Теория Алгоритмов

Александр Киракосян
По лекциям Ивана Близнеца

14 июня 2018 г.

Содержание

1. Машина Тьюринга

13 февраля 2018

1.1. Базовые определения

***Определение* 1.1.**
Одноленточной машиной Тьюринга M будем называть тройку (Q, Γ, δ), где

1. Q — конечное множество состояний, такое, что $Q \supseteq \{q_{\texttt{start}}, q_{\texttt{halt}}\}$
2. Γ — конечный алфавит.
3. δ — функция переходов: $Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$

Так же для моделирования работы одноленточной машины Тьюринга введем бесконечную в одну из сторон ленту.

Замечание.
Дополнительно, для удобства будем считать, что алфавит содержит символ $\square$, соответствующий пустому символу. Далее, все строки, записанные на лентах, будут всегда заканчиваться бесконечным количеством пустых символов и не будут содержать их внутри (т.е. будут являться конечными строками над алфавитом $\Gamma/\{\square\}$).

Так же введем символ $\triangleright$ — означающий начало строки. Будем рассматривать лишь те машины Тьюринга, которые не затирают такой символ и не пишут его.

***Определение* 1.2.**
Конфигурацией машины Тьюринга назовем одну выделенную ячейку ленты, одно зафиксированное состояние (его называют состоянием головки) и написанное на ленте слово. Т.е. конфигурацию можно записать как тройку (i, q, s), где $i \in \mathbb{N}_0$ — индекс выделенной ячейки, $q \in Q$ — состояние головки, $s \in \Gamma^*$ — строка, записанная на ленте.

***Определение* 1.3.**
Работа машины Тьюринга заключается в изменении своей конфигурации согласно функции переходов, а именно, если была конфигурация (i, q, s) и $\delta(q, s[i]) = (q_1, c, d)$, то новой конфигурацией будет тройка (i', q', s'), где $i' := i + d$, $q' := q_1$, $s' := (s[i] = c)$ (заменили i-ый символ на символ c).

Замечание.
В случае, если $i + d$ отрицательно, считаем, что $i' = 0$. Т.е. головка просто не сдвинулась.

Формально, можно написать $i' := \max(0, i + d)$.

***Определение* 1.4.**
Пусть s — входное слово.

Результатом работы машины Тьюринга M на входе s назовем $M(s) = t$ — слово, написанное на ленте в момент, когда состояние головки M будет $q_{\texttt{halt}}$, если запустить машину Тьюринга на начальной конфигурации $(0, q_{\texttt{start}}, \triangleright s)$ (в t мы не учитываем начальный символ $\triangleright$).

Замечание.
Машина Тьюринга может не завершить свою работу (никогда не прийти в состояние $q_{\texttt{halt}}$). В таком случае результат работы машины Тьюринга не определен.

Замечание.

В определениях выше машина Тьюринга определена довольно формально (формальнее, чем на лекицях). Из этого следует, что при рассказе на экзамене не требуется следовать данным определениям дословно, а стоит руководствоваться своим пониманием.

***Определение* 1.5.**

Многоленточная машина Тьюринга на k лентах также состоит из алфавита, множества состояний и функции перехода. Отличие в том, что функция перехода имеет вид $Q^k \times \Gamma^k \to Q^k \times \Gamma^k \times \{-1, 0, 1\}^k$, где k — количество лент.

Замечание.

В случае, когда $k = 1$ получаем одноленточную машину Тьюринга.

Замечание.

Далее, нужно определить конфигурацию и принцип работы. Определяются они довольно естественно. Неформально, мы смотрим на состояния всех головок, символы в выделенной ячейках на всех лентах и делаем переход согласно функции переходов.

Формальные определения выписывать не буду, при желании, можно написать их самим.

***Определение* 1.6.**

Начальной конфигурацией многоленточной машины Тьюринга для строки s назовем конфигурацию при которой все головки указывают на первые символы своих строк и находятся в состоянии $q_{\texttt{start}}$, на первой ленте написана строка $\triangleright s$, а на остальных строках $\triangleright$.

***Определение* 1.7.**

Для определения результата работы ограничим множество всех многоленточных машин Тьюринга, а именно, теперь любая многоленточная машина Тьюринга должна удовлетворять следующим условиям:

1. Машина Тьюринга должна иметь хотя бы две ленты. Первую ленту мы называем входной лентой, а последнюю выходной.
2. Машина Тьюринга не должна перезаписывать значения на первой ленте при запуске на любом входе.

Тогда, результатом работы $M(s)$ назовем строку, написанную на `последней` ленте, в момент, когда все головки находятся в состоянии $q_{\texttt{halt}}$.

Замечание.

Т.к. состояний все еще конечное число (а именно $|Q|^k$), то мы можем считать Q^k — новым множеством состояний, тогда если все компоненты равны $q_{\texttt{start}}$, то такое состояние будет стартовым. Аналогично с конечным.

Пример. Проверка строки на палиндром

TODO:

***Определение* 1.8.**

Машина Тьюринга M вычисляет функцию f, если для любого s: $M(s) = f(s)$.

Утверждение 1.1 (Тезис Черча-Тьюринга)**.**

Для любой алгоритмически вычислимой функции f существует вычисляющая её машина Тьюринга.

***Определение* 1.9.**

Машина Тьюринга M работает за время $T(n)$, если M заканчивает свою работу для любой строки s не более чем за $T(|s|)$ шагов.

***Определение* 1.10.**
Функция T называется конструктивной по времени, если $T(n) \geqslant n$ и существует машина Тьюринга, вычисляющая функцию $\lfloor T(n) \rfloor$ за время не более чем $T(n)$.

Замечание.
$\lfloor x \rfloor$ — двоичная запись числа x.

Замечание.
Смысл конструктивности по времени заключается в том, что если машина Тьюринга M имеет время работы $T(n)$ — конструктивное по времени, то M дополнительно может посчитать свое время работы не увеличив его. К примеру, это будет важно в утверждении про забывчивую машину Тьюринга.

1.2. Эквивалентности машин Тьюринга

Предисловие.
Важно заметить, что состояния машины Тьюринга можно использовать как конечную память, а именно, если в процессе работы МТ мы хотим сохранить какое-либо значение, то мы можем достичь этого посредством увеличения количества наших состояний, а именно пусть мы хотим запомнить k бит. Для этого рассмотрим множество $R = \{0,1\}^k$ и будем использовать состояния вида $Q \times R$, где внутри R мы будем хранить значения наших бит.

Важно заметить, что важна конечность числа k. Т.е. мы не можем так запомнить нашу координату на ленте, т.к. целое число не кодируется конечным числом бит.

Утверждение 1.2 (Об эквивалентной машине Тьюринга с меньшим алфавитом)**.**
Для любой машины Тьюринга M существует такой изоморфизм $h : \Gamma^* \to \{0, 1, \square, \triangleright\}$ и машина Тьюринга M', такая что $\forall x : M(x) = h^{-1}(M'(h(x)))$ и алфавит M' состоит только из символов $0, 1, \square, \triangleright$.

Доказательство.
Сразу определим изоморфизм h, как переводящий $\square \to \square$, $\triangleright \to \triangleright$, а все другие буквы закодируем при помощи $\log|\Gamma|$ бит. Пусть на каждую букву требуется k бит.

Каждое действие исходной машины M будет отвечать одной фазе новой машины M'. Будем поддерживать инвариант, что в конце каждой фазы M' хранит внутри своего состояния состояние головки M и буквы, на которые указывают головки. Так же, если на ленте машины M была записана строка s, то на соответствующей ленте M' будет записана строка $h(s)$, а каждая головка будет указывать на начала соответствующего блока (начала кусочка, в который перешла буква под головкой исходной машины).

Таким образом, можно считать, что ленты машины M' разбиты на блоки длины k, а в каждом блоке записана буква исходного алфавита в двоичном виде.

Опишем, как будет действовать наша машина, чтобы поддержать инвариант. Опишем одну фазу.

1. Мы знаем какой переход исходной машины нужно применить, а значит, знаем новое состояние, новые буквы и сдвиги. Сохраним их внутри состояния.

2. Мы хотим записать в блоки новые значения, сдвинуть головки и поменять состояние.

3. Для записи значения пройдем за k итераций по блоку и запишем в него новое значение.
4. Для сдвига на одну ячейку сдвигаем головку k раз, после чего считываем значение в новом блоке.
5. Считывание производим последовательным сдвигом головки и запоминанием считанного текста в состояние. После этого возвращаем головку на начало блока.
6. Сохраняем новое состояние головки (которое должно быть у M).

Таким образом мы смоделировали действие машины Тьюринга M на нашей машине. Т.к. в конце каждой фазы слова на лентах отличались на применение отображения h, то мы получим верный ответ. □

Замечание.
Заметим, что $T(M') = T(M) \cdot t$, где t — время работы одной фазы. Это какая-то константа, которая зависит только от размера алфавита.

Замечание.
Таким образом мы показали, что рассматривая лишь "бинраные" машины Тьюринга мы не уменьшили количество вычислимых на машине Тьюринга функций.

***Определение* 1.11.**
Переопределим одноленточную машину Тьюринга. Теперь, одноленточная машина Тьюринга дополнительно будет удовлетворять следующим правилам:

1. Алфавит обязательно содержит дополнительный символ $|$, который может использоваться только в описанных далее случаях (в частности, он обязан отсутствовать в любом входе).
2. Первыми своими действиями одноленточная машина Тьюринга отделяет вход от своего рабочего пространства символом $|$.
3. В процессе работы машина Тьюринга не перезаписывает входные данные.
4. В конце работы машина Тьюринга должна отделить рабочую ленту еще одним символом $|$ и за ним выписать ответ, который и будет считаться результатом работы.

Утверждение 1.3 (Об эквивалентной одноленточной машине Тьюринга)**.**
Для любой многоленточной машины Тьюринга M существует одноленточная машина M', такая, что $M(x) = M'(x)$ для любого входа x.

Доказательство.
Рассмотрим машину Тьюринга M. Будем считать, что у M всего k лент, множество состояний Q и алфавит Γ, тогда множеством состояний машины M' будет множество Q', а алфавитом $\Gamma \cup \hat{\Gamma}$, где $\hat{\Gamma} = \{\hat{c} \mid c \in \Gamma\}$.

Разобьем работу M' на фазы (каждая фаза будет отвечать одному действию M), так, чтобы после каждой фазы выполнялись следующие инварианты:

1. В начале ленты M' написан вход, затем символ $|$ (это делается просто во время инициализации), а затем снова вход.
2. Если на рабочих лентах машины M написаны строки $s_1, \ldots, s_k$, то в рабочем пространстве ленты M' написана строка $S = s_1[0]s_2[0]\ldots s_k[0]s_1[1]\ldots s_k[1]s_1[2]\ldots$, т.е. лента разбита на блоки длины k, где в блоке с номером i написаны i-ые символы рабочих лент (либо их аналоги с $\hat{x}$).

3. Если состояние головки для машины M было q, то у машины M' будет в памяти запомнено состояние q.

4. Если головка указывала на набор позиций $i_1, \dots i_k$, то соответствующие им символы на ленте машины M' (и только они) будут помечены $\hat{}$

5. Головка M' указывает на разделитель $|$.

Покажем, как поддерживать такой инвариант. Сначала, проведем инициализацию, а именно поставим символ $|$ и скопируем вход еще раз.

Далее промоделируем один шаг машины M.

1. Мы знаем состояние машины M. Пройдем по ленте до самого конца и сохраним все помеченные символы (для этого потребуется запоминать наш индекс внутри блока). После этого вернемся на начало ленты (на символ $|$).

2. Теперь мы знаем состояние и набор символов под головками, а значит, мы знаем все действия, которые мы должны сделать. Запомним новое состояние и требуемые действия.

3. Снова проходим по ленте и заменяем символы помеченные $\hat{}$ на новые требуемые и возвращаемся на начало.

4. Осталось передвинуть $\hat{}$. Для этого снова проходим по ленте, затираем старые пометки и выставляем новые на требуемых местах (тут нужно немного аккуратнее, но в целом, понятно, как это сделать).

В конце процесса мы получим требуемый ответ на последних позициях блоков. Мы можем записать его куда захотим, в частности, в конец строки через символ $|$. □

Утверждение 1.4.
Если у исходной машины было k лент и время работы $T(n) > n$, тогда время работы построенной нами одноленточной машины будет $(T^2(n))$.

Доказательство.
На каждой из k лент исходной машины будет не больше чем $T(n)$ символов, это означает, что на ленте построенной нами машины в любой момент будет не больше чем $(k+2)T(n)$ символов (лишние 2 добавим на вход и выход), т.е. $\mathcal{O}(T(n))$.

Каждую итерацию исходной машины Тьюринга мы заменили на одну фазу, которая пробегается по всей ленте, т.е. работает не более чем $\mathcal{O}(T(n))$. Тогда суммарное время работы будет $\mathcal{O}(T^2(n))$ (нужно еще учесть линейные инициализации и вывод ответа). □

***Определение* 1.12.**
Двусторонеей машиной Тьринга назовем обыную Машину Тьринга без ограничения на неотрицательность i в конфигурации.

Замечание.
Центральный элемент все еще помечен отдельным символом $\triangleright$.

Утверждение 1.5 (Об эквивалентности машины Тьюринга с двусторонней лентой)**.**
Двустороннюю многоленточную машину Тьринга со временем работы $T(n)$ можно проэмулировать на односторонней многоленточной машине Тьюринга с временем работы $T(n)$.

Доказательство.

Пусть алфавит исходной машины был Г. У нашей машины алфавит будет Γ^2. В состоянии будем хранить еще одно дополнительное число, означающее в какой части ленты мы находимся (слева, справа или в центре). Т.е. три возможных значения.

Будем поддерживать такой инвариант:

1. В каждый момент в ячейке с номером i хранится символ $c_i \times c_{-i}$, т.е. символы в ячейках i и $-i$ исходной ленты.
2. В состоянии хранится корректное число положения головки исходной ленты (слева, справа, центр).

Такой вариант мы можем легко поддерживать, т.к. в каждый момент мы знаем полную информацию о состоянии исходной машины. Подробнее:

1. Мы знаем, на какой символ указывает исходная машина и состояние ее головки, а значит мы можем сделать требуемый переход.
2. Для поддержания корректного состояния нашей головки (в терминах числа положения) нужно отслеживать переходы через контрольный символ $\triangleleft$, отвечающий центру.

Осталось заметить, что время работы не изменилось. □

Замечание.

Таким образом мы показали, что разрешение машине Тьюринга ходить в обе стороны не увеличивает количество вычислимых функций.

***Определение* 1.13.**

Многоленточная машина Тьюринга M называется забывчивой, если при любом фиксированном k, для любого входа x, такого что $|x| = k$ траектория движения каждой каретки одинакова.

Утверждение 1.6.

Для любой многоленточной машины Тьюринга M со временем работы $T(n)$ (конструктивна по времени) существует забывчивая машина Тьюринга M' со временем работы $T^2(n)$, такая, что $\forall x : M(x) = M'(x)$.

Доказательство.

Т.к. функция $T(n)$ конструктивна по времени, то существует вычисляющая ее машин Тьюринга.

Начнем описывать требуемую в утверждение забывчивую машину. Сначала, на отдельных лентах посчитаем значение функции $T(n)$. Эти действия зависят только от числа $n = |x|$, а значит, будут одинаковы для всех входов фиксированной длины.

Далее, введем дополнительную ленту, на которую перезапишем вход (чтобы можно было не думать о запрете на запись).

Теперь, сдвинем каретку на каждой ленте на значение $T(n) + 1$ и установим там особый символ \$. После этого сдвигаем каретки назад на начало и начинаем процесс симуляции. Далее, лента для вычисления $T(n)$ нигде использоваться не будет.

На каждой фазе будем симулировать одно действие исходной машины. Аналогично с симуляцией многоленточной машины будем помечать символы, на которые указывает каретки при помощи дополнительных символов $\hat{x}$.

Опишем одну фазу. За нее требуется сдвинуть крышечки, перезаписать символы, поменять состояние. В начале и в конце каждой фазы все головки будут указывать на начала строк рабочих лент.

1. Сначала пройдем каждой головкой по всей длине ленты (до символа \$), запомнив помеченные символы. Сдвинем головки назад на начало.
2. Зная все буквы под головками и состояние исходной машины определим новые буквы, состояние и сдвиги головок.
3. Состояние исходной машины M как всегда храним внутри своего состояния и меняем в конце фазы.
4. Далее опишем действия на одной из лент. Для остальных будет аналогично.
5. На одной ленте продвигаем головку по всей ее длине. При встрече помеченного символа перезаписываем его значение. Осталось правильно переставить метки.
6. Если головку на ленте сдвигать не надо ничего не делаем. Если нужно сдвинуть вправо, делаем это на нашем пути. Если влево, то делаем на пути назад.

Повторим такие действия $T(n)$ раз (даже если машина закончит вычисления раньше). Стоит заметить, что для этого потребуется таймер, значение которого нужно будет увеличивать после каждой фазы. Таймер можно хранить на той же ленте, на которой была посчитана функция $T(n)$.

Осталось заметить, что движение головки во время симуляции зависит лишь от $T(n)$, т.е. только от длины входа, что и требовалось. □

1.3. Универсальная машина Тьюринга

***Определение* 1.14.**

Две машины Тьюринга M_1 и M_2 мы считаем эквивалентными, если для любого входа x последовательность конфигураций M_1 и M_2 совпадают.

Замечание.

При таком понимании различных машин Тьюринга можно считать, что множеством состояний является множество $\{1 \dots |Q|\}$, а алфавитом множество $\{1 \dots |\Gamma|\}$, а значит машина Тьюринга задается двумя числами ($|Q|$ и $|\Gamma|$) и функцией перехода δ.

Утверждение 1.7.

Существует инъекция $g : M \to \{0,1\}^*$ (т.е. мы можем зашифровать все машины Тьюринга строками из 0 и 1).

Доказательство.

Машина Тьюринга M состоит из двух чисел и функции переходов. Т.е. тройки $(|Q|, |\Gamma|, \delta)$.

Каждая функция перехода записывается в виде $x, y \to (x', y', d)$, где x, y, x', y', d — двоичные числа. Тогда δ можно записать как $[\delta_1; \delta_2; \dots \delta_m]$, где каждая δ_i записана в виде $x, y \to (x', y', d)$. Тогда вся машина M записывается как $a|b|\delta$, где δ записана в описанном ранее виде.

Заметим, что для описания машины M мы использовали конечное количество символов $S = \{01, ; | \to ()\}$. Т.е. построили инъекцию $g' : M \to S^*$. Осталось заметить, что существует инъекция $g'' : S^* \to \{0,1\}^*$ (просто кодируем каждый символ), а значит их композиция подойдет на роль g. □

Обозначение.

Результат работы такой функции g на входе M будем обозначать M_α.

Замечание.

Мы можем построить обратное отображение f, которое по бинарной строчке возвращает машину Тьюринга. Проблема в том, что не для всех строчек она определена. Доопределим функцию f на непокрытых строках как машину M, которая сразу падает (просто зависает)

Замечание.

Мы так же хотим добиться, чтобы каждую машину Тьюринга шифровало бесконечно большое количество строк. Для этого можно рассматривать строки вида $s;t$ (где $s,t \in \{0,1\}^*$) и считать, что такая строка задает машину M_s. Таким образом мы ввели что-то вроде комментария для машины Тьюринга.

Итого, в данный момент мы построили корректное **обратное** отображение из строк в машины Тьюринга $\alpha \to M_\alpha$. Более того, для любой машины Тьюринга M существует бесконечное количество строк α, таких что α переходит в M.

20 февраля 2018

Теорема 1.8 (О существовании универсальной машины)**.**

Существует такая универсальная машина Тьюринга U, что $U(\alpha, x) = M_\alpha(x)$.

Доказательство.

Опишем принцип работы машины требуемой машины U. Пусть она получила на вход машину M и вход x. Первым делом преобразуем M до одноленточной машине M', а затем, преобразуем M' к бинарной машине M''. Т.е. U по входу α, x получает значения α'', x''.

Опишем ленты машины U. Кроме входной и выходной ленты у нее будет еще три рабочие ленты. Опишем их:

Рабочая лента — лента, на которой будет происходить симуляция.

Лента переходов — лента, на которой мы храним переходы машины α''.

Лента состояния — лента, на которой хранится текущее состояние M''.

Будем обозначать эти ленты W, T, S соответственно (под записью $W = s$ понимаем, что на ленте W записана строка s).

Просимулируем один шаг машины M''. Для этого делаем следующие действия:

1. Достаем состояние M'' из ленты S — q.
2. Достаем символ, на который указывает головка (на него указывает головка на ленте W) — символ c.
3. Находим на ленте T требуемое правило перехода, т.е. тройку q', c', d.
4. Изменяем состояние лент в соответствии с правилом перехода (перезаписываем S, меняем один символ на W и сдвигаем головку на W).

Осталось в конце работы переписать результат на выходную ленту (результат берем из W).

□

Замечание.

Свести исходную машину к бинарной требуется т.к. иначе алфавит машины U не будет конечным. Она должна будет уметь работать со всеми возможными алфавитами всех возможных машин Тьюринга, что не является конечным множеством.

Утверждение 1.9.

Если время работы M_α было $T(n)$, то время работы на U будет $CT^2(n)$.

Доказательство.

Действительно, на преобразование α'' работает за время $T^2(n)$. При этом время работы U равно $\texttt{time}(M''_\alpha) \cdot D$, где D — время работы одной фазы, которое зависит лишь от описания α, и не зависит от длины входа M.

Время, требуемое на преобразование так же не зависит от n. □

Замечание.

Если подавать на вход лишь одноленточные машины, то асимптотика времени работы не будет отличаться (т.к. квадрат времени мы получили именно из-за симуляции одноленточной машины).

Замечание.

На самом деле можно добиться времени работы $T(n) \log T(n)$. Доказывать не будем.

1.4. Существование невычислимых функций

Теорема 1.10.

Существует невычисляемая на машине Тьюринга функция f.

Доказательство.

Рассмотрим $f : \{0,1\}^* \to \{0,1\}$ такую, что: $f(x) = 0$, если $M_x(x) = 1$, иначе $f(x) = 1$.

К примеру, если M_x зависает на входе x, то $f(x) = 1$.

Покажем, что f невычислима. Пусть не так. Тогда, существует вычисляющая ее машина Тьюринга M_α.

Это означает, что в частности $M_\alpha(\alpha) = f(\alpha)$.

Если $M_\alpha(\alpha)$ заканчивает вычисления и возвращает 1, то $f(\alpha) = 0 \neq M_\alpha(\alpha)$.

Если же $M_\alpha(\alpha)$ зависает, или возвращает что-либо отличное от 1, то $f(\alpha) = 1 \neq M_\alpha(\alpha)$.

Получили противоречие, значит f невычислима. □

***Определение* 1.15.**

Введем функцию `HALT` такую что $\texttt{HALT}(\alpha, x) = 1$, тогда и только тогда, когда M_α останавливается на входе x.

Теорема 1.11.

Функция `HALT` невычислима.

Доказательство.

От противного. Пусть M_α вычисляет `HALT`. Покажем, что тогда мы можем вычислить описанную в теореме выше функцию f.

Действительно, пусть мы хотим вычислить $f(x)$. Проверим, что $M_x(x)$ останавливается. Если нет, то $f(x) = 1$.

Если же $M_x(x)$ останавливается, то можно ее запустить и проверить является ли результат работы единицей. □

2. Классы P и NP

2.1. Базовые определения

***Определение* 2.1.**
Назовем языком любое подмножество $\{0,1\}^*$.

***Определение* 2.2.**
Машина Тьюринга M решает язык L если $\forall x : x \in L \iff M(x) = 1$.

***Определение* 2.3.**
Язык L называется разрешимым если существует решающая его машина Тьюринга (и неразрешимым в противном случае).

Замечание.
Язык — любая бинарная функция $f : \{0,1\}^* \to \{0,1\}$.

В частности, существуют неразрешимые языки (к примеру, HALT).

***Определение* 2.4.**
Класс $\mathtt{DTime}[\mathtt{T}(\mathtt{n})]$ — множество таких языков L, что для них существуют решающие их машины Тьюринга со временем работы $cT(n)$.

***Определение* 2.5.**
$\mathtt{P} = \bigcup\limits_{d \in \mathbb{N}} \mathtt{DTime}(\mathtt{n}^{\mathtt{d}})$.

***Определение* 2.6.**
Класс NP — множество языков L, таких, что существует полиномиальная машина Тьюринга M, такая что $x \in L \iff \exists u : M(x,u) = 1$, где длина $u = poly(n)$ полиномиальна.

Замечание.
Подходящее слово обычно u называют подсказкой или сертификатом.

***Определение* 2.7.**
Введем недетерминированную машину Тьюринга M.

Отличие недетерминированной машины Тьюринга M от обычной заключается в том, что у нее существует две функции перехода δ_0 и δ_1 и на каждом шаге M меняет свою конфигурацию согласно какой-то из этих функций.

Будем говорить, что недетерминированная машина Тьюринга M принимает x ($M(x) = 1$), если существует последовательность корректных переходов (каждый согласно функции δ_0 или δ_1), такая, что используя их M возвращает 1.

***Определение* 2.8.**
Под временем работы недетерминированной машины Тьюринга понимаем **максимальное** из времен работы по всем последовательностям переходов.

Замечание.
В частности, если хотя бы для одного из путей машина зависает, то ее время работы бесконечно.

***Определение* 2.9.**

Класс $\mathtt{NDTime}[\mathtt{T}(\mathtt{n})]$ — множество языков, для которых существует распознающая их недетерминированная машина Тьюринга M со временем работы $cT(n)$.

Класс $\mathtt{NP} = \bigcup_{d \in \mathbb{N}} \mathtt{NDTime}(\mathtt{n}^{\mathtt{d}})$.

Утверждение 2.1.

Два определения класса NP определяют один и тот же класс (т.е. определения корректны).

Доказательство.

Докажем, в одну сторону. Т.е. если $L \in \mathtt{NDTime}(\mathtt{n}^{\mathtt{c}})$, то $L \in \mathtt{NP}$ из первого определения.

Рассмотрим недетерминированную машину Тьюринга M, решающую L. Пусть $x \in L$.

Тогда существует последовательность состоящая из δ_0 и δ_1, такая, что если M использует эти переходы, то $M(x) = 1$.

Построим M' следующим образом: M' принимает на вход пару (x, u), далее M' на каждом шаге делает переход δ_0 или δ_1, описанный в машине M согласно очередному биту строки u (индекс этого бит можно запомнить в состояние и менять каждую фазу).

Таким образом, получаем, что $x \in L \iff \exists u : M'(x, u) = 1$, что и требовалось.

Покажем в другую сторону, т.е. что если $L \in \mathtt{NP}$ (из первого определения), то существует недетерминированая машина Тьюринга решающая L.

Т.к. $L \in \mathtt{NP}$, то существует $M' : x \in L \iff \exists u : M'(x, u) = 1$. Пусть длина u ограничена полиномом $poly(n)$.

Опишем недетерминированную машину M, решающую язык L. Пусть M сначала пишет случайную строку длины не более чем $poly(n)$ (мы это можем благодаря недетерминизму), а после этого запускает M' на входе, и случайно написанной строке.

Очевидно, что $x \in L \iff M(x) = 1$, т.к. если существовала подходящая строка, то в какой-то момент M ее переберет. □

Пример.

Рассмотрим задачу $\mathtt{INDEPENDENTSET} = \{G\}$ — множество таких графов, что в них существует независимое множество размера $\sqrt{n}$.

Такая задача, очевидно, лежит в классе NP (т.к. можно предоставить это независимое множество в качестве u), но пока никто не умеет решать эту задачу за полином.

Замечание.

Равны ли классы P и NP является открытой проблемой.

***Определение* 2.10.**

SAT — множество таких булевых формул φ, что для них существует выполняющий набор.

Замечание.

Очевидно, что SAT лежит в классе NP.

***Определение* 2.11.**

Класс $\mathtt{EXP} = \bigcup_{d \in \mathbb{N}} \mathtt{DTime}(2^{\mathtt{n}^{\mathtt{d}}})$.

***Определение* 2.12.**

Класс $\mathtt{NEXP} = \bigcup_{d \in \mathbb{N}} \mathtt{NDTime}(2^{\mathtt{n}^{\mathtt{d}}})$.

Утверждение 2.2.

$\mathtt{P} \subseteq \mathtt{NP} \subseteq \mathtt{EXP}$

Доказательство.

Первое включение очевидно, т.к. недетерминизмом можно не пользоваться (иметь одинаковые функции переходов δ_0 и δ_1).

Второе включение. Пусть $L \in$ NP и решается машиной M. Тогда $x \in L \iff \exists u : M(x,u) = 1$, где $|x|$ ограничена полиномом $p(x)$. Пусть машина M' перебирает все возможные строки длины не более $p(x)$ и запускает на них машину M. Тогда M' работает экспоненциальное время и проверяет что x лежит в языке L, т.е. $L \in$ EXP. □

2.2. Понятия сведения и полноты

***Определение* 2.13.**

Будем говорить, что язык L сводится по Карпу к языку L', если существует вычислимая за полином функция f, такая что $x \in L \iff f(x) \in L'$.

Обозначается как $L \leqslant_p L'$.

***Определение* 2.14.**

Язык L называется NP-трудным если любой язык из класса NP сводится к L.

***Определение* 2.15.**

Язык L называется NP-полным если L является NP-трудным и лежит в классе NP.

Свойства.

1. $A \leqslant_p A$
2. Если $A \leqslant_p B$, а $B \in$ NP, то $A \in$ NP.
3. Если $A \leqslant_p B$, а $B \leqslant_p C$, то $A \leqslant_p C$

Доказательство.

TODO: □

Замечание.

В первом свойстве вместо класса NP можно написать почти любой другой класс (P, EXP и.т.д.).

27 февраля 2018

***Определение* 2.16.**

Под снимком многоленточной машины Тьюринга в фиксированный момент времени будем понимать упорядоченный набор символов, на которые указывают головки и состояние машины Тьюринга.

Теорема 2.3.

SAT является NP-полным языком.

Доказательство.

Пусть L — язык из класса NP. Тогда для него существует машина Тьюринга M, такая, что $x \in L \iff \exists u : M(x,u) = 1$. Можно считать, что машина M имеет две ленты, первая из которых входная.

Известно, что для любой машины Тьюринга можно построить забывчивую машину Тьюринга, возвращающую тот же результат. Рассмотрим такую машину для M и назовем ее M'.

Идея первая. Неправильная.

Посмотрим как работает машина M'. Т.к. результат ее работы зависит только от x и u, то можно считать, что M' вычисляет некоторую функцию $f(x,u)$. Зафиксируем x, и получим, что M' занимается вычислением функции $f_x(u)$, которую можно записать в виде формулы $\varphi_x(u)$. При такой интерпретации можно считать, что $x \in L \iff \varphi_x$ — выполнима.

Таким образом, если бы мы могли записать формулу φ_x за полиномиальное время, то мы бы получили корректное сведение.

Идея вторая.

Используем немного другой подход (но все еще с похожей идеей). Заметим, что $x \in L$ если существует u и последовательность снимков $z_1, z_2, \ldots, z_{T(n)}$ такие, что последовательность снимков $M'(x,u)$ совпадает с $z_1, z_2, \ldots, z_{T(n)}$ и $M'(x,u)$ возвращает 1. Запишем это условие при помощи формулы $\varphi(z_1, \ldots, z_{T(n)}, u)$.

Сначала введем функции $\texttt{pos}(i)$ и $\texttt{prev}(i)$, первая из которых возвращает положение головки на ленте для чтения после i итераций, а вторая номер последнего перед i шага, на котором головка M' с рабочей ленты находилась на том же месте что и после i шагов. Важно заметить, что эти функции не зависит от строки u, т.к. машина забывчивая.

Теперь покажем как записать, что очередное z_i корректно. Пусть на запись одного снимка требуется c бит. Рассмотрим от чего зависит $z_i = (a_i, b_i, q_i)$ (a_i — символ с ленты для чтения, b_i — с рабочей ленты, q_i — состояние).

1. Состояние q_i зависит только от предыдущего снимка z_{i-1}.
2. Символ a_i зависит только от $\texttt{pos}(i)$ и входа x.
3. Символ b_i совпадает с символом, который был записан на ленту в момент предыдущего посещения ячейки, т.е. только от снимка $z_{\texttt{prev}(i)}$

Таким образом z_i можно вычислить как функцию $F(z_{i-1}, x_{\texttt{pos}(i)}, z_{\texttt{prev}(i)})$. Тогда F можно записать при помощи формулы константного размера, т.к. длина входа для нее равна $2c+1$ (константа). Осталось научиться записывать итоговую формулу φ_x за полиномиальное время.

Сразу заметим, что при фиксированном x мы можем вычислить все значения $x_{\texttt{pos}(i)}$ и $\texttt{prev}(i)$ запустив $M'(x, 0^{T(n)})$ (вот тут воспользовались забывчивостью машины). Далее, мы можем записать формулу, проверяющую корректность каждого из снимков (на это потребуется $C \cdot T(n)$ времени) и проверяем, что выходной бит $z_{T(n)}$ это единичка, что и требовалось. □

Замечание.

Изначальное ограничение на две ленты реально не требуется. Т.е. в процессе доказательства можно было учитывать символы на сколь угодно большом количестве лент.

2.3. Дополнительно

Утверждение 2.4.

Если задача проверки наличия сертификата размера не больше p (полином) решается за полиномиальное время, то и задача его поиска тоже решается за полиномиальное время.

Доказательство.

Мы уже умеем решать за полином задачу проверки наличия сертификата. Запишем ее в виде: по данным M, x, p проверить, что существует u, такое что $M(x,u) = 1$ и $|u| \leqslant p(|x|)$. Покажем, как тогда по M, x, p найти u, такое что $M(x,u) = 1$ и $|u| \leqslant p(|x|)$.

Сразу проверим, что хотя бы одно такое u существует, а далее будем находить u последовательно. После k фаз у нас будут найдены первые k бит сертификата u.

Опишем очередную фазу. Пусть мы уже знаем первые k бит (обозначим их s). Подставим на место $k+1$ бита 0 и проверим, что существует подсказка длины $p(|x|)-(k+1)$, такая, что $M'(x,u)=1$, где $M'(x,u)=M(x,s0u)$, т.е. что существует сертификат с искомым префиксом. Если такого не найдется, то очередной символ 1.

Таким образом мы восстановили подсказку. □

***Определение* 2.17.**

Класс coNP состоит из языков L, таких что $\overline{L} \in$ NP.

Свойство.

Если $L \in$ coNP, то существует полином p и машина M, работающая за полиномиальное время, такие что $x \in L \iff \forall v : M(x,v)=1$

Доказательство.

Пусть $L \in$ coNP. Тогда $\overline{L} \in$ NP, а значит существует решающая его машина M, такая что $x \in \overline{L} \iff \exists u : M(x,u)=1$. Это означает, что если $x \notin L$, то $\forall v : M(x,v)=1$. Значит, инвертированная машина Тьюринга подойдет в качестве машины Тьюринга из условия. □

Утверждение 2.5.

P = NP $\implies$ EXP = NEXP.

Доказательство.

Рассмотрим язык $L \in$ NEXP. Докажем, что $L \in$ EXP.

Пусть L решается машиной M за время $2^{p(n)}$.

Введем язык $L' := \{z01^{2^{p(|z|)}} | z \in L\}$. Покажем, что $L' \in$ NP.

Действительно, пусть машина Тьюринга M' сначала удалит постфикс из единиц и нуля, а затем запустит M на оставшейся части. Заметим, что время работы M' это $\mathcal{O}(2^{p(n)})$, что является полиномом от длины $x = z01^{2^{p(|z|)}}$. Т.е. M' работает полиномиальное время. Тогда, т.к. NP = P, то язык L' решается за полиномиальное время. Пусть за время $poly(|z| + 2^{p(|z|)}) = exp(p(|z|))$.

Таким образом, мы научились решать язык L' за время $exp(p(|z|))$, а тогда L можно решить дописав ко входу постфикс $10^{p(|z|)}$ и запустив уже известный алгоритм. □

Следствие.

EXP $\neq$ NEXP $\implies$ P $\neq$ NP.

3. Диагонализация

3.1. Теорема об иерархии по времени

Теорема 3.1 (Об иерархии по времени)**.**
Пусть $f(n)\log f(n) = o(g(n))$ — конструктивные по времени. Тогда $\texttt{DTime}[f(n)] \subsetneq \texttt{DTime}[g(n)]$

Доказательство.

Включение очевидно. Покажем нестрогость.

Докажем только для случая $\texttt{DTime}[n] \subsetneq \texttt{DTime}[n^{1.5}]$.

Рассмотрим машину $D(x)$, такую, что $D(x) = M_x(x)$ и останавливается после $n^{1.4}$ шагов. Если на очередном запуске $D(x)$ остановилась, то она возвращает инвертированный ответ. Иначе, возвращает 0.

Теперь предположим, что два класса равны. Тогда существует машина $M = D$, работающая за время cn. Известно, что для каждой машины существует бесконечно много ее записей. Возьмем такую запись M n_0, что $n_0^{1.4} > cn_0 \log n_0$ (такая обязательно существует из асимптотик), где c — константа из времени симуляции D.

Тогда на таком входе D закончит свою работу. Это значит, что $D(n_0) \neq M(n_0)$, что является противоречием с выбором M. □

Замечание.

Решение очевидно протаскивается для любых функций f и g из условия.

3.2. Теорема Ладнера

Теорема 3.2 (Теорема Ладнера)**.**
Если $\texttt{P} \neq \texttt{NP}$, то существует язык $L \in \texttt{NP}$ такой что $L \notin \texttt{P}$ и L не является NP-полным.

Доказательство.

Введем язык $\texttt{SAT}_\texttt{f} = \{\varphi 01^{n^{f(n)}} \mid |\varphi| = n\}$ (f — какая-то функция, а $\varphi \in \texttt{SAT}$).

Теперь определим функцию $H(n)$.

Будем определять $H(n)$ последовательно при всех n.

Сначала определим $H(n)$ как минимальное такое i, что машина M_i возвращает верный ответ о принадлежности x языку $\texttt{SAT}_\texttt{H}$ для всех $x : |x| \leqslant \log n$ за не более чем $i|x|^i$ шагов. При этом, если среди первых $\log\log n$ машин такой нет, то положим $i = \log\log n$.

Лемма. Функцию $H(n)$ можно вычислить за полиномиальное время.

Доказательство.

Будем вычислять $H(n)$ последовательно. На очередном шаге посчитаем $H(n)$.

Сначала заметим, что для любого $x : |x| \leqslant \log n$ мы можем проверить его принадлежность $\texttt{SAT}_\texttt{H}$. Действительно, x имеет вид $\psi 01^{|\psi|^{H(|\psi|)}}$, а значит, $|\psi| \leqslant x \leqslant \log n$, т.е. значение $H(|\psi|)$ нам уже известно и можно проверить, что $\psi \in \texttt{SAT}$ за полиномиальное от n время.

Далее, будем последовательно запускать машины M_1, M_2, …, $M_{\log\log n}$ на $|x|, 2|x|^2, 3|x|^3, \ldots$ шагов на каждом входе $x : |x| \leqslant \log n$. Если очередная машина не успела закончить работу на каком-то входе, либо вернула ответ не совпадающий с принадлежностью x языку $\texttt{SAT}_\texttt{H}$, то

переходим к следующей машине. Если очередная машина подошла для всех x, то ее номер является очередным значением функции H, если никакая машина не подошла то ответом является $\log\log n$.

Заметим, что мы корректно посчитали H. Оценим время работы. Мы просимулировали не более чем $\log\log n$ машин, не более чем по $\log\log n(\log n)^{\log\log n} = o(n)$ итераций на не более чем $\mathcal{O}(n)$ входах. Т.е. вычисление функции для одного k занимает не более чем $\mathcal{O}(n^2)$ времени, тогда суммарное время вычисления не превосходит $\mathcal{O}(n^3)$, что и требовалось.

□

Лемма. $\texttt{SAT}_\texttt{H} \in \texttt{P} \iff H(n) = \mathcal{O}(1)$.

Доказательство.

Стрелочка вправо

Пусть $\texttt{SAT}_\texttt{H} \in \texttt{P}$. Тогда существует машина Тьюринга M_k и число d, такие, что M решает $\texttt{SAT}_\texttt{H}$ за время $d \cdot n^d$.

Т.к. любая машина встречается со сколь угодно большими номерами, то можно считать, что $k > d$. Рассмотрим $n = 2^{2^k}$.

Покажем, что для любого $m > n : H(m) \leqslant k$.

Действительно, по определению H машина M_k должна быть проверена ($\log\log m > \log\log n = k$), а значит, т.к. M_k действительно решает язык $\texttt{SAT}_\texttt{H}$, то $H(m) \leqslant k$.

Таким образом, мы получили, что $H(m)$ ограничена при больших m. При маленьких m (меньших n) $H(m)$ можно ограничить какой-то константой (т.к. множество конечно). Получили требуемое.

Стрелочка влево

Если $H(n)$ ограничено, то какое-то значение $H(n)$ принимает бесконечно часто. Пусть это значение c.

Рассмотрим машину M_c и докажем, что она решает $\texttt{SAT}_\texttt{H}$ за время $c \cdot n^c$.

Действительно, рассмотрим вход x. Пусть его длина равна n. Тогда существует $m > 2^n$, такое что $H(m) = c$. Из того, что $H(m) = c$ можно сделать вывод, что M_c корректно работает на всех входах длины не больше $\log m > n$, в частности, на x, что и требовалось.

□

Вернемся к доказательству теоремы. Докажем, что $\texttt{SAT}_\texttt{H} \notin \texttt{P}$. Пусть не так. Покажем, как тогда решить SAT за полиномиальное время.

Пусть нам дана формула φ. Из леммы знаем, что $H(n)$ ограничено константой C. Рассмотрим строки вида $\varphi 01^k$, где k пробегает все значения от 0 до n^C. Запустим на каждой из них алгоритм для решения $\texttt{SAT}_\texttt{H}$. Если хотя бы на одной строке получили 1, то формула выполнима, иначе нет (т.к. существует ограничение на значение $H(n)$).

Так же нужно заметить, что алгоритм работает все еще полиномиальное время, т.к. мы перебрали полином строк полиномиальной длины.

Заметим, что мы не предъявили алгоритм, а лишь показали что он существует (т.к. на самом деле C нам не известно).

Осталось доказать, что $\texttt{SAT}_\texttt{H}$ не NP-полный.

Пусть не так. Тогда существует сведение SAT к $\texttt{SAT}_\texttt{H}$. Пусть время его работы ограничено функцией n^i. Тогда $\varphi \in \texttt{SAT} \iff f(\varphi) \in \texttt{SAT}_\texttt{H}$. Обозначим $f(\varphi) = \psi 01^{|k|}$.

Тогда $k \leqslant n^i$ (т.к. время работы ограничено n^i). Тогда если $i < H(n)$, то $|\psi| < |\varphi|$, т.к. $|\psi|^{H(n)} = |k| < n^i \implies |\psi| \leqslant n^{\frac{i}{H(n)}} < n$, т.е. мы уменьшили размер формулы.

Мы можем сделать такую операцию для всех длин, таких, что $H(n) > i$. Заметим, что $H(n) \leqslant i$ лишь для конечного числа различных n (см. доказательство второй леммы), а значит, для них можно все проверить за какую-то константу.

Для остальных значений длины формулы мы можем свести задачу проверки $\varphi \in \mathtt{SAT}$ к проверке $\psi \in \mathtt{SAT}$ где $|\psi| < |\varphi|$. Повторив такое сведение не более чем $|\varphi|$ раз получим формулу константой длины (либо на каком-то их шагов формулу длины n, что $H(n) \leqslant i$, которые тоже умеем решать). □

6 марта 2018

3.3. Машины с оракульным доступом

***Определение* 3.1.**

Пусть зафиксирован язык O.

Машиной с оракульным доступом будем называть обычную машину Тьюринга дополненную оракульной лентой. На оракульную ленту можно писать, а также в любой момент при переходе воспользоваться результатом проверки $s \in O$ (т.е. принадлежности строки, написанной на ленте, языку O).

***Определение* 3.2.**

Аналогично можно определить недетерминированную машину Тьюринга с оракульным доступом.

***Определение* 3.3.**

Класс $\mathtt{P}^O$ — класс языков, для которых существует полиномиальная машина Тьюринга с оракульным доступом к O.

Класс $\mathtt{NP}^O$ — класс языков, для которых существует полиномиальная машина Тьюринга с оракульным доступом к O.

Замечание.

Так же можно определить классы и другие классы вроде $\mathtt{EXP}^O$.

Свойства.

1. Для любого языка L: $L, \overline{L} \in \mathtt{P}^L$. В частности, $\overline{\mathtt{SAT}} \in \mathtt{P}^{\mathtt{SAT}}$.

   Доказательство.

   Записываем данный вход на оракульную ленту и проверяем. Если надо, инвертируем ответ. □

2. Если $O \in \mathtt{P}$, то $\mathtt{P} = \mathtt{P}^O$.

   Доказательство.

   Включение $\mathtt{P} \subseteq \mathtt{P}^O$ очевидно, т.к. можно никогда не пользоваться оракульной лентой.

   В другую сторону, каждый раз, когда машине Тьюринга потребуется оракульный доступ к O, вместо этого она может за полиномиальное время сама узнать принадлежность требуемой строки O. Всего таких запросов не более полинома, значит время работы осталось полиномиальным. □

Замечание.
Аналогично, если вместо класса P подставить класс EXP.

***Определение* 3.4.**
Язык $\texttt{EXPCOM} = \{(M, x, 1^n) \mid M \text{ принимает } x \text{ за } 2^n \text{ шагов }\}$

Утверждение 3.3.
$\texttt{EXP} \subseteq \texttt{P}^{\texttt{EXPCOM}}$

Доказательство.
Пусть нам дан язык $L \in \texttt{EXP}$, и машина M решает L за время $2^{p(|x|)}$.

Покажем, что $L \in \texttt{P}^{\texttt{EXPCOM}}$. Проверим лежит ли x в L. Для этого обратимся к оракулу с вопросом про тройку $(M, x, 1^{p(|x|)})$. Заметим, что из определения EXPCOM видно, что $x \in L \iff (M, x, 1^{p(|x|)}) \in \texttt{EXPCOM}$, а значит мы смогли проверить лежит ли x в L при помощи одного оракульного запроса.

Суммарное потраченое время — время на выписывания строки $M, x, 1^{p(n)}$. Значения M — константа, а $1^{p(|x|)}$ и $|x|$ полиномиальны. □

Утверждение 3.4.
$\texttt{NP}^{\texttt{EXPCOM}} \subseteq \texttt{EXP}$

Доказательство.
Заметим, что $\texttt{NP}^{\texttt{EXPCOM}} \subseteq \texttt{EXP}^{\texttt{EXPCOM}} = \texttt{EXP}$.

Первый переход сделан из соображения, что $\texttt{NP} \subseteq \texttt{EXP}$, а второй из того, что $\texttt{EXPCOM} \in \texttt{EXP}$ (можно просимулировать данную на вход машину M на протяжении 2^n шагов). □

Следствие.
$\texttt{P}^{\texttt{EXPCOM}} = \texttt{NP}^{\texttt{EXPCOM}}$

Доказательство.
$\texttt{EXP} \subseteq \texttt{P}^{\texttt{EXPCOM}} \subseteq \texttt{NP}^{\texttt{EXPCOM}} \subseteq \texttt{EXP} \implies \texttt{P}^{\texttt{EXPCOM}} = \texttt{NP}^{\texttt{EXPCOM}}$ □

Замечание.
Из всего этого можно сделать вывод, что используя лишь что:

1. Любая машина Тьюринга описывается строкой
2. Существует универсальная машина Тьюринга

невозможно показать, что $\texttt{P} \neq \texttt{NP}$, т.к. для машин с оракульным доступом такие условия выполняются, но тем не менее аналогичное утверждение про классы P и NP неверно.

Тем не менее, все утверждения которые были доказаны используя лишь описанные два свойства машины Тьюринга естественно получаются и для машин с оракульным доступом. К примеру, теорема об иерархии по времени.

Теорема 3.5.
Существуют такие языки A и B, что $\texttt{P}^A = \texttt{NP}^A$ и $\texttt{P}^B \neq \texttt{NP}^B$.

Доказательство.
Язык A уже построен, это язык EXPCOM.

Приступим к построению языка B.

Введем локальное определение. Пусть B — язык, тогда определим язык U_B как унарный язык, такой что $1^k \in U_B \iff \exists x \in B : |x| = k$.

Построим B, такой что $U_B \in \mathtt{NP}^B$, но при этом $U_B \notin \mathtt{P}^B$. Тогда B подойдет в качестве языка требуемого в условии теоремы.

Шаг 1

Докажем, что для любого языка B язык $U_B \in \mathtt{NP}^B$.

Покажем как проверить, что $x \in U_B$. Переберем все возможные строки длины n при помощи недетерминизма и для каждой проверим лежит ли она в $\mathtt{NP}^B$. Если такая строка найдется, то x лежит в U_B по определению, иначе нет.

Иначе говоря, проверем что $\exists u : |u| = |x|$ && $u \in B$.

Шаг 2

Теперь наша цель — построить язык B, такой что $U_B \notin \mathtt{P}^B$.

На протяжении доказательства будем считать, что M_i обозначает i-ую машину Тьюринга с оракульным доступом (как строить такую нумерацию показывать не будем)

Будем строить язык итеративно. На шаге с номером i язык B будет определен для всех строк с длиной меньшей k_i и будет таким, что любая машина с номером меньшим i будет работать некорректно.

Изначально язык B пуст.

Рассмотрим очередную фазу (с номером i). Пусть на данный момент мы определили язык B для всех строк длины меньше k и только для них. Запустим машину M_i на входе 1^k на не более чем $\frac{2^k}{10}$ шагов (почему так можно поясним позже).

В процессе работы машина M_i иногда обращается к оракулу. Будем доопределять язык B по мере симуляции. Каждый раз, когда M_i обращается к какой-то строке длины меньшей k ответ уже известен (B уже определен для маленьких строк). Если же M_i обращается к строке с длиной хотя бы k, то определим такую строку как не лежащую B и продолжим симуляцию.

В конце работы машина M_i вернет какой-то ответ (если не успела завершиться, считаем, что ответ “нет”). Если этот ответ “да” (т.е. что в языке B существует строка длины k), то доопределим B на строках длины k, как не содержащий их (мы имеем право так сделать, т.к. все неизвестные строки мы определяли как не лежащие в языке). Если же M_i вернула ответ “нет”, то рассмотрим какую-нибудь строку длины k, про которую M_i не спрашивала и добавим в язык. Такая строка обязательно существует, т.к. M_i спросила что-то про не более чем $\frac{2^k}{10}$ строк, т.е. не для всех. Таким образом, если M_i успевает закончить работу, то она ошибается.

Теперь пусть максимальная длина строки, про которую спрашивала M_i равна l. Доопределим язык B на всех еще не определенных строках с длиной не больше l как угодно (к примеру, ответом нет). Таким образом мы сохранили инвариант.

Осталось разобраться с временем работы M_i. Пусть машина M_i работала время $T(n)$. Если $T(n)$ не является полиномом, то такая машина не важна (необходимо только чтобы полиномиальные машины ошибались). Иначе, пусть $T(n) = p(n)$ — полином. Покажем, что существует число k, такое, что M_i на нем ошибается. Действительно, т.к. существует бесконечно много записей M_i, то на какой-то итерации была рассмотрена машина M_j эквивалентная M_i, такая, что на этой итерации число k было таким, что $\frac{2^k}{10} > p(k)$, т.е. машина M_j доработала. Тогда на таком k машина M_j, а значит и M_i ошибается, что и требовалось.

Заключение

Таким образом, мы построили язык B такой что $U_B \notin \mathtt{P}^B$, но тем не менее $U_B \in \mathtt{NP}^B$, что и требовалось в теореме. □

4. Сложность по памяти

4.1. Введение

***Определение* 4.1.**
Машина Тьюринга M использует $S(n)$ памяти, если для любого входа длины n машина M использовала не более чем $S(n)$ памяти без учета входа.

Аналогично определение существует и для недетерминированной машины Тьюринга.-

Замечание.
К примеру, из-за того, что вход не учитывается могут существовать машины Тьюринга с $\mathcal{O}(1)$ памяти, тем не менее, обычно используются только машины с $\mathcal{O}(\log n)$ памяти, т.к. хочется хранить индекс, на который указывает головка на входной ленте.

***Определение* 4.2.**
Класс языков $\texttt{SPACE}[f(n)]$ — те языки, для которых существует машина Тьюринга M и константа C, что M использует не более чем $Cf(n)$ памяти.

Аналогично NSPACE для недетерминированной машины Тьюринга.

***Определение* 4.3.**
Функция $S(n)$ конструктивна по памяти, если $\exists M$, такая что M вычисляет $S(n)$ используя не больше чем $S(n)$ памяти.

Утверждение 4.1.
$\texttt{DTime}[f(n)] \subseteq \texttt{SPACE}[f(n)] \subseteq \texttt{NSPACE}[f(n)]$.

Доказательство.
Первое включение, т.к. за $f(n)$ времени невозможно использовать больше чем $f(n)$ памяти, второе т.к. можно просто не использовать недетерминизм. □

***Определение* 4.4.**
Пусть дана машина Тьюринга M и вход x.

Построим граф конфигураций $G_{M,x}$. Вершинами графа будут достижимые конфигурации машины M начавшей с входа x. Ориентированное ребро между конфигурациями c_1 и c_2 проводится, если из c_1 можно попасть в конфигурацию c_2 за один шаг.

Замечание.
В данном определении и далее считается, что конфигурация **не** содержит строки, записанной на входной ленте. Но все еще хранит индекс на ней.

Замечание.
Если M — детерминированная машина Тьюринга, то исходящая степень каждой вершины не превышает 1.

Если M — недетерминированная, то исходящая степень не превышает двух.

***Определение* 4.5.**
Опять сузим класс рассматриваемых машин Тьюринга. Любая машина Тьюринга в конце своей работы на выходную ленту записывает ответ, а все остальные ленты возвращает в исходное состояние (т.е. очищает рабочие ленты).

Замечание.

Таким ограничением класс распознаваемых языков не был сужен, т.к. по любой машине M можно построить удовлетворяющую нашим условиям машину M'. Для этого вместо перехода в состояние остановки переведем машину в состояние $q_{\texttt{clear}}$ и почистим все рабочие ленты. Также нужно заметить, что это не влияет на асимптотику работы, т.к. затраченная память ограничена затраченным временем.

Утверждение 4.2.

Конфигурацию машины Тьюринга M, можно закодировать битовой строкой длины $\mathcal{O}(S(n))$.

Доказательство.

Нужно закодировать состояние, индекс и содержимое лент.

На состояние требуется константа памяти, на индекс для рабочих лент требуется $\mathcal{O}(\log S(n))$ памяти, на их содержимое $\mathcal{O}(S(n))$ памяти. Для индекса входа требуется $\mathcal{O}(\log n)$ памяти (тут нам важно наше ограничение на память конструктивной по памяти функции). □

Замечание.

В доказательстве утверждения существенно, что конфигурация не содержит в себе входа.

Следствие.

Для любой машины Тьюринга, использующей $S(n)$ памяти существует не более чем $2^{\mathcal{O}(S(n))}$ конфигураций.

Утверждение 4.3.

Для любой машины Тьюринга M (необязательно детерминированной) и входа x существует КНФ формула $\varphi_{M,x}$ размера $\mathcal{O}(S(n))$, такая что для любых описаний конфигураций C_1 и C_2: $\varphi(C_1C_2) = 1 \iff$ из C_1 есть ребро в C_2.

Доказательство.

TODO: не справился...

□

Замечание.

Для честного доказательства нужно использовать формат конфигурации через индекс. Доказательство трудоемко и записываться здесь не будет.

Следствие.

Для любой машины Тьюринга M (необязательно детерминированной) использующей $S(n)$ памяти существует машина Тьюринга M' такая, что $M'(x) = G_{M,x}$ со временем работы $2^{\mathcal{O}(S(n))}$.

Доказательство.

Можно записать все возможные конфигурации машины M, после чего для каждой пары при помощи формулы φ, описанной выше, проверить наличие ребра. Суммарное время работы будет $2^{\mathcal{O}(S(n))} \cdot 2^{\mathcal{O}(S(n))} \cdot \mathcal{O}(S(n))$. □

Теорема 4.4.

$\texttt{NSPACE}[f(n)] \subseteq \texttt{DTime}[2^{\mathcal{O}(f(n))}]$

Доказательство.

Пусть дана машина Тьюринга M, которая решает язык L из класса $\texttt{NSPACE}[f(n)]$. Тогда для нее существует машина Тьюринга M_1, находящая граф конфигурации по входу. Покажем, как с ее помощью решить L за время $2^{\mathcal{O}(f(n))}$.

Покажем, как по x проверить его наличие в языке L. Для этого запустим $M'(x)$ и получим граф $G_{M,x}$. Требуется проверить наличие в нем пути из стартовой вершины в вершину q_{end} и единицей на выходной ленте. Это можно сделать за линейное время, что и требовалось. □

Замечание.

Ищем путь именно в состояние q_{end} и единицей на выходной ленте и ни в какие другие, т.к. были введены ограничения на окончание работы любой машины Тьюринга.

13 марта 2018

***Определение* 4.6.**

Класс $\texttt{PSPACE} = \bigcup_{c=0}^{n} \texttt{SPACE}[n^c]$.

Класс $\texttt{NSPACE} = \bigcup_{c=0}^{n} \texttt{NSPACE}[n^c]$.

Класс $\texttt{L} = \texttt{SPACE}[\log n]$.

Класс $\texttt{NL} = \texttt{NSPACE}[\log n]$.

***Определение* 4.7.**

$\texttt{PATH} = \{(G, s, t) \mid$ в ориентированном графе G существует путь из s в $t\}$

Утверждение 4.5.

$\texttt{PATH} \in \texttt{NL}$.

Доказательство.

Пусть машина Тьюринга последовательно при помощи недетерминизма угадывает очередную вершину на пути из s в t.

Пусть уже найден путь s, v_1, …, v_k. Угадаем вершину v_{k+1}. Требуется проверить, что v_k и v_{k+1} смежны. Для этого надо проверить соответствующее значение во входе (считаем, что граф задан матрицей смежности).

Такая машина найдет корректный путь при его наличии и не найдет при его отсутствии. □

Замечание.

Под словами “угадаем” имеется ввиду, что машина перебирает все возможные последовательности вершин. Т.к. в данном случае, машина недетерминированная, то требуется чтобы хотя бы один путь подошел.

4.2. PSPACE-полные языки

***Определение* 4.8.**

Язык A называется PSPACE-полным если $\forall L \in \texttt{PSPACE} : L \leqslant_p A$, и если $A \in \texttt{PSPACE}$.

***Определение* 4.9.**

TQBF — язык, состоящий из истинных формул с кванторами $\exists, \forall$.

Пример.

$\forall x \exists y : x = y$ — лежит в TQBF.

$\exists x \forall y : x = y$ — не лежит в TQBF.

Утверждение 4.6.

TQBF является PSPACE-трудным языком.

Доказательство.

Принадлежность

TODO:

Полнота

Пусть язык L лежит в классе PSAPCE и решается машиной M. Покажем, как свести принадлежность x языку L к принадлежности формулы языку TQBF.

Известно, что существует граф конфигураций для машины M и входа x. Требуется в терминах формулы записать наличие пути из вершины $q_{\texttt{start}}$ в $q_{\texttt{accept}}$ в таком графе. Важно заметить, что построить весь граф мы не можем, тем не менее существует формула $\varphi_{M,x}$ полиномиального размера способная проверить наличие ребра в этом графе.

Т.к. в графе конфигураций не больше чем $2^{\mathcal{O}(\mathcal{S})}$ вершин, то длина пути не превышает этого же числа.

Будем последовательно записывать формулу, проверяющую, что между $u = q_{start}$ и $v = q_{accept}$ существует путь длины не более 2^i.

Строим по индукции.

База при $i = 0$.

Нужно написать формулу, проверяющую, что из u в v есть путь длины не больше 1. $\varphi_0 = \exists C_1 = u \exists C_2 = v : \varphi_{M,x}(C_1, C_2) \vee C_1 = C_2$.

Переход: $i \to i + 1$

Построим формулу φ_{i+1} имея в наличии формулу i.

Пусть $\varphi_{i+1} := \exists C : \varphi_i(u, C) \wedge \varphi_i(C, v)$. Такая формула кодирует ровно то, что требуется. А именно проверяет наличие вершины C, такой, что существует путь из u в C длины не больше 2^i и из C в v длины не больше 2^i. Тогда это эквивалентно наличию пути из u в v длины не больше 2^{i+1}.

Проблема в том, что такая формула в два раза длиннее формулы изначальной формулы, т.е. формула растет экспоненциально.

Запишем φ_{i+1} иначе.

$\varphi_{i+1} := \exists C \forall D_1 \forall D_2((D_1 = u \wedge D_2 = C) \vee (D_1 = C \wedge D_2 = v)) \implies \varphi_i(D_1, D_2)$.

Теперь формула растет линейно. Осталось понять, что мы записали корректную формулу. Предлагается понять это самостоятельно. □

Замечание.

Утверждается, что если перенести все кванторы в начало, то в правой части формулы (после кванторов) будет записана формула в КНФ формате. Проверять не будем.

Следствие.

PSPACE = NPSPACE

Доказательство.

Из доказательства видно, что детерминизм машины Тьюринга нигде не использовался (т.е. если в графе конфигураций степени больше единицы), то доказательство все еще работает. Это означает, что TQBF одновременно является и PSPACE и NSPACE-сложным языком. Тогда эти классы совпадают. □

4.3. Теорема Савича

Теорема 4.7 (Теорема Савича)**.**

Пусть $T(n)$ — конструктивна по памяти.

Тогда $\texttt{NSPACE}[S(n)] \subseteq \texttt{SPACE}[S(n)^2]$.

Доказательство.

Рассмотрим язык $L \in \texttt{NSPACE}[S(n)]$, пусть L решается при помощи недетерминированной машины M. Построим детерминированную машину M', решающую L используя $S(n)^2$ памяти. Покажем, как работает M' на входе x.

Опять таки, существует граф конфигураций $G_{M,x}$. Если x принадлежал языку L, то в таком графе есть путь из q_{start} в вершину q_{accept}. Введем функцию $\texttt{reach}(u, v, i)$, которая равна единице тогда и только тогда, когда в графе $G_{M,x}$ существует путь из s в t длины не больше 2^i.

Покажем, как ответить на вопрос $\texttt{reach}(u, v, i)$.

Сразу заметим, что $\texttt{reach}(u, v, 0) \in \texttt{PSPACE}(S(n))$, т.к. существует формула $\varphi_{M,x}$.

Теперь заметим, что $\texttt{reach}(u, v, i) = \exists w : \texttt{reach}(u, w, i-1) \wedge \texttt{reach}(w, v, i-1)$, т.е. для проверки $\texttt{reach}(u, v, i)$ можно перебрать вершину w и дважды рекурсивно запуститься.

Оценим требуемое количество памяти. Докажем, что на запрос $\texttt{reach}(u, v, i)$ можно ответить используя $\mathcal{O}(i \cdot S(n))$ памяти. База при $i = 0$ уже проверена. Проверим переход.

Перебираем значение w. Для записи w используем $\mathcal{O}(S(n))$ памяти. После этого, используя $\mathcal{O}((i-1)S(n))$ памяти можно проверить значение функции $\texttt{reach}(u, w, i-1)$. После этого, на том же самом месте можно посчитать значение второй функции ($\texttt{reach}(w, v, i-1)$). Таким образом, потребуется не более чем $\mathcal{O}(S^2(n))$ памяти (вся она тратится на рекурсию с записью очередного w). □

Замечание.

Мы еще раз показали равенство классов PSPACE и NSPACE.

4.4. Обобщенной задача географии

Утверждение 4.8.

Если рассматривать в языке TQBF лишь те формулы, в которых все все кванторы стоят в самом начале и чередуются, то такой язык полон относительно класса PSPACE.

Доказательство.

Доказывать не будем. □

Замечание.

Так же можно считать, что правая часть формулы (после кванторов) записана в формате КНФ.

***Определение* 4.10.**

Пусть есть формула $\exists u_1 \forall u_2 \exists u_3 \dots Q u_i : \varphi(u_1, \dots u_i)$.

Двое играют в игру. Первый игрок выбирает значение u_1, затем второй значение u_2, затем первый u_3 и.т.д.

Первый игрок выигрывает если в результате получилась верная формула φ.

Задача выявления победителя называется QBF.

Замечание.

Первый игрок побеждает тогда и только тогда, когда формула в формате TQBF истина. Т.е. язык QBF эквивалентен TQBF.

***Определение* 4.11.**

Пусть дан ориентированный граф G. В одной из вершин стоит фишка. Два игрока по очереди передвигают фишку по ребрам графа. Проигрывает тот, кто не может передвинуть фишку в еще не посещенную вершину.

Задачу определения победителя назовем задачей обобщенной географии.

Утверждение 4.9.

Задача обобщенной географии является PSPACE-сложной задачи.

Доказательство.

Будет достаточно свести язык QBF к решению задачи обобщенной географии, т.е. по данной формуле φ предоставить граф, для которого победитель совпадает с победителем на исходной формуле.

Рассмотрим кванторы в начале формулы φ, обозначим переменные как $x_1, x_2, \ldots, x_m$.Так же обозначим дизъюнкты правой части как $C_1, C_2, \ldots C_n$.

Начнем строить граф. Для каждой переменной $x_1, x_2, \ldots x_n$ заведем такую конструкцию:

TODO: картинки... □

13 марта 2018

4.5. Классы NL, coNL и NL-полнота

Утверждение 4.10.

$\mathtt{NL} \subseteq \mathtt{P}$

Доказательство. **TODO:**

□

Мотивация.

Для определения NL-полноты требуется как-то иначе определять сведение.

Действительно, если использовать стандартное полиномиальное сведение, то любой непустой язык будет являться NL-полным (покажем это ниже).

Утверждение 4.11.

Если $A \in \mathtt{NL}$ — нетривиальный язык (не пустой, и не все слова), то любой язык $L \in \mathtt{NL}$ полиномиально сводится к A.

Доказательство.

Для того, чтобы показать наличие полиномиального сведения требуется предоставить функцию f, вычислимую за полином, такую, что $x \in L \iff f(x) \in A$.

Т.к. A нетривиальный, то $\exists x \in A$, $\exists y \notin A$.

Пусть дан вход x. Пусть функция f проверит принадлежность x языку L (это можно сделать, т.к. $\mathtt{NL} \subseteq \mathtt{P}$). Если окажется, что $x \in L$, то функция f вернет a, иначе b. □

***Определение* 4.12.**

Будем говорить, что функция f неявно вычислима с использованием логарифмической памяти, если $\exists c : f(x) \leqslant |x|^c$ и языки $L_f = \{(x, i) \mid f(x)_i = 1\}$ и $L_{f'} = \{(x, i) \mid i \leqslant |f(x)|\}$ лежат в классе L.

Замечание.

$f(x)_i$ — i-ый бит $f(x)$.

Замечание.

Иначе говоря, существует возможность вычислить i-ый бит функции f используя логарифмическую память.

***Определение* 4.13.**

Будем говорить, что язык A сводим к языку B с использованием логарифмической памяти, если существует неявно вычислимая с использованием логарифмической памяти функция f, такая что $x \in A \iff f(x) \in B$.

Обозначается $\leqslant_l$.

Свойства.

1. Если $A \leqslant_l B$ и $B \leqslant_l C$, то $A \leqslant_l C$.
2. Если $A \leqslant_l B$ и $B \in L$, то $A \in L$.

Доказательство.

1. Пусть f сводит A к B, а g сводит B к C. Покажем, как вычислить любой бит строки $g(f(x))$.

   Рассмотрим машину M вычисляющую функцию g. Она иногда обращается ко входу, т.е. в нашем случае к битам $f(x)$. У нас нет возможности полностью сохранить значение $f(x)$, так что будем симулировать обращение ко входу. Для этого на отдельной ленте будем хранить номер ячейки, на которую указывает головка на входной ленте, а при обращении к битам входа вычислять их при помощи функции f. В результате будет вычислен любой бит строки $g(f(x))$, что и требовалось.

2. Заметим, что язык $B \in L$, а значит $B \leqslant \{1\}$ — языку, состоящему только из единицы (т.к. функция f, возвращающая 1 если $x \in B$ вычислима с логарифмической памятью). Тогда по свойству 1 получаем, что $A \leqslant_l \{1\}$, тогда можно для любого x проверить его принадлежность A (просто посчитав наличие первого бита в $f(x)$ из сведения).

□

Теорема 4.12.

Язык PATH является NL-полным (относительно сведения $\leqslant_l$).

Доказательство.

Что PATH лежит в классе NL уже доказывалось.

Перейдем ко доказательству полноты. Рассмотрим язык $L \in$ NL и покажем, как свести его к PATH. Пусть машина Тьюринга, распознающая L это M.

Пусть требуется определить лежит ли $x \in L$. Существует граф конфигураций $G_{M,x}$. В таком графе будет полиномиальное количество вершин (мы уже изучали оценки на количество вершин в графе конфигураций). Тогда $x \in L \iff (G_{M,x}, q_{\texttt{start}}, q_{\texttt{accept}}) \in$ PATH.

Таким образом, осталось показать, что можно вычислить любой бит строки $(G_{M,x}, q_{\texttt{start}}, q_{\texttt{accept}})$ используя логарифм памяти. $q_{\texttt{start}}$ и $q_{\texttt{accept}}$ можно вычислить, т.к. это константы.

Осталось научится вычислять биты графа $G_{M,x}$, или, иначе говоря проверять наличие ребра в графе $G_{M,x}$ между двумя вершинами. Т.е. требуется проверить, что машина M из конфигурации C_1 может перейти в конфигурацию C_2. Это делается просто проверкой переходов машины M.

□

***Определение* 4.14.**

Лента называется *ReadOnce* если с нее можно лишь читать, и каретка по ней не может двигаться влево (назад).

***Определение* 4.15** (Определение класса NL через сертификат)**.**

Будем говорить, что язык L лежит в классе NL, если существует детерминированная машина Тьюринга M, использующая *ReadOnce* ленту и $\mathcal{O}(\log n)$ памяти, а так же существует полином p, такие, что $x \in L \iff \exists u \in \{0,1\}^{p(|x|)} : M(x,u) = 1$, где x записывается на входную ленту, а u на *ReadOnce* ленту.

Утверждение 4.13.

Определения класса NL эквивалентны.

Доказательство.

Покажем, что машина с подсказкой может просимулировать недетерминированную машину.

Для этого на каждом шаге, когда есть недетерминизм считываем очередной бит с ленты для подсказки, и поступаем согласно ему.

В обратную сторону. Дана машина M принимающая подсказку и распознающая язык L. Покажем, как при помощи недетерминированной машины (назовем ее M') понять лежит ли данный x в языке L.

M' будет повторять действия M, а в каждый момент, в который M обращается к ленте пользоваться недетерминизмом и угадывать полученый бит. То, что лента доступна лишь один раз на чтение используется в том, что от нас не потребуется повторить когда-то угаданный бит. □

Замечание.

Мы пользуемся тем, что любая разумная машина Тьюринга использующая логарифм памяти использует полиномиальное время. Действительно, общее количество конфигураций ограничено полиномом, а значит и время работы полином (если машина не зависает).

***Определение* 4.16.**

Язык $\overline{\texttt{PATH}} = \{(G, s, t)\}$, такие, что в графе G нет пути из s в t.

Замечание.

$\overline{\texttt{PATH}}$ принадлежит классу coNL.

Утверждение 4.14.

$\overline{\texttt{PATH}} \in \texttt{NL}$.

Доказательство.

Нам даны граф G и вершины s и t. Предоставим сертификат, показывающий, что из s в t нет пути.

Введем обозначения $N^i[s]$ — множество вершин на расстоянии не больше чем i от вершины s. Тогда нас интересует вопрос о принадлежности вершины t множеству $N^n[i]$.

Шаг первый

Рассмотрим задачу принадлежности вершины x множеству $N^i[s]$. Такая задача, очевидно, лежит в классе NL, т.к. сертификатом является последовательность вершин, в которые нужно переходить. А проверяющий алгоритм, соответственно на каждом шаге проверяет, что существует ребро в очередную вершину (храним лишь вершину, в которой находимся). Также алгоритм должен на каждом шаге увеличивать длину пути и в конце проверить, что она не превышает i.

Шаг второй

Пусть известна мощность множества $N^i[s] = m$. Покажем, что задача проверки, что вершина t не принадлежит множеству $N^i[s]$ лежит в классе NL. Передадим в качестве подсказки множество вершин внутри $N^i[s]$ в возрастающем порядке, а после каждой вершины дополнительно передадим сертификат, доказывающий, что эта вершина лежит на расстоянии не более чем i от вершины s (сертификат из первого шага). Тогда алгоритм может проверить, что каждая из переданных вершин действительно лежит в множестве $N^i[s]$, а также понять, что все переданные вершины различны (тут используется то, что номера вершин возрастают). Также алгоритм проверяет, что все переданные вершины отличны от вершины t и считает их количество. Если передано ровно m вершин и все они отличны от t, то $t \notin N^i[s]$.

Шаг третий

Покажем, что зная мощность множества $N^{i-1}[s] = m$ можно показать, что вершина t не лежит в множестве $N^i[s]$. Делается это очень похоже на предыдущий шаг. Передадим в качестве подсказки все вершины из множества $N^{i-1}[s]$. Тогда для каждой вершины нам дополнительно необходимо проверить, что t не лежит в множестве ее соседей. Это, очевидно, делается используя логарифм памяти.

Шаг четвертый

Покажем, что задача проверки, что мощность множества $N^{i+1}[s] =$ при известной мощности $N^i[s] = m$ лежит в классе NL.

Мы показали, что если $x \in N^{i+1}[s]$, то существует доказывающий это сертификат. Также если $x \notin N^{i+1}[s]$, то тоже существует доказывающий это сертификат (см. третий шаг). Тогда, передадим в качестве сертификата последовательно сертификаты для каждой из вершин, доказывающий ее принадлежность/не принадлежность множеству $N^{i+1}[s]$. Тогда алгоритм должен лишь проверить все сертификаты и посчитать количество подходящих вершин.

Заключение

Осталось соединить кусочки вместе и описать сертификат. Итого, сертификат состоит из n уровней. На каждом уровне, передается размер очередного множества $N^i[s]$ и сертификаты для каждой из вершин, доказывающие их принадлежность/не принадлежность этому множеству. Внутренние сертификаты описаны в шагах 1 и 2. В самом конце передается сертификат из шага два, доказывающий, что t не принадлежит множеству $N^n[s]$. □

Следствие.
$\mathtt{NL} = \mathtt{coNL}$

Доказательство.
Пусть $L \in \mathtt{coNL}$. Тогда $L \leqslant_l \overline{\mathtt{PATH}} \in \mathtt{NL} \implies L \in \mathtt{NL}$.

Таким образом получили $\mathtt{NL} \subseteq \mathtt{coNL}$.

Теперь пусть $L \in \mathtt{NL} \implies \overline{L} \in \mathtt{coNL} \implies \overline{L} \in \mathtt{NL} \implies L \in \mathtt{coNL}$, что и требовалось. □

5. Полиномиальная иерархия

27 марта 2018

5.1. Класс полиномиальной иерархии

Обозначение.
$\texttt{INDSET} = \{(G, k)$, такие, что в G есть независимое множество I, $|I| \geqslant k\}$

Обозначение.
$\texttt{EXACTINDSET} = \{(G, K)\}$, где максимальное независимое множество в G имеет размер k.

Обозначение.
$\texttt{MINDNF} = \{\varphi\}$ — где φ является минимальной ДНФ формулой, выражающей функцию.

Замечание.
Т.е. не существует ψ в виде ДНФ, такой, что $|\psi| < |\varphi|$ и $\psi(x) = \varphi(x)$ для любого x.

Мотивация.
Для последних двух примеров не понятно, как предоставить подсказку. Кажется, что этого не сделать. Так что мы вводим новый класс.

***Определение* 5.1.**
$\sum_2^p$ — класс языков L, таких что существует полиномиальная машина Тьюринга M и полином q такие что, $x \in L \iff \exists u \forall v : M(x, u, v) = 1$, где $u, v \in \{0, 1\}^{q(|x|)}$

***Определение* 5.2.**
$\prod_2^p$ — класс языков L, таких что существует полиномиальная машина тьюринга M и полином q такие что, $x \in L \iff \forall u \exists v : M(x, u, v) = 1$, где $u, v \in \{0, 1\}^{q(|x|)}$

Замечание.
$\prod_2^p = \{L \mid \overline{L} \in \sum_2^p\}$, т.к. отрицание утверждения про язык из $\sum_2^p$ даст нам $x \in \overline{L} \iff \forall u \exists v : M(x, u, v) = 0$ и останется только инвертировать машину M.

Утверждение 5.1.
$\texttt{EXACTINDSET}$ лежит в $\prod_2^p$ и в $\sum_2^p$.

Доказательство.
Будем передавать в качестве u независимое множество вершин размера k, а в качестве v — независимое множество вершин размера $k + 1$ (если такое есть).

Тогда, очевидно, что наличие пары в языке равносильно существованию такой подсказки u и отсутствию подсказки v. □

Утверждение 5.2.
$\texttt{NP}$ и $\texttt{coNP}$ лежат в $\sum_2^p$.

Доказательство.
Покажем, что $\texttt{NP} \in \sum_2^p$.

Для любого языка L, лежащего в $\texttt{NP}$ мы знаем, что $\exists M, q : x \in L \iff \exists u : M(x, u) = 1$.

Рассмотрим эту же машину Тьюринга M, но подадим ей на вход дополнительный набор бит (будем его игнорировать). Получим машину Тьюринга, распознающую язык L как язык из класса $\sum_2^p$.

Для coNP аналогично (простое упражнение на понимание). □

***Определение* 5.3.**

$\sum_i^p$ — класс языков L, таких что существует полиномиальная машина Тьюринга M и полином q, такие что

$\forall x : x \in L \iff \exists u_1 \forall u_2 \cdots Q_i u_i : M(x, u_1, \cdots, u_i) = 1$, где все $u_i \in \{0,1\}^{q(|x|)}$.

***Определение* 5.4.**

$\prod_i^p$ — класс языков L, таких что существует полиномиальная машина Тьюринга M и полином q, такие что

$\forall x : x \in L \iff \forall u_1 \exists u_2 \cdots Q_i u_i : M(x, u_1, \cdots, u_i) = 1$, где все $u_i \in \{0,1\}^{q(|x|)}$.

Замечание.

$\prod_i^p = \{L \mid \overline{L} \in \sum_i^p\}$ (тоже простое упражнение на понимание, уже доказывали для $i = 2$).

***Определение* 5.5.**

$\mathtt{PH} = \bigcup\limits_i \sum_i^p$ — класс полиномиальной иерархии.

Свойства.

1. $\sum_1^{\mathtt{p}} = \mathtt{NP}$
2. $\prod_1^{\mathtt{p}} = \mathtt{coNP}$
3. $\sum_{\mathtt{i}}^{\mathtt{p}} \subseteq \sum_{\mathtt{i+1}}^{\mathtt{p}}$
4. $\prod_{\mathtt{i}}^{\mathtt{p}} \subseteq \prod_{\mathtt{i+1}}^{\mathtt{p}}$
5. $\sum_{\mathtt{i}}^{\mathtt{p}} \subseteq \prod_{\mathtt{i+1}}^{\mathtt{p}}$
6. $\mathtt{PH} = \bigcup\limits_i \prod_i^p$

Доказательство.

Все свойства очевидно следуют из определений. Свойства $3, 4, 5$ мы уже доказывали для $i = 1$. Тоже простое упражнение. □

Теорема 5.3.

1. Если $\sum_i^p = \prod_i^p$, то $\mathtt{PH} = \sum_i^p$
2. Если $\mathtt{P} = \mathtt{NP}$, то $\mathtt{PH} = \mathtt{P}$

Доказательство.

1. Рассмотрим язык L из класса $\sum_{i+1}^p$. Он имеет решение вида $\exists u_1 \forall u_2 \ldots Q u_i : M(x, u_1, \ldots, u_{i+1})$.

   Зафиксируем u_1 и рассмотрим язык L_{u_1} описанный условием $\forall u_2 \ldots Q u_i : M(x, u_1, \ldots, u_{i+1})$. По определению L_{u_1} лежит в классе $\prod_i^p$, а тогда и в классе $\sum_i^p$. Значит, существует такая машина Тьюринга M', такая что $x \in L_{u_1} \iff \exists v_2 \ldots Q v_i : M(x, u_1, v_2, \ldots, v_{i+1})$.

   Подставим полученное в решение языка L и получим, что $x \in L \iff \exists u_1 \exists v_2 \ldots Q v_i : M(x, u_1, v_2, \ldots, v_{i+1})$. Два первых квантора можно сжать, и получить, что $L \in \sum_i^p$.

   Таким образом, мы доказали, что $\sum_{i+1}^p = \sum_i^p$. Аналогично, $\prod_{i+1}^p = \prod_i^p$. Продолжаем по индукции и получаем, что $\mathtt{PH} = \sum_i^p$.

2. Пусть $\mathtt{P} = \mathtt{NP}$. Тогда $\mathtt{NP} = \mathtt{coNP}$, а значит из первого пункта $\mathtt{P} = \mathtt{PH}$.

□

5.2. Полные задачи для полиномиальной иерархии

***Определение* 5.6.**
Язык $L \in \texttt{PH}$ называется PH-полным, если любой язык $L' \in \texttt{PH}$ полиномиально сводится к L.

Замечание.

1. $\sum_k^p \subseteq \texttt{PSPACE}$, т.к. мы можем перебрать все возможные наборы u_i, v_i.
2. $\texttt{PH} \subseteq \texttt{PSPACE}$. Следует из определения PH и первого замечания.

Утверждение 5.4.
Если L — PH-полный язык, то $\texttt{PH} = \sum_i^p$, для какого-то i.

Доказательство.
Пусть $L \in \texttt{PH}$ — PH-полный язык. Тогда, т.к. $L \in \texttt{PH}$, то $L \in \sum_i^p$ для какого-то i.
Покажем, что $\texttt{PH} = \sum_i^p$.
Пусть $L' \in \texttt{PH}$. Тогда L' сводится к L, т.е. лежит в классе $\sum_i^p$, что и требовалось. □

Замечание.
Тогда $\texttt{PH} \neq \texttt{PSPACE}$, т.к. PSPACE-полные задачи существуют (в предположении, что полиномиальная иерархия не схлопывается, в частности, $\texttt{P} \neq \texttt{NP}$).

5.3. Альтернирующие машины

***Определение* 5.7.**
Альтернирующая машина Тьюринга работает так же как и обычная недетерминированная машина Тьюринга, т.е. на каждом шаге выбирает один из нескольких своих переходов. Из этого следует, что время работы ATM определяется так же (см. определение ниже).

Отличие заключается в том, что у каждого состояния кроме конечных стоит индикатор $\forall$ или $\exists$, и считается, что ATM принимает слово, если она возвращает 1 на любом "корректном" пути.

Под корректным путем мы понимаем то, что из вершины с индикатором $\forall$ мы обязаны пойти в каждого соседа, а для вершины с индикатором $\exists$ имеем право выбрать любого из соседей.

***Определение* 5.8.**
Альтернирующая машина Тьюринга работает время $T(n)$, если на любом пути она останавливается за $T(n)$ шагов.

***Определение* 5.9.**
$\sum_i \texttt{Time}[T(n)]$ — класс языков L, таких что существует распознающая их ATM M, работающая за время $T(n)$, такая что начальное состояние помечено символом $\exists$, а на любом пути знаки меняются не более чем $i-1$ раз.

Утверждение 5.5.
$\sum_i^p = \bigcup\limits_c \sum_i \texttt{Time}[n^c]$.

Доказательство.
Рассмотрим $L \in \sum_i \texttt{Time}[n^c]$. Докажем, что $\exists M$ со временем работы n^c, такая что $x \in L \iff \exists u_1 \forall v_1 \dots Q q_i : M(x, u_1, v_1, \dots, q_i)$ где все u_i — полиномиальной длины.

Действительно, известно, что существует альтернирующая машина, которая распознает язык L за время n^c. Эта альтернирующая машина на каждом своем шаге действует согласно квантору, написанному в вершине. Это означает, что можно выбрать такой набор действий, что машина примет слово. Это означает, что $x \in L \iff Qb_1Qb_2 \dots Qb_m : M(x, b_1, \dots, b_m)$, где все b_i — биты. Осталось объединить все группы одинаковых кванторов и воспользоваться тем, что количество смен квантора не больше $i - 1$.

В другую сторону, если $L \in \sum_i$, то $x \in L \iff \exists u_1 \forall v_1 \dots Qq_i : M(x, u_1, v_1, \dots, q_i)$. Разобьем каждую из строк на биты, и получим, что $x \in L \iff Q_1 b_1 Q_2 b_2 \dots Q_k b_k M(x, b_1, \dots, b_k)$ (при этом квантор сменяется не более чем $i - 1$ раз). Осталось построить АТМ M'. Пусть M' последовательно пишет биты b_1, b_2, ... согласно кванторам, а затем запускает машину M с полученными битами. Такая M' удовлетворяет условию. □

Замечание.

Доказательство очень похоже на доказательство эквивалентности определений для класса NP.

***Определение* 5.10.**

$\texttt{TISP}[\texttt{T}(\texttt{n}), \texttt{S}(\texttt{n})]$ — класс всех языков распознающихся машиной Тьюринга M, использующей $T(n)$ времени и $S(n)$ памяти.

Теорема 5.6.

Верна теорема об иерархии по времени для недетерминированной машины Тьюринга.

Доказательство.

На данный момент просто верим в это. □

Лемма.

$\texttt{TISP}[\texttt{n}^{12}, \texttt{n}^2] \subseteq \sum_2^{\texttt{p}} \texttt{Time}[\texttt{n}^8]$

Доказательство.

Пусть $L \in \texttt{TISP}[\texttt{n}^{12}, \texttt{n}^2]$. Рассмотрим машину Тьюринга решающую этот язык.

Покажем, как организовать подсказку и машину M', чтобы $x \in L \iff \exists u \forall v : M'(x, u, v) = 1$.

Если $x \in L$, то существует путь по конфигурациям длины не более чем n^{12} до конфигурации $C_{\texttt{accept}}$. Выделим на этом пути n^6 конфигураций через расстояния n^6. Обозначим их $C_1, C_2, \dots C_{n^6}$.

Тогда $x \in L \iff \exists C_1 C_2 \dots C_{n^6} \forall i : M'(x, C_1 \dots C_{n^6}, i)$, где M' проверяет, что M переходит из конфигурации C_i в конфигурацию C_{i+1} за n^6 шагов.

Откуда появилось n^8? Дело в том, что машине M' также требуется считать подсказку, которая имеет размер порядка $n^2 \cdot n^6$. □

Замечание.

Если обобщить, то получится, что $\texttt{TISP}[\texttt{n}^{2\texttt{a}}, \texttt{n}^{\texttt{b}}] \subseteq \sum_2^{\texttt{p}} \texttt{Time}[\texttt{n}^{\texttt{a}+\texttt{b}}]$.

Лемма.

Если $\texttt{NTime}[\texttt{n}] \subseteq \texttt{DTime}[n^{1.2}]$, то $\sum_2^p \texttt{DTime}[n^8] \subseteq \texttt{NTime}[n^{9.6}]$

Доказательство.

Сразу заметим, что если $\texttt{NTime}[n] \in \texttt{DTime}[n^{1.2}]$, то $\texttt{coNTime}[n] \in \texttt{DTime}[n^{1.2}]$

Пусть $L \in \sum_2^p \texttt{DTime}[n^8]$ и решается машиной M, т.е. $x \in L \iff \exists u \forall v : M(x, u, v) = 1$.

Зафиксируем u и рассмотрим задачу вида $\forall v : M(x, u, v)$. Эта задача лежит в $\texttt{coNTime}[n^8]$, а значит и в $\texttt{DTime}[(n^8)^{1.2}] = \texttt{DTime}[n^{9.6}]$. Обозначим решающую ее машину как $D_v(x, u)$.

Подставим эту машину в исходную задачу. Получим $\exists u : D_v(x, u)$.

Эта задача лежит в $\mathtt{NTime}[\mathtt{time}(\mathtt{D})] = \mathtt{NTime}[\mathtt{n}^{9.6}]$, что и требовалось. □

Теорема 5.7.
$\mathtt{NTime}[\mathtt{n}] \not\subseteq \mathtt{TISP}[\mathtt{n}^{1.2}, \mathtt{n}^{0.2}]$

Доказательство.

От противного, пусть

$\mathtt{NTime}[\mathtt{n}] \subseteq \mathtt{TISP}[\mathtt{n}^{1.2}, \mathtt{n}^{0.2}] \subseteq \mathtt{DTime}[\mathtt{n}^{1.2}]$

$\mathtt{NTime}[\mathtt{n}^{10}] \subseteq \mathtt{TISP}[\mathtt{n}^{12}, \mathtt{n}^{2}] \subseteq \sum_2^{\mathtt{p}} \mathtt{Time}[\mathtt{n}^{8}] \subseteq \mathtt{NTime}[\mathtt{n}^{9.6}]$

Получили противоречие с теоремой об иерархии по времени для недетерминированной машины. □

6. Вероятностные алгоритмы

3 апреля 2018

6.1. Базовые вероятностные алгоритмы

***Определение* 6.1.**

Пусть дан язык L и алгоритм A, который решает задачу принадлежности слова языку, но иногда ошибается. Назовем вероятностью ошибки алгоритма A максимальную из вероятностей ошибки по всем возможным входам.

Мотивация.

Рассмотрим язык L и алгоритм A. Пусть A на входах $x \notin L$ всегда возвращает 0, а на входах $x \in L$ возвращает ответ 1 с вероятностью p и 0 с оставшейся вероятностью $1 - p$.

Нашей задачей является модифицировать алгоритм, чтобы уменьшить вероятность ошибки.

Обозначение.

Введем новый алгоритм A_k, который на входе x запускает алгоритм A k раз, и в результате ответа возвращает 1, если хотя бы один из запусков вернул 1 и 0, если все запуски вернули 0.

(Обозначение является временным)

Утверждение 6.1.

Вероятность ошибки алгоритма A_k равна $(1 - p)^k$.

Доказательство.

Пусть мы ошиблись, на входе $x \notin L$, т.е. сказали, что $x \in L$. Это означает, что хотя бы один из запусков вернул 1, а тогда x действительно лежит в L. Это означает, что вероятность ошибки на входах $x \notin L$ равна 0.

Осталось посчитать вероятность ошибки на входе $x \in L$. Т.к. мы ответили, что $x \notin L$, то каждый из запусков алгоритма A вернул 0. Вероятность одного такого события равна $1 - p$ (т.к. $x \notin L$). События независимы, значит общая вероятность равна $(1 - p)^k$, что и требовалось. □

Замечание.

Мы получили последовательность алгоритмов A_k, таких, что вероятность ошибки A_k стремится к нулю, если $k \to \infty$.

Замечание.

Если вероятность ошибки зависит от размера входа как функция $p(n)$, то мы можем выбирать $k = \frac{l}{p(n)}$ раз. Тогда вероятность ошибки нового алгоритма будет стремится к $(1-p(n))^{\frac{l}{p(n)}} \to \frac{1}{e^l}$, т.е. больше не зависит от размера входа.

В частности, мы поняли, что для фиксированной вероятности ошибки p имеет смысл выбирать k порядка $\frac{1}{p}$.

Так же важно заметить, что асимптотика времени работы алгоритма увеличивается пропорционально функции $\frac{1}{p(n)}$.

Алгоритм.

Поиск минимального разреза... См. конспект по алгоритмам, там все было сильно лучше.

Задача (Feedback Vertex Cover)**.**

Дан граф G. Требуется проверить, что существует множество вершин S размера k, такое что G/S — лес.

Утверждение 6.2.
`FVC` является `NP`-трудной задачей.

Доказательство.
Покажем, что существует сведение `VertexCover` к `FVC`.

Рассмотрим данный нам граф G в котором требуется найти минимальное вершинное покрытие.

Преобразуем граф G до графа G', а именно для каждого ребра uv добавим дополнительную вершину V_{uv} и пару ребер $V_{uv}u$ и $V_{uv}v$. Получили граф G' на $|V| + |E|$ вершинах.

Покажем, что если есть вершинное покрытие размера k в G, то есть требуемое множество размера k в графе G' (такое множество S', что G'/S' — лес).

Во-первых покажем, что если S — вершинное покрытие, то G'/S' — лес. Пусть не так. Тогда найдем на множестве вершин G'/S' минимальный цикл C. Заметим, что C не содержит новых вершин, т.к. иначе размер можно уменьшить. Но тогда, C содержит две смежные вершины из множества G'/S', а тогда и из множества G/S. Таким образом, если есть вершинное покрытие размера k, то есть множество S' размера k.

Теперь в другую сторону. Пусть мы нашли множество $|S'| = k$ в графе G'. Покажем, что существует вершинное покрытие размера k в графе G. Для каждой старой вершины множества S' возьмем в независимое множество (назовем S) исходную вершину, а для каждой новой вершины V_{uv} возьмем в множество S любую из вершин u и v. Возможно, некоторые вершины совпадут. Покажем, что S — вершинное покрытие. Пусть не так. Тогда в $\overline{S}$ есть ребро uv. Тогда в S' нет вершинок u, v и V_{uv}, а тогда на них есть цикл. Противоречие.

Таким образом, мы показали, что для проверки наличия вершинного покрытия мощности S в графе G достаточно решить `FVC` в графе G', что и требовалось. □

Теорема 6.3.
Пусть дан граф и число k.

Тогда существует вероятностный алгоритм A, работающий за время $4^k \cdot poly(n)$, проверяющий, что в графе G существует множество S размера k, такое, что G/S — лес.

Алгоритм.
Будем упрощать задачу.

1. Если существует вершина u, со степенью равной единице, то ее можно удалить, т.к. она не лежит ни в каком цикле, а значит она не лежит в минимально возможном множестве S.

2. Если есть тройка вершин u, v, w, такая, что существуют ребра uv и vw, а степень v равна двум, то удалим вершину v с исходящими ребрами и добавим ребро uw (могут появиться как петли, так и кратные ребра).

3. Если есть петля в вершине u, то берем вершину u в множество S и уменьшаем k на единицу (вершину с петлей мы обязаны взять).

В получившемся после упрощений графе степень любой вершины $\geqslant 3$.

Теперь опишем алгоритм. На каждой из k итераций выберем случайное ребро, а затем случайную из инцидентных ему вершин и добавим в множество S. Повторяем такую итерацию

k раз, после чего проверяем получившееся множество S (т.е. проверяем, что G/S — дерево). Оценим вероятность найти множество S, если оно существует.

Оценка вероятности успеха

Назовем обозначим G/S как F. Известно, что S существует, так что $|S| = k$, а F — лес.

Разобьем все вершины множества F на 3 группы $V_1, V_2, V_{\geqslant 3}$ по степени внутри леса.

Обозначим множество ребер внутренних ребер F как $E(F)$, а $E(S, F)$ — множество внешних.

Докажем, что $E(S, F) \geqslant E(F)$. Пишем оценки.

Т.к. для каждой вершины из V_1 ведет хотя бы два ребра наружу, а для вершин из V_2 хотя бы одно, то получаем $2|V_1| + |V_2| \leqslant |E(S, F)|$.

Т.к. из вершин леса исходит не менее чем $|V_1| + 2|V_2| + 3|V_{\geqslant 3}|$ ребер, то $\frac{|V_1|+2|V_2|+3|V_{\geqslant 3}|}{2} \leqslant |E(F)|$

С другой стороны, $|E(F)| < |F| = |V_1| + |V_2| + |V_{\geqslant 3}|$

Итого получаем, что $\frac{|V_1|+2|V_2|+3|V_{\geqslant 3}|}{2} \leqslant |V_1| + |V_2| + |V_{\geqslant 3}| \implies |V_{\geqslant 3}| \leqslant |V_1|$.

Тогда $|E(S, F)| \geqslant 2|V_1| + |V_2| \geqslant |V_1| + |V_2| + |V_{\geqslant 3}| \geqslant E(F)$, что и требовалось.

Таким образом, выбрав случайное ребро с вероятностью $\geqslant \frac{1}{2}$ оно заденет решение, тогда выбрав после этого случайную из инцидентных ему вершин с вероятностью $\geqslant \frac{1}{4}$ она окажется в множестве S.

Завершение алгоритма

После добавления одной вершины в множество S и удалении из графа получаем аналогичную исходной задачу для графа G/v и числом $k = k-1$. Таким образом, итоговая вероятность успеха $\geqslant \frac{1}{4^k}$, тогда нужно повторить алгоритм 4^k раз.

Замечание.

При небольших k ($k \sim \log n$) наш алгоритм работает $4^{\log n} \cdot poly(n)$, что является полиномом.

***Определение* 6.2.**

Вероятностная машина Тьюринга. Рассмотрим недетерминированную машину Тьюринга (т.е. из каждого состояния есть два перехода) и будем искать ответ не по всем переходам, а лишь по одному, но выбирать один из двух переходов на каждом шаге случайно (с вероятностью $\frac{1}{2}$).

***Определение* 6.3.**

$\texttt{RTime}[T(n)]$ — класс языков, для которых существует вероятностная машина Тьюринга M, работающая за время $T(n)$, такая, что

$$\begin{cases} x \in L \implies Prob(M(x) = 1) \geqslant \frac{1}{2} \\ x \notin L \implies Prob(M(x) = 0) = 1 \end{cases}$$

***Определение* 6.4.**

$\texttt{RP} = \bigcup\limits_{c=1}^{\infty} \texttt{RTime}(n^c)$.

***Определение* 6.5.**

$\texttt{ZPP}$ — класс языков, для которых существует вероятностная машина Тьюринга M, такая что математическое ожидание времени работы M полиномиально.

$$\begin{cases} x \in L \implies Prob(M(x) = 1) = 1 \\ x \notin L \implies Prob(M(x) = 0) = 1 \end{cases}$$

***Определение* 6.6.**

Вероятностный алгоритм A относится к типу "Лас-Вегас", если результат работы алгоритма A всегда верен, но время работы может отличаться в зависимости от случайных бит.

***Определение* 6.7.**

Вероятностный алгоритм A относится к типу "Монте-Карло", если время работы алгоритма всегда постоянно, но результат может быть неверен на некоторых наборах случайных бит.

Пример.

Алгоритм `quick_sort` является алгоритмом типа "Лас-Вегас".

Алгоритм поиска минимального разреза является алгоритмом типа "Монте-Карло".

***Определение* 6.8.**

Вероятностная схема — обычная схема (см. главу про схемы), к которой дополнительно добавили m случайных бит (они участвуют в вычислении равноправно с входом).

Будем говорить, что схема C распознает язык L, если

$$\begin{cases} x \in L \implies Prob((x) = 1) \geqslant \frac{1}{2} \\ x \notin L \implies Prob((x) = 0) = 1 \end{cases}$$

Теорема 6.4 (Адлемана)**.**

Класс языков, распознаваемых вероятностными схемами совпадает с классом языков, распознаваемых обычными схемами.

Доказательство.

Рассмотрим язык L. Пусть существует вероятностная схема C_p, которая ее решает. Рассмотрим все слова $g_i \in L$ длины n.

Рассмотрим все возможные случайные входные биты (всего у нас 2^m вариантов, так что это просто числа от 0 до 2^m) и введем таблицу размера $k \times 2^m$. В клеточке (i, j) запишем результат вычисления $C_p(g_i, j)$.

В каждой строке таблицы хотя бы половина клеток является единицами. Тогда и во всей таблице хотя бы половина клеток являются единицами. Тогда существует столбец j, в котором хотя бы половина единиц. Рассмотрим его.

Построим обычную схему C_j, которая совпадает со схемой C_p, но на месте случайных бит зафиксированы биты j. Полученная схема принимает хотя бы половину из строк g_i. Удалим эти строки из таблицы и повторим процесс.

В результате мы получили набор схем $C_{j_1}, C_{j_2}, \ldots, C_{j_l}$, где $l \leq \log n$, т.к. каждый раз количество строк уменьшалось хотя бы в два раза.

Сделаем общую схему C, которая принимает слово, если хотя бы одна из схем C_{i_j} его принимает (дополнительная линия памяти).

Оценим общий размер. Он равен $size(C_p) \cdot \log |G|$, где G — все слова длины n лежащие в L. Тогда $|G| \leqslant 2^n$, т.е. размер не превышает $size(C_p) \cdot n$, что является полиномом. □

10 апреля 2018

6.2. Экономия случайных бит

Мотивация.

Пусть дан алгоритм $A \in \mathtt{RP}$.

Рассмотрим описанный ранее метод повышения точности связанный с многократным запуском и оценим количество затрачиваемых им случайных бит и вероятность понижения ошибки.

Если алгоритм был запущен n раз, то вероятность ошибки будет не превышать $\frac{1}{2^n}$, а количество затраченных бит будет равно $n \cdot \texttt{random\_bits}(A)$.

Нашей целью является уменьшить количество требуемых случайных бит, возможно, увеличив вероятность ошибки.

Лемма.

Пусть a и b — два случайных числа из $\mathbb{Z}_p$.

Тогда случайные величины равные остаткам чисел $ai+b$ и $aj+b$ при делении на p независимы, если $i \neq j$.

Доказательство.

Заметим, что $P(ai + b = c) = \frac{1}{p}$, т.к. для каждого a существует ровно одно подходящее b.

Хотим доказать, что $P(ai + b = c, aj + b = d) = \frac{1}{p^2}$.

Т.е. что у системы $\begin{cases} ai + b = c \\ aj + b = d \end{cases}$ ровно одно решение (a и b переменные).

Это очевидно, т.к. определитель матрицы левой части равен $i - j \neq 0$. □

Лемма.

Если $\xi_1, \ldots \xi_n$ попарно независимы, то $D(\xi_1, \ldots \xi_n) = D(\xi_1) + \ldots + D(\xi_n)$.

Доказательство.

См. конспект по теории вероятности (линейность дисперсии). □

Алгоритм (Понижения ошибки)**.**

Пусть A является RP алгоритмом.

Сгенерируем два случайных числа a и b от 0 до $p-1$, где p — большое простое число. Теперь запустим алгоритм A на входах с наборами случайных бит q_i, где $q_i = ai + b \bmod p$ и $i \in [0..t]$.

Вернем 1, если хотя бы раз была возвращена 1 и 0 в ином случае (суммарно алгоритм был запущен t раз).

Утверждение 6.5.

Вероятность ошибки описанного алгоритма не больше $\frac{1}{t}$.

Доказательство.

Мы ошибемся только если все значения $A(x, q_i) = 0$.

Введем случайную величину $\xi = \sum_{i=1}^{t} A(x, q_i)$. Требуется оценить вероятность $Pr(\xi = 0)$.

Оценим математическое ожидание $E\xi$ и дисперсию $D\xi$.

$E(A(x, q_i)) = P(A(x, q_i) = 1) \geqslant \frac{1}{2}$, т..к. q_i распределены равномерно.

$D(A(x, q_i)) = E(A(x, q_i)^2) - E^2(A(x, q_i)) = \mu - \mu^2 \leqslant \frac{1}{4}$ (т.к. оценка на матожидание уже есть).

$E\xi = \sum_{i=0}^{t} A(x, q_i) \geqslant \frac{t}{2}$, т.к. q_i распределены равномерно, а A является RP алгоритмом.

$D\xi = \sum_{i=0}^{t} D(A(x, q_i)) \leqslant \frac{t}{4}$

Заметим, что если $\xi = 0$, то $|\xi - E\xi| \geqslant \frac{t}{2}$ (просто подставляем $\xi = 0$ внутрь модуля), а тогда $Pr(\xi = 0) \leqslant Pr(|\xi - E\xi| \geqslant \frac{t}{2}) \leqslant \frac{4D\xi}{t^2} \leqslant \frac{1}{t}$

Красное неравенство — неравенство Чебышева. □

6.3. Задача поиска стабильного паросочетания

Задача.

Есть колода из 52 карт. Карты случайно разбиты на 13 групп по 4 карты. Мы начинаем вытаскивать карты (начинаем с группы с номером 13). Если на очередном шаге была достана карта с достоинством k, то следующую карту достаем из колоды с номером k. Процесс останавливается если в очередной колоде больше нет карт.

Найти вероятность того, что будут достаны все карты.

Решение.

Запишем последовательность достанных карт. Процесс заканчивается в тот момент, когда из очередной колоды будет попытка достать пятую карту. Если очередную карту достали из колоды k, то это либо первая карта, либо предыдущая имела достоинство k. Второе событие могло произойти не более четырех раз, от такого стопка закончиться не могла, а значит, если колода закончилась, то мы доставали из нее первую карту, т.е. процесс всегда заканчивается на последней стопке.

Общее количество изначальных раскладок равно 52!. Посчитаем, в каком количестве были достаны все карты.

Рассмотрим последовательность доставаемых карт. Мы показали, что процесс заканчивается на последней стопке, т.е. при извлечении четвертого короля (н.у.о. последняя стопка соответствует королям). С другой стороны, мы достали все карты, значит выписанная последовательность имеет длину 52. Таким образом, нас устраивают ровно те последовательности из 52 карт, которые оканчиваются королем. Их количество равно $\frac{52!}{13}$, тогда вероятность успеха равна $\frac{1}{13}$.

Задача (Stable Marriage)**.**

Есть n юношей и n девушек. У юношей и девушек есть упорядоченный полный список предпочтений. Требуется найти "стабильное" паросочетание.

Замечание.

Паросочетание называется стабильным если не существует двух юношей A_1, A_2 и двух девушек B_1, B_2, таких, что были образованы пары A_1B_2 и A_2B_1, но при этом A_1 больше нравится B_1, а B_1 A_1 (т.е. A_1 и B_1 хотят быть друг с другом больше чем со своими партнерами).

Алгоритм.

Существует решение за квадрат (см. конспект по алгоритмам).

Кратко напомним его суть. Юноши поочередно предлагают образовать пару девушкам из своего списка (в порядке уменьшения приоритета). Каждая девушка получив предложение от некоторого юноши сравнивает его со своей текущей парой (состояние без пары имеет отрицательный приоритет), и если предложивший юноша лучше чем текущий принимает его предложение. Соответственно, ее прошлый партнер остался без пары и переходит к следующей девушке из своего списка.

Утверждается, что алгоритм всегда найдет стабильное паросочетание.

Мотивация.

Попробуем оценить среднее время работы алгоритма.

Время работы алгоритма очевидно пропорционально количеству сделанных юношами предложений, значит требуется оценить их количество.

При оценки максимального времени работы получим квадрат, т.к. каждый юноша суммарно сделал не больше чем n предложений. Нас же интересует среднее время работы алгоритма.

Поставим задачу более строго. Пусть зафиксированы предпочтения девушек. Оценим математическое ожидание количества сделанных юношами предложений по всем возможным спискам приоритетов юношей.

Утверждение 6.6.
Среднее время работы алгоритма равно $\mathcal{O}(n \log n)$.

Доказательство.
Заметим, что на каждой итерации (перед каждым предложением) важно лишь очередная девушка, которая нравится юноше, а не весь список. Из этого сделаем вывод, что можно генерировать предпочтения по ходу действия алгоритма, т.е. в каждый момент определять лучшую из оставшихся девушек. В некотором смысле, данная замена означает, что генерация случайной последовательности эквивалента последовательной случайной генерации каждого ее элемента.

Теперь пусть юноша каждый раз выбирает случайную девушку из всех (даже учитывая тех, кому предложение уже было сделано). От такой замены время работы лишь ухудшится. Действительно, если в какой-то момент юноша предложил образовать пару девушке, которой он уже делал предложение раньше, то он гарантировано получит отказ, т.к. ее партнер строго лучше чем наш юноша.

Поймем, в какой момент алгоритм завершится. Алгоритм завершится в тот момент, когда каждая из девушек получит предложение. Действительно, если в какой-то момент девушка получила предложение то после этого у нее всегда будет пара (возможно партнер будет меняться, но он точно не исчезнет).

Вспомним, что каждый раз очередное предложение приходит случайной девушке, а значит задача эквивалентно задаче *Collection Problem* (см.ниже). □

Задача (Collection Problem)**.**
Есть n различных купонов. На каждом шаге мы получаем случайный купон. Требуется найти математическое ожидание количество операций, требуемое на получение купонов всех видов.

Решение.
Рассмотрим случайные величины $\xi_1, \xi_2, \ldots, \xi_n$, где ξ_i — количество итераций между получением i-го купона и $i+1$-го.

Тогда нас интересует математическое ожидание случайной величины $\xi = \sum\limits_{i=0}^{n-1} \xi_i$.

По линейности математического ожидания $E(\xi) = \sum\limits_{i=1}^{n} E(\xi_i)$. Осталось посчитать $E(\xi_i)$.

Пусть вероятность вытащить новый купон равна p. Тогда с вероятностью p мы закончим за один шаг, а с оставшейся вероятностью нам потребуется начать процесс сначала, т.е $E(\xi_i) = p + (1-p)(E(\xi_i)+1) \implies E(\xi_i) = \frac{1}{p}$.

При этом для шага с номером i $p = \frac{n-i}{n}$.

Т.е. наш ответ это $\sum\limits_{i=0}^{n-1} \frac{n}{n-i} = n \sum\limits_{i=1}^{n} \frac{1}{i} \sim n \log n$, при $n \to \inf$.

Замечание.

Таким образом мы решили как задачу *Collection Problem*, так и задачу *Stable Marriage*.

Утверждение 6.7.

Пусть k — количество потребовавшихся шагов на получение всех купонов.

Тогда $Pr[k > \beta n \ln n] \leqslant \frac{1}{n^{\beta-1}}$

Доказательство.

Оценим вероятность, что купон с номером j не выпал ни разу за первые R шагов. Она равна $\left(1 - \frac{1}{n}\right)^R$, что не превышает $e^{-\frac{R}{n}}$.

Подставляем $R = \beta n \ln n$ и получаем, что вероятность не превышает $n^{-\beta}$.

Осталось домножить на количество купонов (т.е. получили оценку, что какой-то купон не выпал) и получим, что $Pr[k > \beta n \ln n] \leqslant n^{1-\beta}$. □

Утверждение 6.8.

Пусть k — количество потребовавшихся шагов на получение всех купонов.

Так же пусть $c \in \mathbb{R}$, $m = n \ln n + cn$.

Тогда $\lim\limits_{n\to\infty} P(k > m) = 1 - e^{-e^{-c}}$

Доказательство.

Пусть ξ_j^m — событие, отвечающие тому, что купон с номером j не выпал за m ходов.

Тогда $P(k > m) = P(\bigcup\limits_{j=1}^{n} \xi_j^m)$

Правую часть можно расписать по формуле включения-исключения. Для этого введем обозначения: $P_r^n := \sum\limits_{i_1<\ldots<i_r} P(\bigcap\limits_{i_1,\ldots,i_r} \xi_{i_j}^m) = \binom{n}{r}(1 - \frac{r}{n})^m \to \frac{e^{-ck}}{r!}$ (проверять не будем)

$S_r^n = P_1^n - P_2^n + P_3^n - \ldots \pm P_r^n$ — несколько слагаемых из формулы включения-исключения.

Посчитаем, к чему стремится S_r^n при $r, n \to \infty$.

$S_r^n = \frac{e^{-c}}{1!} - \frac{e^{-2c}}{2!} + \ldots \pm \frac{e^{-kc}}{k!} \to 1 - e^{-e^{-c}}$ (очевидно, из разложения в ряд).

При этом значение $P(\bigcup\limits_{j=1}^{n} \xi_j^m)$ можно зажать между соседними значениями S_r^n (тоже из формулы включения-исключения): $S_{2r}^n \leqslant Pr(k > m) \leqslant S_{2r+1}^n$.

Таким образом $P(k > m)$ зажата между двумя последовательностями стремящимся к $1 - e^{-e^{-c}}$, тогда $P(k > m)$ туда стремится. □

Лемма.

$\binom{n}{i} \leqslant \left(\frac{ne}{i}\right)^i$

Доказательство.

$\binom{n}{i} = \frac{n(n-1)\ldots(n-i+1)}{i!} \leqslant \frac{n^i}{i!} \leqslant \left(\frac{ne}{i}\right)^i$ (последний переход по формуле Стирлинга). □

Утверждение 6.9.

Есть n шариков и n корзин. Шарики случайно раскладываются по корзинам. Тогда вероятность того, что в какой-то корзине больше чем $k = \frac{3 \ln n}{\ln \ln n}$ шариков меньше чем $\frac{1}{n}$.

Доказательство.

Посчитаем вероятность, что в первой корзине ровно i шариков:

$\binom{n}{i} \cdot \left(\frac{1}{n}\right)^i \cdot \left(1 - \frac{1}{n}\right)^{n-i} \leqslant \left(\frac{ne}{i}\right)^i \left(\frac{1}{n}\right)^i = \left(\frac{e}{i}\right)^i$

$P(\text{в перовой корзине} \geqslant k) \leqslant \sum_{i=k}^{n} \left(\frac{e}{i}\right)^i \leqslant \left(\frac{e}{k}\right)^k \left(1 + \frac{e}{k} + \left(\frac{e}{k}\right)^2 + \ldots\right) \to \left(\frac{e}{k}\right)^k \frac{1}{1-\frac{e}{k}} \leqslant 2\left(\frac{e}{k}\right)^k \leqslant \frac{1}{n^2}$

Предпоследний переход верен с какого-то момента, т.к. $k \to \infty$, если $n \to \infty$.

Осталось доказать последнее неравенство. Прологарифмируем обе части и получим, что нам нужно проверить неравенство $\ln 2 + k - k \ln k \leqslant -2 \ln n$. Перенесем все в одну часть и оценим:

$\ln 2 + k - k \ln k + 2 \ln n = o(\ln n) - k \ln k + 2 \ln n = 2 \ln n - k(\ln 3 + \ln \ln n - \ln \ln \ln n) + o(\ln n) = 2 \ln n - k \ln 3 - 3 \ln n + o(\ln n) \leqslant o(\ln n) - \ln n$, что не больше 0 при достаточно больших n.

Таким образом, мы оценили вероятность, что в конкретной корзине больше чем k шариков как $\frac{1}{n^2}$. Тогда вероятность, что в какой-то корзине больше чем k шариков не превышает $\frac{1}{n}$. □

6.4. Оценки Чернова

***Определение* 6.9.**

Пусть $x_1, \ldots, x_n$ — независимые случайные величины, такие, что $x_i = 1$ с вероятностью p_i и 0 с вероятностью $1 - p_i$. Схема, в которой нас интересует величина $x = \sum x_i$ называется схемой Пуассона.

Замечание.

В некотором смысле, это обобщение схемы Бернулли на случай, когда монетки различны, т.е. если все p_i равны некоторому p, то получаем обычную схему Бернулли с n подбрасываниями и вероятностью выпадения орла равной p.

Теорема 6.10.

Пусть дана некоторая схема Пуассона и $\delta > 0$.

Тогда $Pr[x > (1+\delta)\mu] < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$, где $\mu = E(x)$ — математическое ожидание результата нашей схемы Пуассона.

Доказательство.

Заметим, что $Pr[x > (1+\delta)\mu] = Pr[e^{xt} > e^{t(1+\delta)\mu}]$ для любого $t > 0$.

Теперь мы можем оценить вероятность по неравенству Маркова: $Pr[e^{xt} > e^{t(1+\delta)\mu}] \leqslant \frac{E(e^{xt})}{e^{t(1+\delta)\mu}}$

Преобразуем числитель:

$E(e^{xt}) = E(e^{x_1 t + \ldots + x_n t}) = E(\prod e^{x_i t}) = \prod (E e^{x_i t}) = \prod (p_i e^t + (1 - p_i)) = \prod (1 + p_i(e^t - 1))$

Оцениваем каждую скобку неравенством $1 + x \leqslant e^x$ и получим:

$\prod (1 + p_i(e^t - 1)) \leqslant \prod e^{p_i(e^t - 1)} = e^{(e^t - 1)\sum p_i} = e^{\mu(e^t - 1)}$

Мы получили неравенство $Pr[x > (1+\delta)\mu] \leqslant \frac{e^{\mu(e^t-1)}}{e^{t(1+\delta)\mu}}$ для каждого положительного t.

Подставим в него $t = \ln(1+\delta)$ и получим требуемое.

□

Замечание.

Функция $E(e^{tx})$, которую мы использовали в доказательстве называется производящей функцией моментов (является экспоненциальной производящей функцией).

Действительно, $E(e^{tx}) = E(\frac{(tx)^0}{0!} + \frac{(tx)^1}{1!} + \ldots) = 1 + tE(x) + \frac{t^2}{2}E(x^2) + \ldots$, что и требовалось.

Обозначение.

Обозначим $\left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$ как $F^+(\mu, \delta)$.

Пример.

Пусть вероятность выпадения орла на монетке равна $\frac{1}{3}$. Тогда вероятность выпадения хотябы половины орлов будет экспоненциально мала.

Обозначим количество выпавших орлов за n подбрасываний как y_n, тогда $Ey_n = \frac{1}{3}$. Воспользуемся оценкой Чернова:

$Pr[y_n > \frac{n}{2}] = Pr[y_n > (1 + \frac{1}{2})Ey_n] \leqslant F^+(\frac{n}{3}, \frac{1}{2}) \approx (0.965)^n$

Теорема 6.11.

Пусть дана некоторая схема Пуассона и $\delta > 0$.

Тогда $Pr[x < (1 - \delta)\mu] < e^{\frac{-\mu\delta^2}{2}}$, где $\mu = E(x)$ — математическое ожидание результата нашей схемы Пуассона.

Доказательство.

Аналогично, $Pr[x < (1 - \delta)\mu] = Pr[e^{tx} < e^{t(1-\delta)\mu}] = Pr[e^{-tx} > e^{-t(1-\delta)\mu}]$

По неравенству Маркова:

$Pr[e^{-xt} > e^{-t(1-\delta)\mu}] < \frac{E(e^{-tx})}{e^{-t(1-\delta)\mu}}$

Оцениваем числитель:

$E(\prod e^{-tx}) = \prod(Ee^{-tx}) = \prod(p_i e^{-t} + (1 - p_i)) = \prod(1 + p_i(e^{-t} - 1))$

Так же оцениваем каждую скобку по неравенству $1 + x \leqslant e^x$ и получаем:

$\prod(1 + p_i(e^{-t} - 1)) \leqslant \prod e^{p_i(e^{-t}-1)} = e^{\sum p_i(e^{-t}-1)} = e^{(e^{-t}-1)\mu}$

Подставляем в полученное неравенство $t = \ln \frac{1}{1-\delta}$ и получаем

$Pr[x < (1 - \delta)\mu] \leqslant \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^\mu \leqslant e^{\frac{-\delta^2}{2}\mu}$

Последнее неравенство мы не доказываем. □

Обозначение.

Обозначим $e^{\frac{-\delta^2}{2}\mu}$ как $F^-(\mu, \delta)$.

Пример.

Пусть вероятность выпадения орла равна $\frac{3}{4}$. Тогда вероятность того, что орел выпадет в менее чем половине случаев экспоненциально мала.

Аналогично, обозначаем количество орлов как y_n, $Ey_n = \frac{3}{4}n$.

$Pr[y_n < \frac{n}{2}] = Pr[y_n < (1 - \frac{1}{3})Ey_n] \leqslant F^-(\frac{3}{4}n, \frac{1}{3}) \approx (0,9592)^n$.

Теорема 6.12.

Для $\delta \in [0, u]$, $F^+(\mu, \delta) \leqslant e^{-C(u)\mu\delta^2}$, где $C(u) = \frac{(1+u)\ln(1+u)-u}{u^2}$.

Доказательство.

Без доказательства. □

Замечание.

Это просто некоторый способ оценки функции $F^+(\mu, \delta)$. Мы никогда им не пользовались.

***Определение* 6.10.**

Пусть дана система линейных неравенств и линейная функция. Задачей линейного программирования является нахождения значений переменных, такие, что линейная функция принимает минимальное (или максимальное) возможное значение.

Задачей целочисленного линейного программирования является аналогичная задача над кольцом целых чисел.

Замечание.

Задача обычного линейного программирования решается за полиномиальное время. В то же время задача целочисленного линейного программирования является NP-трудной.

Замечание.

Если требуется добавить в систему равенство, то это можно сделать посредством добавления двух разносторонних неравенств.

Алгоритм.

Некоторый общий метод решения:

1. Свести задачу к задаче целочисленного линейного программирования
2. Решить ее как задачу простого линейного программирования.
3. Округлить ответ специальным образом.

Задача.

Дана доска $\sqrt{n} \times \sqrt{n}$. Несколько пар клеток отмечены одинаковыми цветами. Требуется соединить клетки каждой пары ломаной, проходящей по клеткам таблицы, и имеющей не более одного угла.

Пусть $c(e)$ — количество пересечений единичного ребра сетки e проведенными ломаными.

Требуется провести ломаные таким образом, чтобы $\max c(e)$ по всем e было минимально.

Алгоритм.

Попробуем решить нашу задачу описанным выше образом. Сделаем три описанных шага:

Шаг 1: Свести задачу к LIP

Для каждой пары концов у нас есть два варианта как провести ломаную. Заведем две переменные x_i и y_i, так, что если ломаная проведена по верхней и правой стороне квадрата, то $x_i = 0$, $y_i = 1$, а если по левой и нижней, то $x_i = 1$, $y_i = 0$. Таким образом мы имеем равенство $x_i + y_i = 1$. Так же нужно ограничить значения x_i и y_i числами из множества $\{0, 1\}$. Для этого добавляем неравенства $0 \leqslant x_i, y_i \leqslant 1$.

Далее, для каждого ребра (к примеру для c) запишем количество пересекающих его ребер. Это число равно $\sum x_i + \sum y_i$, где суммирование ведется по пересекающим c ломаным. Таким образом мы получили систему неравенств $\sum x_i + \sum y_i \leqslant \omega$. При этом, мы хотим минимизировать значение ω, т.е. мы получили задачу LIP.

Шаг 2: Решить ее как задачу обычного линейного программирования.

Тут ничего сложного, просто решаем. Получаем решение вида $\tilde{x_i}$, $\tilde{y_i}$, $\tilde{\omega}$. Где все $x_i, y_i \in [0, 1]$.

Шаг 3: Округлить ответ специальным образом

Для каждого x_i выберем значение равное 1 с вероятностью $\tilde{x_i}$ и 0 с оставшейся вероятностью. Значение для y_i будет получено автоматически. Полученные значения обозначим как $\hat{x_i}$ и $\hat{y_i}$. Соответственно новое значение для ω обозначим как $\hat{\omega}$. Осталось оценить ошибку.

Оцениваем вероятность ошибки.

Оценим какую точность можно гарантировать с ошибкой не больше ε.

Значение каждой из x_i, y_i равно 0 или 1 с вероятностями $\tilde{x_i}$ и $\tilde{y_i}$ соответственно, при этом одна из переменных равна 0, а другая 1, т.е. мы получили схему Пуассона для каждого из неравенств $\sum \hat{x_i} + \sum \hat{y_i}$. Обозначим сумму как s_i.

Мы умеем оценивать вероятность того, что такая сумма больше $(1 + \delta)\mu$, где μ — математическое ожидание. Посчитаем μ.

$\mu = Es_i = E(\sum \hat{x}_i + \sum \hat{y}_i) = \sum E\hat{x}_i + \sum E\hat{y}_i = \sum \tilde{x}_i + \sum \tilde{y}_i \leqslant \tilde{\omega}$.

Оцениваем $Pr[s_i > (1+\delta)\tilde{\omega}] < Pr[s_i > (1+\delta)\mu] < F^+(\mu, \delta)$.

Выбираем такое δ, что $F^+(\mu, \delta) = \frac{\varepsilon}{2n}$.

Тогда мы получили, что вероятность того, что мы ошиблись в неравенстве с номером i больше чем в $1+\delta$ раз не более $\frac{\varepsilon}{2n}$. Всего неравенств $2n$, а значит вероятность, ошибки хотя бы в одном не больше чем $2n \cdot \frac{\varepsilon}{2n} = \varepsilon$.

Т.е. с вероятностью ошибки ε мы можем гарантировать ошибку в $1+\delta$ раз, где $F^+(\mu, \delta) = \frac{\varepsilon}{2n}$.

Замечание.

Почему результат хороший (т.е. δ достаточно маленькое), выяснять не будем.

24 апреля 2018

6.5. Задача поиска пути в графе

Мотивация.

Наша цель — описать RP алгоритм проверки наличия пути из вершины s в вершину t в неориентированном графе использую $\mathcal{O}(\log n)$ памяти, т.е. если путь существует, то алгоритм вернет 1 с вероятностью большей чем $\frac{1}{2}$, а если пути нет, то алгоритм гарантировано вернет 0.

Алгоритм.

Сразу заметим, что можно считать, что граф не содержит кратных ребер, т.к. они не влияют на наличие пути.

Опишем алгоритм. Будем в каждый момент хранить вершину, в которой находимся и количество совершенных итераций. На очередной итерации выбираем случайное ребро из вершины в которой находимся и переходим по нему. Если в какой-то момент достигнута требуемая конечная вершина, то алгоритм сразу вернет 1. Иначе, после $poly(n)$ шагов алгоритм вернет 0. Многочлен $poly(n)$ выберем позже.

Заметим, что мы можем считать, что в графе ровно одна компонента связности, т.к. нас интересуют лишь ситуации в которых мы не нашли существующий путь, а понижение количество итераций лишь повышает вероятность ошибки.

Для удобства добавим к графу петли так, чтобы число петель в сумме с числом ребер было константой для всех вершин (понятно, что петли лишь повышают вероятность ошибки нашего алгоритма).

Рассмотрим матрицу смежности и назовем ее A_G. Пусть в каждой строке сумма чисел будет равна R (она одинакова по построению). Тогда рассмотрим матрицу $A = \frac{A_G}{R}$.

Дополнительно нужно заметить, что $R \leqslant n$, т.к. мы удалили все кратные ребра. Это будет важно ближе к концу доказательства.

Н.у.о. будем считать, что алгоритм начинает работать с вершины с номером 1.

Введем вектора $p_k \in \mathbb{R}^n$ по следующему правилу: координата i вектора p_k равна вероятности оказаться в вершине с номером i после k итераций. В частности, $q_0 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$.

Заметим, что на p_k есть следующая рекуррента: $p_{k+1} = Ap_k$.

Идея решения

Пусть мы нашли такой полином $P(n)$, что $A^{P(n)}p \sim (\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n})$ для всех p с нормой 1.

Тогда, в частности, при $p = p_0$ получим, что через $P(n)$ шагов вероятность оказаться в конечной вершине примерно равна $\frac{1}{n}$. Далее, если повторим алгоритм n раз и получить вероятность ошибки, примерно равную $(1 - \frac{1}{n})^n \to \frac{1}{e}$, т.е. константную.

Вычисление ошибки

Оценим значение $A^m p$.

Известно, что матрица A — симметричная, а значит, существует ортонормированный базис $u_1, \dots u_n$ состоящий из собственных векторов (собственные числа соответственно будут $\lambda_1, \dots, \lambda_n$).

Столбец из всех единичек является собственным вектором для числа 1, а значит, можно считать, что $\lambda_1 = 1$.

Выразим через базис $u_1, \dots, u_n$ вектор $p = c_1 u_1 + \dots c_n u_n$.

Т.к. норма p равна 1, то $1 = \|\sum c_i u_i\| = \sqrt{\sum c_i^2} \geqslant |c_i| \implies |c_i| \leqslant 1$.

Считаем $A^m p$, и получаем: $A^m p = A^m \sum c_i u_i = \sum c_i \lambda_i^m u_i = c_1 u_1 + \sum \lambda_i^m \cdot c_i u_i$.

Пусть мы показали, что $|\lambda_i| \leqslant 1 - \frac{1}{poly(n)}$ при $i \in [2..n]$. Тогда выбрав $m = poly(n) \cdot \log n^{10}$ получим, что $\lambda_i^m = \left(1 - \frac{1}{poly(n)}\right)^m < \frac{1}{n^{10}}$, а значит вся сумма $\sum \|\lambda_i^m \cdot c_i u_i\|$ не превышает $\frac{1}{n^9}$.

Т.е. мы получили, что $A^m p = c_1 u_1 + \mathcal{O}(\frac{1}{n^9})$. Это означает, что каждая из координат $A^m p$ равна $c_1 + \mathcal{O}(\frac{1}{n^9})$, а т.к. сумма всех координат равна 1, то $c_1 = \frac{1}{n}$, что и требовалось (ошибка на $\mathcal{O}(\frac{1}{n^9})$ не влияет на значение предела).

Доказательство оценки на λ_i.

Покажем, что $|\lambda_i| < 1 - \frac{1}{poly(n)}$ при $i \in [2..n]$.

$\max\limits_{2 \leqslant i \leqslant n} \lambda_i = \max \|Au\|$, где максимум берется по всем единичным векторам из пространства $(1, 1, \dots, 1)^\perp$. Действительно, мы выделили подпространство, которое образуют векторы $u_2, \dots, u_n$ и нашли на нем максимум $\|Au\|$. Из курса линейной алгебры известно, что он совпадает с максимальным собственным числом.

Таким образом, осталось показать, что $\|Au\| \leqslant 1 - \frac{1}{poly(n)}$ для всех $u \in (1..1)^\perp$. Рассмотрим какой-то вектор u и обозначим вектор Au как v.

Введем магическую сумму $S = \sum\limits_{i,j} A_{i,j}^n (u_i - v_j)^2$. С одной стороны, S можно вычислить:

$$S = \sum_{i,j}^n A_{ij} u_i^2 + \sum_{i,j}^n A_{ij} v_j^2 - 2 \sum_{i,j}^n A_{ij} u_i u_j = \|u\|^2 + \|v\|^2 - 2\langle Au, v\rangle = \|u\|^2 + \|v\|^2 - 2\|v\|^2 = 1 - \|v\|^2.$$

Пусть $S \geqslant \frac{2}{poly(n)}$. Тогда $S = 1 - \|v\|^2 \geqslant \frac{2}{poly(n)} \implies \|v\| \leqslant 1 - \frac{1}{poly(n)}$, что и требовалось.

Оценка на S

Осталось показать, что $S \geqslant \frac{2}{poly(n)}$. Выберем $poly(n) = 16n^4$. Тогда мы хотим $S \geqslant \frac{1}{8n^4}$.

Будем дополнительно считать, что в исходном графе есть хотя бы одна петля в каждой вершине (опять же, добавление петель лишь повысит вероятность ошибки алгоритма). Тогда можно оценить диагональные элементы матрицы A как $A_{i,i} \geqslant \frac{1}{R} \geqslant \frac{1}{n}$.

Предположим, что $\exists i : |u_i - v_i| \geqslant \frac{1}{2n^{3/2}}$. Тогда $S \geqslant \frac{1}{4n^4} \geqslant \frac{1}{8n^4}$, что и требовалось.

Теперь пусть $\forall i : |u_i - v_i| \leqslant \frac{1}{2n^{3/2}}$.

Рассмотрим $u = (u_1, \dots u_n)$. Н.у.о. $u_1 \geqslant u_2 \dots \geqslant u_n$.

Мы знаем, что $\|u\| = 1$, и $u \in (1, 1, \dots, 1)^\perp$. Это означает, что $\sum u_i^2 = 1$ и $\sum u_i = 0$.

Это означает, что либо $u_1 \geqslant \frac{1}{\sqrt{n}}$, либо $u_n \leqslant -\frac{1}{\sqrt{n}}$, т.е. $u_1 - u_n \geqslant \frac{1}{\sqrt{n}}$.

С другой стороны $u_1 - u_n = (u_1 - u_2) + (u_2 - u_3) + \ldots + (u_{n-1} - u_n)$, а значит $\exists i : u_i - u_{i+1} \geqslant \frac{1}{n\sqrt{n}}$.

Вернемся к графу. Пусть $T = \{u_1, \ldots u_i\}$, а $\overline{T} = \{u_{i+1} \ldots u_n\}$, соответственно.

Т.к. граф связен, то между T и $\overline{T}$ есть ребро, т.е. $A_{lk} \geqslant \frac{1}{n}$ (l и k — номера концевых вершин этого ребра). При этом $l \leqslant i \leqslant i+1 \leqslant k \implies u_l - u_k \geqslant \frac{1}{n\sqrt{n}}$.

Далее, $|u_l - v_k| \geqslant |u_l - u_k| - |u_k - v_k| \geqslant \frac{1}{n\sqrt{n}} - \frac{1}{2n\sqrt{n}} = \frac{1}{2n\sqrt{n}}$, а тогда соответствующее слагаемое $A_{lk}(u_l - u_k)^2 \geqslant \frac{1}{4n^4} \geqslant \frac{1}{8n^4}$.

6.6. Снова понижение ошибки

Мотивация.

Рассмотрим RP алгоритм, использующий $\log n$ случайных битов.

Мы знаем, что если повторить алгоритм $\log n$ раз, то вероятность ошибки будет $\frac{1}{2^{\log n}} = \frac{1}{n}$, при использованных $\log^2 n$ случайных бит.

***Определение* 6.11.**

Рассмотрим двудольный граф $G[X, Y]$ такой что $|X| = n$, $|Y| = n^{\log n} = 2^{\log^2 n}$ обладающий следующими свойствами:

1. $\forall v \in Y : \deg v \leqslant 12 \log^2 n$
2. $\forall S \subseteq X, |S| = \frac{n}{2} : |N(S)| \geqslant n^{\log n} - n$.

Утверждение 6.13.

Пусть A — RP алгоритм использующий $\log n$ случайных бит. Тогда можно понизить вероятность ошибки RP алгоритма A до $\frac{n}{n^{\log n}}$ использовав $\log^2 n$ случайных бит.

Доказательство.

Рассмотрим описанный выше граф.

Каждая вершина левой части соответствует набору случайных бит длины $\log n$. Обозначим такой набор как $\texttt{bits}(\texttt{u})$.

Сгенерируем $\log^2 n$ случайных бит. Они определят ровно одну вершинку правой доли v.

Рассмотрим множество соседей вершины v и запустим алгоритм A используя каждый из наборов случайных бит среди соседей, т.е. $\forall u \in N(v) \ : A(x, \texttt{bits}(\texttt{u}))$.

Оценим вероятность ошибки.

Рассмотрим множество вершин $S \subseteq X$, на которых A работает корректно. Тогда, т.к. $A \in \texttt{RP}$, то $|S| \geqslant \frac{n}{2}$, а значит $N(S) \geqslant n^{\log n} - n$.

Если проверив всех соседей v мы ошиблись, то $v \notin N(S)$, а таких вершин не более n. Это означает, что вероятность ошибки не превышает $\frac{n}{n^{\log n}}$, что и требовалось. □

Замечание.

Дополнительно заметим, что мы запустили A не более чем $12 \log^2 n$ раз, в то время когда прежний подход требовал $\log n$ запусков, т.е. асимптотика времени работы хоть и ухудшилась, но все еще осталось полиномиальна.

Теорема 6.14.

Графы описанного вида существуют для каждого достаточно большого n.

Доказательство.

Сгенерируем случайный граф, а именно проведем $d = n^{\log n} \cdot \frac{4\log^2 n}{n}$ случайных ребер для каждой вершины левой доли (ребра могут повторяться, в таком случае мы все равно проводим лишь одно ребро).

Проверим, что требуемые свойства выполняются с вероятностью большей $\frac{1}{2}$. Это будет означать, что существует граф, для которого будут выполнены оба свойства.

Проверим второе свойство

Рассмотрим множество $S : |S| = \frac{n}{2}$.

Пусть для него условие не выполняется. Это означает, что существует хотя бы n вершин правой доли, такие, что для каждой вершины из S мы не провели в них ребро. Для фиксированного множества вершин правой доли эта вероятность равна $\left(1 - \frac{n}{n^{\log n}}\right)^{\frac{dn}{2}} = p$

Тогда вероятность, что такое множество из n вершин существует не превышает $p \cdot \binom{n^{\log n}}{n}$. Осталось домножить на количество способов выбрать множество S получить оценку на вероятность ошибки равную $p \cdot \binom{n^{\log n}}{n} \cdot \binom{n}{n/2}$, что меньше $\frac{1}{2}$ (проверять не будем).

Проверим первое свойство

Рассмотрим конкретную вершину в правой части u. Обозначим как ξ — случайную величину равную степени u. Тогда из линейности математического ожидания: $E\xi = \frac{dn}{n^{\log n}} = 4\log^2 n$.

Рассмотрим величину $s = x_1 + x_2 + \ldots + x_n$, где все x_i отвечают событию вида: проведено ребро из вершины левой доли с номером i в вершину u.

Напишем оценку Чернова: $Pr[s \geqslant 12\log^2 n] = Pr[s \geqslant (1+2)\mu] \leqslant F^+(\mu, 2) = \left(\frac{e^2}{27}\right)^{\mu} \leqslant \left(\frac{e}{3}\right)^{12\log^2 n}$.

Осталось оценить вероятность ошибиться для одной вершины. Оцениваем, и получаем, что она не превышает $\left(\frac{e}{3}\right)^{12\log^2 n} \cdot 2^{\log^2 n} = \left(\frac{2e^{12}}{3^{12}}\right)^{\log^2 n}$.

Число в скобках меньше 1, а значит для больших n все доказано. □

7. Схемы

25 апреля 2018

7.1. Основные определения

***Определение* 7.1.**
Схема — это ациклический граф, такой что:

1. В нем существует n истоков (вершины входящей степени 0).
2. Существует ровно 1 сток (вершина исходящей степени 0).
3. Каждая вершина кроме истоков помечена одним из символов $\wedge, \vee, \neg$.
4. Входящая степень каждой вершины соответствует арности написанного на ней оператора (она равна двум для $\wedge$ и $\vee$ и одному для $\neg$).

Размером схемы называется количество вершин в соответствующем ей графе, а число n размером входа схемы.

***Определение* 7.2.**
Если C — схема с размером входа n, а $w \in \{0,1\}^n$ — бинарное слово длины n, то результатом работы схемы будет значение в истоке вычисленное согласно следующим правилам:

1. Значение истока с номером i равно i-ому биту слова w (входа).
2. Значение в вершине, помеченной оператором, вычисляется согласно этому оператору (значения аргументов берутся из противоположных концов входящих ребер).

Замечание.
В целом определения являются довольно формальными. На самом деле мы все знаем что такое булева схема из курса дискретной математики.

***Определение* 7.3.**
Пусть $T : \mathbb{N} \to \mathbb{N}$ — функция.

Семейством схем размера $T(n)$ называется множество схем $= \bigcup\limits_{k \in \mathbb{N}} C_k$, такое, что размер схемы C_k равен $T(k)$ (k — размер входа соответствующей схемы).

Замечание.
Теперь мы можем понимать под результатом работы семейства C схем над словом $w \in \{0,1\}^*$, как результат работы схемы $C_{|w|}$ на входе w. Результат работы обозначим как $C(w)$.

***Определение* 7.4.**
Язык L решается схемой размера $T(n)$, если существует семейство схем C размера $T(n)$, такое, что $x \in L \iff C(x) = 1$

***Определение* 7.5.**
Класс $\texttt{Size}[T(n)]$ — множество языков, решаемых схемами размера не больше чем $T(n)$.

Обозначение.
$\texttt{P}_{/\texttt{poly}} := \bigcup_{c \geqslant 1} \texttt{Size}[\texttt{n}^\texttt{c}]$.

Лемма.
Если L — унарный язык, то $L \in \mathtt{P}_{/\mathtt{poly}}$.

Доказательство.
Построим семейство схем C размера $\mathcal{O}(n)$, решающих язык L.

Построим схему C_k. Если 1^k лежит в языке L, то C_k будет схемой, считающих конъюнкцию всех входных битов, если же 1^k не лежит в языке, то C_k будет схемой, возвращающей тождественный 0. □

Утверждение 7.1.
$\mathtt{P} \neq \mathtt{P}_{/\mathtt{poly}}$.

Доказательство.
Рассмотрим язык L не решаемый на машине Тьюринга.

Построим унарный язык L_1, состоящий из слов языка L, записанных в унарной записи. Покажем, что L_1 нам подойдет.

Принадлежность языка L_1 классу $\mathtt{P}_{/\mathtt{poly}}$ следует леммы. Осталось показать, что $L_1 \notin P$. Действительно, если бы мы умели за полиномиальное время решать язык L_1, то умели бы решать и язык L за экспоненциальное время (перевести слово в унарный вид и проверить), мы же рассмотрели такой L, что он не решается на машине Тьюринга. □

Замечание.
Более того, существует разрешимый язык, лежащий в $\mathtt{P}_{/\mathtt{poly}}$, но не лежащий в $\mathtt{P}$.

Теорема 7.2.
$\mathtt{P} \subset \mathtt{P}_{/\mathtt{poly}}$

Доказательство.
Нестрогость мы уже показали. Покажем, что любой $L \in \mathtt{P}$ так же лежит и в $\mathtt{P}_{/\mathtt{poly}}$.

Для языка L существует забывающая машина Тьюринга M работающая за полиномиальное время $T(n)$. Построим систему схем, решающих L.

Построим схему C_k, т.е. схему с k входами, принимающую только слова из языка L. Утверждается, что мы можем построить схему C_k так, что она моделирует работу M. Действительно, наша схема будет состоять из $T(n)$ уровней, где на уровне с номером i будет вычислено состояние ленты и головки машины M после i шагов, в частности, на нулевом уровне будет записан вход и состояние головки q_0.

Теперь заметим, что существует схема C_δ, принимающая на вход состояние головки и символ, на который она указывает и вычисляющая новое состояние головки, новый символ и сдвиг (т.е. вычисляющая δ). Действительно, δ является конечной функцией, а значит вычисляется какой-то схемой. Важно заметить, что размер такой схемы будет константой относительно n.

Осталось построить переходы между уровнями. На каждом уровне мы знаем, на какую именно клетку указывает головка и ее состояние, а значит, мы можем применить функцию перехода. Вставим между двумя уровнями схему C_φ и перезапишем наше состояние согласно ее выходу. □

Замечание.
Тем, что машина забывающая мы пользовались в том, что мы на каждом уровне знаем куда именно указывает головка машины и благодаря этому мы могли зафиксировать вход для схемы C_φ.

Следствие.

$\mathtt{NP} \not\subset \mathtt{P}_{/\mathtt{poly}} \implies \mathtt{P} \neq \mathtt{NP}$.

7.2. Машина Тьюринга с советом

***Определение* 7.6.**

Машина M называется машиной Тьюринга с советом, если для каждого n дополнительно задана подсказка α_n.

M распознает язык L если существует машина Тьюринга $M' : x \in L \iff M'(x, \alpha_{|x|}) = 1$.

***Определение* 7.7.**

Класс $\mathtt{Size}(T(n))/a(n)$ состоит из таких языков L, для которых существует машина Тьюринга с советом M, работающая время $T(n)$, такая, что M распознает L и размер α_n не превышает $a(n)$.

Утверждение 7.3.

$\mathtt{P}_{/\mathtt{poly}} = \bigcup_{c,d} \mathtt{Size}(\mathtt{n}^\mathtt{d})/\mathtt{n}^\mathtt{c}$.

Доказательство.

Шаг первый

Покажем, что $\mathtt{P}_{/\mathtt{poly}} \subseteq \bigcup_{c,d} \mathtt{Size}(\mathtt{n}^\mathtt{d})/\mathtt{n}^\mathtt{c}$

Рассмотрим $L \in \mathtt{P}_{/\mathtt{poly}}$.

Нам нужно задать все подсказки α_i. Зададим подсказку с номером k.

Известно, что существует схема C_k полиномиального размера, такая что:
$x \in L \iff C_k(x) = 1$ для всех слов длины k. Закодируем схему C_k и выберем как подсказку.

Машина M же будет моделировать работу переданной в качестве подсказки схемы, т.е. $M(x, \alpha) = C_\alpha(x)$ (в качестве C_α мы обозначили схему, соответствующую строке α).

Тогда очевидно, что $x \in L \iff M(x, \alpha) = 1$.

Шаг второй

Покажем включение в другую сторону, т.е. $\mathtt{P}_{/\mathtt{poly}} \supseteq \bigcup_{c,d} \mathtt{Size}(\mathtt{n}^\mathtt{d})/\mathtt{n}^\mathtt{c}$.

Построим систему схем C_k решающих язык L.

Построим C_k. Нам известна строка α, такая что $M(x, \alpha_k) = 1 \iff x \in L$ для слов длины k. Машину $M(x, \alpha)$ можно рассматривать как какую-то фиксированную машину $M_{\alpha_k}(x)$, которая будет работать аналогично M. Это означает, что существует полиномиальная схема C_k соответствующая машине M_{α_k}, т.к. ее время работы полиномиально. Такая C_k нам подойдет. □

Теорема 7.4.

Если $\mathtt{NP} \subseteq \mathtt{P}_{/\mathtt{poly}}$, то $\mathtt{PH} = \sum_2^p$.

Доказательство.

Пусть $\mathtt{NP} \subseteq \mathtt{P}_{/\mathtt{poly}}$.

Покажем, что $\prod_2^p \subseteq \sum_2^p$. Для этого достаточно показать, что $\prod_2^p \mathtt{SAT} \in \sum_2^p$.

Рассмотрим язык $\mathtt{SAT}$. Он лежит в классе $\mathtt{NP}$, а значит по предположению существует решающая $\mathtt{SAT}$ полиномиальная схема $C_{\mathtt{SAT}}$.

На практике мы показывали, что существует полиномиальная машина Тьюринга, использующая оракульное обращение к языку $\mathtt{SAT}$, которая находит выполняющий набор. Рассмотрим соответствующую ей полиномиальную схему $\tilde{C}_{\mathtt{SAT}}$.

$\tilde{C}_{\texttt{SAT}}$ иногда использует оракульный доступ к языку SAT, для решения которого существует полиномиальная схема $C_{\texttt{SAT}}$. Это означает, что мы можем построить полиномиальную систему схем C, находящую выполняющий набор формулы, поданной на вход подставив вместо обращений оракульных схему C_φ.

Таким образом мы получили, что существует система схем умеющая находить выполняющий набор.

Рассмотрим язык $L = \prod_2^p \texttt{SAT} = \{\varphi : \forall u \exists v : \varphi(u, v) = 1\}$ и докажем, что он лежит в $\sum_2^p$.

Передадим в качестве первой подсказки конкретный экземпляр схемы C, находящей решения для формул длины φ. Внутренняя же машина Тьюринга $M(\varphi, C, u)$ будет вычислять $\varphi(u, C(\varphi(u, v)))$, т.е. заниматься проверкой выполнимости формулы $\varphi(u)$ или иными словами, что $\exists v : \varphi(u, v)$.

Осталось показать, что $\varphi \in L \iff \exists c \forall u : M(\varphi, c, u) = 1$. Действительно, пусть $\varphi \in L$. Тогда передав описанный экземпляр схемы получим, что $\varphi \in L \iff \forall u : \varphi(u, C(\varphi(u))) = 1$, т.е. что $\forall u \exists v : \varphi(u, v) = 1$.

С другой стороны, если $\varphi \notin L$, то $\exists u \forall v : \varphi(u, v) = 0$. Тогда какую бы схему c мы не подставили $M(\varphi, c, u)$ вернет 0, что и требовалось.

Таким образом мы показали, что $\prod_2^p \subseteq \sum_2^p$. Осталось заметить, что из этого следует, что $\texttt{PH} = \sum_2^p$ (**TODO:** раньше было, но для равенства). □

Теорема 7.5.
Если $\texttt{EXP} \subseteq \texttt{P}_{/\texttt{poly}}$, то $\texttt{EXP} = \sum_2^p$.

Доказательство.
Без доказательства. □

Следствие.
Если $\texttt{P} = \texttt{NP}$, то $\texttt{EXP} \not\subseteq \texttt{P}_{/\texttt{poly}}$.

Доказательство.
От противного. Пусть $\texttt{P} = \texttt{NP}$, но при этом $\texttt{EXP} \subseteq \texttt{P}_{/\texttt{poly}}$. Тогда из теоремы $\texttt{EXP} = \sum_2^p$.

При этом, т.к. $\texttt{P} = \texttt{NP}$, то $\texttt{PH} = \texttt{P}$, в частности $\texttt{P} = \sum_2^p = \texttt{EXP}$.

Противоречие, т.к. мы знаем, что $\texttt{P} \neq \texttt{EXP}$. □

Теорема 7.6.
Существует функция $f : \{0,1\}^n \to \{0,1\}^n$ не вычислимая схемой размера $\frac{2^n}{10n}$

Доказательство.
Было на дискретной математике в первом семестре (Теорема 1.2.3 конспекта). □

Замечание.
При этом схемами размера $n2^n$ вычисляется любая функция, т.к. мы можем честно написать КНФ формулу.

В частности, это значит, что $f \in \texttt{Size}(n2^n)$ для любого f.

Теорема 7.7 (Теорема о иерархии)**.**
Если $\frac{2^n}{n} > T'(n) > 10 \cdot T(n) > n$, то $\texttt{Size}[T(n)] \not\subseteq \texttt{Size}[T'(n)]$.

Доказательство.
Мы покажем лишь, что $\texttt{Size}(n) \not\subseteq \texttt{Size}(n^2)$.

Рассмотрим f, найденную из теоремы выше.

Введем функцию $g(w) = f(x)$, где $w = xy$, а длина x равна $1.1 \log n$, т.е. g берет первые несколько бит входа и вычисляет на них функцию f.

Покажем, что g нам подходит, т.е. что $g \in \mathtt{Size}(n^2)$ но $g \notin \mathtt{Size}(n)$.

Обозначим $l = 1.1 \log n$, т.е. l — длина x.

Из определения g: $g(x, y) \notin \mathtt{Size}(\frac{2^{\mathtt{l}}}{10\mathtt{l}}) = \mathtt{Size}(\frac{\mathtt{n}^{1.1}}{11 \log \mathtt{n}}) \supseteq \mathtt{Size}(\mathtt{n})$, что и требовалось.

С другой стороны, $g(x, y) \in \mathtt{Size}(10\mathtt{l} \cdot 2^{\mathtt{l}}) = \mathtt{Size}(11\mathtt{n}^{1.1} \log \mathtt{n}) \subseteq \mathtt{Size}(\mathtt{n}^2)$.

□

Замечание.

Утверждается, что похожее доказательство работает для любых функций T и T'.

***Определение* 7.8.**

Глубина схемы — самый длинный путь из истока в сток.

Замечание.

В теории, алгоритм можно вычислять за глубину схемы, если распараллелеить.

***Определение* 7.9.**

Класс языков $\mathtt{NC}^{\mathtt{i}}$ — содержит такие языки L, что они решаются схемами полиномиального размера и глубины $\mathcal{O}(\log^i n)$.

Замечание.

$\mathtt{NC}^{\mathtt{i}} \subseteq \mathtt{NC}^{\mathtt{i}+1}$

***Определение* 7.10.**

$\mathtt{NC} = \bigcup_{i=1}^{\infty} \mathtt{NC}^{\mathtt{i}}$.

***Определение* 7.11.**

Разрешим узлам схемы принимать на вход $p(n)$ бит, где p — полином.

Класс языков, разрешимых такими схемами полиномиального размера и глубиной $\mathcal{O}(\log^i n)$ назовем $\mathtt{AC}^{\mathtt{i}}$.

Аналогично, $\mathtt{AC} = \bigcup_{i=1}^{\infty} \mathtt{AC}^{\mathtt{i}}$.

Утверждение 7.8.

$\mathtt{NC}^{\mathtt{i}} \subseteq \mathtt{AC}^{\mathtt{i}} \subseteq \mathtt{NC}^{\mathtt{i}+1}$

Доказательство.

Первое включение очевидно. Докажем второе.

Рассмотрим схему для языка из класса $\mathtt{AC}^{\mathtt{i}}$ и преобразуем ее в схему из класса $\mathtt{NC}^{\mathtt{i}+1}$.

Заменим каждое вхождение узла большой арности на схему, вычисляющую значение в этом узле. Для этого нам нужно посчитать дизъюнкцию или конъюнкцию всех входных бит. Это делается схемой полиномиального размера и логарифмической глубины, а значит, т.к. количество входных бит для каждого узла было полиномиально мы получим схему глубины $\mathcal{O}(\log^{i+1} n)$, что и требовалось. □

8 мая 2018

8. Дополнительно

8.1. Немного про класс BPP

***Определение* 8.1.**

Класс BPP — множество языков L, таких что для них существует вероятностная машина Тьюринга M такая что:

$x \in L \iff Pr[M(x) = 1] > \frac{2}{3}$

$x \notin L \iff Pr[M(x) = 1] \leqslant \frac{1}{3}$

Свойства.

1. $\mathtt{BPP} = \mathtt{coBPP}$
2. Константу $\frac{2}{3}$ можно заменить на любую строго большую $\frac{1}{2}$.
3. Константу $\frac{2}{3}$ можно заменить на $1 - \frac{1}{2^{p(n)}}$, где $p(n)$ — любой полином.

Доказательство.

Считаем, что было. □

Теорема 8.1 (Адлемана)**.**

$\mathtt{BPP} \subseteq \mathtt{P}_{/\mathtt{poly}}$

Доказательство.

Пусть язык $L \in \mathtt{BPP}$. Тогда для него существует полиномиальная машина M:

$x \in L \implies Pr[M(x) = 1] \geqslant 1 - \frac{1}{2^{n+2}}$

$x \notin L \implies Pr[M(x) = 1] \leqslant \frac{1}{2^{n+2}}$

Пусть M использует m случайных бит.

Тогда для любого x существует не более чем $\frac{2^m}{2^{n+2}}$ наборов случайных бит r, таких, что $M(x, r) \neq L(x)$ (т.е. таких, что при их использовании машина ошибается).

Тогда всего плохих наборов (таких, что $\exists x : M(x, r) \neq L(x)$) не больше чем $2^n \cdot \frac{2^m}{2^{n+2}} = \frac{2^m}{4}$. Всего же различных подсказок 2^m, а значит существует такой набор случайных бит, что $\forall x : M(x, r) = L(x)$.

Таким образом мы нашли "хороший" набор для каждой длины входа. Обозначим его как r_n.

Осталось сконструировать машину Тьюринга с советом решающую язык L.

Пусть она для каждой длины входа в качестве совета использует найденный набор r_n и считает значение $M(x, r_n)$. Из принципа выбора r_n следует, что $M(x, r_n) = L(x)$, что и требовалось. □

Теорема 8.2 (Сипсера-Гача)**.**

$\mathtt{BPP} \subseteq \sum_2^p \cap \prod_2^p$

Доказательство.

Покажем, что $\mathtt{BPP} \subseteq \sum_2^p$. Пусть $L \in \mathtt{BPP}$.

Рассмотрим машину Тьюринга M для решения L с ошибкой не превышающей $\frac{1}{2^n}$. Обозначим количество используемых машиной M случайных бит как m.

Пусть $S \subseteq \{0,1\}^m$ — множество наборов случайных бит, на которых M возвращает единицу, а $\overline{S}$ — на которых ноль.

Покажем, что $x \in L \iff \exists u_1, \dots u_k : \forall r \bigvee [M(x, r \oplus u_i) = 1]1$ (под $\oplus$ понимается побитовое сложение, т.е. в каждом u_i ровно m бит)

Существует два принципиально разных случая.

Если $x \notin L$

Нужно показать, что не существует $u_1, u_2, \dots u_k$, таких, что для каждого r $M(x, r + \oplus u_i) = 1$ для какого-то из u_i. Действительно, для каждого u_i существует не более чем $\frac{2^m}{2^n}$ различных r, на которых $M(x, r \oplus u_i) = 1$ (из определения BPP). Тогда количество различных r для которых хотябы один из u_i подойдет не превышает $\frac{k2^m}{2^n}$, что меньше общего количества случайных наборов (2^m) при $k < 2^n$.

Если $x \in L$

Нужно показать, что существует требуемый набор u_i. Для этого докажем следующую лемму:

Лемма.
Пусть $S \subseteq \{0,1\}^m$, такое что $|S| \geqslant 2^m(1 - \frac{1}{2^n})$.

Тогда существует $k \leqslant m$, такое что $\exists u_1, \dots u_k : \forall v \exists j : v \in u_j \oplus S$.

Доказательство.
Покажем, что $k = m$ подойдет.

Выбираем случайные u_i.

$v \in S \oplus u_j \iff u_j \in S \oplus v$.

$Pr[v \in S \oplus u_j] = Pr[u_j \in S \oplus v] = \frac{|S|}{2^m} = 1 - \frac{1}{2^n}$.

Т.е. вероятность того, что конкретный вектор v не покрыт случайным u_i равна $\frac{1}{2^n}$, а вероятность, что v не покрыт никаким из векторов u_i равна $\frac{1}{2^n}^k$.

Тогда вероятность, что какой-то вектор не покрыт не превышает $2^m \cdot \left(\frac{1}{2^n}\right)^k = \frac{2^m}{2^{nk}}$, а значит при $k = m$ вероятность будет строго меньше 1, т.е. требуемый набор найдется. □

Теперь можно закончить доказательство. Т.к. $m < 2^n$ (для достаточно больших n из определения BPP), то k найденное из леммы подойдет, т.е. язык L лежит в классе $\sum_2^p$.

Осталось заметить, что тогда coBPP $\subseteq \prod_2^p$, а т.к. coBPP = BPP, то получаем требуемое. □

8.2. Теорема Вэлианта-Вазирани

***Определение* 8.2.**
Будем говорить, что язык A вероятностно сводится к языку B если существует вероятностная машина Тьюринга M такая, что $Pr[B(M(x)) = L(x)] \geqslant \frac{2}{3}$.

Обозначается $A \leqslant_R B$.

Утверждение 8.3.
Пусть $A \leqslant_R B$ и $B \in$ BPP. Тогда $A \in$ BPP.

Доказательство.
Рассмотрим какой-то x. Запустим сводящую A к B машину Тьюринга m раз и получим набор $y_1, y_2, \dots y_m$.

Для каждого из y_i можно узнать вопрос его принадлежности языку B с вероятностью не больше $\frac{1}{10m}$ (многократно запустив машину для языка B).

Вероятность, что для каждого y_i мы узнаем верный ответ не меньше чем $(1 - \frac{1}{10m})^m \to C$.

Осталось выбрать мажорирующее значение среди полученных результатов для y_i. С вероятностью C каждый полученный локальный результат верен, а тогда, аналогично понижению ошибки в BPP и итоговый результат верен (при помощи стандартного способа понижения ошибки получим вероятность ошибки не больше $\frac{2}{3C}$, а затем перемножим). □

Замечание.

Более строго на лекциях показано не было, я тоже не буду.

Замечание.

Сведение не является транзитивным.

***Определение* 8.3.**

$\mathtt{BP} \cdot \mathtt{NP}$ класс языков L, таких что $L \leqslant_R 3\mathtt{SAT}$.

***Определение* 8.4.**

Множество функций $h : \{0,1\}^n \to \{0,1\}^k$, таких что:

$\forall x \neq x' \in \{0,1\}^n$ и $\forall y, y' \in \{0,1\}^k$ верно что $Pr[h(x) = y, h(x') = y'] = \frac{1}{2^{2k}}$

назовем семейством попарно независимых хеш-функций.

Обозначение.

Множество функций $\{h_{a,b} : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}, h_{a,b}(x) = ax + b\}$, где $a, b \in \mathbb{F}_{2^n}$ обозначим как $\mathcal{H}_{n,n}$.

Если же каждая функция дополнительно оставляет от результата лишь первые $k \leqslant n$ бит, то такое множество функций обозначим $\mathcal{H}_{n,k}$.

Замечание.

Набор функций $\mathcal{H}_{n,k}$ можно воспринимать как набор хеш-функций, т.к. существует биекция между множествами $\mathbb{F}_{2^n}$ и $\{0,1\}^k$.

Теорема 8.4 (Об эффективном семействе попарно независимых хеш-функций)**.**

Семейство функций $\mathcal{H}_{n,k}$ является семейством попарно независимых хеш-функций.

Доказательство.

TODO: Практика / без доказательства □

Мотивация.

Пусть требуется проверить наличие совершенного паросочетания в двудольном графе при условии, что если оно есть, то единственное.

Рассмотрим матрицу двудольного графа (в клетке i, j стоит единица, если есть ребро $v_i u_j$).

Обычно проверка наличия происходит подсчетом пермамента (см. дискретную математику). В случае, если паросочетание единственное, то можно считать определитель.

***Определение* 8.5.**

$\mathtt{USAT} = \{\varphi \mid \exists!$ выполняющий φ набор $\}$

Лемма.

Пусть $S \subseteq \{0,1\}^n$, такое что $2^{k-2} \leqslant |S| \leqslant 2^{k-1}$

$\mathcal{H}_{n,k}$ — семейство попарно независимых функций.

Тогда $Pr[\exists! x \in S : h(x) = 0^k] \geqslant \frac{1}{8}$ (вероятность по выбору h).

Доказательство.

При фиксированном h: $Pr[h(x) = 0^k] = p = \frac{1}{2^k}$.

Пусть N — с. в., равная количеству элементов множества S перешедших в 0^k.

Вычислим $Pr[N = 1]$.

$Pr[N \geqslant 1] \geqslant \sum\limits_{x} Pr[h(x) = 0^k] - \sum\limits_{y<x} Pr[h(x) = 0^k \wedge h(y) = 0^k] = p|S| - \binom{|S|}{2}p^2$

$Pr[N \geqslant 2] \leqslant \sum\limits_{x,y} Pr[h(x) = 0^k \wedge h(y) = 0^k] = \binom{|S|}{2}p^2$

(Оба неравенства — префиксы формулы влючения-исключения)

$Pr[N = 1] = Pr[N \geqslant 1] - Pr[N \geqslant 2] \geqslant |S|p - |S|^2p^2 = |S|p(1 - |S|p) \geqslant \frac{1}{4} \cdot \frac{1}{2} \geqslant \frac{1}{8}$. □

Теорема 8.5 (Вэлианта-Вазирани)**.**

Существует алгоритм $A : \{0,1\}^* \to \{0,1\}^*$, такой что

$\varphi \in \texttt{SAT} \implies Pr[A(\varphi) \in \texttt{USAT}] \geqslant \frac{1}{8n}$.

$\varphi \notin \texttt{SAT} \implies Pr[A(\varphi) \notin \texttt{SAT}] = 1$.

Доказательство.

Общее количество выполняющих наборов не превышает 2^n. Пусть алгоритм A выберет случайное число $1 \leqslant k \leqslant n$.

После этого рассмотрим семейство попарно независимых хеш-функций $\mathcal{H}_{n,k}$ и выберем случайную h. В качестве результата работы алгоритма вернем формулу $\psi(x) = \varphi(x) \wedge h(x) == 0^k$.

Покажем, что A удовлетворяет условию. Если φ была невыполнима, то и ψ невыполнима.

Если же φ была выполнима, то с вероятностью $\frac{1}{n}$ количество решений φ лежит между 2^{k-1} и 2^k, а тогда по лемме выше вероятность, что выбранная хеш-функция оставляет единственное решение будет хотя бы $\frac{1}{8}$.

Тогда общая вероятность успеха $\geqslant \frac{1}{8n}$, что и требовалось. □

9. Задачи с практики

9.1. Полиномиальная иерархия

Задача. Докажите, что $\sum_i$ SAT является полным языком в $\sum_i^p$.

Решение.

Мы хотим, чтобы $\exists x_1 \forall x_2 \dots Q x_i : M(x, x_1, \dots, x_i)$ сводилось к $\exists x_1 \forall x_2 \dots Q x_i : \varphi(x_1, \dots, x_i)$ (где φ — какая-то формула).

Рассмотрим машину $M(x, x_1, \dots, x_i)$. Существует φ, такая что $M(x, x_1, \dots, x_i) = 1 \iff \varphi(x_1, \dots, x_i) = 1$ (т.к. SAT является NP полным). Подставим эту φ в нашу задачу и получим требуемое.

Задача. Покажите, что если язык A оракульно сводится к языку $B \in \sum_i^p$, то $A \in \sum_{i+1}^p$.

Решение.

Мы знаем, что $B \in \sum_i^p$.

Хотим показать, что $A \in \sum_{i+1}^p$. Т.е., $\exists M : x \in A \iff \exists v_1 \forall v_2 \dots Q x_{i+1} : M(x, v_1, \dots v_{i+1}) = 1$.

Пусть M_A — оракульная машина для решения A. Поймем, как она работает. Наш алгоритм что-то делает, потом спрашивает про слово из B, потом снова что-то делает и.т.д. Рассмотрим запросы к языку B.

Пусть для конкретного x мы спрашивали про слова $u_1, u_2, \dots$ и получили ответы $a_1, a_2, \dots$. Тогда положим $v_1 = a_1 a_2 \dots$. Осталось научиться проверять нашу подсказку.

Пусть $u \in B$. Тогда $\exists w_1 \forall w_2 \dots Q w_i : M_B(u_1, w_1, \dots) = 1$. Допишем в конец наших подсказок v_i значения $w_1, w_2, \dots$ (к v_i дописываем в конец w_i). Теперь мы можем понять, что подсказка корректна для ответа, что очередное слово принадлежит языку.

Если же $u \notin B$, то $\forall w_1 \exists w_2 \dots Q w_i : M_B(u_1, w_1, \dots) = 0$. В таком случае допишем w_i в конец подсказок $v_2, v_3, \dots$ (к подсказке v_i дописываем w_{i-1}). Таким образом мы научились проверять, что подсказка корректна для слов не лежащих в языке, что и требовалось.

9.2. Вероятностные классы

Задача. Докажите, что NP $\subseteq$ PP.

Решение.

Пусть размер подсказки для нашего NP алгоритма ограничен полиномом $p(n)$. Проверим одну случайную подсказку. Если она подошла — вернем 1. Иначе, с вероятностью $\frac{1}{2} + \frac{1}{2 \cdot 2^{p(n)}}$ вернем 0 и 1 с оставшейся.

Тогда для слова не лежащего в языке вероятность выдать верный ответ равна $\frac{1}{2} + \frac{1}{2 \cdot 2^{p(n)}} > \frac{1}{2}$.

Если же слово лежало в языке, то вероятность правильного ответа будет хотя бы

$\frac{1}{2^{p(n)}} + (1 - \frac{1}{2^{p(n)}})(\frac{1}{2} - \frac{1}{2 \cdot 2^{p(n)}}) = \frac{1}{2} + \frac{1}{2 \cdot 2^{2p(n)}} > \frac{1}{2}$, что и требовалось.

Задача. Докажите, что если NP $\subseteq$ BPP, то NP = RP.

Решение.

Покажем, что SAT лежит в RP. Подставим $x_1 = 0$ и $x_1 = 1$ много раз (так, чтобы вероятность не найти значение x_1 или найти неверное была меньше $\frac{1}{2n}$). Если мы и для $x_1 = 1$ и для $x_1 = 0$ получили ответ, что формула невыполнима то вернем 0.

Пусть мы нашли какое-то значение для x_1. Подставим его навсегда и начнем искать значение x_2 и так далее. Если в результате работы алгоритма мы нашли какой-то выполняющей набор, то проверим его и вернем 1 если он подойдет и 0, если нет.

Тогда если $\varphi \notin SAT$, то мы точно ответим правильно (т.к. мы проверили набор).

Оценим вероятность ошибки, если $\varphi \in SAT$. Она не больше, чем сумма вероятностей того, что мы переменную x_i выбрали неверно, т.е. строго меньше $\frac{1}{2}$.

9.3. Оценки Чернова

Задача.

Дано конечное множество A и семейство его подмножеств S. Известно, что для любого $x \in A$ существует не более f множеств в S его покрывающих. Требуется найти f-оптимальное покрытие (т.е. отличающиеся не более чем в f раз от минимального).

Решение.

Для каждого множества заведем переменную x_i, которая будет отвечать событию вида "мы взяли множество с номером i".

Запишем условие про то, что мы покрыли каждый элемент. Для этого ля элемента c запишем неравенство вида $\sum x_i \geqslant 1$, где суммирование происходит по всем $i : c \in S_i$. При этом, т.к. мы хотим минимизировать количество выбранных множеств, так что будем минимизировать функцию $\sum x_i$. Полученная задача является задачей целочисленного линейного программирования.

Решим ее как задачу обычного линейного программирования и получим ответ вида $\tilde{x_i}$, где все $\tilde{x_i}$ — вещественные числа от 0 до 1. Осталось округлить ответ.

Для каждой переменной x_i выберем ее значение как 1, если $\tilde{x_i} \geqslant \frac{1}{f}$. Тогда мы каждую переменную увеличили не более чем в f раз, а значит, и общую их сумму увеличили не более чем в f раз, т.е. получили f-оптимальное решение.

Осталось проверить, что найденное решение корректно, т.е. что для каждого элемента c существует покрывающее его множество. Действительно, у нас была сумма $\sum \tilde{x_i} \geqslant 1$, при этом всего у нас не больше f слагаемых. Это означает, что $\exists i : x_i \geqslant \frac{1}{f}$. Тогда мы покроем c множеством S_i.

9.4. Схемы

Задача.

Докажите, что `PATH` в ориентированном графе лежит в классе `NC`.

Решение.

Рассмотрим матрицу смежности A. Количество путей длины k из вершины s в t равно значению в клеточке (s, t) матрицы A^k.

Задача вычисления каждой из таких матриц лежит в `NC`. Осталось запустить все такие схемы параллельно и посчитать `and` для всех ответов.

Задача.

Докажите, что в $\sum_2^p$ существует язык со схемной сложностью $\Omega(n^k)$.

Решение.

Рассмотрим два случая.

Если $\mathtt{NP} \subseteq \mathtt{P}_{/\mathtt{poly}}$, то $\mathtt{PH} = \sum_2^p$. С практики мы знаем, что в классе PH есть язык L со сложностью $\Omega(n^k)$, а тогда он есть и в $\sum_2^p$ есть.

Если же $\mathtt{NP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$, то в NP есть лежит язык со схемной сложностью $\Omega(n^k)$, а тогда этот язык лежит и во множестве $\sum_2^p$ (т.к. $\mathtt{NP} \subseteq \sum_2^p$).