1 Билет 1

Утверждение 1.1. *Пусть M – одноленточная МТ, которая распознает язык бинарных палиндромов. Тогда существует константа $C : \exists n_o : \forall n > n_0$ существует вход длины n, на котором $M(x)$ делает $\geq Cn^2$ шагов.*

Доказательство. В начале очевиден принцип несжимаемости, нельзя инъективно перевести строки из $\{0,1\}^n$ в $\{0,1\}^*$ так, чтобы все образы по длине были меньше чем n.
Будем доказывать для входов, длина которых кратна 3. По $x \in \{0,1\}^n$ строим вход $x0^n x^{rev}$ и скармливаем МТ все такие входы. Возьмем все перегородки после нулей, их всего n, существует перегородку, через которую МТ прошла $\leq \frac{T(x)}{n}$ раз.
Теперь строим отображение $f : \{0,1\}^n \to \{0,1\}^*$, переводим строку x в протокол работы МТ на строке $x0^n rev(x)$. Для этого выпишем набор состояний, в которые переодила МТ, переходя через "хорошую"перегородку и номер этой перегородки.
Утверждается, что такая f – инъекция, чтобы доказать, предположите обратное и рассмотрите работу на строке $x0^n y^{rev}$.
Пусть $|x| = n$, тогда $f(x) \leq \log n + \frac{T(x)}{n}C$, но при этом существует $|y| = n$, такой что $f(y) \geq n$. Получаем, что

$$n \leq logn + \frac{T(x)}{n}C$$

$T(x) = \omega(n^2)$ на таких входах.
Для некратных 3 входов делаем также, но по-середине пишем вместо n нулей, на один ноль больше или меньше – это не влияет на оценки. ■

2 Билет 2

Определение 2.1. *k – ленточная машина Тьюринга. (Добавляется куча лент и функция перехода теперь действует по всем лентам).*

Утверждение 2.1. *Для любой k – ленточной МТ, которая на входе x работает время $T(x)$, существует 1 ленточная МТ, которая работает $O(T(x)^2)$.*

Доказательство. Будем хранить в одном символе МТ символы всех лент (а также спец символы, помеченные головкой). На каждом шаге будем идти вправо и делать все изменения, которые нужны на лентах. ■

Определение 2.2. *Универсальная МТ – эмулирует МТ по описанию.*

Утверждение 2.2. *Для любой k-ленточной МТ существует универсальная k-ленточная МТ с линейным замедлением.*

Доказательство. Понятно как получить квадратичное замедление, нужно положить описание в начало, например, первой ленты. Далее постоянно возвращаться, чтобы узнать, какой шаг сделать. Если же хотим линейного – давайте возить описание с собой, это будет давать $O(1)$ действий из-за его константного размера, при этом эмуляция будет работать за линейное время. ■

Утверждение 2.3. *k ленточную МТ можно эмулировать на 2-ленточной с логарифмическим замедлением.*

Доказательство. ■

3 Билет 3

Основная модель вычислений – многоленточная МТ.

Определение 3.1. *$f : \mathbb{N} \to R_+$, тогда $L \in DTime[f(n)]$, если существует многоленточная МТ, такая что*

1. $\forall x \in L \Rightarrow M(x) = 1$.
2. $\forall x \notin L \Rightarrow M(x) = 0$.
3. $\forall x$ *МТ работает* $O(f(|x|)$ *шагов.*

Определение 3.2. $P = \cup_{i>0} DTime[n^i]$.

Определение 3.3. *Про семейство схем, распознающих язык.*

Определение 3.4. *$L \in Size[f(n)]$, если есть последовательность схем, распознающих L и для достаточно больших n выполнено $|C_n| \leq f(n)$.*

Определение 3.5. $P/Poly = \cup_{i>0} Size[n^i]$.

Пример 3.1. *Неразрешимый язык может лежат в $P/Poly$. Например $1^H = \{1^n | n \in H\}$, для некоторого языка тоже является разрешимым и лежит в $P/Poly$, так как на каждую длину мы можем предоставить схему.*

Утверждение 3.1. *Существует такой алгоритм A, который получает на вход T, n, m и*

1. *A работает $poly(n + T + |m|)$ шагов.*
2. *Если МТ m на всех входах из $\{0,1\}^*$ выдает ответ за $\leq T$ шагов, то алгоритм A выдает схему C, которая имеет n входов и 1 выход и распознает на входах длины n также как m.*

Доказательство. Будем возвращать схему размера $T \times T \cdot O(1)$.
На уровне i будет T ячеек, в каждой из которых будет вычисляться некоторая информация: сивмол, написанный в этой ячейке, есть ли тут головка в момент i, а также, если есть головка, то состояние, в которой МТ сейчас находится. Понятно, что для пересчета этих параметров нужно обратиться к нескольким соседним ячейкам предыдущей строки. Для того, чтобы узнать ответ, посмотрим, принималось ли где-нибудь состояние q_{yes}.

■

Утверждение 3.2. $P \subseteq P/Poly$.

Замечание 3.1. *Таким образом хотели доказывать, что $P \neq NP$, взять, к примеру, SAT и показать, что он не лежит в $P/Poly$, однако доказывать нижние оценки на схемы пока что не научились.*

4 Билет 4

Определение 4.1. *$L \subseteq \Sigma^*$, система доказательств для языка L – это такой алгоритм Π, который обладает следующими свойствами:*

1. *(Полнота) $\forall x \in L \Rightarrow \exists w : \Pi(x, w) = 1$*
2. *(Корректность) $\forall x \notin L \Rightarrow \forall w \Pi(x, w) \neq 1$ (Ну или равно 0).*

3. Π *всегда останавливается*

Определение 4.2. *Система доказательств* Π *называется эффективной, если* $\Pi(x, w)$ *работает за* $\leq poly(|x| + |w|)$ *шагов.*

Замечание 4.1. *Системы доказательств существуют для перечислимых языков, как мы знаем из предыдущий главы курса. Также можно доказать, что для всех перечислимых языков существует и эффективная система доказательств (TCS12), искуственно увеличивая подсказку.*

Определение 4.3. *класс* NP *состоит языков, для которых существует эффективная система доказательств* Π*, а также полином* q*, такой что* $\forall x \in L \exists w, |w| \leq q(|x|), \Pi(x, w) = 1$*, то есть, существует полиномиальная подсказка.*

Пример 4.1. *Примеры языков из* NP*.*

1. SAT *– множество выполнимых пропозициональных формул.*
   $SAT \in NP$*, но не выяснено,* $UNSAT \notin / \in NP$*.*

2. $Hampath$ *– язык графов, в которых есть гамильтонов путь, также лежит в* NP*.*

3. $CLIQUE$ *– язык пар (граф, число) такой, что в графе есть клина на числе вершин. Лежит в* NP*.*

4. $Composite$ *– язык составных натуральных чисел, лежит в* NP*, а также, известно, что* $Primes \in NP$ *(TCS11) и, более того, человечество умеет показывать* $Primes \in P$*.*

Определение 4.4. *Недетерминированные МТ. Вместо одной функции перехода теперь две и машина сама выбирает, в какую идти. (:)).*
Говорят, что недетерминированная M *принимает слово, если существует последовательность корректных переходов, при которых она придет в состояние* q_{yes} *на входе этом слове.*
Время работы НМ – максимум по всем возможным применениям функции перехода.

Определение 4.5. $NTime[f(n)]$ *– множество языков, которые принимаются многоленточными НМТ за* $O(f(n))$ *шагов, где* n *– длина входа.*

Определение 4.6 (Второе определение NP)**.** $NP = \cup_{c>0} NTime[n^c]$.

Определение 4.7 (МТ с подсказкой)**.** *. К обычной МТ добавляется лента подсказки, на которую записывается некоторая строка, к которой может обращаться МТ во время работы. Машина принимает слово, если существует подсказка, для которой она придет в состояние* q_{yes}*. Время работы такой машины – максимум по всем подсказкам.*

Теорема 4.1. *Следующие условия эквивалентны:*

1. $L \in NP$ *(на языке систем доказательств)*

2. L *распознается машиной Тьюринга с подсказкой за полиномиальное время.*

3. $L \in \cup_{c>0} Ntime[n^c]$.

Доказательство. (1) $\Rightarrow$ (2): построим МТ с подсказкой. Пусть наша МТ при обращениях к подсказке будет теперь обращаться на ленту с подсказкой вместо ленты входа. Тогда понятно, что подсказки аналогичны друг другу.
(2) $\Rightarrow$ (3) : пусть МТ порождает подсказку, а далее действует детерминированно. Подсказка более чем полиномилального размера не нужна, так как наш алгоритм не успеет ее обработать из-за своего времени работы.
(3) $\Rightarrow$ (1) : подсказка – какую функцию перехода выбирать на каждом шагу. ■

5 Билет 5

Определение 5.1. *Язык A сводится по Карпу к языку B, если существует полиномиально вычислимая f: $\forall x, x \in A \iff f(x) \in B$. Обозначается $A \leq_p B$.*

Утверждение 5.1. *Свойства сведения*

1. $A \leq_p B$, $B \in P \Rightarrow A \in P$.
2. $A \leq_p B, B \leq_p C \Rightarrow A \leq_p C$.

Определение 5.2. *Язык A называется NP-трудным, если $\forall L \in NP, L \leq_p A$. Язык A NP-полный, если $A \in NP$ и A – NP-трудный.*

Определение 5.3. $BH = \{(M, x, 1^t) | \ \exists y, M(x, y)$ *выдает 1 за* $\leq t$ *шагов* $\}$.

Теорема 5.1. *BH – NP-полный.*

Доказательство. Проверим, что $BH \in NP$. Подсказка как раз и будет этот y. Запускаем на t шагов и проверяем, приняло или нет. Если тройка лежит в языке, то по определению найдется такая подсказка, что выдаст *yes*. Иначе – нет.
Возьмем $L \in NP$, хотим проверить $L \leq_p BH$. У L есть нмт эффективная система доказательств П, работающая за $q(|x| + |y|)$, при этом также для лежащих в языке слов есть маленькая подсказка, длины $\leq p(x)$. Давайте сделаем следующее отображение $f(x) = (M, x, 1^{q(|x|+p(|x|))})$. Очевидно, что это корректное сведение. ■

Определение 5.4. *$Circuit - SAT$ – язык, в котором лежат схемы, у которых есть выполняющий набор.*

Теорема 5.2. *$Circuit - SAT$ – NP полный.*

Доказательство. $Circuit - SAT \in NP$, подсказка – выполняющий набор.
Возьмем $L \in NP$. Хотим полиномиально свести L к $Circuit - SAT$. Давайте по МТ, решающей L с помощью подсказки и x построим схему, в которой длина входа равна $q(|x|)$, где q – полином, ограничивающий подсказку. Тогда осталось понять, существует ли подсказка, при которой схема даст значение 1, а это инстанс задачи $Circuit - SAT$. ■

Определение 5.5. *$3SAT$ – язык из формул, записанных в КНФ так, что каждый дизъюнкт состоит из не более чем 3 литералов.*

Теорема 5.3. *SAT и $3SAT$ – NP полные.*

Доказательство. Будем сводить $Circuit - SAT$ к этим задачам. Можно сводить только к $3SAT$, так как дальше сведение тривиально. Просто запишем кнф формулу для схемы, добавив новые переменные для каждого узла в ней и связав их нужными условиями. ■

6 Билет 6

Определение 6.1. *Q – предикат. Он полиномиально ограниченный, если $Q(x, y) = 1 \Rightarrow |y| \leq p(|x|)$ для некоторого полинома p.*

Каждый язык $L \in NP$ задается полиномиально ограниченным, полиномиально проверяемым предикатом, таким что $x \in L \iff \exists y Q(x, y) = 1$.

Определение 6.2 (Задача поиска)**.** *Q – полиномиально ограниченный, полиномиально проверяемый предикат двуместный предикат. Задача поиска, заданная предикатом Q это задача: по x найти y такой, что $Q(x, y) = 1$. Множество задач поиска для таких предикатов обозначим $\widetilde{NP}$.*

Будем говорить, что задача поиска решается агоритмом A, если для любого x, для которого существует такой y, что $Q(x, y) = 1$, выполнено $Q(x, A(x)) = 1$. Множество задач поиска, для которых существуют полиномиальные по времени алгоритмы будем обозначать $\widetilde{P}$. Очевидно, что задачи распознавания, соответствующие задачам из $\widetilde{P}$, лежат в P.

Определение 6.3. *MT с оракулом, лента для вопросов к оракулу.*

Определение 6.4 ($A \leq_{pT} B$)**.** *Вычислительная задача (задача распознавания или NP задача поиска Q) сводится по Куку (= По Тьюрингу за полином) к языку A, если она может быть решена за полином с оракулом A.*

Утверждение 6.1 (Свойства сведений по Куку)**.** .

1. *если $L_1 \leq_p L_2$, то $L_1 \leq_{pT} L_2$*
2. *$\leq_{pT}$ транзитивно, даже если цепочка начинается на задаче поиска.*
3. *Если мы свелись по Куку к задаче из P, то мы сами лежим в P. Аналогично и для задач поиска.*

Доказательство. .

1. В начале сведемся, потом сделаем один запрос к оракулу.
2. Пусть начиналось на задаче распознавания. $L \leq_{pT} A \leq_{pT} B$. Давайте всякий раз, когда МТ для L делает запрос к оракулу A будем запускать полиномиальный алгоритм проверки для языка A, который делает запросы к оракулу B. Таким образом обойдемся только оракулом для B. Аналогичо и для задач поиска.
3. Будем эмулировать запрос к оракулу, запуская полиномиальный алгоритм.

■

Теорема 6.1. *Каждая задача поиска из $\widetilde{NP}$ сводится по Куку к некоторому языку из NP.*

Доказательство. В начале введем нестрогий порядок на строках. $x \preceq y$. Сначала сортим по длине, потом лексико. Пусть Q – полиномиально ограниченный предикат полиномом p. Рассмотрим язык $L = \{(x, a) \mid |a| \leq p(|x|), \exists y \preceq a : Q(x, y) = 1\}$. Тогда такой язык лежит в NP, так как подсказка это нужный y. Задача поиска сводится к этому языку по Куку с помощью бинпоиска по ответу. Делаем бп, в котором для определения, куда сдвигать границу, обращаемся к оракулу. ■

Утверждение 6.2. *Если $P = NP$, то $\widetilde{P} = \widetilde{NP}$.*

Это из-за того, что все задачи поиска из $\widetilde{NP}$ в таком случае сводятся по Куку к задачам из P, а значит, сами лежат в $\widetilde{P}$.

Утверждение 6.3. *Если NP-полный язык L задается предикатом Q, то задача поиска Q сводится по Куку к L.*

Задача поиска Q сводится по Куку к какому-то языку из NP. Этот язык сводится по Карпу к L, а значит сводится к нему по Куку и мы получаем утверждение из транзитивности сведения по Куку.

7 Билет 7

Определение 7.1. *$DTime^A[f(n)]$ как и раньше, но машина с оракулом. Аналогично, $NTime^A[f(n)]$.*
$P^A = \cup_{c>0} DTime^A[n^c]$.
$NP^A = \cup_{c>0} NTime^A[n^c]$.

Определение 7.2. $EXP = \cup_{c>0} Dtime[2^{n^c}]$.

Утверждение 7.1. *$EXPComp = \{(M, x, t),\ M(x) = 1$ за не более t шагов $\}$ – полный язык в классе EXP.*

Доказательство. Понятно включение. Действительно, будем просто эмулировать M, это займет не более t шагов, что экспонента от длины входа. Возьмем $L \in EXP$. Покажем $L \leq_p EXPComp$. По x как всегда выдадим $(M, x, 2^{|x|^c})$, где c – константа из работы M. Очевидно, сведение корректно. ■

Утверждение 7.2. $P^{EXPComp} = NP^{EXPComp} = EXP$.

Доказательство. $EXPComp$ – EXP полный, поэтому $EXP \subseteq P^{EXPComp}$. А также $NP^{EXPComp} \subset EXP$, так как можно перебрать подсказу, а дальше полином обращений к оракулу решить тоже за экспоненту. ■

Теорема 7.1 (Бэйкер, Гилл, Солвэй)**.** *Существует такой язык B, что $P^B \neq NP^B$.*

Доказательство. B – какой-то язык. $U_B = \{1^n \mid \exists x \in \{0,1\}^n \cap B\}$. Если $B \in NP$, то $U_B \in NP$. Подсказка – строка нужного размера и подсказка для нее.
Теперь будем строить B так, чтобы $P^B \neq NP^B$.
Рассмотрим нумерацию Машин Тьюринга с оракулом U_B – M_i. Будем параллельно достраивать язык B, идя по машинам. Возьмем машину M_k. Посмотрим, что она выдает на входе 1^k. Причем дадим ей поработать максимум $2^k/10$ шагов. Если она выдала 1, то не будем добавлять в языке B слов длины k. Иначе добавим какую-нибудь строку длины k. Это можно сделать, так как все предыдущие машины сделали меньше обращений к оракулу, чем 2^k.
Почему мы обманули все полиномиальные машины? Давайте возьмем какую-нибудь полиномиальную МТ и покажем, что она ошибается. Пусть она работает за $p(n)$. Возьмем ее номер k в нумерации такой, что $2^k/10 > p(k)$. Тогда эта машина ошибается на строке 1^k по построению. ■

8 Билет 8

Теорема 8.1 (Ладнер)**.** *Если $P \neq NP$, то существует $L \in NP$, такой, что, $L \notin P$ и L – не полный в NP.*

Доказательство. Интуиция следующая: хотим немного ослабить язык SAT. Давайте рассматривать

$$SAT_H = \{\varphi 01^{n^{H(n)}} | \varphi \in SAT, |\varphi| = n\}$$

Поймем, как определить H. Возьмем нумерацию всех машин Тьюринга $M_1, M_2, M_3, \ldots$.
$H(n) = i$, если i – такое минимальное число, что $i < \lfloor \log\log n \rfloor$, что M_i решает SAT_H на всех входах $|x| \leq \log n$ за время $i|x|^i$, или, если такого числа нет, то $\lfloor \log\log n \rfloor$.
Заметим, что $H(n)$ определяется через SAT_H и наоборот.
Но заметим, что чтобы определить $H(n)$ нам не потребуется значений $H(x)$ при $x > \log n$, так как $H(n)$ нужно для дописывания только к строкам длины $n + 1$.

Утверждение 8.1. *$H(n)$ вычисляется по значениям $H(1), \ldots, H(n-1)$ за $poly(n)$ действий.*

Будем вычислять по определению. Переберем

1. $i \le \log\log n$
2. $x, |x| \le \log n$, это $2^{\log n} = O(n)$ действий
3. запустим машину i на $(\log\log i)(\log i)^{\log\log i} =$ действий.
4. Сравним результат с реальным: мы можем понять, лежит или нет, так как знаем предыдущие значения H, SAT будем решать перебором за $O(n)$.

Таким образом вычислим $H(n)$ за $O(n^3)$ по предыдущим значениям. Тогда мы сможем вычислить $H(n)$ за $O(n^4)$, вычисляя по очереди все значения.

Утверждение 8.2. *$H(n)$ не убывает.*

Действительно, если $H(n) = i$, то такое i подошло и для всех предыдущих, либо i – верхняя граница для $H(n)$, но тогда оно точно больше предыдущих.

Утверждение 8.3. *$SAT_H \in P \iff H(n) \le C$.*

$\Rightarrow$: есть МТ, решающая SAT_H за $p(|x|)$. Но эта МТ встречается в нумерации бесконечное число раз. Она там также встречается как M_i при $i|x|^i > p(|x|)$. Но тогда $H(n) \le i$, так как такая машина нам все решит и мы ее запустим на такое число шагов.
$\Leftarrow$: Ограниченная неубывающая функция это такая функция, что с некоторого момента она равна j. Но заметим, что тогда M_j решит нам язык за $j|x|^j$, что есть полином. ■

Утверждение 8.4. *$SAT_H \in NP$*

Действительно, давайте просто давать как подсказку решение для внутренней SAT, при этом определить, верное ли кол-во единиц мы можем, вычислив H.

Утверждение 8.5. *$SAT_H \notin P$.*

Пусть это не так. Тогда $H(n)$ ограничена. Тогда $SAT \le_p SAT_H$ и $P = NP$.

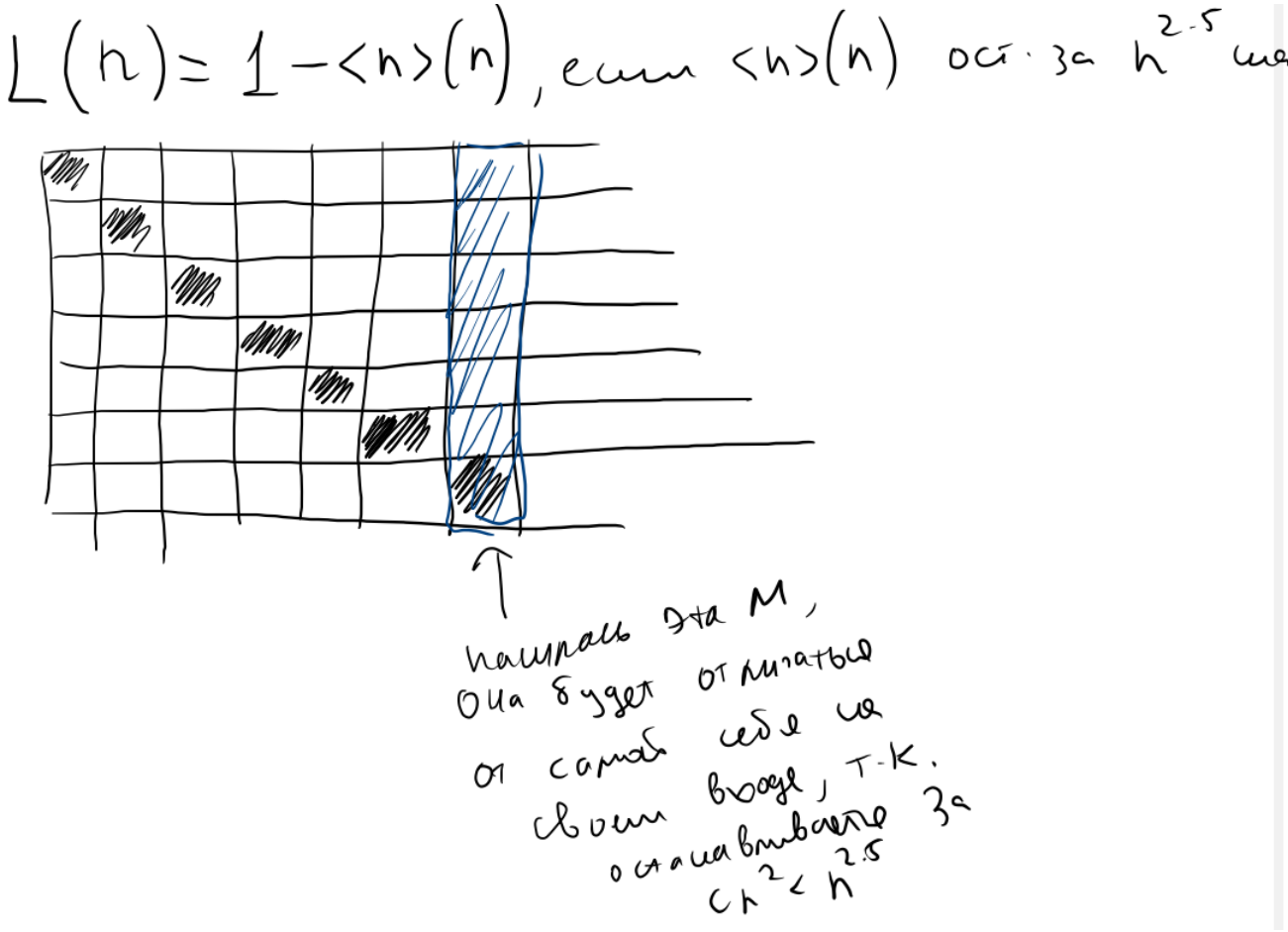
Утверждение 8.6. *SAT не сводится к SAT_H полиномиально.*

Предположим обратное. Покажем тогда, что мы сможем решить SAT за полином. Пусть сведение работает за n^c. Тогда величина формулы, которую выдаст сведение $\le n^c$, где n – длины формулы, поступившей нам на вход.
$H(n)$ не ограничена, возьмем n_0 такое, что $H(n) > 3c$ при $n > n_0$. Тогда пусть сведение выдало формулу, длина которой $> n_0$ (иначе сделаем полный перебор, который займет $O(1)$) и у которой правильное число единиц на конце (иначе мы это легко проверим за полином).
Тогда длина строки, к которой свелись, $m \ge |\varphi| + 1 + |\varphi|^{3c}$, отсюда $|\varphi| \le m^{1/3c}$, при этом $m \le n^c$. Получается, $|\varphi| \le n^{1/3}$. При больших n такое значение хотя бы в 2 раза меньше n. Значит, мы свелись к формуле меньшего размера в 2 раза. Таким образом мы сделаем не более log полиномиальных сведений и выдадим верный ответ. Отсюда $P = NP$ и противоречие.

9 Билет 9

Пример 9.1. $DTime[n^2] \subsetneq DTime[n^3]$.

Доказательство. Давайте диагонализировать.



Рассмотрим язык $L = \{M | M$ отвергает M за не более $|M|^{2.5}\}$. Пусть он решается квадратичной M за Cn^2 шагов. Тогда давайте найдем эквивалентную ей M' такую, что $|M'|^{2.5} > C|M'|^2$. Тогда получится, что M' не равна себе же в строчке, соответствующей M'.
Как показать, что $L \in DTime[n^3]$? Давайте эмулировать МТ, которая работает $O(n^{2.5})$ с логарифмической задержкой.
Тогда получили нужный язык L. ■

Определение 9.1. *$h : \mathbb{N} \to \mathbb{N}$ – конструктивная по времени, если $n \to h(n)$ можно вычислить за $O(h(n))$ шагов на ДМТ.*

Теорема 9.1 (Об иерархии по времени для детерменированных вычислений)**.** *$f, g, h : \mathbb{N} \to \mathbb{N}$, h – конструктивная по времени.*
$f(n) = o(h(n)), h(n) \log h(n) = o(g(n))$, тогда $DTime[f(n)] \subsetneq Dtime[g(n)]$.

Доказательство. Конструкция такая же, как и в предыдущем утверждении. Конструктивная функция нужна для будильника в МТ. log – для эмуляции. ■

Утверждение 9.1. *$P \subsetneq EXP$.*

Доказательство. $P \subseteq DTime[2^n] \subsetneq DTime[2^{n^2}] \subseteq EXP$. ■

Замечание 9.1. *Доказательство не работает для НМТ, потому что мы не можем реверснуть ответ за такое же время.*

Пример 9.2. $NTime[n^2] \subsetneq NTime[n^3]$.

Доказательство. Определим для M_i из перечисления всех НМТ отрезок $[n_i, n_i^*]$ так, что $n_i^* = 2^{n_i^{2.5}}$. $n_{i+1} = n_i^* + 1$.
$[n_1, n_1^*], [n_2, n_2^*], [n_3, n_3^*], \ldots$.
Определим язык L так:

1. $L(n_i^*) = 1 - M_i(0^{n_i})$, если $M_i(0^{n_i})$ завершилось за менее чем $n_i^{2.5}$ шагов и 0 иначе.
2. $L(n) = M_i(0^{n+1})$ для $n_i \le n < n_i^*$

Утверждение 9.2. $L \notin NTime[n^2]$.

Пусть это не так, тогда есть M, решающая L за Cn^2. Возьмем такое ее вхождение, что $\forall n \in [n_i, n_i^*], n^{2.5} > Cn^2$. Тогда:

$$M(0^{n_i}) = L(0^{n_i}) = M(0^{n_i+1}) = L(0^{n_i+1}) = \ldots = L(0^{n_i^*}) \neq M(0^{n_i})$$

так как все машины успеют отработать.

Утверждение 9.3. $L \in NTime[n^3]$. *Разбираем случаи, если надо проэмулировать, то эмулируем, иначе нам нужно реверснуть выход недетерменированной машины. Мы можем сделать это за* $2^{n_i^{2.5}} \cdot n_i^{2.5} < ((n_i)^*)^3$.

■

Теорема 9.2 (Об иерархии по времени для недетерменированных вычислений)**.** . *$f, g, h : \mathbb{N} \to \mathbb{N}$, h – конструктивная по времени. $f(n) = o(h(n))$, $h(n+1) = o(g(n))$, тогда $NTime[f(n)] \subsetneq NTime[g(n)]$.*

Утверждение 9.4. $NP \notin NEXP$.

Доказательство. аналогично детерминированному случаю. ■

10 Билет 10

Определение 10.1 (Модель вычислений с ограничением по памяти.)**.** *МТ с дополнительной лентой входа, она read-only, также по ней нельзя уйти правее первого пробела. Также есть лента выхода, по которой нельзя двигаться влево, а только выдавать очередной символ ответа. Затраченная память – максимальный уход вправо на какой-то из рабочих лент.*

Определение 10.2. *$DSpace[f(n)]$ – класс языков, которые принимают ДМТ, использующие $O(f(n))$ памяти. $NSpace[f(n)]$ – то же самое, но для НМТ.*

Определение 10.3. $PSPACE = \cup_{c>0} DTime[n^c]$, $NPSPACE = \cup_{c>0} NTime[n^c]$.

Определение 10.4. $L = LOGSPACE = DSPACE[\log(n)]$, $NL = NSPACE[\log n]$.

Утверждение 10.1. *$\forall s(n) : DTime[s(n)] \subseteq DSpace[s(n)] \subseteq NSpace[s(n)]$. Причем первое включение работает лишь для некоторых моделей вычислений. В частности, для МТ.*

Определение 10.5. *$S : \mathbb{N} \to \mathbb{N}$ – конструктивная по памяти, если $1^{S(n)}$ можно вывести за $O(S(n))$ памяти.*

Замечание 10.1. *Для хороших функций выполняется также иерархия по памяти. То есть, например, известно, что* $DSpace[n^2] \subsetneq DSpace[n^3], NSpace[n^2] \subsetneq NSpace[n^3]$.
(Смотри $TCS30, TCS31$*).*

Теорема 10.1. $s(n)$ *– конструктивная по памяти и* $s(n) \geq \log n$*. Тогда*

$$NSpace[s(n)] \subseteq DTime[2^{O(s(n)}] = \cup_{c>0} DTime[2^{cs(n)}]$$

.

Доказательство. Для доказательства используется идея с графом конфигураций. Конфигурация МТ это набор параметров:

1. Положение головок на всех лентах
2. Содержание рабочих лент
3. Состояние

Проблема выяснения того, принимает ли МТ вход x может быть рассмотрена как проблема выяснения существования пути в графе конфигураций. Поймем, сколько есть конфигураций:
$n \cdot (Cs(n))^k$ – положение головок, $2^{c's(n)k}$ – содержимое рабочих лент. Если $s(n) \geq \log n$, то это число есть $2^{O(s(n))}$. В таком случае мы можем сгенерировать такой граф и проверять наличие пути полиномиальным алгоритмом. Получится время работы $poly(2^{O(s(n))}) = 2^{O(s(n))}$.
Зачем пользовались конструктивностью функции по времени? Для того чтобы сгенерировать конфигурацию нужно отмерить максимальную длину конфигурации и дальше уже перебирать все возможные строчки. Поэтому хочется уметь отмерить за нормальную память.
То есть итоговый алгоиртм такой: по M, x строим граф конфигураций и ищем путь из K_0 в K_{accept}. Можно сделать одно состояние K_{accept}, попросив МТ стирать все рабочие ленты перед тем как завершиться. Так мы унифицируем конечные состояние, но, очевидно, не изменим вычислительную мощь (:)). ■

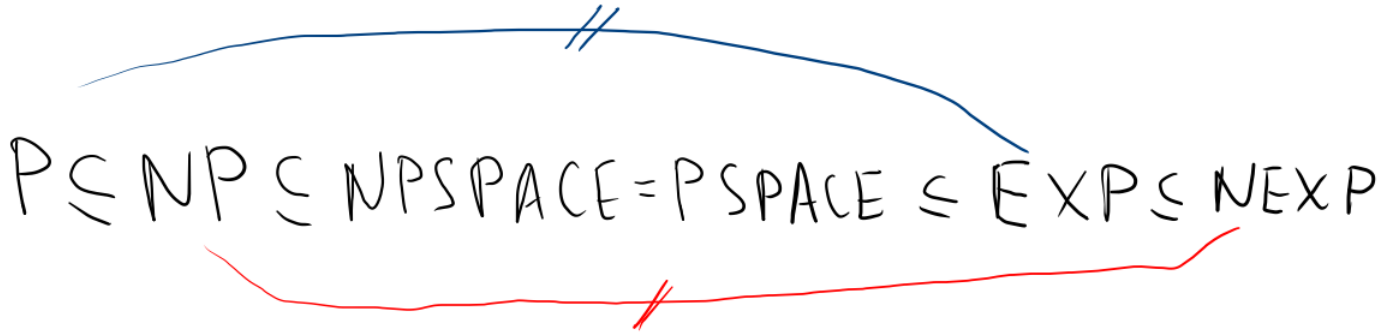
Утверждение 10.2. .

1. $NSPACE \subseteq EXP$
2. $NP \subseteq EXP$

Теорема 10.2 (Савич)**.** $s(n)$ *– конструктивная по времени и* $s(n) \geq \log n$*, тогда* $NSpace[s(n)] \subset DSpace[s(n)^2]$.

Доказательство. Все сводится к тому, чтобы по графу понять, есть ли в нем путь от вершины до другой за память $s(n)^2$, где $s(n)$ – память рассматриваемой НМТ. Воспользуемся предикатом $PATH(u, v, i)$ – есть ли путь от u до v длины не более 2^i и рекурсивным перебором. ■

Утверждение 10.3. $NPSPACE = PSPACE$



11 Билет 11

Рассмотрим кванторные пропозициональные формулы: $Q_1 x_1, \ldots, Q_n x_n \varphi(x_1, \ldots, x_n)$. Язык таких истинных формул это $TQBF$. Понятно, что $SAT \leq_p TQBF$ просто дописываниям ко всем переменным квантора существования.

Утверждение 11.1. *$TQBF$ лежит в $PSPACE$*

Действительно, можно просто сделать перебор рекурсивно и проверить выполнимость.

Утверждение 11.2. *$TQBF$ – полный в классе $PSPACE$.*

Доказательство. Берем граф конфигураций. Хотим записать формулу $PATH(K_0, K_{accept}, cp(n))$. (время работы не более $2^{cp(n)}$, так как иначе все зациклится).
Будем записывать при помощи следующего выражения:
$PATH(u, v, i) = \exists z \forall A, B((A = u, B = z) \vee (A = z, B = v)) \to PATH(A, B, i - 1)$, тогда мы сможем записать всё за полиномиальное количество бит. Остаются детали, как записать в виде формулы утверждения вида $PATH(u, v, 0)$. Мы можем с помощью полиномиального алгоритма проверить такой предикат.
Еще нужно подумать о том, как записать равенство конфигураций, это можно сделать просто побитово. ■

Замечание 11.1. *Выяснение вопроса детерменированной победы в конечных играх – задача из $PSPACE$, так как ее можно записать в $TQBF$ в виде $\exists step_{1,1} \forall step_{2,1} \exists step_{1,2} \ldots Q(..)$, что означает, что есть ход первого игрока, что при любом ходе второго есть ход первого и тд, что первый игрок выиграл.*

12 Билет 12

Определение 12.1 (Логарифмическое сведение)**.** *$A \leq_l B$, если существует p, вычислимая с использованием логарифмической памяти такая что $x \in A \iff p(x) \in B$.*

Утверждение 12.1 (Свойства лог сведений)**.**

1. *$A \leq_l B \Rightarrow A \leq_p B$.*
2. *$A \leq_l B, B \leq_l C \Rightarrow A \leq_l C$.*
3. *$A \leq_l B, B \in L \Rightarrow A \in L$.*

Чтобы доказать эти утверждения нужна лемма.

Утверждение 12.2. *f, g – вычислимые с логарифмической памятью. Тогда $f \circ g$ тоже вычислима с логарифмической памятью.*

Доказательство. Будем вычислять $f(g(x))$, когда для вычисления f будет требоваться очередной бит входа, будем вычислять i-тый бит выхода $g(x)$ заново и ждать пока выведется бит под номером i. ■

$L \in NL$, а вот есть ли равенство – открытый вопрос.

Определение 12.2 (Определение NL через систему доказательств)**.** *У нас появляется лента для подсказки, по которой можно двигаться только вправо (потому что НМТ не может записывать результат выбора на ленту). Это задает такой же класс языков, это можно видеть также как мы видели для разных определений NP.*

Определение 12.3. $DPATH = \{(G, u, v) |$ *в ор. графе* G *есть путь* $u \to v$ $\}$.

$DPATH \in NL$, подсказкой является собственно путь. Нужно проверить, что первая вершина в нем есть u, последняя v и что есть ребро между соседними, это можно сделать за log памяти.

Теорема 12.1. $DPATH$ *– NL полный (относительно* $\leq_l$*).*

Доказательство. пусть $A \in NL$, M – нмт, решающая A. Сведение будет следующим. Оно будет генерировать конфигурации, которые составляют максимум log памяти и выводить ребра между ними, то есть строить граф конфигураций. Далее выведем начальное и конечное состояние и это и будет искомый инстанс задачи $DPATH$. ■

Замечание 12.1. *Неизвестно, лежит ли* $DPATH$ *в* L*, или нет.*

Теорема 12.2. $\overline{DPATH} \in NL$

Доказательство. Нам нужно сертифицировать, что между s, t нет пути, причем сделать такое доказательство, которое можно читать слева направо и использовать логарифмическую память.
U_i – множество вершин, до которых есть путь от s длины не более i.

1. для v можно сертифицировать, что $v \in U_i$, сертификат – просто путь, как в языке $DPATH$.
2. если известно $|U_{i-1}| = k$, то можно сертифицировать, что $u \notin U_i$. Для этого предоставим список вершин из U_{i-1} с сертификатами, что они действительно оттуда. Причем список будет у нас возрастающим, чтобы не было повторяющихся вершин. Проверим, что там правильное число вершин + нет вершины u и также нет ребра между вершинами из списка и u.
3. если известно $|U_{i-1}| = k$, то можно сертифицировать, что $|U_i| = t$. Просто для каждой вершины напишем одно из двух, либо сертификат того, что она лежит в U_i, либо сертификат того, что не лежит, при условии $|U_{i-1}| = k$.
4. Таким образом мы можем сертифицировать, что размер $U_{n-1} = k$ и что $t \notin U_n$.

■

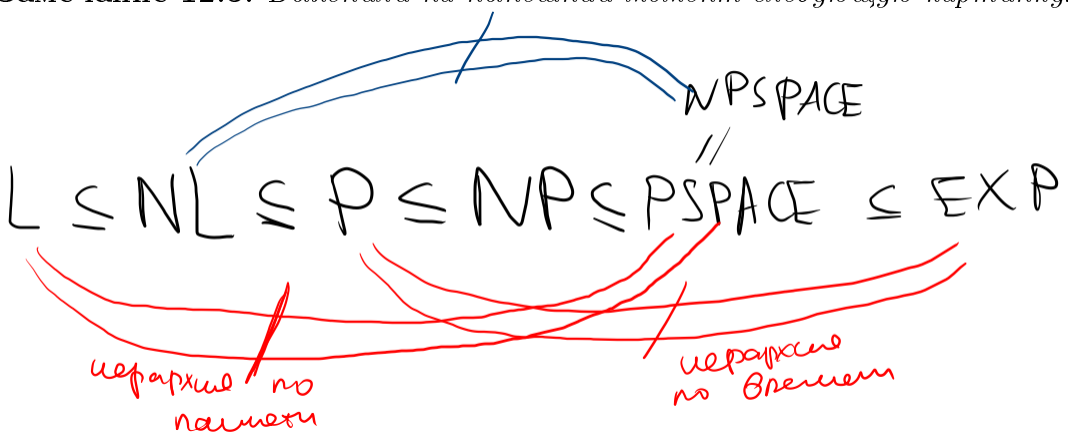
Определение 12.4 (соЯзык)**.** X *– класс языков. Тогда* $coX = \{\Sigma^* \setminus L | L \in X\}$.

Замечание 12.2. *Как мы уже видели,* $NP = coNP$ *– неизвестно.*

Теорема 12.3. $s(n) \geq \log n$ *– конструктивная по памяти. Тогда* $NSpace[s(n)] = coNSpace[s(n)]$.

Доказательство. $A \in NSPACE[s(n)]$, хотим показать, что $A \in coNSpace[s(n)] \iff \overline{A} \in NSpace[s(n)]$. Причем, если покажем это, то автоматически последует равенство, так как для $A \in coNSpace[s(n)]$ надо показать, что $\overline{A} \in Nspace[s(n)]$, тогда из той стрелочки, которая описана это последует.
Рассмотрим M – НМТ, которая распознает A с памятью $cS(n)$. $x \in \overline{A} \iff x \notin A \iff$ в графе конфигураций G_x из K_0 нет пути в K_{accept}.
Сертификатом будет тот же самый, сертификат, что и был в теореме выше. Размер графа конфигураций будет $2^{O(s(n))}$, тогда нам потребуется $O(s(n))$ памяти, чтобы проверить сертификат. Причем граф строить полностью не надо, нужно всего лишь уметь проверять, есть ли некоторые ребра в нем. ■

Замечание 12.3. *Выяснили на нынешний момент следующую картинку:*



13 Билет 13

Определение 13.1. *Определение классов* $\Sigma_0^P, \Sigma_1^P, \ldots,$ *а также* Π_0^P, Pi_1^P.

Интуиция про $NP, coNP$ и Σ_1^P, Π_1^P.
Можно развернуть определение: $\forall x, x \in L \iff \exists y_1 |y_1| \leq p(|x|) \forall y_2, \ldots, y_i Q(x, y_1, \ldots, y_i)$.

Определение 13.2. $PH = \cup_{i \geq 0} \Sigma_i^P$

Утверждение 13.1 (Свойства полиномиальной иерархии)**.** .

1. $\Sigma_i^P \cup \Pi_i^P \subseteq Sigma_{i+1}^P \cap Pi_{i+1}^P$.

   Доказательство. Добавляем фиктивные переменные и кванторы. ■

2. $PH = \cup_{i \geq 0} Pi_i^P$.

3. $\Sigma_i^P = co\Pi_i^P$

4. $\Sigma_i^P = \Pi_i^P \Rightarrow PH = \Sigma_i^P$.

   Доказательство. Доказательство индукцией по $j \geq i$, что $\Sigma_j^P = \Pi_j^P$. Используйте равенство предыдущего уровня для объединения кванторов в один. ■

5. Σ_i^P *и* Π_i^P *замкнуты относительно* $\leq_p$.

   Доказательство. Легко видеть, что надо просто взять вычислимый Q из определения и попросить его использовать сведение чтобы охарактеризовать нужным образом язык. ■

6. *Если в* PH *есть полный язык, то полиномиальная иерархия схлопывается.*

   Доказательство. Просто все языки, начиная с уровня, на котором лежит полный, будут лежать в этом же уровне. ■

7. *Теперь введем полные языки на уровнях иерархии. Пусть* ΣSAT *это язык, в котором лежат истинные формулы вида* $\exists x_1 \forall x_2 \exists x_3, ..., x_i Q(x_1, \ldots, x_i)$, *где* x_i *– возможно вектор значений. Аналогично, только с противоположными кванторами определяется* Π_i.

Утверждение 13.2. *при $i \geq 1$ выполняется то, что $\Sigma_i SAT$, $\Pi_i SAT$ полны в соответствующих классах на i-том уровне полиномиальной иерархии.*

Доказательство. Покажем про $\Sigma_i SAT$. В начале то, что он лежит в Σ_i^P. Характеристика его такова: так как мы не знаем точно сколько там переменных внутри векторочков, то будем в каждый блок класть столько переменных, чтобы нам хватило. То есть, характеристика следующая: $\exists x_1, |x_1| = |\varphi|, \forall x_2 | x_2| = |\varphi|, \ldots, x_i : A(\varphi, x_1, \ldots, x_i)$. A – просто полиномиальный алгоритм, который берет и подставляет из наших блоков переменные в φ и проверяет то, что она истинна. Таким образом показали включение.
Теперь полнота. Возьмем некоторый язык $L \in \Sigma_i^P$. Его характеризация имеет вид: $\exists y_1, |y_1| = p(|x|), \ldots, y_i Q(x, y_1, \ldots, y_i)$. Можно в характеризации сделать все игрики фиксированной длины (в отличии от того что у нас раньше была оценка сверху на длину), так как можно допихать нулей в случае чего.
$Q(x, y_1, \ldots, y_i)$ – какой-то полиномиально вычислимый предикат. Давайте переделаем его в схему. Это получится, так как $y_1, \ldots, y_i$ и x имеют фиксированную полиномиальную длину. Теперь переделаем схему в формулу при помощи введения дополнительных переменных для всех узлов схемы и обеспечивания соответствующих равенств. Получится формула вида $\exists t \varphi(x, y_1, \ldots, y_n)$. Если вдруг нам и нужен был квантор существования в конце (i – нечетно), то мы уже победили. Иначе воспользуемся трюком с существованием для отрицания и получим запись с квантором всеобщности. Тогда этот квантор можно совместить с последним квантором из характеризации L.

■

14 Билет 14

Теорема 14.1. $\Sigma_{i+1}^P = NP^{\Pi_i SAT}$

Доказательство. .
$\subseteq$: $L \in \Sigma_{i+1}^P$ тогда есть следующая характеризация $\exists L' \in \Pi_i^P, \forall x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}, (x, y) \in L'$. Тогда пусть наша НМТ генерирует этот x, потом делает запрос к оракулу при помощи сведения (x, y) к $\Pi_i SAT$.
$\supseteq$: пусть $L \in NP^{\Pi_i SAT}$ и его решает НМТ M за $p(n)$ с использованием оракула Π_i.
$x \in L \Longleftrightarrow \exists z \in \{0,1\}^{p(n)}, T_1(x, z) \wedge T_2(x, z) \wedge T_3(x, z)$, где

1. z отвечает за недетерменированные выборы НМТ и ответы оракулов.

2. $T_1(x, z)$ проверяет, что мы действительно принимаем x действуя согласно строке z

3. $T_2(x, z)$ проверяет, правда ли, что оракул дал верные положительные ответы. Все вопросы к оракулу выглядит как формулы вида: $\forall x_1, \ldots, x_k \varphi(x_1, \ldots, x_k)$. Тогда давайте T_2 будет выглядеть так: $\forall r_1 \in \{0,1\}^{p(n)^2}, \ldots r_k \in \{0,1\}^{p(n)^2} R(x, z, r_1, \ldots, r_k)$, где R поблочно подставляет переменные во все формулы, про которые спрашивал наш алгоритм у оракула и проверяет истинность.

4. T_3 проеряет отрицательные ответы оракула. Для того, чтобы проверить, что φ ложна – нужно проверить, что истинно отрицание φ, то есть, что $\exists x_1, \ldots, x_k ! \varphi(x_1, \ldots, x_k) = 1$. Тогда давайте запишем характеристику следующим образом: $\exists x_1 \in \{0,1\}^{p(n)^2}, \ldots, x_k \in \{0,1\}^{p(n)^2} T'(x, z, x_1, \ldots, x_k)$ и T' выдает 1 тогда и только тогда, когда все формулы обнулились.

Теперь вынесем кванторы независимо из T_1, T_2. Получится нужное чередование. ■

15 Билет 15

Определение 15.1 (Вычисления с неравномерной подсказкой)**.** *Языки, принимаемые MT с неравенномерной подсказкой длины $K(n)$ ($P/K(n)$) это такие языки, для которых существует M – MT с лентой для подсказки и α_n – последовательность подсказок длины $\leq K(n)$ такие, что MT понимает, лежит ли x в L, используя подсказку $\alpha_{|x|}$.*

Замечание 15.1. *$P/1$ содержит неразрешимые языки.*

Утверждение 15.1. *$\cup_c P/n^c = P/poly$.*

Доказательство. .
$\subseteq$: давайте переделаем MT в схемы, потом встроив туда подсказку. Получим последовательность схем.
$\supseteq$: в качестве подсказки возьмем схему. ■

Теорема 15.1. *Существует функция $f : \{0,1\}^n \to \{0,1\}$ такая, что она не вычислима схемой размера менее $2^n/10^n$.*

Доказательство. Схему размера T можно задать за не более чем $4T \log T$ битов. Тогда таких схем не более $2^{4T \log T}$. Пусть $T < 2^n/10n$. Тогда покажем, что $2^{4T \log T} < 2^{2^n}$. Нужно показать: $4T \log T < 2^n$, действительно: $4T \log T < 42^n/(10n) \log(2^n/10n) < 4 * 2^n/10 < 2^n$. ■

16 Билет 16

Теорема 16.1 (Карп-Липтон)**.** *Если $NP \in P/poly$, то полиномиальная иерархия схлопывается на втором уровне.*

Доказательство. За счет того, что мы умеем сводить задачи распознавания к задачам поиска, у нас существует и семейство схем, которое выдает выполняющие наборы для схем. Пусть это семейство C_n. Причем пусть размеры ограничены полиномом $p(n)$
Хотим показать, что $\Sigma_2^P = \Pi_2^P$. Нам хватит показать, что $\Pi_2 SAT$ лежит в Σ_2^P, так как это *со* языки друг друга.
Что у нас лежит в $\Pi_2 SAT$? Истинные формулы вида $\forall x \exists y Q(x,y)$. Давайте попробуем написать для этого языка Σ_2^P характеристику.
Пусть $T_z(y) = Q(z,y)$.
$\varphi \in \Pi_2 SAT \iff \exists C_1, \dots C_{|\varphi|} \forall z \in \{0,1\}^{|\varphi|} T_z(C_{|T_z|}(T_z)) = 1$. Мы легко можем проверить последний предикат за полиномиальное время. Причем из того, что $NP \in P/poly$ будем существовать нужный набор схем в случае, если формула действительно лежит в языке. ■

17 Билет 17

Теорема 17.1 (Первая теорема Каннана)**.** *PH не лежит в $Size[n^k]$ ни для какого k.*

Доказательство. Давайте построим язык, который будет содержаться в PH, но при этом не решаться схемами размера n^k.
Посмотрим на булевы функции от $(k+1)\log n$ переменных. Тогда по теореме про существование сложной фунцкии, среди таких функций найдется функция, схемная сложность которой более $2^{(k+1)\log n}/10n = \dfrac{n^{k+1}}{10(k+1)\log n}$, что больше n^k при больших n.
Будем с помощью кванторов задавать первую такую функцию. Скажем, что $x \in L \to \forall f (\forall C |C| \leq$

$n^k \iff \exists x C(x) \neq f(x)), \forall g(g \prec f, \exists C |C| \leq n^k \forall x C(x) = g(x)) \to f(x) = 1$. Мы сможем проверить предикат лексикографической меньшести и равенства функции, так как функцию из $(k+1)\log n$ битов можно задать таблицей истинности полиномиального размера. ■

Теорема 17.2 (Вторая теорема Каннана)**.** *Уже в $\Sigma_2^P \cap \Pi_2^P$ есть язык, который не распознается семейством полиномиальных схем.*

Доказательство. Пусть это не так. Тогда $NP \in \Sigma_2^P \cap \Pi_2^P \leq P/poly$, но тогда из теоремы Карпа-Липтона получим, что полиномиальная иерархия схлопывается на 2 уровне и $PH = \Sigma_2^P$, но это противоречие с первой версией теоремы Каннана. ■

18 Билет 18

Определение 18.1. *Семейство схем C_n, имеющих n входов, называется равномерным, если есть логарифмическая по памяти машина Тьюринга, генерирующая нам по входу 1^n выход C_n.*

Теорема 18.1. *Класс языков, распознающихся семейством равномерных схем есть класс P.*

Доказательство. Если распознается семейством равномерных схем, то давайте будем генерировать схему и проверять на наличие. Будет работать за полином, потому что алгоритм, генерирующий схему, лежит в L. Теперь заметим, что наш алгоритм, который переводит МТ в схему – тоже логарифмический. Потому что ему для построения нужно было выписывать какие-то ячейки на одном уровне и при этом добавлять ребра между некоторым константным количеством соседних ячеек, тут нет сильно затратных по памяти операций. ■

Замечание 18.1. *Часто считают, что задача поддается эффективному распараллеливанию, если ее можно решить на $poly(n)$ процессоров за $\log n$, где n – длина входа.*

Замечание 18.2. *Если смогли распараллелить на n процессорах, то можно поделить это число на какую-нибудь константу, степень распараллеливания тоже поделится примерно на константу.*

Можно представлять себе распараллеленные вычисления как схему малой глубины.

Определение 18.2. $NC = \cup_{i>0} NC^i$.
$L \in NC^i$ тогда и только тогда, когда существует последовательность равномерных схем C_n, распознающих L и глубина $C_n \leq O(\log^i n)$.

Замечание 18.3. *$NC \subseteq P$, потому что это множество слабее, чем то, которое равно P: мы добавили ограничение на глубину схемы. При этом открытым вопросом является равенство NC и P.*

В каких-то P–подобных классах интересно рассматривать $\leq_l$ сведения, потому что по Карпу и так почти все сводится друг к другу тривиально.

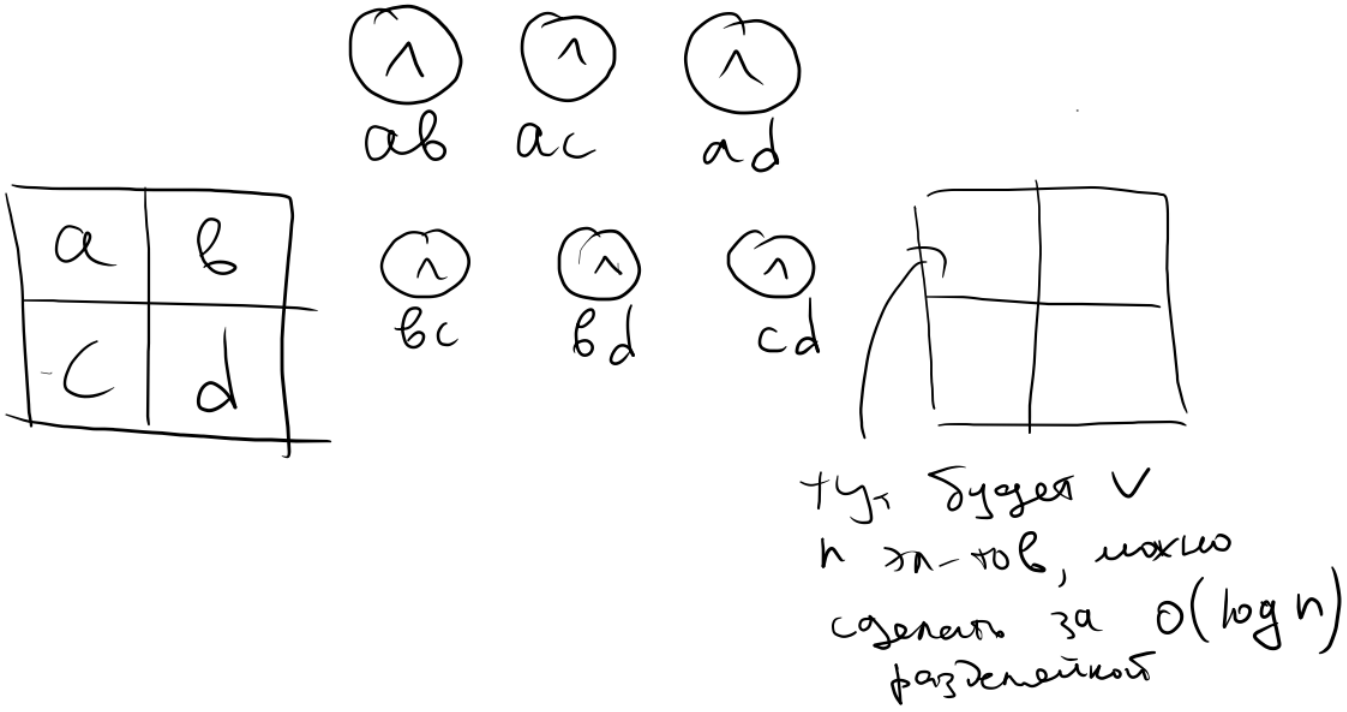
Определение 18.3. *$Circuit - Value = \{(C, x)$ C – булева схема, $C(x) = 1$ $\}$ является P-полным относительно $\leq_l$ сведений.*

Доказательство. Понятно, что этот язык лежит в P. Как свести? Давайте построим по МТ схему как обычно логарфмическим алгоритмом. ■

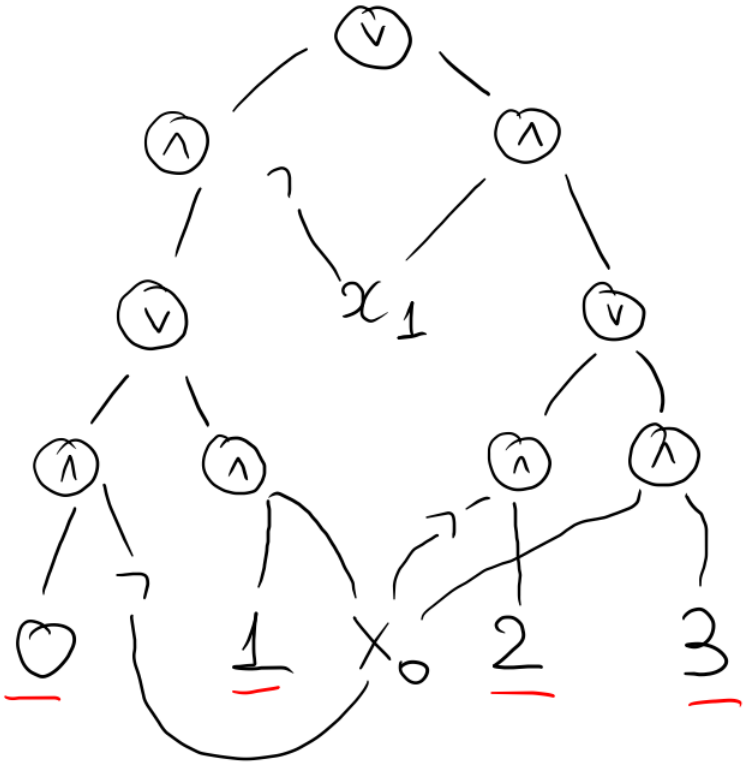
Замечание 18.4. *Попытка объяснения про задачу, почему она не распараллеливается могла бы быть такой: давайте сведем ее к $Circuit - Value$. Поскольку, неизвестно равны ли P и NC, то если бы это было не так, то попытка объяснения бы сработала.*

Утверждение 18.1. *$DPATH \in NC^2$, если граф задан матрицей смежности.*

Доказательство. Чтобы понять, есть ли путь из вершины в другую, надо k раз возвести в квадрат матрицу смежности, где $k = \lceil \log n \rceil$. Тогда давайте строить схему, делающую это. Там будет k возведений матрицы в квадрат. Причем каждое возведение добавляет в схему $O(\log n)$ высоты, что нетрудно видеть, посмотря на пример ниже.



Теперь надо взять элемент (s, t) у этой матрицы.
Покажем схему, которая поможет выбрать i-тый элемент массива, если i задано в двоичном виде. Основываемся на идее, что если старший бит i равен 1, то брать надо из правой половины, иначе – из левой. Построим для половинок схемы на $n - 1$ бите и так далее. Получится глубина $O(\log n)$, пример на рисунке для $n = 4$:



Тогда выберем из каждой строчки элемент t, а дальше из всех выбранных выберем элемент s, получится схема глубину $2\log n$. Таким образом, построили схему нужной глубины.
Важно, что все схемы, которые мы тут обсудили можно сгенерировать с использованием логарифмической памяти. Проверять это довольно нудно, но видно, что, например в последней схеме, нужно просто бежать по уровням и выводить ребра с элементами с фиксированными номерами, получающимися по какой-нибудь формуле типа $2i$, незачем хранить всё целиком. ■

Утверждение 18.2. *Классы NC^i при $i \geq 2$ замкнуты относительно логарифмических сведений.*

Доказательство. Доказательство очень глиняное. Пусть есть $B \in NC^i, A \leq_l B$, f – сведение. Тогда хотим сделать следующее: в начале вычислять сведение при помощи схемы глубины $\log^2$, а дальше считать уже спокойненько функцию схемой для языка B. Будем считать сведение так: посчитаем отдельно каждый его бит. Всего будет какой-то полином битов. Нам нужно проверить, что при нашем входе сведение выдаст на j-тый бит, бит 1. Давайте возьмем МТ, которая такая же как сведение, но, она еще принимает бит, который надо выдать и в конце приходит в состояние q_yes или q_no в зависимости от значения этого бита. Далее переделаем всё это в схему при помощи предыдущего утверждения и графа конфигураций. ■

Замечание 18.5. *Вспомним, что $DPATH$ – NL полный. Отсюда получается, что $NL \subseteq NC^2$.*

Теорема 18.2. $NC^1 \subseteq L \subseteq NL \subseteq NC^2$

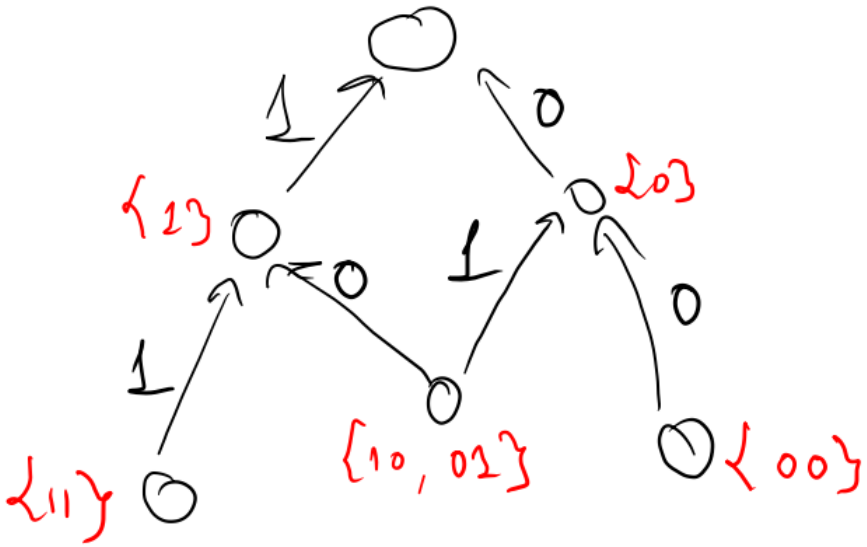
Доказательство. Из всего вышесказанного, остается показать лишь $NC^1 \subseteq L$. Возьмем какой-нибудь язык $L \in NC^1$. Хотим решить его за логарифмическую память, сделаем следующие действия по x:

1. Построим схему, решающую данный язык с помощью алгоритмы из NC^1
2. Переделаем схему в формулу
3. Вычислим формулу с помощью рекурсии

Если покажем, что каждое действие можно сделать за $O(\log n)$ памяти, то можно будет и всё сделать за такую память, так как композиция функцих, вычислимых за $O(\log n)$ памяти тоже функция такого вида.

1. Работаем тут за $O(\log)$ памяти, так как нужно просто применить алгоритм, который нам предоставляет схему за такую память из NC^1, подав ему на вход нужное число единиц.

2. Понятно, что из схемы высотой $O(\log)$ получится формула тоже не более чем такой высоты. Будем копировать вершины, построив что-то вроде двоичного бора на нашей схеме и всякий раз доходя до какой-то вершины, копируя её и выдавая номер, соответствующий пути в боре. Нам понадобится $O(\log)$ памяти, так как по сути нужно хранить нынешний путь в виде числа (можно изначально задать соответствие вершин и таких путей, чтобы понимать, в какой мы сейчас вершине). Подробнее на рисунке:



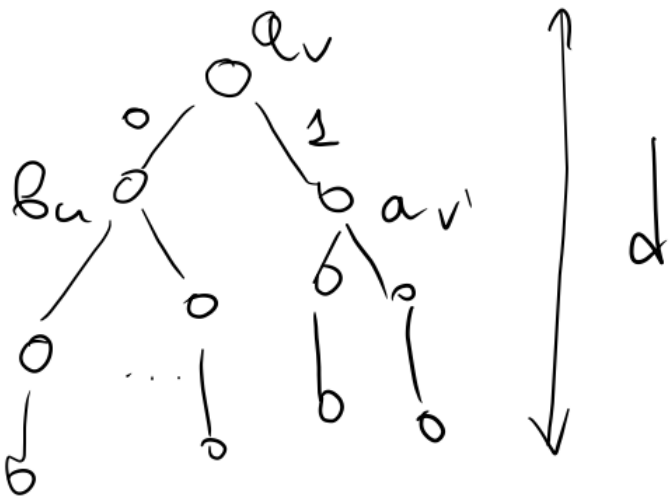
3. Вычисление формулы занимает $O(\log n)$ памяти, примерно по тому же, почему и в предыдущем пункте.

■

19 Билет 19

Определение 19.1. *Какая-то интуиция коммуникационной сложности + пример про делимость и определение EQ.*

Определение 19.2 (Коммуникационный протокол)**.** *Это дерево, в котором в каждой вершине либо что-то передает Алиса, либо Боб. То есть там написана функция Алисы из ее входа в $\{0,1\}$, либо Боба. История взаимодействия учитывается, так как везде может быть своя функция.*



Определение 19.3. *Коммуникационная сложность протокола – его высота. Обозначается как* $D(\Pi)$*. Коммуникационная сложность функции это минимальная коммуникационная сложность протокола, вычисляющего её. Обозначется аналогично.*

M_f – матрица для $f : X \times Y \to Z$, если $M_{i,j} = f(i, j)$. Комбинаторный прямоугольник – множество $X' \times Y'$ для некоторых $X' \subseteq X, Y' \subseteq Y$. Комбинаторный прямоугольник назовем одноцветным тогда и только тогда, когда все элементы матрицы, соответствующие ему равны попарно.
Заметим, что каждый ход в протоколе разбивает какой-то прямоугольник на еще несколько в зависимости от значения функции, а листья протокола соответствуют как раз одноцветным прямоугольникам, так как в них значения определены корректно и однозначно.
Отсюда получается:

Утверждение 19.1. *Если в протоколе для* f *есть* k *листьев, то* f *можно разбить на* k *одноцветных прямоугольников.*

Другими словами, число листьев в протоколе $\geq$ минимальное число одноцветных прямоугольников, на которое можно разбить M_f.

Теорема 19.1. *Сложность протокола* EQ *есть* $n + 1$.

Доказательство. Очевидно, как сделать за $n+1$. Нужна оценка снизу. Будем доказывать при помощи подсчета одноцветных прямоугольников в матрице для EQ. Эта матрица это E_{2^n}. В ней никакие 2 единицы не могут принадлежать одному прямоугольнику. А значит, прямоугольников хотя бы $2^n + 1$, что свидетельствует о том, что высота протокола будет $\geq n + 1$. ■

20 Билет 20

Оказывается, что можно доказывать некоторые забавные вещи при помощи построенной конструкции.

Теорема 20.1. *Для одноленточной МТ, рещающей язык палиндромов, выполнено* $T \cdot S = \Omega(n^2)$*, где* T *– время работы,* S *– затраченая память.*

Доказательство. Возьмем такую МТ M. По ней построим коммуникационный протокол для вычисления EQ. Будем рассматривать строки $x0^n y^{rev}$, которые являются палиндромами тогда и только тогда, когда $x = y$. Пока МТ не вышла за самую правую перегородку, стоящую после искуственно

добавленных нулей, Алиса будет эмулировать ее, а дальше отправит Бобу состояние ленты, а также состояние МТ. Далее также будет делать Боб, но наоборот, до момента, пока МТ не дойдет до самой левой перегородки после искуственно добавленных нулей.

Тогда сложность такого протокола равна $(O(1) + O(S(x0^n y^{rev}))\frac{T(x0^n y^{rev})}{n} \geq n + 1$, отсюда искомое неравенство. n в знаменателе, так как между передачами проходит хотя бы n действий МТ. ■

21 Билет 21

Определение 21.1 (Про отношения)**.** .

1. *Отношением назовем множество троек $R \subseteq X \times Y \times Z$.*
2. *Отношение тотально, если $\forall x, \forall y, \exists z, (x, y, z) \in R$.*
3. *Функция f реализует тотальное отношение, если $\forall x \in X, \forall y \in Y, (x, y, f(x, y)) \in R$.*
4. *Коммуникационный протокол для тотального отношения R – это коммуникационный протокол для какой-нибудь функции его реализующей.*
5. *Коммуникационная сложность тотального отношения – глубина наимненьшего коммуникационного протокола для него.*

Определение 21.2 (Отношение Карчмар-Вигдерсон)**.** . $f : \{0,1\}^n \to \{0,1\}$. $f(x) = 0$, $f(y) = 1$. $KW_f = \{(x, y, z) \mid \ f(x) = 0, f(y) = 1, x_i \neq y_i \ \}$.

Утверждение 21.1 (Лемма Карчмар-Вигдерсон)**.** *Если у булевой функции f есть формула, вычисляющая ее с l листьями, то есть коммуникационный протокол, который ее вычисляет с не более чем l листьями.*

Доказательство. В начале перенесем отрицания к листьями в формуле. Далее поддерживаем инвариант в каждой вершине, что у Алиса прообраз нуля функции, вычисляемой в этой вершине, у Боба – прообраз единицы. В узлах, где последняя операция ИЛИ – выбирать будет Боб – ему надо пойти в единичное поддерево, а там где И – Алиса. В конце в листьях как раз и будут отличающиеся биты, так как там вычисляются тривиальные функции. ■

Другими словами, кол-во листьев в формуле $\geq$ минимальное количество листьев в протоколе для отношения Карчмар-Вигдерсон этой функции.

22 Билет 22

Теперь применим то, что было в предыдущих билетах. А именно, цепочку неравенств для булевой функции f:

$$\#min\_leafs\_in\_formula(f) \geq min\_height(KW_f) \geq min\_cnt\_monochrome(M_f)$$

Утверждение 22.1. *X, Y – непересекающиеся подмножества $\{0,1\}^n$, $C = \{(x, y) \mid x \in X, y \in Y,$ x, y различаются ровно в одном бите $\}$. M – матрица на $X \times Y$. $\forall x \in X, y \in Y$ выполнено $M_{x,y} =$ бит различия x, y. Тогда*

$$T \geq \frac{|C|^2}{|X||Y|}$$

где T – минимальное число одноцветных комбинаторных прямоугольников, на которое можно разбить M.

Доказательство. Пусть $R_1 \sqcup R_2 \sqcup \ldots \sqcup R_T$ – разбиение M на одноцветные прямоугольники. $m_i = |R_i \cap C|$.
Посмотрим на R_i, утверждается, что в нем нет двух элементов из C в одной строке. Дейсвительно, все пары отличаются в одинаковом бите. Если ровно в одном, то не может быть такого, что такие пары две в строке, иначе у нас какой-то элемент написан дважды. Аналогично для столбцов. Отсюда выполняется $|R_i| \geq m_i^2$.

$$|C|^2 = (\sum_{i \in [T]} m_i)^2 \leq T \sum_{i \in [T]} m_i^2 \leq T|X||Y|$$

. Отсюда искомое неравенство. Использовали КБШ в оценке (представили как $1 \cdot m_i$). ■

Утверждение 22.2. *Формула для $Parity_n$ имеет размер $\Omega(n^2)$. Воспользуемся цепочкой неравенств выше. Для оценки самой правой величины в цепочке используем предыдущее утверждение. Возьмем $X = Parity_n^{-1}(0)$, $Y = Parity_n^{-1}(1)$. Тогда сколько есть для элемента из X парных с ним таких, что они отличаются в одном бите из Y? Понятно, что все такие лежат в Y и их n штук. Тогда неравенство говорит нам, что $T \geq n^2$, что и хочется.*

23 Билет 23

Определение 23.1. *Вероятнотсная МТ может быть определена двумя эквивалентнтыми способами:*

1. *Это НМТ, у которой следующий переход выбирается равновероятно.*
2. *это ДМТ, у которой есть лента случайных битов, которую разрешается только читать и бежать по ней только вправо.*

Определение 23.2. *$L \in BPP$, если существует вероятностная МТ M, которая всегда останавливается и работает за полином $p(|x|)$ времени и при этом $Pr[M(x) \leq L(x)] < 1/3$.*

Замечание 23.1. *$P \subseteq BPP$, а обратное включение пока не доказано.*

Определение 23.3 (Классы с односторонней ошибкой)**.** *$L \subseteq RP$, если существует вероятностная МТ M, которая всегда останавливается и работает за полином $p(|x|)$ времени и при этом:*

1. *$x \notin L \Longrightarrow M(x) = 0$*
2. *$x \in L \Longrightarrow Pr[M(x) = 1] \geq 1/2$.*

Например, $Primes \in coRP$, так как мы знаем какие-то вероятностные тесты, которые могут составное число назвать простым, но не наоборот.

Утверждение 23.1 (Лемма Шварца-Зиппеля)**.** *$\mathbb{F}$ – поле, $|\mathbb{F}| < +\infty$, $p \in \mathbb{F}[x_1, \ldots, x_n], p \not\equiv 0$, тогда при случайном независимом выборе $x_1, \ldots, x_n$ из $\mathbb{F}$ выполняется*

$$Pr[p(x_1, \ldots, x_n) = 0] \leq \frac{d}{|\mathbb{F}|}$$

где $d = \deg p$.

Доказательство. индукция по n. В случае $n = 1$ утверждение следует из теоремы Безу.
переход: $f = a_n^{i_1} f_{i_1} + \ldots + a_n^{i_k} f_{i_k}$ для некоторых ненулевых f_{i_j} и $i_1 < i_2 < \ldots i_k$.
Обозначим $S = |\mathbb{F}|$, $\mathbb{F} = \{0, \xi_2, \ldots, \xi_S\}$.

1. Если $i_1 \neq 0$. Распишем по формуле полной вероятности, пробегая все значения a_n.

$$Pr[f=0] = Pr[f=0|a_n=0]Pr[a_n=0] + \ldots + Pr[f=0|a_n=\xi_S]Pr[a_n=\xi_S]$$

Подстановка $a_n = 0$ обнулит f, тогда $Pr[f=0|a_n=0] = 1$. Все остальные подстановки понизят степень хотя бы на 1. Тогда выполняется

$$Pr[f=0] \leq \frac{1}{S} + \frac{1}{S} \cdot \frac{S(d-1)}{S} = \frac{d}{S}$$

2. доказательство аналогично кроме того, что $a_n = 0$ не обнуляет f.

■

Пример 23.1 (Polynomial identetity testing)**.** *Хотим проверять $P \equiv 0$ для некоторого полинома P.*

1. *Если P задан над некотором полем $\mathbb{F}$, таким что $|\mathbb{F}| >> \deg P$, то нужно подставлять случайную точку поля, если $P \neq 0$, то мы с большой вероятностью получим не 0.*

2. *Если хотим проверить равенство 0, как многочлену от целых значений, то возьмем какое-нибудь поле остатков по простому модулю, такое что, например, $|\mathbb{F}| > 1000 \deg P$ и будем подставлять точки в нем. Это не даст гарантии, что многочлен нулевой. Возможно, его значения совпадают по взятому модулю. Можно в таком случае оценить максимальную разность двух значений многочлена и взять побольше модулей так, чтобы если уж эта разность на всех них поделилась, то она точно нулевая.*

В частности, язык нулевых многочленов, записанных в виде формулы или схемы, лежит в классе RP.

24 Билет 24

Вспомним про ветвящиеся программы.

Определение 24.1 (1-BP, read-once branching problem)**.** *Ветвящаяся программа лежит в $1-BP$, если на любом пути от входа до выхода любая переменная встречается не более 1 раза.*

Хотим сравнивать на эквивалентность две ветвящиеся программы с помощью сравнения многочленов на равенство.

Определение 24.2. *Мультилинейный многочлен – каждая переменная в каждом мономе не более чем в 1 степени.*

Утверждение 24.1. *h, g – мультилинейные многочлены. h, g – совпадают на булевом кубе. Тогда h, g совпадут всюду.*

Доказательство. Индукция по степени. Для нулевой степени понятно, это константы, совпадают и совпадают.
Переход. $h = h_{|x_n=0}(1-x_n) + h_{|x_n=1}x_n$, аналогично для g. Применяем для кусочков без x_n индукционное предположение. ■

Пример 24.1 (Вероятностная проверка на эквивалентность $1-BP$ программ)**.** *Построим по $1-BP$ программе мультилинейный многочлен. Построим рекурсивно. В начале построим для левого сына f_l и для правого f_r. Тогда $f = (1-x)f_l + xf_r$. Тогда у нас степень этого многочлена $\leq n$, где n – размер программы. Берем поле $|\mathbb{F}| > 1000n$ и делаем проверку с ошибкой $1/1000$.*

25 Билет 25

Теорема 25.1 (Оценка Чернова). *$X_1, \ldots, X_n$ – независимые случайные величины, распределенные на $\{0,1\}$, то есть $E[x_1] = \ldots = E[x_n]$, тогда*

$$Pr[|\frac{X_1 + \ldots + X_n}{n} - \mu| \geq \varepsilon] \leq 2e^{-2\varepsilon^2 n}$$

Теорема 25.2 (Понижение ошибки в BPP). *Если $L \in BPP$ и $m(n)$ – некоторый полином, то существует полиномиальная вероятностная МТ M, которая распознает L и вероятность ее ошибки не превосходит $2^{-m(n)}$.*

Доказательство. Берем машину из определения M_0 с вероятностью ошибки $< 1/3$.
Теперь предположим, что мы повторили N раз ее вычисления и выдали самый частый результат. Оценим вероятность ошибки.

$$X_n = \begin{cases} 1 & \text{запуск n отработал верно} \\ 0 & \text{запуск n отработал неверно} \end{cases}$$

$E[X_n] \geq 2/3$.

$$Pr[\text{машина ошиблась}] \leq Pr[X_1 + \ldots + X_n \leq N/2] = Pr[\frac{X_1 + \ldots + X_N}{N} \leq 1/2] \leq$$

$$\leq Pr[|\frac{X_1 + \ldots + X_N}{N} - E[X_n]| \geq 1/6] \leq e^{-NC}$$

, для некоторого $C > 0$. Тогда возьмем $N = O(m(n))$ с достаточно большой константой, чтобы перебить $2^{-m(n)}$ и получим нужную полиномиальную машину. ■

Теорема 25.3 (Адлеман). *$BPP \in P/Poly$.*

Доказательство. Покажем, что для всех входов длины n есть одинаковые случайные биты, с помощью которых МТ может верно решить наш язык.
Для этих целей давайте возьмем МТ с ошибкой $\frac{1}{2^{n^2}}$. Пусть эта МТ на входах длины n работает $m(n)$ времени. Тогда рассмотрим множество "плохих"случайных битов, таких, которые приводят нас к неверному ответу. R_x – множество масок из $\{0,1\}^{m(n)}$, таких что $M_r(x)$ работает неправильно. Тогда, так как вероятность ошибки небольшая, то $|R_x| \leq 2^{m(n)-n^2}$.
Давайте все такие плохие биты объединим по всем входам.

$$|\cup_{x \in \{0,1\}^{m(n)}} R_x| \leq 2^n 2^{m(n)-n^2} = 2^{(n-n^2)m(n)} < 2^{m(n)}$$

при $n > 1$. Отсюда понимаем, что есть "универсальный"набор случайных битов. Давайте использовать его в качестве неравномерной подсказки. ■

Утверждение 25.1. *Пусть $n < m < 2^n$ и $S \subseteq \{0,1\}^m$, $k = \lceil m/n \rceil + 1$. Тогда:*

1. *$|S| \leq 2^{m-n} \Longrightarrow$ не существует $r_1, \ldots, r_k$ таких, что $\cup_{i \in [k]}(S + r_i) = \{0,1\}^m$.*
2. *$|S| \geq 2^m(1 - 2^{-n})$, тогда такие $r_1, \ldots, r_k$ существуют.*

Доказательство. 1. $|\cup| \leq k2^{m-n} \leq 2^{m-n}(n/k + 2) \leq 2^m$.

2. Давайте нагенерим масочки r_i независимо и равновероятно. Тогда оценим вероятность того, что какой-то элемент не покрыт:

$$Pr[x \text{ не покрыт}] = Pr[\forall i, x + r_i \notin S] \leq 2^{-nk}$$

$$Pr[\text{какой-то } x \text{ не покрыт}] \leq 2^{m-nk} < 1$$

значит, есть решение, в котором всё покрыто.

■

Теорема 25.4 (Сипсер-Гач)**.** $BPP \in \Sigma_2^P \cap \Pi_2^P$.

Доказательство. Возьмем $L \in BPP$. Достаточно показать, $L \in \Sigma_2^P$, так как нетрудно видеть, $BPP = coBPP$, отсюда последует $coBPP = BPP \in \Pi_2^P$, что и надо.
Пусть МТ M для L использует $m(n)$ случайных битов на входе длины n и ошибка у нее не превосходит 2^{-n}. Возьмем некоторый $x \in L, |x| = n$. Пусть $R_x \subseteq \{0,1\}^{m(n)}, r \in R \iff M_r(x) = 1$.

1. $x \in L$, тогда $|R_x| \geq 2^{m(n)}(1 - 2^{-n})$
2. $x \notin L$, тогда $|R_x| \leq 2^{m(n)-n}$

Применим предыдущее утверждение.
$x \in L \Leftrightarrow \exists r_1, \ldots, r_{\lceil m(n)/n \rceil + 1} \forall a \in \{0,1\}^{m(n)} Q(r_1, \ldots, r_k, x)$, где Q – проверяет, что для какого-нибудь i: $M_{a+r_i}(x) = 1$. Таким образом получили нужную характеристику. ■

26 Билет 26

Определение 26.1 (Интерактивный протокол)**.** *Есть V и P, которые общаются и V вероятностно проверяет доказательство, которое ему предоставлет P. Протокол состоит из передачи информации от V к P по очереди. V является вероятностным алгоритмом, работающим полиномиальное время от входа.*
$P(x, h)$, где x – вход, h – история взаимодействия.
$V(x, h, r)$, где r – случайные биты.
Результат взаимодействия V и P на входе x обозначается как $< V, P > (x)$. Это некоторая случайная величина, потому что зависит от случайных битов.

Определение 26.2. *$L \in IP[K(n)] \iff \exists$ функция V и полиномиальный относительного первого аргумента алгоритм P:*

1. *$\forall P'$ V заканчивает общение с P' на входе x за $K(|x|)$ шагов.*
2. *$\forall x \in L$ выполнено: $Pr[< V, P > (x) = 1] > 2/3$.*
3. *$\forall x \notin L$ выполнено: $Pr[< V, P > (x) = 1] < 1/3$.*

Пример 26.1. *GNI – язык, в котором лежат пары неизоморфных графов на одном числе вершин.*

Утверждение 26.1. $GNI \in IP[2]$

Доказательство. Протокол будет устроен так:

1. Рассмотрим протокол для графов $G_0, G_1, |V| = n$
2. $Verifier$ генирирует число $i \in \{0,1\}$
3. $Verifier$ генерирует $P \in S_n$

4. $Verifiter$ перемешивает граф G_i сгенерированной перестановкой и отсылает этот граф

5. $Prover$ посылает ответ на вопрос: какого графа была выдана перестановка ему сейчас?

6. Если $Prover$ прислал $j = i$, то выдаем, что графы неизоморфны, иначе – изоморфны.

Заметим, что если графы были неизоморфны, то найдется честный $Prover$, который это подтвердит, он просто пришлет нам верный ответ на наш вопрос и мы будем всегда верно говорить ответ. Если же графы были изоморфны, то на всех пруверах мы будем не более чем в половине случаев говорить неправду (Если прувер угадал наше число i).
Чтобы соответствовать определению, повторим данный протокол 2 раза. Но заметим, что можно не делать 4 передачи, просто передадим сразу пару и запросим пару. ■

Определение 26.3. $IP = \cup_{c>0} IP[n^c]$

Замечание 26.1. *В нашем примере существенно использовалось то, что у двух сторон разные случайные биты. Потому что иначе Prover смог бы всегда нас обманывать. Также заметим, что в примере, честный прувер всегда убеждает нас в верных фактах. Чуть позже станет ясно, что первого из этих условий можно избежать, а второе можно добавить в определение и язык не изменится.*

Теорема 26.1 ($IP \subseteq PSPACE$)**.**

Доказательство. Наша цель – по протоколу найти прувер, максимизирующий вероятность того, что мы примем x и сравнить эту вероятность с $2/3$.
Построим полное дерево взаимодействия. Можно считать, что его высота $t(n)$, каждая переданная строка тоже оценивается полиномом $q(n)$. Теперь хотим посчитать некоторую динамику:

$$p_u = \max_{P-Prover} Pr[P \text{ доходит до } u, \text{при взаимодействии с этим прувером получим 1}]$$

.
Как считать такую штуку для листьев? Если в листе написан 0, то вероятность равна 0. Для единичных листов нам известно то, как должен вести себя прувер, переберем случайные биты проверяльщика и посчитаем, какая доля случайных битов приводит нас к этому листу. Всего понадобится полином случайных битов, каждая эмуляция проводится также за полином.
Как считать такую штуку для вершин, соответствующих ходам прувера? Наш прувер, дошедший до этой вершины дальше сделает ход в какую-то из вершин – детей. Он хочет максимизировать вероятность дохождения до 1, значит нужно взять максимальный ответ для детей.
Как считать такую штуку для вершин, соответствующих ходам проверяльщика? Так как его алгоритм нам известен, то для него значение равно просто вероятности попасть в 1, начиная с этой вершины. Это равно сумме p_{v_k} по всем сыновьям.
Посчитав такую динамику, сравним значение в начальной вершине с $2/3$ и выдадим ответ. ■

27 Билет 27

Теорема 27.1 (Шамир)**.** $PSPACE \subseteq IP$

Доказательство. Можно показать $TQBF \subseteq IP$. Тогда вместо того чтобы доказывать протоколом $x \in L$ будем показывать $f(x) \in TQBF$ для f – сведения.
Будем переводить формулу φ, $|\varphi| = m$ из $TQBF$ в формулу над полем. Такой процесс называется алгебраизация.

$$x \Longrightarrow x$$

$$\neg\varphi \Longrightarrow 1 - \widetilde{\varphi}$$
$$a \wedge b \Longrightarrow \widetilde{a}\widetilde{b}$$
$$a \vee b \Longrightarrow 1 - (1 - \widetilde{a})(1 - \widetilde{b})$$

Также введем операторы над многочленами для кванторов. A_x, E_x. Значение следующее:

$$A_x\widetilde{\varphi} = \varphi_{|x=0} \cdot \varphi_{|x=1}$$

$$E_x\widetilde{\phi} = 1 - (1 - \varphi_{|x=0})(1 - \varphi_{|x=1})$$

Заметим сразу, что A_x и E_x убивают переменную, по которой они написаны.
На данном этапе получилась формула такого вида: $Q_{1_{x_1}} \dots Q_{n_{x_n}}\widetilde{\varphi}$. Хотим сделать следующее: переделать ее, чтобы на каждом суффиксе получающийся многочлен имел маленькую степень. Введем оператор линеаризации по переменной: $L_x\varphi = \varphi_{|x=0}(1 - x) + x\varphi_{|x=1}$. Он введен так, чтобы степень x в формуле была максимум 1.
Тогда запишем формулу так: $\widetilde{Q_1}x_1Lx_1\widetilde{Q_2}x_2Lx_2Lx_1 \dots \widetilde{\varphi}$. Тогда по каждой переменной степень многочлена-суффикса данной строки не превосходит m, но сама строка увеличилась несильно, длина возвелась в квадрат.
Заметим, что мы так строили полином, чтобы значение его на булевом кубе совпадали со значением формулы из $TQBF$.
Теперь строим протокол. Протокол будет заниматься тем, что доказывать про значение формулы, что оно равно 1 при некоторых подставленных уже переменных. Будем снимать кванторы слева направо.
Пусть хотим доказывать, что у какого-то многочлена, соответствующего суффиксу строки, значение равно c на подстановке p.

1. В самом начале хотим доказывать про формулу, что она тождественно истина на пустой подстановке.

2. Если в формуле не осталось операторов, то $Verifier$ может сам проверить на истинность свое утверждение, подставив в формулу значения и посчитав.

3. Пусть последний оператор Ax_i. Попросим у прувера многочлен $h(x_i)$, который получится, если снять последний оператор и сделать подстановку. Проверим, что выполнено $h(0)(1) = c$, если это не так – то нам точно прислали фейковый многочлен.

4. Пусть последний оператор Ex_i. Попросим аналогично у прувера многочлен $h(x_i)$ и проверим, $c = 1 - (1 - h(0))(1 - h(1))$.

5. Последний оператор Lx_i. Это возможно, если x_i уже задано значение r_1 подстановкой p. Попросим аналогично многочлен h и проверим: $(1 - r_1)h(0) + r_1h(1) = 1$.

Далее пополняем подстановку. Подставим вместо x_i случайное значение r из поля. Запустимся рекурсивно.
Если формула была истинной, то существует хороший $Prover$, так как он может просто давать нам правильные коэффициенты многочлена.
Теперь поймем, как нам добиться, чтобы нас не могли сильно обмануть.
Если формула была ложна, а потом стала в какой-то момент истинна, то в какой-то момент произошел переход. Если нам дали неверный h, то он совпадает с верным не более чем в m точках и вероятность, что мы попадем в истинное утверждение на следующем шаге не превосходит $\dfrac{m}{|\mathbb{F}|} < \dfrac{1}{m^3}$, если взять $|\mathbb{F}| > m^4$. Тогда вероятность перейти из лжи в истину хотя бы раз не превосходит $\dfrac{m^2}{m} = \dfrac{1}{m}$. ■

Замечание 27.1. *Из доказательства следует, что можно сделать с определением IP следующее:*

1. *потребовать, чтобы на лежащих в языках элементах честный прувер всегда убеждал проверяльщика.*

2. *можно рассматривать только доказывающих, которые использует полиномиальную память. Действительно, пруверу надо уметь только интерполировать многочлены. Это можно для многочленов, заданных в нашем виде, делать рекурсивным алгоритмом с полиномиальной памятью.*

3. *проверяющий может не скрывать от доказывающего случайные биты. Заметим, что мы в нашем протоколе можем не скрывать выбранное значение, так как дальше оно по сути не используется.*