Обзорный курс по теоретической информатике (ЧЕРНОВИК краткого конспекта*лекций)

Д.М. Ицыксон †

26 октября 2020 г.

Аннотация

Этот текст содержит основные определения, утверждения и доказательства из курса. Неформальные объяснения, мотивация и примеры в этот текст скорее всего не попадут.

Содержание

1 Существование доказательств и перечислимые языки **2**

2 Короткие доказательства, класс NP **4**
- 2.1 Сводимость и NP-полнота . . . 6
- 2.2 Машины Тьюринга . . . 8
  - 2.2.1 Одноленточная машина Тьюринга . . . 8
  - 2.2.2 Многоленточные машины Тьюринга . . . 9
- 2.3 Теорема Кука-Левина . . . 10
- 2.4 Сведение задач поиска к задачам распознавания . . . 10

3 Вычисления с ограничением по памяти **12**
- 3.1 Вычисления с константной памятью . . . 13
- 3.2 Вычисления с логарифмической памятью . . . 19

4 Параллельные вычисления **22**
- 4.1 Параллельный алгоритм для достижимости . . . 22

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

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

1 Существование доказательств и перечислимые языки

Алфавитом называется конечное множество, элементы которого мы называем символами. Если Σ — это конечный алфавит, то Σ^* обозначает множество всех конечных строк над этим алфавитом.

Важнейшим понятием, про которое мы будем говорить, — это понятие алгоритма. Точное математическое определение этого понятия (машина Тьюринга) будет дано позже, для начала достаточно интуитивного понимания, что такое алгоритм. Вместо того, чтобы давать определения, мы просто примем некоторые соглашения, что все интуитивно представляли одно и то же понятие.

- Алгоритм получает на вход строку над каким-то алфавитом. Алгоритм может остановиться и не остановиться. Если алгоритм остановился, то результатом его работы является некоторая строка. Если алгоритм не останавливается, то его работа тоже может быть осмысленной, к примеру, он может выводить какую-то бесконечную последовательность.

- Каждому алгоритму соответствует строка, текст этого алгоритма, которая этот алгоритм полностью определяет.

- Алгоритм можно исполнять по шагам.

Кроме этого нам понадобиться понятие сложности алгоритма. На первых порах нам хватит только сложности по времени. Временной сложностью алгоритма на данном входе мы называем число шагов, которое нужно сделать для того, чтобы выдать результат. С понятием временной сложности не все так гладко, как кажется, поскольку не все шаги могут быть равноценными, и надо как-то различать долгие операции от быстрых. Мы будем считать, что наше определение понятия шага устроено разумно, к примеру для перемножения целых чисел используется число шагов, пропорциональное суммарной длине этих чисел. Точное определение будет дано позже.

Определение 1.1. 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) Если существует такая строка w, что $\Pi(x, w) = 1$, то $x \in L$.

Определение 1.2. Язык L называется разрешимым, если существует такой алгоритм A, что $A(x) = 1$ для всех $x \in L$ и $A(x) = 0$ при $x \notin L$.

Для разрешимых языков существует система доказательств с пустым доказательством.

Определение 1.3. Язык L называется полуразрешимым, если существует такой алгоритм A, что при $x \in L$ алгоритм $A(x)$ останавливается и выдает 1, а при $x \notin L$ алгоритм A не останавливается.

Определение 1.4. Язык L называется перечислимым, если существует такой алгоритм A, который работает бесконечно долго на пустом входе и печатает на выход все элементы языка L без повторений через запятую.

Теорема 1.1. Для языка L существует система доказательств тогда и только тогда, когда L перечислим, тогда и только тогда, когда L полуразрешим.

Доказательство. Пусть язык L имеет систему доказательств, докажем, что L перечислим. Пусть Π система доказательств для L. Рассмотрим бесконечную в две сторны клетчатую четверть плоскости, клетки по горизонтали и по вертикали пронумерованы строчками. Поставим в клетку с координатами (x, y) крестик, если $\Pi(x, y) = 1$. Нам надо перечислить координаты всех столбцов, в которых есть хотя бы один крестик. Будем обходить все клетки по диагоналям и проверять, есть ли там крестик, запускаем алгоритм Π, и если крестик есть, то выводить соответствующую координату. Избавиться от повторений можно, например, просматривая уже выведенные строки.

Пусть L перечислим. Тогда полуразрешающий алгоритм устроен так: запускаем перечисляющий и если появится строка x, то остановиться и выдать 1.

Пусть L полуразрешим. Тогда систему доказательств Π можно определить так $\Pi(x, n) = 1$, если полуразрешающий алгоритм останавливается на входе x за n шагов. □

Теорема 1.2 (Пост)**.** Если язык и его дополнение перечислимы, то язык разрешим.

Доказательство. Запустим параллельно полуразрешающие алгоритмы для языка и его дополнения. Какой остановится, такой ответ и выдать. □

Пусть A — это алгоритм, тогда $\sharp A$ будет обозначать текст алгоритма A. И наоборот, если s — некоторая строка, то $<s>$ обозначает алгоритм, который эта строка задает. Если строка s не задает никакого алгоритма, то будем считать, что $<s>$ обозначает алгоритм, который ничего не делает.

Будем называть универсальным алгоритмом такой алгоритм, который по тексту алгоритма и входу исполняет этот алгоритм на этом входе. Иными словами $U(s, x)$ запускает $<s>(x)$. В программировании универсальные алгоритмы называют интерпретаторами.

Рассмотрим язык $H = \{(s, x) \mid < s > (x) \text{ останавливается}\}$ (Halting problem, задача об остановке алгоритма). Очевидно, что H полуразрешим, следовательно перечислим.

Существуют ли неразрешимые языки? Да, конечно, поскольку алгоритмов счетное число, а языков несчетное. Аналогично можно доказать, что существует неперечислимый язык. А как доказать, что существует неразрешимый перечислимый язык? Мы покажем, что H — это такой язык. Мы даже докажем более сильное утверждение, что язык $W = \{s \mid <s>(s) \text{ останавливается}\}$ является неразрешимым. В качестве упражнения докажите, что из неразрешимости W следует неразрешимость H.

Теорема 1.3. Язык W не является разрешимым.

Доказательство. Допустим противное, пусть язык W является разрешимым. Тогда язык $\overline{W} = \{s \mid <s>(s)$ не останавливается$\}$ тоже является разрешимым, следовательно полуразрешимый. Пусть A полуразрешающий алгоритм для языка $\overline{W}$. $\sharp A$ — это текст алгоритма A. Рассмотрим два случая:

1. Пусть $\sharp A \in W$, тогда $A(\sharp A)$ останавливается, следовательно (поскольку A полуразрешает $\overline{W}$) $\sharp A \in \overline{W}$. В этом случае получается противоречие.

2. Пусть $\sharp A \notin W$, тогда $A(\sharp A)$ не останавливается, следовательно (поскольку A полуразрешает $\overline{W}$) $\sharp A \notin \overline{W}$. И в этом случае получается противоречие.

Таким образом, язык W не является разрешимым. □

2 Короткие доказательства, класс NP

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

В дальнейшем мы будем считать, что системы доказательств эффективны.

Определение 2.1. 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 $p(|x|)$, 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.

Лемма 2.1 (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 $n-1 = p_1 p_2 \ldots p_k$ 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 $p_1 p_2 \ldots p_k = n-1$ и сертифицировать простоту чисел $p_1, p_2, \ldots, p_k$. 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.

Определение 2.2. Язык L называется разрешимым за полиномиальное время (или принадлежит классу P), если существует такой алгоритм A, что $A(x) = 1$ для всех $x \in L$ и $A(x) = 0$ при $x \notin L$ и существует такой полином p, что для каждого входа x время работы $A(x)$ ограничено $p(|x|)$.

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

2.1 Сводимость и NP-полнота

Определение 2.3. 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 2.1.

- Если $A \leq_p B$ и $B \leq_p C$, то $A \leq_p C$.
- Если $A \leq_p B$ и $B \in$ P, то и $A \in$ 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.

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 2.2. $BH \in$ 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 2.1. Для любого языка $L \in$ 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 $x \in L$. □

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

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.

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

Чуть позже мы докажем, что задача BH сводится к $Circuit - SAT$, следовательно $Circuit - SAT$ является NP-полной. Пока мы это утверждение примем на веру.

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

Теорема 2.2. (Кук, Левин) SAT и $3SAT$ являются NP-полными

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

Доказательство. 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. □

2.2 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.

2.2.1 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 2.1. 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.

2.2.2 Многоленточные машины Тьюринга

k-ленточная машина Тьюринга содержит k лент, на каждой 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.

2.3 Теорема Кука-Левина

Наша ближайшая цель доказать теорему Кука-Левина об NP-полноте задачи SAT. С учетом того, что мы сделали нам достаточно свести задачу BH к задаче $Circuit-SAT$.

Мы покажем, что все что можно сделать машиной Тьюринга, можно сделать и схемой, причем размер схемы будет связан полиномом с временем работы машины Тьюринга. Это мы формализуем в следующей лемме.

Лемма 2.2. Существует алгоритм A, который получает на вход T, n и M, который работает время полиномиальное относительно n, T и $|M|$, что если детерминированная машина Тьюринга M на всех входах длины n работает не больше T шагов и она выдает ответ из множества $\{0,1\}$, то результат работы алгоритма A — это схема 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. □

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 2.3. 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 2.2. 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 $C_x \in Circuit - SAT$. □

2.4 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 2.4. 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 2.5. 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 2.3.

- 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 2.4. 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 2.1. Если $P = NP$, то $\widetilde{\mathrm{P}} = \widetilde{\mathrm{NP}}$.

Следствие 2.2. 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 2.1. Для задачи SAT сведение задачи поиска к задаче распознавания можно было бы сделать (и именно это сведение рассказывалось на лекции) 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.

3 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.

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

PSPACE $= \cup_{c>0}$DSpace$[\mathrm{n}^c]$ — это множество языков, которые вычислимы за полиномиальную память.

Нетрудно заметить, что P $\subseteq$ NP $\subseteq$ PSPACE, второе включение выполняется, так как с использованием полиномиальной памяти можно перебрать все возможные доказательства (их длина ограничена полиномом).

Определим кванторную пропозициональную формулу: она имеет вид $Q_1x_1Q_2x_2\dots Q_nx_n\varphi(x_1,x_2,\dots,x_n)$, 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 3.1. Язык $TQBF$ является PSPACE-полным. Сначала мы докажем, что $TQBF \in$ PSPACE. Рассмотрим следующий рекурсивный алгоритм: пусть на вход подана формула вида $Q_1x_1\phi$. 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. Конфигурация машины Тьюринга — 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 $2^{\mathrm{poly}(n)}$ различных конфигураций (тут используется, что машина использует полиномиальную память). Для каждого входа x 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 из конфигурации K_1 попадает 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, поэтому мы будем считать . Рассмотрим предикат $s(u, v, i)$, который истинен, если из вершины u графа конфигураций 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.

3.1 Вычисления с константной памятью

Конечным автоматом называется пятерка $(Q, q_0, F, \Sigma, \delta)$, где

- Q — конечное множество состояний
- $q_0 \in Q$ — начальное состояние
- $F \subset Q$ — множество конечных состояний
- Σ — конечный входной алфавит
- $\delta : Q \times \Sigma \to Q$ — функция перехода

Автомат начинает работу в состоянии q_0. Он читает символы один за другим и переходит согласно функции перехода в новое состояние. Если, прочитав вход, автомат придет в конечное состояние, то он этот вход принимает, в противном случае вход отвергается.

Пример 3.1. Опишем конечный автомат, который принимает все строки, в которых содержится подстрока “PDMI”.

- $Q = \{q_0, q_\mathrm{p}, q_\mathrm{d}, q_\mathrm{m}, q_\mathrm{i}, q_f\}$, $F = \{q_f\}$
- $(q_0, \Sigma \setminus \{\mathrm{P}\}) \mapsto q_0$
- $(q_0, \mathrm{P}) \mapsto q_\mathrm{p}$
- $(q_\mathrm{p}, \Sigma \setminus \{\mathrm{P,D}\}) \mapsto q_0$
- $(q_\mathrm{p}, \mathrm{D}) \mapsto q_\mathrm{d}$
- $(q_\mathrm{d}, \Sigma \setminus \{\mathrm{P,M}\}) \mapsto q_0$
- $(q_\mathrm{d}, \mathrm{M}) \mapsto q_\mathrm{m}$
- $(q_\mathrm{m}, \Sigma \setminus \{\mathrm{P,I}\}) \mapsto q_0$
- $(q_\mathrm{m}, \mathrm{I}) \mapsto q_f$
- $(q_{0,\mathrm{p,d,m}}, \mathrm{P}) \mapsto q_\mathrm{p}$
- $(q_f, \Sigma) \mapsto q_f$

Недетерминированный конечный автомат (НКА) отличается от детерминированного тем, что функция перехода неоднозначна: $\delta : Q \times \Sigma \to 2^Q$. НКА принимает слово, если существует легальная последовательность шагов, приводящих после прочтения слова в конечное состояние.

Теорема 3.2. По любому НКА можно построит детерминированный конечный автомат, принимающий тот же язык.

Доказательство. У недетерминированного автомата множеством состояний будут подмножества Q. $S = 2^Q, s \subset Q$. Начальное состояние $s_0 = \{q_0\}$. Правило перехода: $(s, a) \mapsto \bigcup\limits_{q \in s} \delta(q)$. Конечные состояния $F_s = \{s \in S | F \cap S \neq \emptyset\}$. □

Регулярные выражения. Определим язык регулярных выражений. Алфавитом этого языка будет некий алфавит Γ и не входящие в него символы $\Lambda, \epsilon, (,), *, |$.

- Буквы алфавита Γ — регулярные выражения;
- Символы Λ, ϵ — регулярные выражения;
- Если $A_1, A_2, \ldots A_n$ — регулярные выражения, то $(A_1 A_2 \ldots A_n)$ — регулярное выражение;

- Если $A_1, A_2, \dots A_n$ — регулярные выражения, то $(A_1|A_2|\dots|A_n)$ — регулярное выражение;
- Если A — регулярное выражение, то $A*$ регулярное выражение.

Каждому регулярному выражению мы сопоставим язык:

- Букве из Γ соответствует одноэлементный язык из этой буквы;
- Символу ϵ — пустой язык, Λ — язык из пустого слова;
- Выражению $(A_1 A_2 \dots A_n)$ соответствует конкатенация языков для $A_1, A_2, \dots, A_n$.
- Выражению $(A_1|A_2|\dots|A_n)$ соответствует объединение языков для $A_1, A_2, \dots, A_n$.
- Выражению $A*$ соответствует язык, слова которого можно разрезать на части, принадлежащие A.

Примеры регулярных выражений:

- $(a|b)*$ — все слова из букв a и b;
- $(aa)*$ — все слова из четного числа букв a;
- $(\Gamma * (\text{ПОМИ})\Gamma*)$ — все слова, содержащие подстроку ПОМИ.

Теорема 3.3. По любому регулярному выражению можно построить конечный автомат, который принимает язык, задаваемый этим выражением.

Доказательство. Индукция по построению выражения. Достаточно построить недетерминированный автомат, он по теореме 3.2.

- Для ϵ: $Q = F = \{q_o\}$. Для Λ: автомат, в котором конечное состояние недостижимо;
- $a \in \Gamma$: $Q = \{q_0, q_1, q_f\}, F = \{q_f\}, (q_0, a) \mapsto q_f, (q_0, \Gamma \setminus \{a\}) \mapsto q_1, (q_1, \Gamma) \mapsto q_1$;
- Покажем, как по автоматам A_1 и A_2 построить автомат, который примет язык, конкатенацию языков для A_1 и A_2. Добавим во все конечные состояния автомата A_1 правила, соответствующие начальному состоянию A_2.
- Покажем, как по автоматам A_1 и A_2 построить автомат, который примет язык, объединение языков для A_1 и A_2. Добавим в начальное состояние автомата A_1 правило, соответствующее начальному состоянию автомата A_2.
- Покажем, как по автомату A, принимающему язык L, построить автомат, принимающий язык $L*$. Для этого сделаем новое начальное состояние q_0' (старое оставим) Добавим все правила, которые были у старого, сделаем q_0' конечным состоянием, кроме того во все конечные состояния добавим правила, какие есть у q_0'.

□

Теорема 3.4. По любому конечному автомату можно построить регулярное выражение, которое задает тот же язык, что принимает автомат.

Доказательство. Для НКА можно считать, что у него есть всего 1 конечное состояние. Его можно модифицировать следующим образом: добавить состояние q'_f, из всех состояний, откуда можно попасть в конечное, добавить аналогичный переход в q'_f. Добавить еще одно неконечное состояние q_∞, и все шаги из q'_f направить в q_∞, и из q_∞ все правила ведут в него самого.

Пусть вершины НКА пронумерованы числами $1, 2, \ldots, k$. Вершина с номером 1 — начальная, вершина с номером k — единственная конечная. Обозначим через $D_{i,j,s}$ множество всех слов, которые можно прочитать, на пути из вершины i в j, если по пути можно заходить только в вершины с номерами $1, 2, \ldots, s$. Язык, задаваемый автоматом, совпадает с $D_{1,k,k}$ Индукцией по s докажем регулярность выражения $D_{i,j,s}$. База $s = 0$: $D_{i,j,0}$ либо пуст либо состоит из пустого слова, либо из буквы. $D_{i,j,s+1}$ можно определить по следующей формуле:

$$D_{i,j,s+1} = D_{i,j,s} | (D_{i,s+1,s} D_{s+1,s+1,s} * D_{s+1,j,s})$$

□

Языки, которые распознаются конечными автоматами, называются автоматными или регулярными.

С помощью следующей леммы можно доказывать, что языки не являются автоматными.

Лемма 3.1 (Лемма о накачке, pumping lemma)**.** Для любого регулярного языка L существует число n, что любую строку $\alpha \in L, |\alpha| > n$ можно представить в виде $\alpha = uvw$, где $|v| \geq 1$ и для всех k слово $uv^k w \in L$.

Доказательство. Пусть n — число состояний детерминированного КА, принимающего L. Если $\alpha \in L, |\alpha| > n$, то в процессе принятия слова α какое-то состояние q встретилось не менее 2-х раз. Обозначим строчку до первого вхождения в состояние q за u, строчку между первым и последним состоянием q за v, и оставшуюся часть за w. Очевидно, что строчка $uv^k w$ тоже будет приниматься. □

Пример 3.2. Язык $L = \{ \underbrace{11..1}_{p \text{ штук}} \mid p - \text{ простое}\}$ не является регулярным.

Доказательство. Если бы он был регулярным, то нашлось бы такое n, что для $p > n$ $\underbrace{11..1}_{p \text{ штук}} = uvw, |v| > 1$ и для всех k $uv^k w \in L$. Пусть $|v| = d > 1$, $|u| + |w| = a$.
Тогда при всех $k > 0$ число $kd + a$ должно быть простым. Выберем $k = d + a + 1$: $(d + a + 1)d + a = (d + a)(d + 1)$ — не является простым. □

Для каждого языка L определим бинарное отношение $\sim$, для двух строчек x и y выполняется $x \sim y$, если для всех строчек z строка xz лежит в L тогда и только тогда, когда yz лежит в L. Нетрудно проверить, что отношение $\sim$ является отношением эквивалентности.

Теорема 3.5. Язык L является автоматным тогда и только тогда, когда число классов эквивалентности по отношению $\sim$ конечно.

Доказательство. Пусть язык L распознается конечным автоматом с множеством состояний Q. Для каждого состояния q определим множество L_q, которое состоит из всех строк, прочитав которые, автомат останавливается в состоянии q. Все строки из L_q эквивалентны по отношению $\sim$, следовательно, классы эквивалентности являются объединением нескольких L_q, следовательно их число не больше, чем $|Q|$.

Пусть для языка L число классов эквивалентности по отношению $\sim$ конечно. Сопоставим каждому такому классу состояние, обозначим множество всех состояний буквой Q. Назовем начальным состоянием то, которое соответствует классу эквивалентности пустой строки. Множество конечных состояний соответствует тем классам эквивалентности, которые целиком содержатся в L. $\delta(q, a) = q'$, если для какой-то строки x в классе q строка xa лежит в классе q'. Индукцией по длине строки можно доказать, что так построенный автомат на строке x придет в состояние, которое соответствует классу эквивалентности строки x. □

Теорема 3.6. Класс DSpace[1] совпадает с множеством регулярных языков.

Доказательство. Пусть язык L распознается конечным автоматом, тогда его можно распознать на одноленточной машине Тьюринга без дополнительной памяти. Множество состояний Q у машины будет состоять из состояний конечного автомата и двух финальных состояний $\{q_{yes}, q_{no}\}$, у машины будет то же самое начальное состояние q_0. Если у конечного автомата было правило перехода, которое отображало $(q, a) \mapsto q'$, то у машины Тьюринга будет правило перехода $(q, a) \mapsto (q, a, \rightarrow)$. Кроме того для каждого финального состояния q нужно добавить правило $(q, \_) \rightarrow (q_{yes}, a, \cdot)$, а для всех остальных состояний правило: $(q, \_) \rightarrow (q_{no}, a, \cdot)$.

Пусть язык $L \in$ DSpace[1], он решается машиной Тьюринга M, которая использует константную память. Мы докажем, что число классов эквивалентности по отношению $\sim$ будет конечно. Для машины Тьюринга M определим понятие псевдоконфигурации, которое включает текущее состояние, положение головок на всех рабочих лентах и содержание рабочих лент. (В конфигурацию машины Тьюринга кроме псевдоконфигурации включают положение головки на входной ленте.) Поскольку машина M использует константную память, то число псевдоконфигураций конечно. Обозначим множество всех псевдоконфигураций P.

Каждой строке x соответствует пара (p, f), где $p \in P$, а $f : P \rightarrow P \cup \{\perp\}$. Псевдоконфигурация p — это либо финальная псевдоконфигурация, если машина Тьюринга M на входе x заканчивает работу, не выходя за пределы строки x, либо псевдоконфигурация, в которую попадает машина, когда в первый раз головка на входной ленте

выходит за пределы x. А f — это отображение, преобразует псевдоконфигурацию, которую мы интерпретируем как псевдоконфигурацию входа головки машины на строку x (т.е. головка указывает на последний символ строки x), в псевдоконфигурацию выхода (либо финальная псевдоконфигурацию, либо псевдоконфигурацию, которая соответствует выходу за пределы строки x). Поскольку не все псевдоконфигурации реально достижимы, то возможна ситуация, что машина не остановится, не выходя за пределы строки x, в этом случае отображение выдаст символ $\perp$. Заметим, что если строкам x и y соответствует одинаковая пара (p, f), то $x \sim y$. Следовательно, число классов эквивалентности по отношению $\sim$ не превосходит числа пар (p, f), которых конечное число. □

Пусть $b(i)$ обозначает бинарную запись числа i. Рассмотрим язык $L_* = \{b(1) * b(2) * \cdots * b(n) \mid n \in \mathbb{N}\}$.

Предложение 3.1. Язык L_* не является регулярным.

Доказательство. Утверждение следует из леммы 3.1. Если v содержит звездочку, то для больших k строка uv^kw не может лежать в языке, так в каждой строчке языка L_* не больше, чем 2^m пар соседних звездочек, которые находятся на расстоянии m. Если же v не содержит звездочку, то все слова uv^kw не могут содержаться в L_*, так как там ровно одна строчка с данным количеством звездочек. □

Предложение 3.2. $L_* \in \mathrm{DSpace}[\log \log n]$.

Доказательство. Опишем машину Тьюринга M, которая решает язык L_*. Машина выполняет много проверок, как только какая-то проверка не прошла, она переходит в состояние q_{no}, если все проверки прошли, то она переходит в состояние q_{yes}. Машина сначала проверит, что до первой звездочки записана строка 1. Дальше машина проверяет, с самого начала, что соседние строки s_1 и s_2 обладают следующим свойствам:

1. Длина строки s_2 либо равна длине строки s_1, либо на 1 больше. Для этой проверки нужно посчитать длину строки s_2, затем вычитать по единичке для каждого символа строки s_2. Для этого потребуется памяти $O(\log |s_1|)$.

2. Проверить, что строка s_2 равняется $s_1 + 1$ в двоичной системе счисления. Для этого надо посчитать число единиц k, на которых заканчивается s_1, убедиться, что строка s_2 кончается на 1 и k нулей. И кроме того нужно проверить, что остальные цифры у s_1 и s_2 совпадают. Для этого потребуется константное число счетчиков для чисел не больших, чем $|s_1| + 1$.

Проверим, что машина $M(x)$ использует память $O(\log \log |x|)$. Память для проверок разных пар строк может быть переиспользовано. Пусть пара s_1, s_2 последняя, которую машина проверяла. Это значит, что до s_1 встречались все строки от 1 до $s_1 - 1$, следовательно, длина строки x как минимум $2^{|s_1|}$. Для последней проверке понадобиться больше всего памяти, т.е. всего памяти не больше, чем $O(\log |s_1|) = O(\log \log |x|)$. □

Оказывается, что с меньшим объемом памяти распознать язык L_* нельзя, это следует из следующего утверждения:

Теорема 3.7. Если $s(n) = o(\log\log n)$, то $\mathrm{DSpace}[s(n)] \subseteq \mathrm{DSpace}[1]$.

Доказательство. Расмотрим язык $L \in \mathrm{DSpace}[s(n)]$, пусть он распознается машиной M, которая использует $O(s(n))$ памяти. Пусть S некоторое натуральное число, рассмотрим самую короткую строку x, что машина, работая, на x использует как минимум S ячеек памяти на рабочих лентах. Пусть $M(x)$ использует $S' \geq S$ ячеек памяти. Пусть P — это множество всех псевдоконфигураций, которые затрагивают только S' ячеек памяти, которые использует $M(x)$. Аналогично доказательству теоремы 3.6 каждому префиксу y строки x соответсвует пара (p, f), где p — псевдоконфигурация, в которой головка машины в первый раз выйдет за пределы строки y, а f отображение из $P \to (P \cup \{\perp\}) \times \{0, 1\}$. Отображение f по псевдоконфигурации, когда головка машины возвращается на последний символ строки y, выдает псевдоконфигурацию выхода за пределы y (ответ $\perp$ выдается, если машина хочет использовать больше, чем S' ячеек памяти; если машина закончит работу до того, как выйдет за пределы y, то финальная псевдоконфигурация и будет считаться выходом) дополнительный бит означает, верно ли, что во время того, пока головка машины находится строке y, впервые произошло использование как минимум S ячеек памяти.

Пусть двум различным префиксам y_1 и y_2 строки x соответсвует одинаковая пара (p, f), при этом $|y_1| < |y_2|$ и $x = y_2 z$. Тогда поведение машины (результат и факт использования не менее S ячеек памяти) на входе $y_2 z$ совпадает с поведением машины на входе $y_1 z$, что противоречит тому, что строка x самая короткая, когда машина M использует не больше, чем S ячеек памяти.

Пусть $n = |x|$, число пар (p, f) должно быть как минимум $n-1$. Число псевдоконфигураций $|P|$ не превосходит $2^{cS'}$ для некоторой константы c, а число отображений f не превосходит $2^{cS'2^{cS'}}$. Должно выполняться неравенство $2^{cS'2^{cS'}} \geq n - 1$, отсюда следует, что $S' \geq \Omega(\log\log n)$.

Таким образом, если память, которую использует машина M, не ограничена константой, то найдется сколь угодно длинная строка x, на которой машина использует как минимум $\Omega(\log\log|x|)$ памяти. Это противоречит тому, что машина использует $o(\log\log n)$ памяти. □

3.2 Вычисления с логарифмической памятью

Вычисления с логарифмической памятью можно представлять себе так: программа использует конечное число переменных, которые принимают целочисленные значения полиномиального от длины входа размера и не использует массивов и другой памяти.

Легко показать, что если машина Тьюринга использует $O(\log n)$ памяти, то она работает полиномиальное время. Конфигурацией машины Тьюринга называется: текущее состояние, положение головок на всех лентах и содержимое рабочих лент. Действительно, число конфигураций такой машины на входе x длины n не превосходит $poly(n)$. Если

бы две конфигурации могли бы повториться, то такая машина не закончила бы работу. Следовательно, такая машина обязательно остановится за $poly(n)$ шагов.

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

Теорема 3.8 (Савич)**.** $DPATH \in \mathrm{DSpace}[\log^2 n]$.

Доказательство. 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 $O((\log n)^2)$. □

Класс $\mathrm{PSPACE} = \bigcup_{\mathrm{c}>0} \mathrm{DSpace}[\mathrm{n}^{\mathrm{c}}]$ состоит языков, которые распознаются машиной Тьюринга с использованием детерминированной памяти. Аналогично классу NP можно определить класс NPSPACE. Класс NPSPACE состоит из языков, для которых существует система доказательств $\Pi(x, y)$, которая вычисляется машиной Тьюрингна, в которой x располагается на входной ленте, y на специальной ленте, которая используется только для чтения и головка по этой ленте движется слева направо, при этом машина использует $poly(|x|)$ памяти на рабочих лентах.

Из теоремы Савича следует:

Следствие 3.1. PSPACE = NPSPACE.

Доказательство. Включение PSPACE $\subseteq$ NPSPACE очевидно, так как машина, распознающая язык из PSPACE сама и является системой доказательств с пустым доказательством. Рассмотрим язык $L \in$ NPSPACE, пусть $\Pi(x, y)$ — система доказательств. Для Π существует машина Тьюринга, которая принимает доказательство на отдельной ленте, доступной только для чтения, при этом головка на этой ленте может двигаться только слева направо. Конфигурацией такой машины назовем: символ на ленте с доказательством, положение головки на остальных лентах и содержимое рабочих лент. Из каждой конфигурации можно попасть в одну (если головка на ленте с доказательством не двигается) или в две конфигурации. Получается граф из конфигураций. Число конфигураций $2^{poly(n)}$, нужно узнать, есть ли путь из начальной конфигурации в принимающую, по теореме Савича это можно узнать, используя $poly(n)$ памяти. □

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

Для пути в неориентированном графе есть более сильная теорема:

Теорема 3.9 (Рейнгольд)**.** $UPATH \in \mathrm{DSpace}[\log \mathrm{n}]$.

Доказательство этой теоремы достаточно трудное, вместо этого мы приведем вероятностный алгоритм для достижимости в неориентированном графе, который использует $O(\log n)$ памяти.

Алгоритм 3.1. Вход: неориентированный граф $G(V, E)$, вершины s, t.

1. Запустить случайное блуждание из вершины s на $4|V||E|$ шагов. При блуждании каждый раз переходить в случайного соседа вершины с равными вероятностями.
2. Если во время блуждания встретилась вершина t, то вернуть 1, иначе 0.

Алгоритм 3.1 использует $O(\log n)$ памяти, так как ему нужно хранить только счетчик числа ходов.

Пусть для графа G и вершин u, v случайная величина $X_{u,v}$ означает число шагов, которые требуются при случайном блуждании из вершины u, чтобы попасть в вершину v.

Лемма 3.2. Если вершины u и v соединены ребром, то $\mathrm{E}[X_{u,v}] \leq 2|E|$.

Эту лемму мы докажем позже.

Теорема 3.10. Если s и t не соединены путем, то алгоритм 3.1 выдает 0, а если s и t соединены путем, то алгоритм 3.1 выдает 1 с вероятностью хотя бы $\frac{1}{2}$.

Доказательство. Если s и t не соединены путем, то утверждение теоремы очевидно.

Пусть s и t соединены путем $v_0 = s, v_1, v_2, \ldots, v_k = t$. Рассмотрим случайную величину $Y_{s,t}$, которая равняется числу шагов в блуждании от вершины $s = v_0$ до v_1 затем от вершины v_1 до v_2 и так далее до вершины $v_k = t$. Очевидно, что $\mathrm{E}[Y_{s,t}] \geq \mathrm{E}[X_{s,t}]$. $Y_{s,t} = \sum_{i=0}^{k-1} Z_i$, где Z_i — это случайная величина, которая равняется длине пути в блуждании от v_i до v_{i+1}. Очевидно, что $\mathrm{E}[Z_i] = \mathrm{E}[X_{v_i,v_{i+1}}]$, а по лемме 3.2 $\mathrm{E}[X_{v_i,v_{i+1}}] \leq 2|E|$. Поэтому можно оценить $\mathrm{E}[X_{s,t}] \leq \mathrm{E}[Y_{s,t}] = \sum_{i=0}^{k-1} \mathrm{E}[Z_i] = \sum_{i=0}^{k-1} \mathrm{E}[X_{v_i,v_{i+1}}] \leq 2|V||E|$. Из неравенства Маркова следует, что $\Pr[X_{s,t} > 4|V||E|] < \frac{1}{2}$, следовательно вероятность того, что алгоритм выдаст 0 меньше, чем $\frac{1}{2}$. □

Лемма 3.2 следует из следующей леммы.

Лемма 3.3. Будем представлять случайное блуждание, как блуждание по ребрам связного графа, а не по его вершинам. Тогда среднее число шагов, которое требуется сделать, чтобы блуждание, начавшееся по ребру (u, v) в данном направлении, снова прошло по ребру (u, v) в таком направлении, равняется $2|E|$.

А лемма 3.3 вытекает из следующей леммы:

Лемма 3.4. Для каждого связного графа G обозначим $\pi_{u,v}^{(n)}$ — вероятность того, что случайное блуждание (считаем, что оно начинается в фиксированной вершине) на n-м шаге проходит через ребро (u, v) в данном направлении. Тогда если граф G не является двудольным, то $\pi_{u,v}^{(n)}$ стремится к $\frac{1}{2|E|}$. А если граф является двудольным, то каждый четный или нечетны член последовательности $\pi_{u,v}^{(n)}$ равен нулю, а подпоследовательность на остальных номерах стремится к $\frac{1}{|E|}$.

Лемма 3.4 следует из следующей леммы:

Лемма 3.5. Для каждого связного графа G обозначим $p_i^{(n)}$ — вероятность того, что случайное блуждание (считаем, что оно начинается в фиксированной вершине) на n-м шаге проходит через вершину i в данном направлении. Пусть d_i — степень вершины i. Тогда если граф G не является двудольным, то $p_i^{(n)}$ стремится к $\frac{d_i}{2|E|}$. А если граф является двудольным, то каждый четный или нечетны член последовательности $p_i^{(n)}$ равен нулю, а подпоследовательность на остальных номерах стремится к $\frac{d_i}{|E|}$.

Доказательство. Обозначим $p_i^{(n)} = c_i^{(n)} d_i$. Тогда $p_i^{(n+1)} = \sum_{(i,j)\in E} c_j^{(n)}$. Получается, что $c_i^{(n+1)} = \frac{1}{d_i} \sum_{(i,j)\in E} c_j^{(n)}$. Значение c_i^{n+1} является выпуклой комбинацией нескольких предыдущих. Это значит, что максимальное c_i не увеличивается, а минимальное не уменьшается. Мы докажем, что если граф недвудольный, то разница между минимальным и максимальным c_i уменьшается в константное число раз через константное число шагов. В двудольном графе будет выполняться тоже самое, только на шагах с каждой четностью отдельно: c_i будут равны нулю на шагах с одной четностью в одной доле и с другой четности в другой доле.

Пусть граф недвудольный, тогда существует такое число N, что между любыми двумя вершинами есть путь длины N. Произвольный путь можно удлинить на 2, пройдя по ребру два раза в одну и другую сторону. В недвудольном графе есть цикл нечетной длины, поэтому, дойдя до цикла можно исправить четность до нужной.

Получаем, что $c_i^{(n+N)}$ является выпуклой комбинацией $c_j^{(n)}$ и все коэффициенты будут положительными. Это значит, что разница между минимальной и максимальной c_j уменьшается в константу раз, следовательно, они все стремятся к одному пределу. Предел $p_i^{(n)}$ в d_i раз больше, утверждение леммы следует из того, что сумма $\sum_i p_i^{(n)} = 1$.

Для двудольного графа доказательство аналогично, только надо рассматривать отдельно четные и нечетные шаги. □

4 Параллельные вычисления

Булевы схемы могут являться простой моделью для параллельных вычислений. Вершины схемы разбиваются на уровни, нулевой уровень — это входы, первый уровень — это вершины, в которые входят ребра из вершин нулевого уровня, вершины i-го уровня — это те вершины, которые еще не имеют уровня (меньшего i), но все ребра входят из вершин уровня меньше, чем i. Число уровней в схеме — это глубина схемы. Все операции на одном уровне можно выполнять одновременно, т.е. глубина схемы — это число параллельных тактов, необходимых для вычисления схемы.

4.1 Параллельный алгоритм для достижимости

Лемма 4.1. Язык $DPATH$ решается схемами полиномиального размера и глубины $O(\log n^2)$, если граф представлен в виде матрицы смежности.

Доказательство. Булево произведение 0/1 матрицы — это такое произведение, когда вместо сложения берется дизъюнкция, вместо умножения конъюнкция. Булево произведение матриц является ассоциативным. Если матрицу смежности возвести в степень k, то в клетке с координатами (i, j) стоит 1 тогда и только тогда, когда из вершины i есть путь в вершину j длины не более k. Пусть n — это число вершин в графе, нам нужно узнать, есть ли путь между вершиной s и t, для этого нужно возвести матрицу в степень n (или любую степень большую, чем n). Для этого достаточно матрицу смежности возвести в квадрат $\lceil \log n \rceil$ раз. Одно возведение в квадрат требует глубины $O(\log n)$, итого, общая глубина равняется $O(\log^2 n)$. □

Следствие 4.1. Любой язык из класса DPSpace[log n] можно решить полиномиально по размеру схемой глубины $O(\log^2 n)$.

Доказательство. Рассмотрим язык $L \in DPSpace[\log n]$, для этого языка существует машина Тьюринга M, которая распознает этот язык с использованием $O(\log n)$ памяти. Пусть конфигурация машины — это текущее состояние машины, положение головок на всех лентах и содержимое рабочих лент. Поскольку машина использует $O(\log n)$ памяти, то число конфигураций не превосходит $poly(n)$. Рассмотрим граф конфигураций этой машины: из одной конфигурации идет ребро в другую, если машина за один шаг попадает из этой конфигурации в другую. В этом графе нужно определить, верно ли, что из начальной конфигурации на входе x машина попадает в конфигурацию, которая соответствует принимающей. Это можно сделать с помощью схемы полиномиального размера и глубины $O(\log^2 poly(n)) = O(\log^2 n)$. Есть некоторая проблема, связанная с тем, что принимающих конфигураций может быть несколько, эту проблему можно решить, переделав машину так, что после того, как новая машина придет в старое принимающее состояние q_{yes}, она почистит всю память и головку вернет на первый символ и только после этого переходит в новое принимающее состояние q'_{yes}. □

4.2 Параллельное сложение и умножение

Теорема 4.1. Существует схема размера $O(n)$ и глубины $O(\log n)$, вычисляющая сумму двух n-битных чисел.

Доказательство. Если вычислять сложение чисел в столбик, то глубина такой схемы получится линейной, поскольку переносы вычисляются последовательно через предыдущие переносы. Поэтому достаточно построить схему глубины $O(\log n)$ и размера $O(n)$, которая вычислит переносы во всех разрядах. В каком случае будет перенос в k-м разряде при сложении двух чисел $\overline{\alpha_n \alpha_{n-1} \dots \alpha_1}$ и $\overline{\beta_n \beta_{n-2} \dots \beta_1}$? Перенос будет, если $\overline{\alpha_k \alpha_{k-1} \dots \alpha_1} + \overline{\beta_k \beta_{k-1} \dots \beta_1} > 2^k - 1$, если мы перенесем в другую часть неравенства второе число, то получим $\overline{\alpha_k \alpha_{k-1} \dots \alpha_1} > \overline{\beta'_k \beta'_{k-1} \dots \beta'_1}$, где $\beta'_i = 1 - \beta_i$. Таким образом, задача вычисления всех переносов сводится к задаче сравнения всех суффиксов первого числа с суффиксами инвертированного второго числа.

Для простоты будем считать, что $n = 2^\ell$, в общем случае старшие биты можно заполнить нулями. Для сравнения всех суффиксов мы будем делать промежуточные

вычисления. Каждое промежуточное вычисление — это сравнение двух чисел одинаковой длины, результат сравнения можно записать с использованием двух битов. Первая фаза предварительных вычислений состоит из ℓ шагов. На i-м шаге будут сравниваться числа $\overline{\alpha_{2^{i-1}t}\dots\alpha_{2^{i-1}(t-1)+1}}$ и $\overline{\beta'_{2^{i-1}t}\dots\beta'_{2^{i-1}(t-1)+1}}$ для всех t от 1 до $2^{\ell-i+1}$. На первом шаге придется сделать все сравнения, на это потребуется схема глубины $O(1)$ и размера не больше $O(n)$. На i при $i>1$ шаге надо произвести $2^{\ell-i+1}$ сравнений, каждое такое сравнение можно вычислить с помощью уже вычисленного на предыдущем шаге: сравниваем первую половину чисел (т.е. число образованное первыми 2^{i-2} цифрами), если они не равны, то можно выдать результат сравнения, если же равны, то надо выдать результат сравнения вторых половин. Для реализации i-го шага глубина схемы увеличится на $O(1)$, а размер на $O(2^{\ell-i+1})$. Итого, мы выполним всю первую фазу с помощью схемы глубины $O(\log n)$ и размера $O(n)$.

На второй фазе мы сравним все суффиксы $\overline{\alpha_k\alpha_{k-1}\dots\alpha_1}$ и $\overline{\beta'_k\beta'_{k-1}\dots\beta'_1}$. Эта фаза разбивается на $\ell-1$ шагов. На i-м шаги обрабатываются такие k, что в двоичной записи числа k встречается i единиц. Числа, которые нужно сравнить на первом шаге, уже сравнены на первой фазе. Если $k=2^t+2^{t+1}s$, то можно воспользоваться сравнить части, соответствующие битам от $2^{t+1}s+1$ до k (эти части сравнены на первой фазе) и, если они равны, то сравнить концы чисел, которые сравнены на предыдущем шаге, так как в числе $2^{t+1}s$ единичек в двоичном представлении на 1 меньше, чем в k. Итого, на каждом шаге мы увеличили глубину схемы на константу, а размер схемы увеличился не больше, чем на константу для каждого k, т.е. размер увеличился не больше, чем на $O(n)$. Таким образом, мы вычислили все переносы схемой размера $O(n)$ и глубины $O(\log n)$. □

Теорема 4.2. Существует схема размера $O(n^2)$ и глубины $O(\log n)$, вычисляющая произведение двух n-битных чисел.

Доказательство. Если моделировать умножение в столбик, то умножение двух n-битных чисел схемой константной глубины и квадратичного размера сводится к сложению n чисел из $2n$ битов. Складывать числа будем по принципу 3 в 2. Складывая три числа, мы будем в одно число записывать поразрядную сумму цифр по модулю два, а во второе число записывать поразрядный перенос (при сложении трех двоичных цифр перенос может возникнуть только на один разряд). Сумма исходных трех чисел равняется сумме получившихся двух. Такое сложение можно выполнить схемой константной глубины и линейного размера. Каждый раз мы будем разбивать имеющиеся числа на тройки (если количество чисел не кратно трем, то с оставшимися числами никаких действий производить не будем) и параллельно складываем их в 2 числа. Через $O(\log n)$ шагов останется всего два числа, которые мы сложим с помощью схемы из теоремы **??**. Итоговая схема будет иметь глубину $O(\log n)$ и размер $O(n^2)$. □

5 Коммуникационная сложность

Рассмотрим следующую задачу: есть функция $f : X \times Y \to Z$. Алиса знает $x \in X$, Боб знает $y \in Y$, им нужно вместе вычислить функцию $f(x, y)$ и при этом минимизировать количество битов пересылаемой информации.

Коммуникационным протоколом, вычисляющим $f : X \times Y \to Z$ называется бинарное дерево, каждый лист этого дерева помечен элементом множества Z. Каждая вершина v дерева помечена либо некоторой функцией $a_v : X \to \{0, 1\}$, либо $b_v : Y \to \{0, 1\}$. Для каждой вершины, кроме листа, одно ребро к сыновьям помечено 0, другое 1. Алиса и Боб изначально договариваются о протоколе. Чтобы вычислить $f(x, y)$ Алиса и Боб на своих копиях дерева ставят фишку в корень дерево. Каждый раз, когда фишка находится в вершине, которая помечена функцией a_v, Алиса посылает Бобу $a_v(x)$ и они передвигают фишку в потомка, по ребру, которое соответствует биту $a_v(x)$; если же вершина помечена функцией b_v, то Боб посылает Алисе $b_v(y)$ и они передвигают фишку аналогичным образом. Когда фишка пришла в лист, то значение в листе должно совпадать с $f(x, y)$. Коммуникационной сложностью f называется минимальная возможная глубина коммуникационного протокола.

Рассмотрим пример $X = Y = \{0, 1\}^n$, функция $EQ(x, y) = 1$, если $x = y$ и 0 иначе. Нетрудно видеть, что коммуникационная сложность функции EQ не превосходит $n+1$: Алиса посылает всю строку x Бобу, после чего Боб посылает ответ, равны ли строки. Покажем, что коммуникационная сложность функции EQ не меньше, чем $n + 1$.

Мы можем представить каждую функцию $f(x, y)$ в виде матрицы $M[X, Y]$, строки которой проиндексированы элементами X, а столбцы элементами Y, при этом $M(x, y) = f(x, y)$. Назовем прямоугольником произвольную подматрицу $M[X', Y']$, где $X' \subseteq X$, а $Y' \subseteq Y$. Прямоугольник $M[X', Y']$ называется одноцветным, если все элементы матрицы этого прямоугольника равны между собой.

Рассмотрим коммуникационный протокол, вычисляющий функцию $f(x, y)$. Каждой вершине дерева сопоставим некоторый прямоугольник. Корню соответствует прямоугольник $M[X, Y]$. Пусть вершине v соответствует прямоугольник $M[X_v, Y_v]$, v_0 и v_1 — сыновья вершины v. Пусть вершина v помечена функцией a_v, обозначим $X_{v_0} = \{x \in X_v \mid a_v(x) = 0\}$, а $X_{v_1} = \{x \in X_v \mid a_v(x) = 1\}$, тогда вершинам v_0 и v_1 будет соответствовать прямоугольники $M[X_{v_0}, Y_v]$ и $M[X_{v_1}, Y_v]$. Пусть вершина v помечена функцией b_v, обозначим $Y_{v_0} = \{y \in Y_v \mid b_v(y) = 0\}$, а $Y_{v_1} = \{y \in Y_v \mid b_v(y) = 1\}$, тогда вершинам v_0 и v_1 будут соответствовать прямоугольники $M[X_v, Y_{v_0}]$ и $M[X_v, Y_{v_1}]$. Листам дерева будут соответствовать одноцветные прямоугольники, цвета которых совпадают с ответом в листе. Таким образом, если в дереве N листьев, то матрицу $M[X, Y]$ разбить на N одноцветных прямоугольников.

Лемма 5.1. Коммуникационная сложность функции EQ равняется $n + 1$.

Доказательство. Достаточно доказать нижнюю оценку $n + 1$. Если бы существовал коммуникационный протокол для EQ глубины не больше, чем n, то листьев в этом протоколе было бы не больше, чем 2^n, следовательно, матрицу $M[X, Y]$ для функции

EQ можно было бы разбить на не больше, чем 2^n одноцветных прямоугольников. Матрица $M[X, Y]$ имеет размер $2^n \times 2^n$, на ее диагонали стоят единицы, остальные элементы нули. Нетрудно заметить, что две единицы не могут лежать в одном прямоугольнике: если бы прямоугольник содержал точки (x, x) и (y, y), то он бы содержал и (x, y) и (y, x). Следовательно, при любом разбиении $M[X, Y]$ на одноцветные прямоугольники должно быть хотя бы 2^n прямоугольников цвета 1 и хотя бы один прямоугольник цвета 0, т.е. всего прямоугольников должно быть не менее $2^n + 1$, противоречие. □

Предложение 5.1. Для каждой многоленточной машины Тьюринга, которая распознает язык палиндромов за $T(n)$ шагов с использованием $S(n)$ памяти, выполняется $T(n)S(n) \geq \Omega(n^2)$. Как обычно считается, что входная лента доступна только для чтения.

Доказательство. Будем рассматривать входы, которые имеют вид $x0^n y$, где $|x| = |y| = n$. Алиса знает строку x, Боб знает строку y, они будут исполнять работу машины, начинает Алиса и моделирует работу машину, пока головка на входной ленте в первый раз не попадет на клетку с номером $2n + 1$ (там начинается строка y, которую Алиса не знает), в этот момент Алиса посылает Бобу текущее состояние и содержание всех рабочих лент. После этого Боб моделирует машину, пока головка машины на входной ленте не попадет в клетку с номером n, в этот момент Боб передает управление Алисе и посылает ей текущее состояние и содержимое рабочих лент. И так далее, если машина заканчивает работу, то тогда тот, кто моделирует работу, посылает результат.

Если машина на всех входах длины $3n$ работает не больше, чем $T(n)$ шагов, то пересылок сообщений было не больше $T(n)/n$, так как между любыми двумя пересылками машина должна была сделать как минимум n шагов, каждый раз пересылается не больше, чем $CS(n)$ битов информации, где C — некоторая константа, а $S(n)$ — максимальная память, которая используется на входах длины $3n$. Поскольку коммуникационная сложность предиката равенства равняется $n + 1$ (а машина фактически проверяет равенство строк x и перевернутой строки y), то $CT(n)S(n)/n \geq n + 1$, следовательно $T(n)S(n) \geq \Omega(n^2)$. □