i \leq n$, для всех положительных ϵ выполняется $\Pr\left[|\frac{\sum_{i=1}^{n} X_i}{n} - \mu| \geq \delta\right] \leq 2e^{-2\delta^2 n}$.

Доказательство. Введем обозначение $\overline{X}_i = X_i - \mu$.

$$\Pr[\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \geq \delta] = \Pr[\frac{1}{n}\sum_{i=1}^{n} \overline{X}_i \geq \delta] = \Pr[e^{t\sum_{i=1}^{n} \overline{X}_i} \geq e^{t\delta n}] \leq \frac{\mathrm{E}[e^{t\sum_{i=1}^{n} \overline{X}_i}]}{e^{t\delta n}}$$

Поскольку X_i взаимно независимые, то $\mathrm{E}[e^{t\sum_{i=1}^{n} \overline{X}_i}] = \prod_{i=1}^{n} \mathrm{E}[e^{t\overline{X}_i}]$.

$$\mathrm{E}[e^{t\overline{X}_i}] = \mu e^{t(1-\mu)} + (1-\mu)e^{t(0-\mu)} \le e^{t^2/8},$$

последнее неравенство следует из разложения экспоненты в ряд Тейлора.

Получаем $\Pr[\frac{1}{n}\sum_{i=1}^n X_i - \mu \ge \delta] \le \frac{(e^{t^2/8})^n}{e^{t\delta n}} = e^{(t^2/8 - t\delta)n}$. Подставим $t = 4\delta$ (и аналогично оценим второй способ раскрытия модуля):
$\Pr[|\frac{1}{n}\sum_{i=1}^n X_i - \mu| \ge \delta] \le 2e^{-2\delta^2 n}$. □

Доказательство оценки Чернова не приводилось на лекции. Вместо этого на лекциях был рассказан неформальный комментарий, почему убывание экспоненциальное. Рассмотрим пример, когда все X_i принимают значения 0 и 1 с равной вероятностью. Покажем, что вероятность того, что $\frac{1}{n}\sum_{i=1}^n X_i \ge \frac{2n}{3}$, экспоненциально маленькая. Рассмотрим случайную величину $X_1X_2\dots X_n$ значения этой величины — это строки из 0 и 1 длины n. Рассмотрим набор случайных величин $Y_1, Y_2, \dots, Y_n$, каждая из которых независимо от остальных принимает 1 с вероятностью $\frac{2}{3}$, а 0 с вероятностью $\frac{1}{3}$. Рассмотрим строку s, в которой не менее, чем $\frac{2}{3}n$ единиц. Случайная величина $X_1X_2\dots X_n$ равняется этой строке с вероятностью $\frac{1}{2^n}$. Случайная величина $Y_1Y_2\dots Y_n$ равняется s с вероятностью как минимум $(\frac{2}{3})^{\frac{2}{3}n}(\frac{1}{3})^{\frac{1}{3}n} = (\frac{\sqrt[3]{4}}{3})^n$. Поскольку $\frac{1}{2} < \frac{\sqrt[3]{4}}{3}$, то вероятность строки s при распределении $X_1X_2\dots X_n$ как минимум в $\left(\frac{1}{c}\right)^n$ раз меньше, чем вероятность этой строки при распределении $Y_1Y_2\dots Y_n$, где $c = \frac{3}{2\sqrt[3]{4}} < 1$. Пусть $w(s)$ обозначает число единиц в строке s.

Запишем $\Pr[\frac{1}{n}\sum_{i=1}^n X_i \ge \frac{2n}{3}] = \sum_{s:w(s)\ge\frac{2}{3}n}\Pr[X_1X_2\dots X_n = s] < c^n\sum_{s:w(s)\ge\frac{2}{3}n}\Pr[Y_1Y_2\dots Y_n = s] < c^n$.

Теорема 11.1. Повторением можно добиться понижения вероятности ошибки в алгоритме, решающем язык из класса BPP до 2^{-N}, где $N = poly(n)$.

Доказательство. Запустим $m = 18N+1$ копий машины M независимо с независимыми случайными битами и выдадим наиболее частый ответ. Пусть случайная величина X_i равна 1, если был дан верный ответ и 0 иначе. $E[X_i] = \mu \ge \frac{2}{3}$. Вероятность ошибки нового алгоритма $\Pr\left[\sum_{i=1}^m X_i \le \frac{1}{2}m\right] \le \Pr\left[|\frac{\sum_{i=1}^m X_i}{m} - \mu| \ge \frac{1}{6}\right] \le 2e^{-\frac{2}{36}m} < 2^{-N}$.

□

Теорема 11.2 (Адлеман)**.** BPP $\subseteq$ P/poly

Доказательство. Пусть $L \in$ BPP, существует вероятностная машина M, решающая L, вероятность ошибки которой понижена до 2^{-n^2}. На входе длины n машина использует не более $p(n)$ случайных битов. Назовем строку случайных битов плохой, если машина с этой строкой дает неправильный хотя бы для одного входа длины n. Всего плохих строк не более $2^{p(n)-n^2+n} < 2^{p(n)}$, значит есть строка случайных битов, которая подходит для всех срок длины n. Эту строку можно использовать как неравномерную подсказку. □

Теорема 11.3 (Сипсер-Гакс)**.** BPP $\subseteq \Pi_2^{\mathrm{P}} \cap \Sigma_2^{\mathrm{P}}$.

Доказательство. Достаточно доказать, что $\mathrm{BPP} \subseteq \Sigma_2^{\mathrm{P}}$, теорема будет из этого следовать, так как BPP замкнут относительно операции дополнения.

Пусть $L \in \mathrm{BPP}$, существует машина M, которая решает L с ошибкой 2^{-n}, где n — длина входа. Пусть машина M на входах длины n использует $m(n)$ случайных битов, где m — некоторый полином. Для каждого входа $x \in \{0,1\}^n$ существует множество $S_x \subseteq \{0,1\}^m$ — множество случайных битов, на которых машина M выдает ответ 1. При $x \in L$ выполняется $|S_x| \geq 2^m(1-2^{-n})$, а при $x \notin L$ $|S_x| < 2^{m-n}$.

Покажем, что эти два случая различаются тем, что в первом случае множество $\{0,1\}^m$ можно покрыть k копиями множества S_x, а во втором случае нельзя, где $k = \lceil \frac{m}{n} \rceil + 1$.

Для $A \subseteq \{0,1\}^m$ и $a \in \{0,1\}^m$ обозначим $a + A = \{a + x \mid x \in A\}$. Покажем, что если $|S_x| < 2^{m-n}$, то не найдется таких строк $s_1, s_2, \ldots, s_k \in \{0,1\}^m$, что $\bigcup_{i=1}^k S_x + s_i = \{0,1\}^m$.

Это следует из того, что $|\bigcup_{i=1}^k S_x + s_i| < k|S_x| < 2^{m-n}k < 2^m$ при достаточно больших n.

А если $|S_x| \geq 2^m(1-2^{-n})$, то найдутся такие строки $s_1, s_2, \ldots, s_k \in \{0,1\}^m$, что $\bigcup_{i=1}^k S_x + s_i = \{0,1\}^m$. Действительно, выберем $s_1, s_2, \ldots, s_k \leftarrow \{0,1\}^m$ независимо случайным образом и оценим вероятность того, что как минимум один элемент S будет не покрыт. Вероятность того, что элемент $t \in \{0,1\}^m$ будет не покрыт множеством $s_1 + S_x$ не превосходит 2^{-n}, поскольку эта вероятность равняется вероятности того, что $s_1 + t \notin S_x$. Все k событий: t не покрыт множеством $s_i + S_x$ являются независимыми, поэтому вероятность того, что t не покрыт $\bigcup_{i=1}^k s_i + S$ не превосходит 2^{-nk} Вероятность того, что хотя бы один такой элемент из $\{0,1\}^m$ будет не покрыт, не превосходит $2^m 2^{-nk} < 1$. Т.е. с положительной вероятностью все элементы будут покрыты. Значит, существуют такие $s_1, s_2, \ldots, s_k \in \{0,1\}^m$, что $\bigcup_{i=1}^k S_x + s_i = \{0,1\}^m$.

Теперь мы можем записать

$$x \in L \iff \exists s_1, s_2, \ldots, s_k \in \{0,1\}^m \forall r \in \{0,1\}^m r \in \bigcup_{i=1}^k S_x + s_i$$

Условие $r \in \bigcup_{i=1}^k S_x + s_i$ можно переписать как дизъюнкцию условий $\bigvee_{i=1}^k r + s_i \in S_x$. Заметим, что $r + s_i \in S_x$ тогда и только тогда, когда машина M, используя вместо случайных битов строку $r + s_i$, на входе x выдает 1 (это можно кратко записать $M_{r+s_i}(x) = 1$). Теперь можно переписать условие в форме, которая докажет принадлежность $L \in \Sigma_2^P$:

$$x \in L \iff \exists s_1, s_2, \ldots, s_k \forall r \in \{0,1\}^m \bigvee_{i=1}^k M(x, r + s_i) = 1$$

□

12 Интерактивные протоколы

Рассмотрим пару функция P (доказывающий) и алгоритм V (проверяющий), которые обладают следующими свойствами. V получает на вход три строки s_1, s_2, s_3 и выдает либо строку, либо элемент множества $\{\mathbf{1}, \mathbf{0}\}$. Функция P по паре строк выдает третью строку. Цель доказывающего убедить проверяющего, что строка x принадлежит L. Интерактивное взаимодействие происходит следующим образом: сначала проверяющий вычисляет $V(x, r, \sharp) = s_1$, где x — это строка, для которой доказывающий хочет убедить проверяющего, что она принадлежит языку L, а r — это строка случайных битов, и посылает результат доказывающему. Затем доказывающий вычисляет $P(x, s_1) = s_2$ и посылает результат проверяющему. Далее проверяющий вычисляет $V(x, r, \sharp s_1 \sharp s_2) = s_3$ и т.д. Работа протокола заканчивается, как только проверяющий выдал ответ $\mathbf{1}$ или $\mathbf{0}$. На время вычисления функции P у нас нет никаких ограничений, доказывающий обладает неограниченными возможностями, проверяющий алгоритм V должен быть полиномиальный по времени. Результат работы протокола обозначим $<V, P>(x)$.

Определение 12.1. Пусть $k(n)$ — это некоторая функция. Язык L принадлежит классу IP$[k(n)]$, если существует такой полиномиальный от длины первого параметра алгоритм V и функция P:

- Для любого x и для любой функции P' в протоколе взаимодействия P' и V на входе x не более $k(|x|)$ посылок сообщений.
- Для любого $x \notin L$ для любой функции P' выполняется $\Pr[out_{P,V}(x) = 0] \geq \frac{2}{3}$.
- Для любого $x \in L$, $\Pr[<V, P>(x) = 1] \geq \frac{2}{3}$.

Класс IP $= \bigcup_{c>0} IP[n^c]$.

Пример 12.1. *Язык GNI, состоящий из пар неизоморфных графов на одинаковом числе вершин, лежит в $IP[2]$.*

Доказательство. На вход подана пара графов (G_0, G_1), в каждом графе n вершин Проверяющий генерирует случайное $i \leftarrow \{0, 1\}$ и случайную перестановку $\pi \leftarrow S_n$. После этого перестановка π применяется к графу G_i и полученный граф $\pi(G_i)$ посылается доказывающему. Проверяющий ожидает получить от доказывающего один бит. Пусть доказывающий посылает проверяющему $j \in \{0, 1\}$. Если $i = j$, то проверяющий выдает $\mathbf{1}$, иначе $\mathbf{0}$.

Если графы были неизоморфны, то честный доказывающий может понять, какому из двух графов G_0 или G_1 изоморфен присланный и прислать правильный ответ. Пусть теперь графы изоморфны. Пусть доказывающий по присланному графу выдает ответ, но в случае изоморфных графов, присланный граф с равной вероятностью может быть перестановкой G_0 и перестановкой G_1. Значит, доказывающий угадывает с вероятностью ровно $\frac{1}{2}$, с этой же вероятностью проверяющий выдаст $\mathbf{1}$. Чтобы понизить вероятность ошибки, надо повторить этот протокол два раза. Заметим, что это можно сделать за один раунд, сразу генерировать i_0, i_1 и π_0, π_1 и ожидать от доказывающего, что он выдаст правильный ответ в обоих случаях. □

Теорема 12.1 (Шамир)**.** IP = PSPACE.

Доказательство. Докажем сначала простое включение: IP $\subseteq$ PSPACE. Пусть $L \in$ IP, а V — это соответствующий проверяющий алгоритм. Поскольку V работает полиномиальное от x время, то длины всех сообщений можно считать ограниченными полиномом $q(n)$, где $n = |x|$. Не умаляя общности можно считать, что число раундов взаимодействия равняется $t(n)$, где t — некоторый полином и на каждом раунде посылается сообщение длины ровно $q(n)$. В таком случае мы можем представить работу интерактивного протокола в виде дерева T глубины $t(n)$, каждая вершина дерева (кроме листов) имеет $2^{q(n)}$ потомков, которые соответствуют различным сообщениям, которые могут на очередном шагу послать проверяющий или доказывающий. На каждом пути от корня до листа нечетные ребра соответствуют сообщениям проверяющего, а четные ребра сообщениям доказывающего. Будем считать, что в самом низу дерева есть разветвление по итоговому ответу проверяющего: **1** или **0** и листы дерева помечены решениями проверяющего. Не по всем путям этого дерева протокол в реальности может работать, поскольку некоторые сообщения проверяющий может послать с вероятностью ноль. Для каждого листа дерева, помеченного **1** посчитаем вероятность того, что проверяющий своими решениями придет в этот лист, если доказывающий будет посылать сообщения, которые согласуются с путем из корня T_P в этот лист. Заметим, что если путь от корня до листа задан, то вероятность этого пути можно посчитать с использованием полиномиальной памяти: просто перебираем все случайные биты и проверяем, пойдем ли мы вдоль данного пути.

Каждая конкретная функция доказывающего P задает поддерево T_P этого дерева: в каждой вершине, в которой осуществляется ход доказывающего, остается ровно один потомок. Для каждого доказывающего P вероятность того, что P убедит V на входе x равняется сумме вероятностей листов дерева T_p, помеченными **1**. Покажем, что с использованием полиномиальной памяти можно посчитать максимальную (по всем возможным доказывающим) вероятность, с которой доказывающий сможет убедить проверяющего. Обозначим через p_v максимальную вероятность события, что во время работы протокола мы побывали в вершине v и при этом протокол выдал **1**, которую может обеспечить доказывающий. Нам требуется посчитать p_r, где r — это корень дерева T. Как считать эту вероятность для листьев мы уже описали выше. Вероятность p_v удовлетворяет такими рекуррентными соотношениями: для вершины v, соответствующей ходу проверяющего $p_v = \sum_u p_u$, где суммирование ведется по непосредственным потомкам вершины v в дереве T. Если вершина v соответствует ходу доказывающего, то $p_v = \max_u p_u$, где вершина u пробегает по всем непосредственным потомкам вершины v в дереве T. По этим рекуррентным формулам вычислить значение в корне дерева можно с помощью рекурсивного алгоритма, который использует полиномиальную память. Память этого алгоритма получается полиномиальной за счет полиномиальной глубины рекурсии и того, что в каждом листе вероятность можно посчитать с использованием полиномиальной памяти.

Теперь докажем сложное включение PSPACE $\subseteq$ IP. Для доказательства нам достаточно показать, что $TQBF \in$ IP. Язык $TQBF$ состоит из истинных формул вида

$Q_1x_1Q_2x_2\dots Q_nx_n\varphi(x_1,\dots,x_n)$, где Q_i — это кванторы, а φ — это пропозициональная формула. Пусть $|\varphi| = m$, не умоляя общности можно считать, что $n \le m$.

Сейчас мы опишем способ как представлять формулу в виде многочлена. Многочлены мы будем рассматривать над полем $\mathbb{F}$ из p элементов, где $p > m^4$, но интересоваться его значениями мы будем только для $0/1$ переменных. Строить многочлен мы будем индуктивно:

- пропозициональной переменной x соответствует многочлен $\widetilde{x}$
- формуле $\varphi \wedge \psi$ соответствует многочлен $\widetilde{\varphi} \cdot \widetilde{\psi}$
- формуле $\varphi \vee \psi$ соответствует многочлен $1 - (1 - \widetilde{\varphi}) \cdot (1 - \widetilde{\psi})$
- формуле $\neg\varphi$ соответствует многочлен $1 - \widetilde{\varphi}$

При этом раскрывать скобки мы не будем, тогда длина записи многочлена не сильно больше длины записи самой формулы. Про построенный многочлен можно утверждать, что его значения на $0/1$ переменных совпадает со значением исходной пропозициональной формулы.

Кванторы мы будем представлять в виде операторов над многочленами. Квантор всеобщности мы будем заменять на такой оператор $A\,x_iP(x_1,\dots,x_n)$, который обозначает многочлен $P(x_1,\dots,0,\dots x_n)P(x_1,\dots,1,\dots,x_n)$. Квантор существования будем заменять на оператор $E\,x_iP(x_1,\dots,x_n)$, который обозначает многочлен $1-(1-P(x_1,\dots,0,\dots x_n))(1-P(x_1,\dots,1,\dots,x_n))$.

Дополнительно нам понадобится оператор линеаризации, который понижает степень многочлена по переменной до 1. Формально он определяется так: $L\,x_iP(x_1,\dots,x_n)$ обозначет многочлен $(1-x_i)P(x_1,\dots,0,\dots x_n)+x_iP(x_1,\dots,1,\dots,x_n)$. Заметим, что оператор линеаризации не меняет значение многочлена при $0/1$ переменных.

Соответственно, формулу $Q_1x_1Q_2x_2\dots Q_nx_n\varphi(x_1,\dots,x_n)$ мы будем представлять в виде многочлена $\widetilde{Q_1}x_1\,L\,x_1\widetilde{Q_2}x_2\,L\,x_1\,L\,x_2\widetilde{Q_3}x_3\dots\widetilde{Q_n}\,L\,x_1\dots L\,x_n\widetilde{\varphi}$, где $\widetilde{Q_i} = A$, если Q_i — это квантор всеобщности и $\widetilde{Q_i} = E$, если Q_i — это квантор существования.

Нетрудно видеть, что степень многочлена $\widetilde{\varphi}$ по каждой переменной не превосходит m. Благодаря операторам линеаризации мы получаем, что все многочлены, которые получаются, отбрасыванием нескольких первых операторов имеют степени по каждой переменной не превосходящие m.

Сейчас мы опишем интерактивный протокол, с помощью которого честный доказывающий сможет убедить проверяющего, что формула истинна, потом мы проверим, что если формула ложна, то вероятность убедить проверяющего в обратном очень маленькая.

Изначально мы хотим доказать, что многочлен тождественно равен 1. Мы будем один за одним снимать операторы из начала формулы. После снятия кванторов A или E, соответствующей переменной будет присваиваться какое-то значение из поля $\mathbb{F}$. Наш протокол будет рекурсивный, на каждом шаге рекурсии будет сниматься один из операторов. Опишем общую ситуацию: у многочлена сняты первые ℓ кванторов, при этом из

них k операторов A или E и переменным $x_1, \ldots, x_k$ присвоили значения $r_1, r_2, \ldots, r_k \in \mathbb{F}$. Доказывающему нужно убедить проверяющего, что значение текущего многочлена равняется c. Посмотрим на последний не снятый оператор. Разберем три случая.

1. Последний оператор $A\,x_1$. В таком случае проверяющий попросит доказывающего прислать многочлен $h(x_1)$, который задает многочлен, если у него снять оператор A и подставить значения остальных переменных. Проверяющий проверяет, что $h(0)h(1) = c$, если это не так, то проверяющий отвергает.

2. Последний оператор $E\,x_1$. В таком случае проверяющий попросит доказывающего прислать многочлен $h(x_1)$, который задает многочлен, если у него снять оператор E и подставить значения остальных переменных. Проверяющий проверяет, что $1 - (1 - h(0))(1 - h(1)) = c$, если это не так, то проверяющий отвергает.

3. Последний оператор $L\,x_1$. Это возможно, только если переменной x_1 уже задано какое-то значение r_1. В таком случае проверяющий попросит доказывающего прислать многочлен $h(x_1)$, который задает многочлен, если у него снять оператор L и подставить значения остальных переменных. Проверяющий проверяет, что $(1 - r_1)h(0) + r_1h(1) = c$, если это не так, то проверяющий отвергает.

Далее проверяющий снимает последний оператор, выбирает случайное значение $r \leftarrow \mathbb{F}$, подставляет его значение переменной x_1 (даже если у этой переменной раньше было другое значение, что могло быть в случае L-оператора). И запускает рекурсивно протокол для значения $h(r)$.

Заметим, что степень многочлена h не превосходит m. Поэтому если многочлен h, присланный доказывающим, отличался от истинного многочлена, то он совпадает с ним не более, чем в m точках, т.е. вероятность того, что в следующем раунде будет доказываться верное утверждение, не превосходит $\frac{m}{|\mathbb{F}|} < \frac{1}{m^3}$.

В самом конце протокола все кванторы сняты и у всех переменных есть значение, остается только их подставить и проверить значение многочлена с заявленным, это проверяющий может выполнить сам без помощи доказывающего.

Итак, если исходная формула была истинной, то честный доказывающий может всегда честно выдавать запрашиваемые многочлены и убедить проверяющего. Покажем, что если формула была ложной, то вероятность того, что принимающий выдаст $\mathbf{1}$, будет маленькой. Если формула ложна, то первое утверждение, которое доказывалось в протоколе, было ложным. Если все утверждения во всех рекурсивных вызовах были бы ложными, то на последнем шаге, когда операторов не осталось, эта ложь была бы обнаружена. Значит, в какой-то момент утверждение из ложного стало истинным. Всего у нас было не более, чем m^2 операторов, как мы отмечали, вероятность смены утверждения с ложного на истинное при каждом снятии оператора не превосходит $\frac{1}{m^3}$, поэтому вероятность перехода ложь-истина не превосходит $\frac{1}{m}$. □

Следствие 12.1. Из доказательства следует, что если мы наложим на класс IP такие ограничения, то класс не изменится:

1. В определении класса IP можно потребовать, что если $x \in L$, то $\Pr[<V, P>(x) = 1] = 1$.

2. В определении класса IP достаточно рассматривать доказывающих, которые вычисляются алгоритмами, использующими полиномиальную память. Доказывающему нужно уметь выдавать коэффициенты многочлена h, с помощью интерполяции это можно делать, если уметь считать значения h в каждой точке. Вычислять значения многочлена h можно с помощью рекурсивного алгоритма, который реализуется на полиномиальной памяти.

3. В случае, если число раундов полиномиальное, то проверяющий может не скрывать от доказывающего случайные биты, которые он использовал для генерации очередного сообщения.

13 Вероятностно проверяемые доказательства

13.1 Примеры приближенных алгоритмов

Задача $MAX3SAT$ состоит в том, чтобы по формуле в 3-КНФ найти набор, который выполняет максимальное возможное число дизъюнктов. Эта задача является NP-трудной, поскольку NP-трудной является задача проверки того, можно ли выполнить все дизъюнкты.

Будем говорить, что алгоритм решает задачу $MAX3SAT$ с приближением ρ, если он выдает набор, который выполняет как минимум ρm дизъюнктов, которые можно выполнить.

В случае, если требовать, что в каждом дизъюнкте встречаются 3 различных переменных, то можно построить вероятностный $\frac{7}{8}$-приближенный алгоритм. Пусть формула содержит m дизъюнктов, рассмотрим случайный набор значений переменных. Пусть X_i — это случайная величина, которая равна 1, если i-й дизъюнкт выполнен данным набором. $X = \sum_{i=1}^{m} X_i$ — случайная величина, которая говорит, сколько дизъюнктов выполнено данным набором. $\Pr[X_i = 0] = \frac{1}{8}$, так как ровно одно из восьми возможных значений переменных, входящих в i-й дизъюнкт, не выполняет его. Значит $\Pr[X_i = 1] = \frac{7}{8}$, отсюда $\mathrm{E}[X_i] = \frac{7}{8}$. По линейности $\mathrm{E}[X] = \frac{7}{8}m$. Значит, существует такой набор значений переменных, который выполнит не менее, чем $\frac{7}{8}m$ дизъюнктов. С помощью неравенства Маркова можно доказать, что если запускать этот алгоритм несколько раз, то нужный набор встретиться с большой вероятностью.

Рассмотрим задачу о минимальном вершинном покрытии. Вершинным покрытием графа называется такое множество вершин, что любое ребро смежно как минимум с одной вершиной этого множества. Известно, что язык $\{(G, k) \mid$ в графе G есть вершинное покрытие размере не более $k\}$ является NP-полным. Задача $MinVertexCover$ состоит в нахождении покрытия минимального размера. Рассмотрим следующий $\frac{1}{2}$-приближенный алгоритм для задачи $MinVertexCover$: начать с пустого множество, каждый раз находить непокрытое ребро и добавлять в множество оба его

конца. Построенное таким образом множество будет не более, чем в два раза больше минимального, поскольку хотя бы одну из вершин всех рассматриваемых ребер покрыть надо.

13.2 Вероятностно проверяемые доказательства

Определение 13.1. Будем говорить, что язык L содержится в классе $\mathrm{PCP}(r(n), q(n))$, если существует такой полиномиальный вероятностный алгоритм V, который имеет оракульный доступ к доказательству длины не больше, чем $2^{O(r(n))}q(n)$ и обладает следующими свойствами:

1. На входе x алгоритм V использует $O(r(n))$ случайных битов и делает $O(q(n))$ неадаптивных запросов к доказательству.
2. Если $x \in L$, то существует такое доказательство π, что $\Pr[V^\pi(x) = 1] = 1$.
3. Еслим $x \notin L$, то для любой строки π выполняется $\Pr[V^\pi(x) = 1] \leq \frac{1}{2}$.

Покажем в качестве примера, что $GNI \in \mathrm{PCP}(poly(n), 1)$. Корректное доказательство для пары графов G_0, G_1 на n вершинах состоит из строки размера $2^{\frac{n(n-1)}{2}}$, каждый бит этой строки соответствует графу на n вершинах, в корректном доказательстве на i-ой позиции стоит ноль, если i-й граф изоморфен G_0, 1, если i-й бит изоморфен G_1 и 0 в противном случае. Проверяющий алгоритм V выглядит следующим образом:

- Выбрать случайное $i \leftarrow \{0, 1\}$
- Выбрать случайную перестановку n-элементного множества σ
- Запросить бит доказательства, который соответствует графу $\sigma(G)$.
- Если этот бит равен i, то выдать 1, иначе выдать 0.

Если графы были неизоморфны, то V примет корректное доказательство с вероятностью 1. Если графы изоморфны, то граф $\sigma(G)$ мог быть с равной вероятностью образом G_0 и G_1, следовательно V принимает с вероятностью $\frac{1}{2}$.

Теорема 13.1 (PCP-теорема)**.** $\mathrm{NP} = \mathrm{PCP}(\log n, 1)$.

Полностью мы эту теорему не докажем. Докажем просто включение.

Доказательство $\mathrm{PCP}(\log n, 1) \subseteq \mathrm{NP}$. Рассмотрим $L \in \mathrm{PCP}(\log n, 1)$, для этого языка существует проверяющий алгоритм V, который использует $O(\log n)$ случайных битов. Если $x \in L$, то существует строка π размера $poly(n)$, которую V принимает с вероятностью 1, а если $x \notin L$, то любая строка π отвергается с вероятностью как минимум $\frac{1}{2}$. Рассмотрим алгоритм $A(x, \pi)$, который запускает V, давая ему использовать в качестве случайных битов все возможные строчки (таких строк $poly(n)$), а в качестве доказательства будет выступать строка π. Алгоритм A принимает, если V принимает на всех строках случайных битов и отвергает иначе. Время работы алгоритма A ограничено полиномом, следовательно $L \in \mathrm{NP}$. □

13.3 CSP

Мы рассмотрим некоторое обобщение задачи SAT, а именно задачу выполнения ограничений (CSP). CSP задачей арности q называется набор функций $f_1, f_2, \ldots, f_m : \{0,1\}^n \to \{0,1\}$, каждая из которых зависит не более, чем от q входных битов. Это значит, что каждую функцию можно задать строкой длины 2^q. Будем говорить, что набор переменных $x \in \{0,1\}^n$ выполняет CSP, если для всех $i \in \{1,\ldots,m\}$ выполняется $f_i(x) = 1$. Функции f_i называются ограничениями. Язык $qCSP$ состоит из выполнимых задач арности q. Поскольку $qSAT$ является частным случаем $qCSP$, то $qCSP$ является NP-полным.

Задача $MAXqCSP$ состоит в поиске набора, который выполняет максимальное возможное число ограничений. Для задачи ϕ обозначим максимальную долю ограничений, которые можно одновременно выполнить через $val(\phi)$. Для каждой константы $0 < \rho < 1$ Рассмотрим задачу $\rho GAPqCSP$, которая состоит в том, чтобы различать задачи ϕ у которых $val(\phi) = 1$ и задачи ϕ у которых $val(\phi) < \rho$.

Будем говорить, что задача $\rho GAPqCSP$ является NP-трудной, если для любого языка $L \in$ NP найдется такой полиномиальный алгоритм A, что для любого $x \in L$ выполняется $val(A(x)) = 1$, а если $x \notin L$, то $val(A(x)) < \rho$.

Теорема 13.2. PCP-теорема эквивалентна тому, что для некоторой константы q и для некоторой константы $0 < \rho < 1$ задача $\rho GAPqCSP$ является NP-трудной.

Доказательство. Пусть выполняется PCP-теорема, тогда для любого языка из NP существует проверяющий алгоритм V, который делает q запросов к доказательству и использует $c \log n$ случайных битов. Построим по строке x CSP задачу следующим образом. Переменными в этой задаче будут биты доказательства π, ограничений будет n^c и они соответствуют каждой последовательности случайных битов. Для каждой последовательности случайных битов функция ограничения зависит только от q битов, к которым делал запрос алгоритм V и ее ответ совпадает с ответом алгоритма V, который бы он сделал, получив данные q битов в качестве доказательства. Если $x \in L$, то можно выполнить все ограничения, если $x \notin L$, то одновременно выполняются не более половины ограничений. Таким образом задача $\frac{1}{2}GAPqCSP$ является NP-трудной.

Пусть задача $\rho GAPqCSP$ является NP-трудной. Опишем вероятностно проверяемую систему доказательств для произвольного языка $L \in$ NP. Пусть x сводится к CSP задаче ϕ, в которой m ограничений. Доказательством принадлежности $x \in L$ будет выполняющий набор, проверяться этот выполняющий набор будет следующим образом: случайно выбирается ограничение и запрашиваются значения q переменных, от которых это ограничение зависит. Принимаем, если это ограничение выполнилось, отвергаем в противном случае. Если $x \in L$, то доказательство будет принято с вероятностью 1, а если $x \notin L$, то доказательство будет принято с вероятностью меньше, чем ρ. Можно повторить проверку $O(1)$ раз, чтобы добиться вероятности ошибки меньше $\frac{1}{2}$. □

Следствие 13.1. Для некоторого ρ задача $\rho GAP3SAT$ является NP-трудной.

Доказательство. Доказательство остается в качестве упражнения. □

Предложение 13.1. Существует такая константа $\rho < 1$, что из существования полиномиального ρ-приближенного алгоритма для MAX3SAT следует, что P = NP.

На самом деле это утверждение известно для $\rho = \frac{7}{8} + \epsilon$ для произвольного $\epsilon > 0$.

Доказательство. Пусть число ρ такое, что задача $\rho GAP3SAT$ является NP-трудной. Пусть есть полиномиальный ρ-приближенный алгоритм A для задачи $MAX3SAT$. Тогда, если ϕ выполнима, то он выдаст набор, который удовлетворяет хотя бы долю ρ дизъюнктов. Если $val(\phi) < \rho$, то любой набор выполняет строго меньше, чем ρ дизъюнктов, поэтому с помощью алгоритма A можно решить задачу $\rho GAP3SAT$, которая является NP-трудной. □

13.4 Коды Уолша-Адамара

Для двух строк $x, y \in \{0,1\}^n$ внутренним (или скалярным) произведением будем называть $x \cdot y = \sum_{i=1}^{n} x_i y_i \bmod 2$.

Кодом Уолша-Адамара для строки x называется строка $WH(x) = (x \cdot y)_{y \in \{0,1\}^n}$. Иными словами мы умножаем строку x на все возможные строки y и выписываем ответ. Таким образом, $WH : \{0,1\}^n \to \{0,1\}^{2^n}$.

- Функция WH является линейной, т.е. $WH(x+y) = WH(x) + WH(y)$.
- Если $x \neq 0^n$, то в строке $WH(x)$ половина нулей, половина единиц. Действительно, если $x \neq 0^n$, то найдется такой индекс i, что $x_i = 1$, все строчки $y \in \{0,1\}^n$ можно разбить на пары, которые отличаются только в i-м бите, на одном элементе пары произведение с x будет 1, а на другом — 0.
- Если $x \neq y$, то расстояние Хемминга между $WH(x)$ и $WH(y)$ равняется 2^{n-1}. Действительно, расстояние Хемминга между строчками $WH(x)$ и $WH(y)$ равняется числу единиц в $WH(x) + WH(y) = WH(x+y)$, но $x + y \neq 0^n$, поэтому в строке $WH(x+y)$ ровно 2^{n-1} единица.
- Коды Уолша-Адамара — это таблицы истинности линейных функций $f : \{0,1\}^n \to \{0,1\}$. Это следует из того, что любая линейная функция $\{0,1\}^n \to \{0,1\}$ задается вектор-строкой.
- Пусть функция $\tilde{f}$ отличается от линейной функции f на доле $\alpha < \frac{1}{4}$ входов. Допустим у нас есть оракульный доступ к функции $\tilde{f}$. Покажем, как можно с хорошей вероятностью восстановить значение функции f в любой точке x. Выберем случайную строку $r \leftarrow \{0,1\}^n$ и посчитаем сумму $\tilde{f}(r) + \tilde{f}(x+r)$. Вероятность того, что $\tilde{f}(r) \neq f(r)$ или $\tilde{f}(x+r) \neq f(x+r)$ не превосходит $2\alpha < \frac{1}{2}$. Значит с вероятностью хотя бы $1 - 2\alpha > \frac{1}{2}$ выполняется равенство $f(x) = \tilde{f}(r) + \tilde{f}(x+r)$. Эту вероятность можно увеличить с помощью оценок Чернова, если делать этот тест несколько раз и выдавать самый частый ответ. Это свойство кода Уолша-Адамара называется локальным декодированием. Если есть кодовое слово, испорченное не

более чем в доле $\alpha < \frac{1}{4}$ позиций, то каждый символ правильного кода можно восстановить с хорошей вероятностью, делая при этом константное число запросов к битам кодового слова.

- Будем говорить, что две функции $f, g : \{0,1\}^n \to \{0,1\}$ являются ρ-похожими для константы $\rho \leq 1$, если $\Pr_{x \leftarrow \{0,1\}^n}[f(x) = g(x)] \geq \rho$.

- Оказывается, что можно тестировать, является ли функция 0.99-похожей на линейную, сделав только константное число запросов к ней. Это следует из следующей теоремы, которую мы докажем позже.

  Теорема 13.3. Если $\Pr_{x,y \leftarrow \{0,1\}^n}[f(x+y) = f(x) + f(y)] \geq \rho$, то f является ρ-похожей на линейную функцию.

  Действительно, построим тест, который делает только константное число запросов к функции, примет линейную функцию с вероятностью 1 и отвергнет не 0.99-похожую на линейную функцию с большой вероятностью. Если функция не является 0.99-похожей на линейную, то $\Pr_{x,y \leftarrow \{0,1\}^n}[f(x+y) \neq f(x)+f(y)] > \frac{1}{100}$. Повторим 1000 раз: выберем случайные x и y и проверим, что $f(x+y) = f(x)+f(y)$, если хотя бы один раз это равенство не выполняется, то отвергнуть, в противном случае принять. Если функция линейная, то она принимается с вероятностью 1. Если функция нелинейная, то отвергается с вероятностью хотя бы $1 - \left(1 - \frac{1}{100}\right)^{100} \geq 1 - \frac{1}{e^{10}}$.

13.5 $\mathrm{NP} \subseteq \mathrm{PCP}(poly(n), 1)$

Мы будем рассматривать системы квадратных уравнений над полем $\mathbb{F}_2$. Система состоит из уравнений $f(x_1, \ldots, x_n) = 0$, где f — это многочлен степени 2. Язык QE состоит из совместных систем квадратных уравнений.

Предложение 13.2. QE — это NP-полный язык.

Доказательство. Проще всего задачу $CIRCUIT{-}SAT$ свести к QE. Заведем переменные для входов и для гейтов схемы. И напишем такие уравнения:

- Для гейта выхода: $V^2 - 1 = 0$;
- Для гейта отрицания $X = \neg Y$: $X^2 + Y^2 - 1 = 0$
- Для гейта конъюнкции $Z = X \wedge Y$: $Z^2 - XY = 0$
- Для гейта дизъюнкции $Z = X \vee Y$: $Z^2 + X^2 + Y^2 + XY = 0$

□

Можно считать, что уравнения в системах квадратных уравнений не содержат мономов степени 1, поскольку над полем из $\mathbb{F}_2$ выполняется равенство $x^2 = x$.

Мы докажем, что $QE \in \mathrm{PCP}(poly(n), 1)$, из этого будет следовать, что $\mathrm{NP} \subseteq \mathrm{PCP}(poly(n), 1)$, поскольку можно сначала применить сведение языка из NP к L, затем воспользоваться проверять доказательство для задачи QE.

Тензорным произведением векторов $x = (x_1, x_2, \ldots, x_n)$ и $y = (y_1, y_2, \ldots, y_m)$ называется вектор $x \otimes y = (x_1y_1, \ldots, x_1y_m, x_2y_1, \ldots, x_2y_m, \ldots, x_ny_1 \ldots x_ny_m)$.

Систему из m квадратных уравнений от n неизвестных можно записать в матричном виде $Ax \otimes x = b$, где A — это матрица $m \times n^2$, x — вектор неизвестных размера n, а b — вектор размерности m.

Доказательством выполнимости система $Ax \otimes x = b$ будет являться пара, состоящая из кода Уолша-Адамара от x и от $x \otimes x$. Формально доказательство — это строка размера $2^n + 2^{n^2}$. Будем считать, что первые 2^n бит этой строки — это таблица истинности функции $f : \{0,1\}^n \to \{0,1\}$, а последние 2^{n^2} битов — это таблица истинности $g : \{0,1\}^{n^2} \to \{0,1\}$. В правильном доказательстве функция f задает код Уолша-Адамара x, а g задает код Уолша-Адамара $x \otimes x$.

Покажем, что такое доказательство можно проверить с хорошей вероятностью за полиномиальное от n время, делая лишь константное число запросов к функциям f и g. Проверка будет состоять из множества тестов, если хотя бы один тест провален, то мы отвергаем доказательство и принимаем, если все тесты пройдены. Все тесты будут иметь некоторую маленькую вероятность ошибки, но суммарная вероятность ошибки будет меньше $\frac{1}{10}$.

1. Тест на линейность. Проведем тест, который с вероятностью 10^{-3} отвергнет функции, которые не являются 0.999-похожими на линейные. После того, как этот тест пройден, мы будем считать, что f и g линейные функции, а когда мы хотим вычислить их значение в какой-то точке, то будем применять свойство локального декодирования и будем точно знать значение функций f и g с вероятностью 0.999, делая лишь константное число запросов к функции.

2. Тест на согласованность. Если $f : \{0,1\}^n \to \{0,1\}$ — линейная функция, то f является кодом Уолша-Адамара для некоторой строки x. Если $g : \{0,1\}^{n^2} \to \{0,1\}$ — линейная функция, то g является кодом Уолша-Адамара для некоторой строки y. Согласованность f и g означает, что $x \otimes x = y$.

   Тест состоит в том, чтобы 10 раз выполнить проверку: для случаных векторов r и r' проверить, что $f(r)f(r') = g(r \otimes r')$.

   Проверим, что для согласованных f и g эта проверка должна пройти. Действительно, $f(r)f(r') = \sum_i x_i r_i \sum_j x_j r'_j = \sum_{i,j} x_i x_j r_i r'_j = g(r \otimes r')$.

   Пусть f и g не являются согласованными, f является кодом Уолша-Адамара строки x, а g является кодом Адамара строки $y \neq x \otimes x$. Произведение $f(r)f(r')$ равняется $r^T U r'$, где матрица $U = x^t x$. А $g(r \otimes r') = r^T W r'$, где матрица W отличается от матрицы U.

С вероятностью как минимум $\frac{1}{2}$ вектор Wr' отличается от вектора Ur'. Значит, с вероятностью как минимум $\frac{1}{4}$ $r^T Ur'$ отличается от $r^T Wr'$. Так как тест повторяется 10 раз, то вероятность выявить несогласованность как минимум $1 - \left(\frac{3}{4}\right)^{10}$.

3. Проверка того, что g — это код Уолша-Адамара от правильного решения линейной системы $Ay = b$.

   Пусть $A_1, A_2, \ldots, A_m$ — строки матрицы A. Функция g задает код Уолша-Адамара системы $Ay = b$, если для всех i от 1 до m выполняется $g(A_i) = b_i$. Но мы можем сделать только константное число запросов к функции g, поэтому поступим иначе. Рассмотрим случайную строку $r \leftarrow \{0,1\}^m$. Если $Ay = b$, то $r^T Ay = r^T b$. Если же $Ay \neq b$, то с вероятностью $\frac{1}{2}$ выполняется $r^T Ay \neq r^T b$ (причина такая же, как и у того, что код Уолша-Адамара имеет расстояние $\frac{1}{2}$). Повторив это 10 раз можно добиться вероятности ошибки $\frac{1}{2^{10}}$.

13.6 Тестирование функции на линейность

Наша ближайшая цель доказать теорему 13.3.

Перейдем к $-1/1$ переменных с помощью замены $0 \mapsto 1$, $1 \mapsto -1$. Сложение после такой замены перейдет в умножение. Линейные функции перейдут в мультипликативные, т.е. для всех $x, y \in \{-1,1\}^n$ выполняется $f(xy) = f(x)f(y)$, где xy — это покоординатное произведение векторов. Каждая функция задается таблицей значений, т.е. вектором из $\{-1,1\}^{2^n} \subseteq \mathbb{R}^{2^n}$. Для функций $f, g : \{-1,1\}^n \to \mathbb{R}$ определим скалярное произведение $<f,g> = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x)g(x)$. Нормирующий множитель $\frac{1}{2^n}$ нужен для того, чтобы скалярный квадрат функции из $\{-1,1\}^n \to \{-1,1\}$ был равен единице.

Для каждого $I \subseteq \{1,2,\ldots,n\}$ определим функцию $\chi_I : \{-1,1\}^n \to \mathbb{R}$ следующим образом: $\chi_I(x) = \prod_{i \in I} x_i$.

Убедимся, что множество функций χ_I задают ортонормированный базис (этот базис называют базисом Фурье): Для этого посчитаем скалярные произведения. $<\chi_I, \chi_I> = 1$, при $I \neq J$ выполняется $<\chi_I, \chi_J> = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} \prod_{i \in I} x_i \cdot \prod_{j \in J} x_j = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} \prod_{i \in I \oplus J} x_i = 0$, где $I \oplus J$ обозначает симметрическую разность множеств I и J.

Каждая функция χ_I является мультипликативной, а в старых переменных линейной.

В терминах скалярного произведения можно выразить похожесть функций. Если f и g являются $\left(\frac{1}{2} + \epsilon\right)$-похожими тогда и только тогда, когда $<f,g> \geq 2\epsilon$.

Для функции f будем обозначать $\hat{f}_I$ — коэффициент Фурье, соответствующий элементу базиса χ_I, т.е. $f = \sum_{I \subseteq \{1,\ldots,n\}} \hat{f}_I \chi_I$. Число $\hat{f}_I = <f, \chi_I>$ соответствует похожести функции f на мультипликативную.

Теорема 13.3 следует из следующей леммы:

Лемма 13.1. Если $\Pr_{x,y \leftarrow \{-1,1\}^n}[f(xy) = f(x)f(y)] \geq \frac{1}{2} + \epsilon$, то существует $I \subseteq \{1,2,\ldots,n\}$, что $\hat{f}_I \geq 2\epsilon$.

Доказательство. Из условия леммы следует, что

$$2\epsilon \le \frac{1}{2^{2n}} \sum_{x,y} f(xy)f(x)f(y) =$$

$$\frac{1}{2^{2n}} \sum_{x,y} \sum_{I} \chi_I(xy)\hat{f}_I \sum_{J} \chi_J(x)\hat{f}_J \sum_{K} \chi_I(y)\hat{f}_I =$$

$$\sum_{I}\sum_{J}\sum_{K} \hat{f}_I\hat{f}_J\hat{f}_K \frac{1}{2^n} \sum_{x} \chi_I(x)\chi_J(x) \sum_{y} \chi_I(y)\chi_J(y) =$$

$$\sum_{I} \hat{f}_I{}^3 \le \max_I \hat{f}_I \sum_I \hat{f}_I^2 = \max_I \hat{f}_I.$$

□