1. *Класс NP*

Определение 1. *$L \in NP$ т. и т. т. когда существует некоторая машина Тьюринга M и полином $p(n)$, такие что на любом входе (x, y) M делает не более $p(|x| + |y|)$ шагов и*

$$\begin{cases} \forall x \in L\ \exists y : |y| \leqslant p(|x|) \text{ и } M(x,y) = 1; \\ \forall x \notin L\ \forall y : |y| \leqslant p(|x|),\ \text{выполнено } M(x,y) = 0. \end{cases}$$

Последнее условие в более короткой форме выглядит так

$$x \in L \iff \exists y : |y| \leqslant p(|x|) \text{ и } M(x,y) = 1.$$

Таким образом языки из NP это те, для которых существует некоторый общий полиномиальный алгоритм, который может по слову и сертификату проверить правильность сертификата, а существование правильного полиномиального сертификата равносильно принадлежности слова языку.

2. *Полиномиальная сводимость и NPC*

Легко доказать, что $P \subset NP$. Действительно, пусть $L \in P$. Тогда существует M, распознающая L за полиномиальное время. Тогда рассмотрим M', которая будет получать на вход x и y через разделитель (как в определении 1), но совершенно игнорировать y и работать так же как M на x. Тогда для этой M' будет выполнено условие из определения 1, поэтому $L \in NP$.

Можно задаться вопросом, строгое ли это включение, то есть существуют ли языки из NP, которые не распознаются за полиномиальное время. Этот вопрос является открытым. Тем более удивительным становится существование некоторых языков, которые имеют очень простые описания, понятные школьникам, и при этом вопрос о равенстве P и NP сводится к вопросу о их принадлежности классу P. Для того, чтобы понять откуда берутся такие языки нужно ввести еще одно из ключевых понятий курса и вообще в теории сложности (во всяком случае везде так пишут) – понятие сводимости.

Определение 2. *L_1 сводится к L_2 по Карпу (обозначается $L_1 \leqslant_P L_2$), если существует полиномиально вычислимая функция $f : \Sigma^* \to \Sigma^*$, такая что*

$$x \in L_1 \iff f(x) \in L_2.$$

Смысл этого определения становится понятен после доказательства следующих простых утверждений.

Утверждение 1. *Если $L_1 \leqslant_P L_2$ и $L_2 \in P$, то $L_1 \in P$.*

Докажите его самостоятельно исходя из определений. Идея доказательства в том, что если мы имеем алгоритм M_2 распознающий L_2, то алгоритм M_1, который на вход получает x, вычисляет $f(x)$ (f – сводящая функция из определения 2), а затем применяет M_2 к $f(x)$ будет распознавать L_1.

Утверждение 2. *Если $L_1 \leqslant_P L_2$ и $L_2 \leqslant_P L_3$, то $L_1 \leqslant_P L_3$.*

Его тоже легко доказать по определению. Сводящей L_1 к L_3 функцией будет композиция сводящих L_1 к L_2 и L_2 к L_3.

Эти два утверждения позволяют сформулировать следующую концепцию. Предположим, что существует язык $L \in NP$, к которому сводятся все языки из NP. Если $L \notin P$, то $P \neq NP$. Если же $L \in P$, то по утверждению 1 весь класс NP вложен в P, то есть эти классы совпадают. Более того, если L сводится к какому-то другому языку $L' \in NP$, то по утверждению 2 принадлежность L' классу P снова равносильна совпадению классов P и NP.

Определение 3. *L называется NP-полным ($L \in NPC$), если*

a) $L \in NP$;

b) $\forall L' \in NP$ выполнено, что $L' \leqslant_P L$.

Оказывается, что существуют очень естественные NP-полные языки. Дальше мы будем действовать следующим образом. Сначала мы опишем каким образом можно получить первый NP-полный язык. После этого, мы сможем доказывать NP-полноту какого-то языка, сведя к нему один из уже известных NP-полных языков. Эту логику можно изобразить следующим образом

$$\text{Произвольный } L \in NP \leqslant_P L_1 \in NPC \leqslant_P L_2 \quad \Longrightarrow \quad L_2 \in NPC.$$

3. *Первый NP-полный язык*

Так как у нас пока нет в запасе NP-полных языков, придется доказывать для какого-то языка, что во-первых он в NP, а во-вторых, что любой язык из NP к нему сводится. Это можно сделать для нескольких языков, которые похожи друг на друга, но имеют и принципиальные различия.

$$\text{CIRCUIT-SAT} = \{C \mid C \text{ – выполнимая булева } \textbf{схема}\}.$$

Следующие два языка мы рассматриваем в алфавите $\Sigma_\varphi = \{0, 1, x, (,), \vee, \wedge, \neg\}$.

$$\text{SAT} = \{\varphi \mid \varphi \text{ – выполнимая булева } \textbf{формула}\}.$$

$$\text{3SAT} = \{\varphi \mid \varphi \text{ – выполнимая булева формула в 3-КНФ}\}.$$

Верещагин в своих лекциях доказывает NP-полноту языка CIRCUIT-SAT (4 лекция).

Гач и Ловас в своей книге доказывают NP-полноту языка SAT (при этом они под SAT понимают язык выполнимых булевых формул в КНФ). Их доказательство основывается на определении NP через недетерминированные машины Тьюринга, которое я надеюсь мы обсудим.

Наконец, Мусатов доказывает NP-полноту SAT, но у него это написано очень кратко и, возможно, не очень понятно.

После этого во всех трех источниках полученный NP-полный язык сводится к языку 3SAT. Мы приведем это сведение для языка SAT.

Утверждение 3. SAT $\leqslant_P$ 3SAT.

Доказательство. Нам нужно привести полиномиальную процедуру f построения по любому слову $\varphi \in \Sigma_\varphi^*$ слова $f(\varphi) \in \Sigma_\varphi^*$, такую что $\varphi \in \text{SAT} \Leftrightarrow f(\varphi) \in \text{3SAT}$.

Заметим, что булевые операции $\vee$ и $\wedge$ бинарные, то есть у них два аргумента. Поэтому формула имеет вид $\varphi = (\varphi_1 * \varphi_2)^\alpha$, где $*$ – какая-то из операций, возведение в степень 0 означает отрицание, а в степень 1 ничего. При этом формулы φ_1 и φ_2 имеют аналогичный вид.

Процедура будет следующей. Проходом по φ и подсчетом скобок можно отделить φ_1 и φ_2. После этого введем новые булевые переменные y, y_1 и y_2, которые будут отвечать значениям формул φ, φ_1 и φ_2. Запишем формулу $y \equiv (y_1 * y_2)^\alpha$ и приведем ее к виду 3-КНФ, например $y \equiv \neg(y_1 \wedge y_2)$ приведем к

$$(\neg y \vee \neg y_1 \vee \neg y_2) \wedge (y \vee y_1) \wedge (y \vee y_2) \tag{1}$$

Несложно понять, что формулу $y \equiv (y_1 * y_2)^\alpha$ всегда можно привести к 3-КНФ, содержащей не более 3 дизъюнктов.

После этого рекурсивно вызовем эту процедуру от φ_1 и φ_2. Дойдя до литералов, то есть когда $\varphi_j = x_i^{\alpha_j}$, мы уже не будем вводить новую переменную y_j, а будем использовать $x_i^{\alpha_j}$. Таким образом мы получим формулы вида 1, содержащие переменные y_k и x_k. После этого мы берем конъюнкцию всех этих формул и еще добавляем в эту конъюнкцию y (который отвечает исходной формуле φ).

Заметим, что получившаяся формула, во-первых имеет вид 3-КНФ, а во-вторых выполнима т. и т. т. когда выполнима φ (обязательно докажите это сами!).

Остается показать, что эта рекурсивная процедура занимает полиномиальное время. Запишем рекуррентное соотношение на количество элементарных операций

$$T(n) = T(n-l) + T(l) + O(n), \quad 1 \leqslant l \leqslant n-1, \text{ причем } l \text{ зависит от } n.$$

Легко доказать по индукции, что $T(n) = O(n^2)$. $\square$

Я привел это рассуждение очень подробно, потому что оно неплохо иллюстрирует технику сведения. Заметьте, что получившаяся формула $f(\varphi)$ не эквивалента φ (она даже содержит большее количество аргументов). Важно то, что она выполнима т. и т. т. когда выполнима φ.

4. *Следующие NP-полные языки*

Сначала выпишем некоторые классические NP-полные языки.

$$\text{BLOCKING-SET} = \{(\{A_1, \ldots, A_m\}, k) \mid \text{существует множество } B \text{ мощности не больше } k\text{, пересекающееся со всеми } A_i\}.$$

$$\text{SET-COVER} = \{(\{A_1, \ldots, A_m\}, k) \mid \exists I \subset \{1, \ldots, m\} : |I| = k \text{ и } \textstyle\bigcup_{i \in I} A_i = \bigcup_{i=1}^{m} A_i\}.$$

$$\text{CLIQUE} = \{(G, k) \mid \text{в } G \text{ есть клика размера } k\}.$$

$$\text{INDEPENDENT-SET} = \{(G, k) \mid \text{в } G \text{ есть независимое множество размера } k\}.$$

$$\text{VERTEX-COVER} = \{(G, k) \mid \text{в } G \text{ есть вершинное покрытие размера } k\}.$$

$$\text{SUBSET-SUM} = \{(a_1, \ldots, a_n, b) \mid \exists I \subset \{1, \ldots, n\} : \textstyle\sum_{i \in I} a_i = b\}.$$

$$\text{PARTITION} = \{\{a_1, \ldots, a_n\} \mid \text{числа можно разбить на две группы, с равной суммой в каждой группе}\}. \quad (2)$$

$$\text{3-COLOR} = \{G \mid G \text{ можно раскрасить в 3 цвета}\}.$$

$$\text{HAM-CYCLE} = \{G \mid \text{в } G \text{ есть гамильтонов цикл}\}.$$

$$\text{HAM-PATH} = \{G \mid \text{в } G \text{ есть гамильтонов путь}\}.$$

$$\text{MAX-CUT} = \{(G,k) \mid \text{множество вершин } G \\ \text{можно разбить на два непересекающихся подмножества,} \\ \text{так чтобы между ними было хотя бы } k \text{ ребер}\}.$$

$$\text{EXACTLY-3SAT} = \{\varphi \mid \varphi \text{ – выполнимая ровно-3-КНФ формула, т.е. такая что} \\ \text{в каждом дизъюнкте ровно три литерала, полученных из различных переменных}\}$$

$$\text{NAE-EXACTLY-3SAT} = \{\varphi \mid \varphi \text{ – ровно-3-КНФ формула, для которой существует} \\ \text{набор, такой что в каждом дизъюнкте есть истинный и ложный литералы}\}.$$

Теперь приведем несколько утверждений о сводимости из которых следует NP-полнота всех приведенных языков. Для некоторых утверждений я дам ссылку на доказательство. Остальные пойдут в домашку, к некоторым, я дам подсказку.

Утверждение 4. 3SAT $\leqslant_P$ BLOCKING-SET.

Доказательство. Смотреть стр. 110 в Гаче и Ловасе. □

Утверждение 5. BLOCKING-SET *и* SET-COVER *сводятся друг к другу.*

Подсказка: Сведение простое, но немного взрывает мозг. Если придумаете, то личный респект от меня.

Утверждение 6. 3SAT $\leqslant_P$ CLIQUE.

Доказательство. Смотреть 4 лекцию Верещагина. □

Утверждение 7. CLIQUE*,* INDEPENDENT-SET*,* VERTEX-COVER *сводятся друг к другу.*

Подсказка: Доказательство основано на двух наблюдениях. Первое: клика в графе – это независимое множество в его дополнении. Второе: вершины образуют независимое множество т. и т. т. когда оставшиеся образуют вершинное покрытие.

Утверждение 8. VERTEX-COVER $\leqslant_P$ SUBSET-SUM.

Доказательство. Смотреть 4 лекцию Верещагина. □

Утверждение 9. SUBSET-SUM *и* PARTITION *сводятся друг к другу.*

Подсказка: Чтобы свести SUBSET-SUM к PARTITION добавьте к числам $a_1, \ldots, a_n$ число $\sum_i a_i - 2b$.

Утверждение 10. 3SAT $\leqslant_P$ 3-COLOR.

Доказательство. Смотреть 5 лекцию Верещагина. □

Утверждение 11. VERTEX-COVER $\leqslant_P$ HAM-CYCLE.

Доказательство. Смотреть 5 лекцию Верещагина. □

Утверждение 12. HAM-PATH *и* HAM-CYCLE *сводятся друг к другу.*

Утверждение 13. 3SAT $\leqslant_P$ EXACTLY-3SAT.

Утверждение 14. EXACTLY-3SAT *и* NAE-EXACTLY-3SAT *сводятся друг к другу.*

Доказательство. Потом напишу. □

Утверждение 15. NAE-EXACTLY-3SAT $\leqslant_P$ MAX-CUT.

Подсказка: Для каждой переменной заводим вершины x_i и $\neg x_i$. Соединяем ребрами все пары x_i и $\neg x_i$, а так же для каждого дизъюнкта, проводим три ребра, между литералами в нем (в графе могут образоваться кратные ребра). Если переменных n, а дизъюнктов m, то всего вершин $2n$, а ребер $n + 3m$. Утверждается, что существует разрез не меньше $n + 2m$ тогда и только тогда, когда исходная формула принадлежит NAE-EXACTLY-3SAT.

Еще есть менее известный, но по-моему прикольный язык

$$\text{STEINER-TREE} = \{(\text{взвешенный } G = (V, E), S \subset V, w) \mid \text{в } G \text{ есть дерево, суммарного веса не более чем } w \text{ и содержащее все вершины } S\}.$$

Утверждение 16. SET-COVER $\leqslant_P$ STEINER-TREE.

Подсказка: вершины графа – элементы и множества, и еще одна вспомогательная вершина. Соединяем ребрами веса 0 соединяем элементы и множества, в которых они лежат, а так же ребрами веса 1 вспомогательную вершину и множества.

Пример. Сведем CLIQUE к языку $L = \{G \mid \text{в } G \text{ есть клика размера } \geqslant \frac{|V|}{2}\}$.

Рассмотрим следующую полиномиально вычислимую функцию: на входе, который не является парой (G, k) она возвращает пустой граф на 3 вершинах (любое слово не принадлежащее L). На входе который является парой граф G и число k, она сравнивает k с $\frac{|V|}{2}$. Если $k \geqslant \frac{|V|}{2}$, то добавляет $2k - |V|$ вершин, не соединенных ни с чем. Если $k < \frac{|V|}{2}$, то добавляет $|V| - 2k$ вершин, соединенных со всеми и между собой. Обозначим полученный граф G'.

Докажем, что в G есть клика на k вершинах т. и т. т. когда в G' есть клика на половине вершин.

Пусть в G есть клика на k вершинах. Если $k \geqslant \frac{|V|}{2}$, то в полученном графе будет клика на k вершинах, а всего вершин будет $2k$, что и требовалось. Если $k < \frac{|V|}{2}$, то в полученном графе будет клика на $|V| - 2k + k = |V| - k$ вершинах, а всего вершин будет $2|V| - 2k$, что и требовалось.

Пусть в G' есть клика на половине вершин. Разберем два случая как мог быть получен G. Если первым способом, то в нем $2k$ вершин и значит есть клика на k вершинах. Значит в исходном была клика на k вершинах (так как в первом случае мы добавили пустой граф). Если же он был получен вторым способом, то в нем $2|V| - 2k$ вершин, значит есть клика на $|V| - k$ вершинах. Значит в исходном была клика на k вершинах (так как мы добавили $|V| - 2k$ вершин, соединненых со всеми).