Семинары 4-5.
$\mathcal{NP}$-полнота, пространственная сложность, **L** и **NL**

Составил Р. Делла Пиетра

(7,14).2.20

1 $EXACTCOVER$

Покажем, что задача точного покрытия множества $\mathcal{NP}$ полная.
Формулировка такая: на вход подаётся число n семейство множеств $S \subset 2^{\overline{1,n}}$, нужно определить, можно ли из S выбрать дизъюнктное подсемейство, покрывающее $\overline{1,n}$.
Сведём $3SAT$ к этой задаче.
Для каждой переменной, используемой в булевой формуле, «бронируем» 3 числа: p, p_0, p_1. Для каждого дизъюнкта 4 числа: C, C_1, C_2, C_3. С каждой переменной добавляем множества $\{p, p_0\}$ и $\{p, p_1\}$ в семейство, с каждым дизъюнктом добавляем $\{C\}, \{C, C_i\}, \{C, C_i, C_j\}\ \forall i \neq j$. Также добавляем множества такого вида: для каждой переменной делаем множества $\{p_1, \dots\}$ и $\{p_0, \dots\}$, где в первом троеточие — C_i, если в дизъюнкте C переменная на i-м месте, а во втором C_j если отрицание переменной на j-м месте.
Пример: $\varphi = (x \vee y)(\overline{y} \vee z \vee x) \implies A = (x \vee y), B = (\overline{y} \vee z)$.
Добавляем числа $x, x_0, x_1, y, y_0, y_1, z, z_0, z_1, A, A_1, A_2, A_3, B, B_1, B_2, B_3$, то есть $n = 17$,
и $S = \{\{x, x_0\}, \{x, x_1\},$ *то же самое для* y *и* z,
$\{A\}, \{A, A_1\}, \{A, A_2\}, \{A, A_3\}, \{A, A_1, A_2\}, \{A, A_2, A_3\}, \{A, A_3, A_1\},$ *то же самое для* B,
$\{x_1, A_1, B_3\}, \{y_1, A_2\}, \{y_0, B_1\}, \{z_1, B_2\}\}$.
Покажем, что сводимость корректна.
Если в выполняющем наборе p принимает значение i, выбираем $\{p, p_{1-i}\}$ и $\{p_i, \dots\}$. После выбора всех таких множеств для всех переменных, выбираем остатки для дизъюнктов.
На том же примере: выполняющий набор $x = 1, y = 0, z = 1$.
$\{x, x_0\}, \{y, y_1\}, \{z, z_0\}, \{x_1, A_1, B_3\}, \{y_0, B_1\}, \{z_1, B_2\}, \{A, A_2, A_3\}, \{B\}$ дизъюнктно покрывают всё S.
В другую сторону: есть дизъюнктное покрытие, значит можно составить выполняющий набор. Это ясно интуитивно: из множеств вида $\{p, p_i\}$ для каждой переменной будет выбрано одно, которое как раз будет говорить о значении переменной. Из множеств вида $\{p_i, \dots\}$ тоже будут выбраны по одному для каждой переменной, показывающему, какие дизъюнкты выполняются. Остаток будет состоять из множеств вида $\{A\}, \{A, A_i\}, \{A, A_i, A_j\}$, где i, j — места в дизъюнкте, указывающие на невыполненные переменные. При этом в S нет множеств вида $\{A, A_1, A_2, A_3\}$, поэтому для каждого дизъюнкта хотя бы одна переменная будет выполняться, то есть формула выполнима.
Таким образом, $3SAT \leqslant_p EXACTCOVER$. По теореме Кука-Левина $3SAT \in \mathcal{NP}c$,
поэтому $EXACTCOVER \in \mathcal{NP}h$.
Тривиально проверяется, что $EXACTCOVER \in \mathcal{NP}$, поэтому $EXACTCOVER \in \mathcal{NP}c$.

2 Полиномиальная иерархия

Введём группу классов, более сложных, чем $\mathcal{NP}$ и $co\mathcal{NP}$, но всё ещё требующих полиномиальные по времени вычисления.

$$\Sigma_0 = \mathcal{P} \quad \Sigma_1 = \mathcal{NP} \quad \Sigma_k = \{L | x \in L \iff \exists y_1 \forall y_2 \ldots (\forall, \exists) y_k : R(x, y_1, \ldots, y_k) = 1 : R \in \mathfrak{F}_p(|x|), y_i < poly(|x|)\}$$

$$\Pi_0 = \mathcal{P} \quad \Pi_1 = co\mathcal{NP} \quad \Pi_k = \{L | x \in L \iff \forall y_1 \exists y_2 \ldots (\forall, \exists) y_k : R(x, y_1, \ldots, y_k) = 1 : R \in \mathfrak{F}_p(|x|), y_i < poly(|x|)\}$$

$$\mathcal{PH} = \bigcup \Sigma_k = \bigcup \Pi_k$$

2.1 Примеры задач

2.1.1 SAT

Для Σ_k есть обобщение задачи SAT, так называемая $\Sigma_k SAT$: язык булевых формул от k групп переменных: $\exists \overrightarrow{y_1} \forall \overrightarrow{y_2} \ldots \varphi(\overrightarrow{y_1}, \ldots, \overrightarrow{y_k}) = 1$. Эта задача Σ_k полная.

2.1.2 Графы

Задача клики также обобщается: язык $\{(G, k)\}$, где G — описание графа, а k — максимальный размер клики в графе, лежит в Σ_2.

2.2 Схема вложений

В каждом из предикатов можно игнорировать крайние справа и слева переменные, поэтому легко видеть, что $\Sigma_k \cup \Pi_k \in \Sigma_{k+1}$, $\Sigma_k \cup \Pi_k \in \Pi_{k+1}$, но утверждение $\Sigma_{k+1} = \Pi_{k+1}$ нетривиально и может быть неверно.

2.3 Схлопывания

Что происходит, если $\mathcal{P} = \mathcal{NP}$? Вся полиномиальная иерархия схлопывается в $\mathcal{P}$. Также могут происходить другие сценарии: если $\exists k : \Sigma_k = \Sigma_{k+1}$ или $\Sigma_k = \Pi_k$, то полиномиальная иерархия коллапсирует до этого уровня.

3 Пространственная сложность

$$DSPACE(s(n)) = \{L | \exists \textit{детерминированная МТ, разрешающая } L\textit{, затрагивая } O(s(n)) \textit{ ячеек ленты}\}$$

$$NSPACE(s(n)) = \{L | \exists \textit{недетерминированная МТ, разрешающая } L\textit{, затрагивая } O(s(n)) \textit{ ячеек ленты}\}$$

$$\mathfrak{F}_s(n) = \{\textit{функции, вычислимые на } poly(n) \textit{ ячейках ленты}\}$$

$$\mathcal{P}SPACE = \bigcup_{c=1}^{\infty} DSPACE(n^c) = \{L \mid \exists \chi_L(x) \in \mathfrak{F}_s(|x|),\ \chi_L \textit{ — характеристическая функция}\}$$

$$\mathcal{NP}SPACE = \bigcup_{c=1}^{\infty} NSPACE(n^c)$$

Во всех вопросах, касающихся пространственной сложности, измеряется дополнительная память, то есть память, которая требуется разрешающему алгоритму, не учитывая сам вход.

3.1 Теорема Сэвича

$DSPACE(s(n)) \subseteq NSPACE(s(n)) \subseteq DSPACE(s^2(n))$, если $s(n)$ растёт быстрее логарифма.
Из теоремы следует, что $\mathcal{P}SPACE = \mathcal{NP}SPACE$

3.2 Примеры задач

3.2.1 *SPACE TM SAT*

$SPACE\,TM\ SAT = \{(M, x, 1^s) | M(x) = 1$ и вычисление $M(x)$ требует s ячеек$\}$ — $\mathcal{P}SPACE$-полная задача.

3.2.2 *True Quantified Boolean Formulae*

$TQBF = \{\varphi(x_1, x_2, \ldots, x_k) | \forall x_1 \exists x_2 \ldots (\forall, \exists) x_k : \varphi(x_1, \ldots x_k) = 1\}$. Покажем по индукции, что она лежит в $\mathcal{P}SPACE$.
Для того, чтобы определить значение формулы без кванторов, требуется логарифмическая память (об этом подробнее на следующем семинаре).
Для формулы из одной переменной делаем такие действия: если попался квантор всеобщности, подставляем по очереди 0 и 1, вычисляем формулу, если что-то оказалось 0 прерываем с ответом 0. Если квантор существования, делаем то же самое, но прерываем, если попался 1.
Легко видеть, что здесь нужна полиномиальная память от длины формулы, т. к. для ответов нужна константа, а для вычислений логарифм.
База индукции есть, далее пусть мы умеем решать задачу для любого количества
переменных $k < n$ с любыми кванторами. Для формулы с n переменными поступим таким же образом: вместо первой переменной подставляем фиксированное значение 0 или 1. Новая формула будет булевой формулой от $n-1$ оставшихся переменных, то есть её можно проверить на полиномиальной памяти по предположению индукции. Таким образом, мы можем вычислить ответ для формулы из n переменных, и пройтись по 0 и 1 и определить выполнение квантора. За новый шаг индукции мы снова использовали константу памяти и логарифм для вычислений.
Всего переменных не больше, чем $|\varphi|$, глубина рекурсии не больше длины входа, поэтому используемая память линейна.

4 $\mathbf{L}, \mathbf{NL}$

$$\mathbf{L} = DSPACE(\log n)$$
$$\mathbf{NL} = NSPACE(\log n)$$

4.1 Примеры задач из L

4.1.1 LE

$LE = \{(x, y) \mid x \leqslant y\}$. Для того, чтобы сравнить два числа, нужно проверить их длины и, если они одинаковые, сравнить посимвольно.
Для проверки длин потребуются два счётчика, каждый из которых занимает $\max(\log\log x, \log\log y)$ памяти. Двойной логарифм т. к. длина входа задачи $\log x + \log y$. Посимвольное сравнение требует 1 бит памяти.

4.1.2 ADD

$ADD = \{(x, y, z) \mid x + y = z\}$. Будем вычислять сумму x и y побитово и сравнивать с соответствующим битом z. Для этого понадобится 1 бит — результат сложения, 1 бит для переносов и $\log\log z$ бит для счётчика текущего сравниваемого бита.

4.1.3 MUL

$MUL = \{(x, y, z) \mid xy = z\}$. Также будем вычислять побитово, но нам понадобится большее количество счётчиков, но идея вычислять ответ побитово и сравнивать с z тоже работает.

4.2 Сертификатное определение NL

$L \in \mathbf{NL} \iff \exists V(x, y) : x \in L \iff \exists y : V(x, y) = 1$ с данными ограничениями:
доступ к x не ограничен, y можно читать только один раз слева направо, вычисления должны требовать логарифмической памяти.

4.3 $LOGSPACE$ сводимость

$A \leqslant_l B \iff \exists f$ требует для вычислений логарифмическую дополнительную память от длины входа, и $x \in A \iff f(x) \in B$.
Композиция $LOGSPACE$ функций тоже $LOGSPACE$ функция, поэтому работают аналогичные свойства, как и для m и p сводимостей: рефлексивность и транзитивность. Кроме того, если $B \in (\mathbf{N})\mathbf{L}$, то A лежит в том же классе.

4.4 $PATH$

Покажем, что задача $PATH = \{(G, s, t) \mid$ в графе G есть путь из вершины s в вершину $t\}$ NL-полная относительно l-сводимости.
Для начала покажем, что он лежит в NL: сертификатом будет последовательность вершин, а верификатор пусть проходит по ней и проверяет, что соседние вершины в пути связаны ребром. Для этого из дополнительной памяти понадобится номер текущей вершины, то есть $\log n$ при входе длины $n^2 + 2\log n$.
Покажем полноту: пусть есть некоторая задача из **NL**. Это значит, что для этой задачи есть недетерминированная МТ, разрешающая её, требуя логарифмическую дополнительную память. По определению недетерминированная МТ принимает слово, если существует путь в графе конфигураций от начального состояния к принимающему. Идея понятна, осталось доказать, что такая сводимость корректна по памяти.