Классы $\mathcal{P}$, $\mathcal{NP}$ и co-$\mathcal{NP}$. Полиномиальная сводимость. Семинар 3.

Подготовил: Горбунов Э.

Ключевые слова: временная и ёмкостная сложности, полиномиальный алгоритм, классы $\mathcal{P}$, $\mathcal{NP}$, co-$\mathcal{NP}$, полиномиальная сводимость, классы $\mathcal{NP}$-hard и $\mathcal{NP}$-complete, теорема Кука-Левина

Литература: [Кормен 1, Глава 36], [Кормен 3, Глава 34], [ДПВ, Глава 8], http://www.ict.edu.ru/ft/004803/Comp.pdf

Неформальное введение

Мы подошли к большому разделу нашего курса, который посвящён сложностным классам алгоритмов. Начнём мы этот раздел с так называемых *полиномиальных алгоритмов*. Оказывается, что почти все алгоритмы, которые мы рассматривали до этого, имеют *полиномиальную сложность по длине входа*[1]. Например, рассмотрим алгоритм, который на вход получает число n и печатает букву a на экран n раз. Полиномиален ли такой алгоритм? И “да”, и “нет” — ответ зависит от способа задания числа n. Если число n задаётся своей двоичной записью, то такой простой алгоритм не полиномиален, т. к. работает экспоненциальное от длины входа время: $O(2^{\log_2 n})$. С другой стороны, если задавать число n в унарном алфавите, то есть используя только один символ, скажем 1, то вход будет просто представлять из себя n единичек, и в таком случае алгоритм работает за линейное от длины входа время. Мы видим, что перед тем, как определить, какие алгоритмы мы считаем полиномиальными, а какие — нет, нужно договориться о том, в каком формате задатся вход (это всегда нужно чётко понимать).

Пусть фиксирован алфавит Σ (если специально не оговорено, то будем считать, что $\Sigma = \{0, 1, *(\text{разделитель})\}$). Вспомним, что *предикат*— это булева функция на словах $P(\cdot) : \Sigma^* \to \{0, 1\}$, и любому предикату можно поставить в соответствие язык всех слов, на которых он истинен: $\{x \in \Sigma^* | P(x) = 1)\}$. Класс $\mathcal{P}$ состоит из всех *полиномиально вычислимых предикатов* или *языков*, которые распознаются *полиномиальными алгоритмами*. Иными словами, любой предикат $P(\cdot) \in \mathcal{P}$ вычисляется на произвольном входе x за время $poly(|x|)$, где $|x|$— длина слова x или длина кодировки входа x. А любому полиномиальному алгоритму T — вычислимой функции, перерабатывающей *слова-входы* x_i в слова-выходы *ответы* y_i, $i = 1, 2, \dots$ можно сопоставить ее *график* полиномиальный предикат: $L_T = \{x_1 * y_1, x_2 * y_2, \dots\} \in \mathcal{P}$.

Отметим, что в данной концепции мы рассматриваем только задачи *разрешения*, то есть такие задачи, на которые возможно только два ответа: “да” или “нет” (например, результат можно интерпретировать следующим образом: 1 = “да” и 0 = “нет”). Более того, некоторые задачи, не являющиеся задачами разрешения, можно рассматривать в рамках данного подхода. Для этого нужно преобразовать задачу к задаче разрешения. Например, есть целый класс задач, называемых *оптимизационными задачами*. В них как правило нужно найти максимальное или минимальное значение какой-либо величины. Такую задачу можно преобразовать к задаче разрешения следующим способом: проверять, что какое-либо число является нижней или верхней гранью величины, которую мы хотим оптимизировать.

Остаётся резонный вопрос: а почему используются именно полиномы? Ответ: прежде всего, поскольку они замкнуты относительно суперпозиции, поэтому, если программа, выполняющаяся за полиномиальное по входу время, будет фиксированное число раз вызывать любые подпрограммы, также выполняющиеся за полиномиальное время, то и результирующая программа также будет выполняться за полиномиальное время.

Задача №1. Показать, что класс $\mathcal{P}$ замкнут относительно операции пересечения, дополнения, объединения, конкатенации и замыкания Клини (последнее вас ждёт в домашнем задании).

[1] В этом месте стоит быть осторожным. Как мы увидим далее, один и тот же алгоритм может быть и полиномиальным, и не полиномиальным по длине входа при разных описаниях входа. Поэтому всегда стоит оговаривать, как именно задаётся вход, прежде чем делать выводы о его полиномиальности по длине входа.

Задача №2. Дан язык $L = \{\omega \in \{1\}^* \mid |\omega| \ -$ простое число$\}$. Доказать[2], что $L \in \mathcal{P}$.

Решение. По сути в данной задаче нам нужно проверить, что число $n = |\omega|$ является простым. В данной постановке нам подойдёт простой алгоритм перебора всех чисел от 2 до $\lfloor\sqrt{n}\rfloor$ и проверки того, что n не делится ни на одно из этих чисел. Если оно делится хотя бы на одно из этих чисел, то число составное. Иначе — простое. Время работы такого алгоритма: $O(\sqrt{n})$. Отметим, что если число n задаётся своей двоичной записью, то алгоритм перестаёт быть полиномиальным, так как время его работы $O(2^{\frac{\log_2 n}{2}})$.

Задача №3. Дан язык $L = \{\omega \in \{0,1\}^* \mid \omega = \omega^R\}$, то есть L — язык палиндромов. Доказать, что $L \in \mathcal{P}$.

Решение. Основная идея: сравнивать символы с двух концов слова. Число сравнений равно $\Theta(n)$, значит, язык принадлежит $\mathcal{P}$.

Задача №4. Число вида $\frac{n(n+1)}{2}$, где $n \in \mathbb{N}$, называют *треугольным числом.* Покажите, что язык двоичных записей треугольных чисел принадлежит классу $\mathcal{P}$.

Решение. Пусть на вход подаётся двоичная запись числа n (её длина $\log n$). Проверим, что оно треугольное. Во-первых, $\forall n > 1$ $\frac{n(n+1)}{2} \geqslant n + 1 > n$ (для $n = 1$ неравенство $\frac{n(n+1)}{2} \geqslant n$ тоже выполнено). Следовательно, можно бинарным поиском искать k такое, что $\frac{k(k+1)}{2} = n$. Для этого определим границы: $l = 1, r = n + 1$. Далее за $O(\log(n + 1))$ найдём такое k, что $\frac{k(k+1)}{2} = n$, или докажем, что такого k не существует.

Задача №5. Оцените сложность алгоритма сортировки пузырьком, как количество операций соответствующей машины Тьюринга (на вход ей подаются двоичные записи чисел массива, на выход она должна вернуть их же, но уже в отсортированном порядке). Ясно, что можно придумать разные МТ, которые будут реализовывать пузырёк, постарайтесь выбрать наиболее эффективную реализацию.

Решение. Пусть МТ на вход получает n чисел длины m, которые записаны в двоичной записи и разделены символом #. Для оценки работы алгоритма сортировки пузырьком на МТ для начала оценим сравнение двух соседних чисел и перестановку местами двух соседних чисел. Для сравнения двух соседних чисел нужно сравнивать биты слева-направо. Это можно сделать за $O(m^2)$ (для сравнения одного бита нужно $O(m)$ тактов). Чтобы поменять два соседних числа местами, нужно тоже сделать $O(m^2)$ тактов.

Теперь рассмотрим следующий алгоритм для МТ:

- Будем использовать разделитель & для отделения отсортированной части массива от неотсортированной. Поэтому & сначала поставим перед первым числом и после каждого прохода по массиву нужно будет передвигать & на m знаков вперёд, то есть за $O(m)$.
- Далее за $O(nm)$ сдвинем головку МТ на последнее число. Затем будем сравнивать два соседних числа и менять их местами, если они стоят не в том порядке. Это делается за $O(m^2)$. Таких действий придётся совершить $n - 1$.
- Затем передвинем & вперёд на одно число за $O(m)$, после чего вернёмся в конец массива за $O(nm)$. После чего повторим проход.

В худшем случае (когда массив отсортирован в обратном порядке) таких проходов будет $(n - 1)$. В i-м проходе будет сделано $n - i$ сравнений и перестановок. Отсюда следует, что сложность алгоритма $O\left(((n-1) + (n-2) + \ldots + 1)\, O(m^2)\right) = O\left(\frac{n(n-1)}{2} \cdot m^2\right) = O(n^2 m^2)$.

Напоминание

В этом разделе мы кратко обсудим базовые понятия, которые должны быть вам известны из курса ТФС и дадим другое определение класса $\mathcal{P}$, эквивалентное тому, что было в предыдущем разделе.

[2] В 2002-м году был придуман алгоритм, который проверяет, что число простое за $O(poly(\log n))$. Это был первый *детерминированный* алгоритм, который за полиномиальное по входу время проверяет, что число простое, если вход задаётя двочиной записью числа. Подробности: https://en.wikipedia.org/wiki/AKS_primality_test.

Определение. *Машина Тьюринга (МТ)* — это четвёрка $M = (Q, \Sigma, S, \Pi)$, где Σ — «ленточный» алфавит (содержит специально выделенный символ: $*$ — разделитель), Q — *конечное* множество состояний (среди них есть стартовое и финальные состояния), $S = \{-1, 0, +1\}$ — алфавит сдвигов и Π — программа, представляющая собой отображение $\Pi : Q \times \Sigma \to Q \times \Sigma \times S$.
Определение. Пусть $k \in \mathbb{N}$. Тогда *k-ленточная машина Тьюринга* — это пятёрка $M = (k, \Sigma, Q, S, \Pi)$, где $\Pi : Q \times \Sigma^k \to Q \times (\Sigma \times S)^k$.
С принципами работы одноленточных и многоленточных машин Тьюринга вы должны быть уже знакомы. Отмечу лишь, что шаг многоленточной машины Тьюринга определяется символами в текущих ячейках на всех лентах.

Определение. *Временно́й сложностью* машины Тьюринга M на входе x мы будем называть величину $T_M(x)$ — количество шагов, сделанных машиной M при обработке входа x. Временная сложность *в худшем случае* определяется следующим выражением: $T_M(n) = \max\{T_M(x) \mid |x| = n$ и машина M останавливается на входе $x\}$.
Определение. *Ёмкостной сложностью* машины Тьюринга M на входе x мы будем называть величину $S_M(x)$ — максимум по всем лентам количества ячеек, на которых побывала головка машины M при обработке входа x. Ёмкостная сложность *в худшем случае* определяется следующим выражением: $S_M(n) = \max\{S_M(x) \mid |x| = n$ и машина M останавливается на входе $x\}$.
Если на входе x машина Тьюринга не останавливается, то соответствующие сложности не определены.

Из определений следует следующий простой факт.

Теорема 1. Для любой машины Тьюринга M и любого входа x, на котором она определена, выполняется неравенство $S_M(x) \leqslant T_M(x)$, т. е. ёмкостная сложность не превышает временную.

Более того, теперь мы можем формально зафиксировать (без доказательства) тот факт, что k-ленточные и одноленточные машины Тьюринга в некотором смылсе эквивалентны.

Теорема 2. Для любой k-ленточной машины Тьюринга M, имеющей временную сложность $T(n)$, существует одноленточная машина Тьюринга M' с временной сложностью $T'(n) = O(T^2(n))$.

Лирическое отступление. Все мы хорошо помним из курса ТФС, что язык L называется *разрешимым*, если существует машина Тьюринга M, которая для входов $x \in L$ на выходе печатает 1, а для входов $x \notin L$ на выходе печатает 0. Существуют ли **неразрешимые** языки? Конечно! Это следует хотя бы из того, что языков — несчётное число (подумайте, почему это так), а описаний машин Тюринга (алгоритмов) — счётное число. Кроме того, из курса ТФС вам должно быть известно, что язык $L = \{(M, x) \mid$ машина M останавливается на входе $x\}$, состоящий пар (описание машины Тьюринга для *универсальной* машины Тьюринга, вход), является неразрешимым.

Временная сложность

Далее в этом семинаре мы сосредоточим своё внимание на временной сложности распознавания (разрешения) языков.

Определение. $\mathcal{D}\mathrm{TIME}(f(n))$ — это класс языков, для каждого из которых существует машина Тьюринга, распознающая его и имеющая временную сложность $O(f(n))$.
Буква «D» в названии говорит о том, что рассматриваются *детерминированные* машины Тьюринга (или просто — машины Тьюринга).
Определение. $\mathcal{P} = \bigcup\limits_{k \in \mathbb{N}} \mathcal{D}\mathrm{TIME}\left(n^k\right)$ — класс языков, разрешимых за полиномиальное время.
Несложно проверить, что «новое» определение класса $\mathcal{P}$ эквивалентно определению класса $\mathcal{P}$ с предыдущего семинара.
Определение. $\mathrm{EXPTIME} = \bigcup\limits_{k \in \mathbb{N}} \mathcal{D}\mathrm{TIME}\left(2^{n^k}\right)$ — класс языков, разрешимых за экспоненциальное время.

Из определения $\mathcal{D}\mathrm{TIME}$ мгновенно следует, что если $f(n) > g(n)$ для любого n, то $\mathcal{D}\mathrm{TIME}(g(n)) \subseteq \mathcal{D}\mathrm{TIME}(f(n))$, т. к. если задачу можно решить за время $O(f(n))$, то её можно решить и за время $O(g(n))$. На этой почве возникает интересный вопрос: есть ли такие задачи, которые можно решить за время $O(g(n))$, но нельзя решить за время $O(f(n))$? Интуиция подсказывает, что в «общем случае» так и должно быть. Но на интуицию всегда полагаться нельзя (тем более в математике). Например, если $g(n) = 2f(n)$, то классы в точности совпадают. Это следует из того, что мы использовали в определении выражение $O(\cdot)$.

Оказывается, что эти классы были бы равны даже в том случае, если бы мы в определении класса $\mathcal{D}$TIME не использовали обозначение $O(\cdot)$, а определили бы $\mathcal{D}\mathrm{TIME}(f(n))$ как класс языков, разрешимых за время не превосходящее $f(n)$. Это объясняется *теоремой о линейном ускоренииi*, которую я привожу без доказательства.

Теорема 3. (О линейном ускорении). Для любой машины Тьюринга M с временной сложностью $T(n)$ и **любой константы** $c > 0$ существует эквивалентная машина Тьюринга M' с временной сложностью $T'(n) = cT(n) + n$.

Поэтому, чтобы $\mathcal{D}\mathrm{TIME}(g(n))$ оказался шире класса $\mathcal{D}\mathrm{TIME}(f(n))$, функция g должна расти *значительно быстрее* функции f. Перед тем, как сформулировать теорему, которая прольёт свет на данную проблему, сформулируем ещё одно определение.

Определение. Функция f называется *конструируемой по времени*, если существует машина Тьюринга M, которая для данного входного слова длины n останавливается ровно через $f(n)$ шагов.

Например, функции $n, n^k, 2^n, 2^{n^k}$ являются конструируемыми по времени.

Теорема 4. (Об иерархии). Пусть f и g — две вычислимые конструируемые по времени функции, и[3] $f(n) = \omega(g(n)\log g(n))$. Тогда класс $\mathcal{D}\mathrm{TIME}(g(n))$ **строго** вложен в класс $\mathcal{D}\mathrm{TIME}(f(n))$.

Отмечу два важных следствия теоремы об иерархии.

Следствие 1. $\forall k \in \mathbb{N} \hookrightarrow \mathcal{D}\mathrm{TIME}(n^k) \subsetneq \mathcal{D}\mathrm{TIME}(n^{k+1})$.

Следствие 2. $\mathcal{P} \subsetneq \mathrm{EXPTIME}$.

Класс $\mathcal{NP}$

Вот мы и подошли к одной из самых важных тем всего курса — к классу $\mathcal{NP}$ (non-deterministic polynomial). Начнём с исторического определения класса $\mathcal{NP}$ (которое и объясняет такое название класса).

Начнём мы с понятия *недетерминированной машины Тьюринга*, или *недетерминированного алгоритма*. Недетерминированные алгоритмы являются, в некотором смысле, обобщением детерминированных: на некоторых (возможно — на всех) шагах у недетерминированного алгоритма имеется выбор действия из несколько вариантов. Причем этот выбор не зависит ни от каких внутренних или внешних факторов (ничем не детерминирован). Дадим теперь формальное определение недетерминированной машины Тьюринга.

Определение. *k-ленточной недетерминированной машиной Тьюринга* называется четвёрка $M = (Q, \Sigma, S, \Pi)$. Значения всех компонент четвёрки — такие же, как и в случае k-ленточной детерминированной МТ, за исключением того, что программа Π представляет собой не отображение, а *отношение*[4], заданное на множестве $(Q \times A^k) \times (Q \times (A \times S)^k)$.

Заметим, что при обработке любого входного слова x недетерминированная машина M может пройти разные последовательности конфигураций (за счет того, что на некоторых шагах выбор следующей конфигурации недетерминирован). Считают, что НМТ M допускает входное слово x, если хотя бы одна такая последовательность конфигураций приводит к допускающему состоянию (такая последовательность называется *допускающей*). В противном случае, т. е. если *ни одна* последовательность конфигураций не приводит к допускающему состоянию, машина M отвергает слово x. Таким образом, в отличие от детерминированных машин, ситуации допуска или отвержения входного слова не симметричны[5]. Определение языка, распознаваемого НМТ, полностью аналогично соответствующему определению для ДМТ.

Определение. Говорят, что НМТ M имеет *временную сложность* $T(n)$, если для всякого допускаемого входного слова длины n найдется последовательность, состоящая не более чем из $T(n)$ шагов, приводящая в допускающее состояние.

[3] Вспомните первый семинар, где было введено обозначение $\omega(\cdot)$.

[4] Иными словами, $\Pi \subseteq \{(x, y) \mid x \in Q \times A^k,\ y \in Q \times (A \times S)^k\}$.

[5] Зафиксируем это важное свойство. Мы о нём ещё вспомним, когда определим язык co-$\mathcal{NP}$

Определение. Говорят, что НМТ M имеет *ёмкостную сложность* $S(n)$, если для всякого допускаемого входного слова длины n найдется последовательность, приводящая в допускающее состояние, в которой число просмотренных ячеек на каждой ленте не превышает $S(n)$.

Для НМТ справедли аналог теоремы 1.

Теорема 5. Для любой k-ленточной недетерминированной машины M, имеющей временную сложность $T(n)$, существует одноленточная НМТ M', моделирующая M с временной сложностью $T'(n) = O(T^2(n))$.

Определение. $\mathcal{N}\mathrm{TIME}(f(n))$ — это класс языков, для каждого из которых существует недетерминированная одноленточная МТ, разрешающая этот язык с временной сложностью $O(f(n))$.
Определение. $\mathcal{NP} = \bigcup\limits_{k\in\mathbb{N}} \mathcal{N}\mathrm{TIME}(n^k)$ — класс языков, разрешимых недетерминированными алгоритмами за полиномиальное время (отсюда и название — non-deterministic polynomial).
Определение. $\mathcal{N}\mathrm{EXPTIME} = \bigcup\limits_{k\in\mathbb{N}} \mathcal{N}\mathrm{TIME}\left(2^{n^k}\right)$ — класс языков, разрешимых недетерминированными алгоритмами за экспоненциальное время.

Так как ДМТ — это частный случай НМТ, то $\mathcal{P} \subseteq \mathcal{NP}$. Однако сказать что-то ещё про это вложение (строгое ли это вложение или вообще выполнено равенство) никто до сих пор не может. Задача о равенстве классов $\mathcal{P}$ и $\mathcal{NP}$ является одной из задач тысячелетия и до сих пор никто её не решил.

Приведём *эквивалентное* определение класса $\mathcal{NP}$, которое придумали исторически позднее, но которое оказалось весьма удобным.

Определение. Будем говорить, что $L \in \mathcal{NP}$ если существует полиномиально вычислимая функция двух аргументов $R(\cdot,\cdot)$, такая что

$$L = \{x \in \{0,1\}^* \mid \exists y:\ |y| \leqslant \mathrm{poly}(|x|) \text{ и } R(x,y) = 1\}.$$

Слово y часто называют *сертификатом* (доказательством, подсказкой и т. д.). Неформально говоря, язык L лежит в $\mathcal{NP}$, если для любого слова x из L можно быстро (полиномиально) проверить доказательтво того, что слово действительно принадлежит L. При этом подразумевается, что доказательство полиномиально ограничено по отношению ко входу x.

Приводим следующий факт без доказательства.

Теорема 6. $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathrm{EXPTIME}$.

Отсюда следует, например, что **все языки класса $\mathcal{NP}$ являются разрешимыми**.

Остановимся теперь немного на определении класса $\mathcal{NP}$ и приведём несколько примеров задач из этого класса.

1. **SAT.** Язык ВЫПОЛНИМОСТЬ (Satisfability) — это язык всех выполнимых булевых формул, заданных в *конъюктивной нормальной форме*[6] от переменных $x_1, x_2, \ldots, x_n, \wedge, \vee, \neg$. Сертификат — это набор значений переменных, на котором формула выполняется. Проверить это можно быстро (за полиномиальное время). Однако быстро находить такой набор никто не умеет. Подробнее про это мы ещё поговрим далее.

2. **TSP.** Даны n вершин, а также попарные расстояния между ними (считаем, что данный граф с весами на рёбрах задаётся матрицей весов). Кроме того, есть некоторый бюджет b. Задача коммивояжёра (traveling salesman problem, TSP) состоит в отыскании (или в доказательстве того, что такого нет) такого маршрута, проходящего через все вершины ровно по одному разу, так чтобы сумма весов рёбер маршрута не превосходила b. Соответствующий язык — язык описаний пар (взвешенный граф, бюджет), таких что существует требуемый маршрут. Сертификатом служит сам маршрут. Проверка сертификата требует полиномиальное от входа время.

[6]Это *конъюнкция* (логическое «и» одного или нескольких *дизъюнктов*, каждый из которых представляет из себя *дизъюнкцию* (логическое «или») одного или нескольких *литералов*, где под литералом мы будем понимать либо булеву переменную, либо её отрицания ($x, \bar{x}$ — литералы)

3. **HAM-CYCLE.** Язык гамильтоновых графов[7] HAM-CYCLE принадлежит классу[8] $\mathcal{NP}$ (графы задаются матрицами смежности). Сертификатом является гамильтонов цикл.
4. Язык двоичных записей составных чисел тоже является языком из класса $\mathcal{NP}$. Сертификат — два делителя, которые в произведении равны заданному числу.
5. Оказывается, что язык двоичных записей простых чисел тоже лежит в классе $\mathcal{NP}$. В домашнем задании вы ещё встретите этот язык. Более того, на предыдущем семинаре упоминался полиномиальный алгоритм проверки простоты числа. Однако, если вы будете где-то использовать этот факт в нашем курсе, то нужно приводить доказательство, ибо этот полиномиальный алгоритм выходит за рамки нашего курса.

Класс co-$\mathcal{NP}$

Определение. co-$\mathcal{NP} = \{L \subseteq \Sigma^* \mid \bar{L} \in \mathcal{NP}\}$ — класс языков, дополнения которых принадлежат $\mathcal{NP}$.

Иными словами, язык L лежит в co-$\mathcal{NP}$, если для любого слова, не принадлежащего L, можно быстро проверить доказательство того, что слово не принадлежит языку. Заметим, что $\mathcal{P} \subseteq$ co-$\mathcal{NP}$, так как язык $\mathcal{P}$ замкнут относительно операции дополнения. Отсюда получаем, что $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$. Однако никто не знает, верно ли, что $\mathcal{P} \subsetneq \mathcal{NP} \cap$ co-$\mathcal{NP}$ или что $\mathcal{NP} \neq$ co-$\mathcal{NP}$.

Приведём несколько примеров языков из co-$\mathcal{NP}$.

1. **TAUTOLOGY** — язык, состоящий из описаний формул, являющихся *тавтологиями*[9]. Сертификат — набор, на котором формула равна нулю.
2. Язык двочиных записей простых чисел лежит co-$\mathcal{NP}$. Сертификат — два делителя, которые в произведении дают заднное число.

Полиномиальная сводимость. $\mathcal{NP}$-complete

Оказывается, что в классе $\mathcal{NP}$ есть «наиболее сложные» задачи, т. е. такие, к которым *сводятся* все задачи из $\mathcal{NP}$. Например, это означает, что если хоть одну из таких задач можно решать за полиномиальное время, то и любую задачу из $\mathcal{NP}$ можно решать за полиномиально время, т .е. что $\mathcal{P} = \mathcal{NP}$. Определим теперь, что мы будем понимать под сводимостью.

Определение. Говорят, что *язык A сводится полиномиально по Карпу к языку B* (и пишут $A \leq_p B$), если существует такая полиномиально вычислимая функция f, что

$$\forall x \in \Sigma^* \ (x \in A \Longleftrightarrow f(x) \in B)\,.$$

Определение. Говорят, что *язык A сводится полиномиально по Куку к языку B* (и пишут $A \leq_T B$), если существует МТ с полиномиальной временной сложностью с оракулом для языка B, которая разрешает язык A (оракул работает за 1 такт).

Далее мы будем в подавляющем большинстве случаев работать со сводимостью по Карпу.

Определение. Говорят, что $L \in \mathcal{NP}$-hard, если $\forall A \in \mathcal{NP} \hookrightarrow A \leq_p L$.

Определение. Говорят, что $L \in \mathcal{NP}$-complete, если $L \in \mathcal{NP}$-hard $\cap\, \mathcal{NP}$.

Следующая теорема показыввает, что существуют $\mathcal{NP}$-полные задачи.

Теорема 7. (Теорема Кука-Левина). Язык SAT$\in \mathcal{NP}$-complete.

Более подробно про сводимости и $\mathcal{NP}$-полные задачи мы поговорим на следующей неделе.

[7] Граф называется гамильтоновым, если в нём существует цикл, который проходит через каждую вершину ровно по одному разу.

[8] Рассмотрим похожий на первый взгляд язык EULER-CYCLE — язык описаний эйлеровых графов, то есть таких графов, в которых существует эйлеров цикл — цикл проходящий через все рёбра ровно по одному разу. Эйлер доказал следующий критерий эйлеровости графа: *граф G имеет эйлеров цикл тогда и только тогда, когда все вершины графа G имеют чётную степень.* Нетрудно построить полиномиальный алгоритм отыскания эйлерового цикла. Однако для похожей задачи HAM-CYCLE полиномиального алгоритма никто не знает.

[9] Тавтология — такая булева формула, которая равна единице при любых значениях булевых переменных.