Условные обозначения

Далее в тексте будем использовать следующие термины и обозначения

- алфавитом Σ будет называться произвольное непустое множество;
- строкой w над алфавитом Σ будет называться конечный упорядоченный набор элементов алфавита:
$$w = x_1 x_2 \dots x_n,\ x_i \in \Sigma,\ i = \overline{1, n};$$
- пустая строка, т.е. строка, не содержащая ни одного символа, будет обозначаться через ε;
- через $\square$ будет обозначаться пустой символ;
- через $|w|$ будет обозначаться длина строки w, т.е. число символов, из которых состоит w;
- языком A будет называться любое множество строк;
- через $\overline{A}$ будет обозначаться дополнение языка A до множества всех возможных строк над выбранным алфавитом Σ;
- через $A \cap B$ будет обозначаться пересечение двух языков A и B;
- через $A \cup B$ будет обозначаться объединение двух языков A и B;
- через $A \times B$ будет обозначаться декартово произведение двух множеств A и B;
- через $\mathcal{P}(A)$ будет обозначаться множество всех подмножеств некоторого множества A.
- через оператор $\langle \cdot \rangle$ будет обозначаться описание некторого объекта в виде строки символов. Например, для МТ M через $\langle M \rangle$ будет обозначено её некоторая символьная запись (через «четвёрки», «пятёрки» или на конкретном языке программирования).

1 Машина Тьюринга

Рассмотрим более сложную вычислительную модель, позволяющую распознавать значительно больший класс языков, которая называется машина Тьюринга. Данная модель предполагает, что

1. входная строка записана на бесконечной рабочей ленте;
2. пишущая головка машины Тьюринга может перемещаться по ленте влево и вправо, производя чтение и запись;
3. вычисления завершаются немедленно при переходе машины Тьюринга в допускающее или отвергающее состояние.

Определение. *Машиной Тьюринга (МТ)* называется семёрка $M = (Q, \Sigma, F, \delta, q_0, q_{accept}, q_{reject})$, где

1. Q – конечное непустое множество состояний;
2. Σ – входной алфавит, не содержащий пустого символа: $\square \notin \Sigma$;

3. Γ – рабочий алфавит такой, что $\Sigma \cup \{\square\} \subset \Gamma$;

4. $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – переходная функция;

5. $q_0 \in Q$ – начальное состояние;

6. $q_{accept} \in Q$ – допускающее состояние;

7. $q_{reject} \in Q \setminus \{q_{accept}\}$ – отвергающее состояние.

Определение. *Конфигурацией МТ* $M = (Q, \Sigma, F, \delta, q_0, q_{accept}, q_{reject})$ называется совокупность текущего содержимого ленты, состояния и положения головки МТ. Для $u, v \in \Gamma^*$, $q \in Q$ через uqv обозначается конфигурации МТ, находящейся в состоянии q, у которой на ленте написано uv при этом головка расположена на первом символе v.

Начальной конфигурацией M на строке $w \in \Sigma^*$ называется конфигурация $q_0 w$, *допускающей* или *отвергающей конфигурацией M* называется любая конфигурация uqv, где $q = q_{accept}$ или $q = q_{reject}$ соответственно.

Определение. Будем говорить, что для МТ $M = (Q, \Sigma, F, \delta, q_0, q_{accept}, q_{reject})$, $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, $q_i, q_j \in Q$

1. конфигурация uaq_ibv *порождает* конфигурацию uq_jacv, если $\delta(q_i, b) = (q_j, c, L)$;

2. конфигурация uaq_ibv *порождает* конфигурацию $uacq_jv$, если $\delta(q_i, b) = (q_j, c, R)$.

Определение. Говорят, что МТ $M = (Q, \Sigma, F, \delta, q_0, q_{accept}, q_{reject})$ допускает (отвергает) строку $w \in \Sigma^*$, если существует набор конфигураций $C_1, \ldots, C_k$ такой, что

1. C_1 – начальная конфигурация M на w;

2. C_i порождает C_{i+1}, $i = \overline{1, k-1}$;

3. C_k – допускающая (отвергающая) конфигурация.

Определение. Множество всех строк, которые допускает M обозначается $L(M)$ и называется *языком, распознаваемым M*.

Язык называется *распознаваемым по Тьюрингу* или просто *распознаваемым*, если существует МТ, которая распознаёт его.

Замечание. Согласно определению и принципам проведения вычислений, в отличие от конечных автоматов и МП-автоматов, машина Тьюринга может не завершить вычисления на входной строке за конечное число тактов (переходов между последовательными конфигурациями). Возможна ситуация, при которой машина Тьюринга зацикливается, т.е. некоторая конфигурация uqv последовательно порождает саму себя. Т.е. из того, что МТ M не допускает некоторую строку w, не следует, что M её отвергнет. Данный факт делает существенным выделение более узкого класса языков, чем распознаваемые по Тьюрингу.

Определение. МТ, которая либо допускает, либо отвергает любую входную строку (но не зацикливается), называется *решателем*.

Язык называется *разрешимым по Тьюрингу* или просто *разрешимым*, если существует решатель, который распознаёт его.

Замечание. Как правило, машины Тьюринга описывают в виде «четвёрок» или «пятёрок» – специального набора команд, которые определяют переходную функцию МТ. Здесь для краткости записи будет использоваться более высокоуровневое описание машин Тьюринга посредством словесного описания действий, которые она должна совершить с входной строкой.

Пример 1. Построим МТ, которая распознаёт язык

$$A = \{w\#w \colon w \in \{0,1\}^*\}.$$

$M_1 =$ *на входе* w :

1. *Проверить, удовлетворяет ли* w *формату* $\{0,1\}^*\#\{0,1\}^*$, *если нет, то отвергнуть.*
2. *Найти крайний левый символ, расположенный слева от* $\#$, *который является 0 или 1.*
3. *Найти крайний левый символ, расположенный справа от* $\#$, *который является 0 или 1.*
4. *Если символы, найденные на шагах 1 и 2 совпадают, то заменить их на символ* x *и перейти к шагу 1.*
5. *Если данные символы не совпадают, то отвергнуть.*
6. *Если данные символы не были найдены и в строке все символы заменены на* x, *то допустить.*
7. *Иначе отвергнуть.*

Пример 2. Построим МТ, которая решает язык

$$B = \{0^{2^n} : n \geqslant 0\}.$$

$M_2 =$ *на входе* w :

1. *Проверить, имеет ли* w *формат* 0^*, *если нет, то отвергнуть.*
2. *Если остался единственный 0, то допустить.*
3. *Удалить каждый второй 0.*
4. *Если 0 было нечётное число, то отвергнуть, иначе перейти к шагу 2.*

Пример 3. Построим МТ, которая решает язык

$$C = \{a^i b^j c^k \colon i \cdot j = k,\ i,j,k \geqslant 1\}.$$

$M_3 =$ *на входе* w :

1. *Проверить, имеет ли* w *формат* $a^+b^+c^+$, *если нет, то отвергнуть.*
2. *Вернуться в начало строки.*
3. *Удалить один символ* a *и столько же символов* c, *сколько на ленте записано символов* b.
4. *Если символов* c *не хватило, то отвергнуть.*
5. *Повторить шаги 3 и 4, пока символы* a *или* c *на ленте не закончатся.*
6. *Если они закончились одновременно, то допустить, иначе отвергнуть.*

С точки зрения класса распознаваемых языков, существуют различные разновидности вычислительных моделей, эквивалентных машине Тьюринга. Рассмотрим некоторые из них.

Многоленточная МТ. Предполагается, что у данной МТ несколько рабочих лент и пишущих головок. При этом вычисления производятся одновременно на всех лентах. Основное отличие от обычной МТ в смысле описания заключается в определении переходной функции:

$$\delta \colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{L,R\}^k,$$

где k – число лент МТ.

Теорема 1. *Для каждой многоленточной МТ существует эквивалентная ей одноленточная МТ.*

Доказательство. Пусть M – некоторая k-ленточная МТ. Построим одноленточную МТ M', симулирующую M. Организуем содержимое ленты M' следующим образом. Если i-й ленте M в некоторый момент времени соответствует конфигурация $u_i q a_i v_i$, где $u_i, v_i \in \Gamma^*$, $a_i \in \Gamma$, $q \in Q$, $i = \overline{1,k}$, то на ленте МТ M', симулирующей такт M будет записана строка $\#u_1\dot{a}_1v_1\# \dots \#u_k\dot{a}_kv_k\#$. Здесь постановка точки над символом a_i обозначает метку, определяющую положение головки МТ M на i-й ленте, специальный символ $\#$ обозначает разделение областей ленты M', отводящихся для симуляции различных лент M. Далее опишем работу M'.

$M' =$ *на входе* $w = w_1w_2 \dots w_n$:

1. *Представить входные данные в виде* $\#\dot{w}_1w_2 \dots w_n\#\dot{\square}\# \dots \#\dot{\square}\#$.
2. *По очереди производить вычисления на каждой из виртуальных лент, симулируя* M.
3. *Если в некоторый момент виртуальная головка МТ при движении вправо (влево) попадает на символ* $\#$*, то сдвинуть его и всё последующее содержимое ленты на одну ячейку вправо, вписав на исходное место символ* $\dot{\square}$*, и продолжить вычисления.*

По построению $L(M') = L(M)$. □

Следствие 1. *Язык распознаваем по Тьюрингу тогда и только тогда, когда некоторая многоленточная МТ распознаёт его.*

Недетерминированная МТ (НМТ). Предполагается, что данная МТ производит недетерминированные вычисления, т.е. допускает переход из текущей конфигурации в некоторое конечное множество производных конфигураций, вычисления на которых продолжаются параллельно. При этом НМТ допускает входную строку w, если хотя бы одна из ветвей вычислений завершается допускающей конфигурацией. Переходная функция НМТ имеет вид

$$\delta\colon Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

Теорема 2. *Для каждой НМТ существует эквивалентная ей детерминированная МТ.*

Доказательство. Пусть N – некоторая НМТ. Построим четырёхленточную МТ M, симулирующую N. Пусть l – максимальное число недетерминированных ветвлений, которое N может сделать на одном такте. Тогда любую ветвь вычислений за k тактов можно однозначно описать набором чисел $i_1, \dots, i_k = \overline{1,l}$. Организуем содержимое лент M следующим образом. На первой ленте M перманентно будет хранится входная строка w, на второй ленте будет симулироваться работа N на выбранной ветви параллельных вычислений, на третьей будет записано число k симулируемых последовательно тактов N, на четвёртой будет хранится описание выбранной ветви недетерминированных вычислений в виде k чисел. Далее опишем работу M.

$M =$ *на входе* $w = w_1w_2 \dots w_n$:

1. *Скопировать содержимое 1-й ленты на 2-ю ленту, записать на 3-ю ленту число 1, на 4-ю ленту описание ветви вычислений в виде 1.*
2. *Симулировать вычисление ветви указанной на 4-й ленте такое число тактов, какое указано на 3-й ленте.*
3. *Если при симуляции строка была допущена, то допустить, иначе перейти к шагу 4.*
4. *Записать на 4-й ленте лексикографически следующее описание ветви вычислений и перейти к шагу 2.*
5. *Если таковых описаний заданной длины не осталось, то увеличить число на 3-й ленте на 1, записать на 4-й ленте первое лексикографичеки описание ветви вычислений и перейти к шагу 2.*

По построению $L(N) = L(M)$. С учётом теоремы 1 это эквивалентно существованию одноленточной МТ M' такой, что $L(N) = L(M')$. □

Следствие 2. *Язык распознаваем по Тьюрингу тогда и только тогда, когда некоторая НМТ распознаёт его.*

Перечислитель. Под перечислителем понимается МТ, совмещённая с печатающим устройством. Предполагается, что в любой момент перечислитель может отправить на печать текущее содержимое своей рабочей ленты и продолжить вычисления. Под языком перечислителя E понимается множество строк, которые E печатает.

Теорема 3. *Язык распознаваем по Тьюрингу тогда и только тогда, когда существует перечислитель, перечисляющий его.*

Доказательство. Пусть перечислитель E перечисляет некоторый язык A. Построим следующую МТ:
$M =$ *на входе* w :

1. *Запустить* E.
2. *Каждый раз, когда* E *печатает строчку, сравнить её с* w.
3. *Если строка совпала с* w, *то допустить.*

Тогда $L(M) = A$, т.е. A распознаваемый.

Пусть A – распознаваемый язык, M – МТ, которая распознаёт A. Поскольку Σ – конечное множество, то Σ^* является счётным и допускает представление

$$\Sigma^* = \{s_1, s_2, s_3, \ldots\}.$$

Построим перечислитель E:
$E =$ *игнорирует вход* :

1. *Повторить шаги 2 и 3 для всех* $i = 1, 2, 3, \ldots$.
2. *По очереди симулировать* M *на каждой строке* $s_1, \ldots, s_i$ *ровно* i *тактов.*
3. *Если какая-либо строка была допущена, то напечатать его.*

Тогда E перечисляет язык A. □

Построение такой вычислительной модели, как машина Тьюринга, позволило формализовать понятие алгоритма. Суть этого вопроса можно рассмотреть на примере исследования 10-й проблемы Гильберта, которая в 1900 г. была сформулирована следующим образом: требуется построить алгоритм, который определял бы, имеет ли многочлен с целыми коэффициентами и целыми показателями степеней целочисленные корни. Проблематика заключалась в том, что изначально предполагалось, что такой алгоритм есть, и его просто необходимо найти. Однако из-за отсутствия формального определения термина «алгоритм» доказать или опровергнуть его существование было невозможно.

Определение. *Алгоритмом* называется МТ.

В таком случае 10-я проблема Гильберта сводилась к вопросу, является ли разрешимым язык

$$D = \{p\colon p \text{ полином с целыми степенями и коэффициентами,}$$

$$\text{обладающий целочисленным корнем}\}.$$

В 1970 г. было доказано, что язык D не является разрешимым по Тьюрингу, хотя он и является распознаваемым. Построим МТ, которая распознаёт D:
$M =$ *на входе* p :

1. *Проверить, является ли* p *полином с целыми степенями и коэффициентами.*
2. *Последовательно перебрать все целочисленные векторы, вычисляя на них значение* p.
3. *Если* p *обратится в 0 при очередной подстановке, то допустить.*

M не является решателем D, так как в случае полинома с целыми коэффициентами $p \notin D$ МТ M никогда не завершит свои вычисления.

2 Измерение сложности

Для изучения вопросов сложности алгоритмов рассмотрим следующий пример. Пусть $A = \{0^k 1^k \colon k \geqslant 0\}$. Очевидно, что язык A разрешим, но с практической точки зрения представляет интерес, какое количество времени потребуется машине Тьюринга, чтобы его разрешить. Рассмотрим машину Тьюринга M_1:
$M_1 =$ *на входе* w :

1. *Просмотреть посимвольно* w *и отвергнуть, если где-то 0 стоит справа от 1.*
2. *Просмотреть содержимое ленты и пометить один 0 и одну 1.*
3. *Повторить шаг 2 до тех пор, пока на ленте есть непомеченные и 0, и 1.*
4. *Если все символы помечены, то допустить. Иначе отвергнуть.*

Для того, чтобы оценить временную сложность M_1 введём определения.

Определение. Пусть M – детерминированная МТ, которая останавливается на любом входе. Тогда *временем работы* или *временной сложностью* называется функция $f \colon \mathbb{N} \to \mathbb{N}$, где $f(n)$ – максимальное число тактов машины Тьюринга M, которые она совершит на входе $w \in \Sigma^*$, где $|w| = n$.

Замечание. Истинную временную сложность, как правило, тяжело вычислить. Поэтому зачастую используется только её оценка в смысле O-символики или o-символики.

Определение. Пусть $f, g \colon \mathbb{N} \to [0; +\infty)$. Говорят, что $f(n) = O(g(n))$, если существуют $C > 0$ и $n_0 \in \mathbb{N}$ такие, что для всех $n \geqslant n_0$ верно неравенство $f(n) \leqslant Cg(n)$.

Также говорят, что $f(n) = o(g(n))$, если

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Пример 4. Пусть $f_1(n) = 5n^3 + 2n^2 + 22n + 6$. Тогда справедлива оценка $f_1(n) = O(n^3)$. Однако также справедлива оценка $f_1(n) = O(n^4)$.

Пример 5. Поскольку для любого $b > 0$, $b \neq 1$ верно равенство

$$\log_b(n) = \frac{\log_2 n}{\log_2 b},$$

то $O(\log_b(n)) = O(\log_2(n)) = O(\log n)$.

Например, $f_2(n) = 3n \log_2 n + 5n \log_2 \log_2 n + 2 = O(n \log n)$.

Пример 6. Верны следующие равенства:

$$\begin{aligned}
&1)\ \sqrt{n} = o(n),\\
&2)\ n = o(n \log\log n),\\
&3)\ n \log\log n = o(n \log n),\\
&4)\ n \log n = o(n^2),\\
&5)\ n^2 = o(n^3),\\
&6)\ f(n) \neq o(f(n)).
\end{aligned}$$

Проанализируем теперь сложность M_1. Шаг 1 занимает $O(n)$ действий. Шаги 2 и 3 повторяются не более $\frac{n}{2}$ раз и каждый раз проверяется не более n символов. Т.е. их сложность можно оценить в виде $O\left(\frac{n}{2}\right) O(n) = O(n^2)$. Шаг 4 заключается в проверке n символов и его сложность равна $O(n)$. Окончательно M_1 имеет сложность $O(n^2)$.

Определение. Пусть $t\colon \mathbb{N} \to [0; +\infty)$. Тогда

$$\mathrm{TIME}(t(n)) = \{A \subset \Sigma^* \colon \text{существует детерминированная машина Тьюринга,}$$

$$\text{которая решает } A \text{ за } O(t(n)) \text{ время}\}.$$

Пример 7. Справедливо включение $A = \{0^k 1^k \colon k \geqslant 0\} \in \mathrm{TIME}(n^2)$. Тем не менее, не исключено, что A можно решить за меньшее время.
$M_2 =$ *на входе* w :

1. *Просмотреть посимвольно w и отвергнуть, если где-то 0 стоит справа от 1.*
2. *Если на ленте написано нечетное число 0 и 1 без меток, то отвергнуть.*
3. *Пометить каждый второй 0 и каждую вторую 1.*
4. *Повторить шаги 2 и 3, если на ленте остались 0 и 1.*
5. *Если все символы помечены, то допустить. Иначе отвергнуть.*

Заметим, что $L(M_2) = A$, но при этом шаги 2–4 имеют сложность $O(n \log n)$. Т.е. $A \in O(n \log n)$.

Замечание. Для оценки сложности языка снизу можно использовать тот факт, что если некоторый язык B может быть решён за $o(n \log n)$ времени детерминированной одноленточной машиной Тьюринга, то B – регулярный язык.

Пример 8. Сложность решателя вообще говоря зависит от выбранной вычислительной модели. Построим двухленточную машину Тьюринга M_3:
$M_3 =$ *на входе* w :

1. *Просмотреть посимвольно w и отвергнуть, если где-то 0 стоит справа от 1.*
2. *Скопировать все нули с первой ленты на вторую.*
3. *Читать одновременно 1 с первой ленты и 0 со второй ленты, помечая их.*
4. *Если помечены все 1 на первой ленте и все 0 на второй ленте, то допустить.*
5. *Иначе отвергнуть.*

Верно, что $L(M_3) = A$, но при этом M_3 решает A за $O(n)$ времени.

Теорема 4. *Пусть $t(n)$ – функция, такая что $t(n) \geqslant n$. Тогда для каждой многоленточной машины Тьюринга с $t(n)$ временной сложностью существует эквивалентная одноленточная машина Тьюринга с временной сложностью $O(t^2(n))$.*

Доказательство. Для доказательства теоремы 4 используем алгоритм построения одноленточной машины Тьюринга, эквивалентной данной многоленточной машине Тьюринга, использованный в доказательстве теоремы 1.

Пусть S – k-ленточная машина Тьюринга. Тогда согласно теореме 1 существует эквивалентная ей одноленточная машина Тьюринга M, которая симулирует все ленты S. Для симуляции одного шага S машине Тьюринга M необходимо просканировать не более $kt(n)$ ячеек, т.к. за $t(n)$ шагов S может использовать не более $t(n)$ ячеек на каждой из своих k лент. Т.е. симуляция одного шага S на M занимает не более $O(kt(n)) = O(t(n))$ времени. При этом число шагов, которые необходимо симулировать не превосходит $t(n)$. Тогда симуляция займёт не более $t(n)O(t(n)) = O(t^2(n))$ времени. □

Определение. Пусть N – недетерминированная машина Тьюринга, которая решает язык A. Тогда *временем работы* или *временной сложностью* называется функция $f\colon \mathbb{N} \to \mathbb{N}$, где $f(n)$ – максимальное число тактов машины Тьюринга N в любой ветви вычислений (т.е. высота дерева разбора) на входе $w \in \Sigma^*$, где $|w| = n$.

Теорема 5. *Пусть $t(n)$ – функция такая, что $t(n) \geqslant n$. Тогда для каждой одноленточной недетерминированной машины Тьюринга с $t(n)$ временной сложностью существует эквивалентная ей одноленточная машина Тьюринга с временной сложностью $2^{O(t(n))}$.*

Доказательство. Для доказательства теоремы 5 используем алгоритм построения детерминированной машины Тьюринга, эквивалентной данной недетерминированной машине Тьюринга, использованный в доказательстве теоремы 2.

Пусть N – недетерминированная машины Тьюринга. Тогда согласно теореме 2 существует эквивалентная ей четырёхленточная машина Тьюринга S, которая симулирует поочерёдно все ветви вычислений N. Если b – максимальное число недетерминированных ветвлений в коде N (максимальное число ветвей, выходящих из вершин дерева разбора), то S симулирует не более $b^{t(n)}$ различных ветвей вычислений. При этом количество шагов при симуляции каждой ветви составляет не более $t(n)$ шагов. Тогда временная сложность S составляет $O\left(t(n)b^{t(n)}\right) = 2^{O(t(n))}$.

В силу теоремы 4 существует эквивалентная S одноленточная детерминированная машина Тьюринга M с временной сложностью

$$O\left(\left(2^{O(t(n))}\right)^2\right) = 2^{2\cdot O(t(n))} = 2^{O(t(n))}.$$

□

3 Класс P

Определение. *Классом* P называется класс всех языков разрешимых за полиномиальное время на детерминированной машине Тьюринга:

$$\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(n^k).$$

Замечание. 1) Согласно теореме 4 класс P инвариантен к выбору детерминированной вычислительной модели.

2) Класс P приблизительно соответствует множеству тех задач, которые могут быть решены при помощи ЭВМ за приемлимое время.

Через $PATH$ обозначим задачу обнаружения пути из одной вершины в другую в ориентированном графе G:

$$PATH = \{\langle G, s, t\rangle : G \text{ – ориентированный граф,}$$

$$\text{в котором есть путь из вершины } s \text{ в вершину } t\}.$$

Теорема 6. *Справедливо включение* $PATH \in \mathrm{P}$.

Доказательство. Построим машину Тьюринга M_1:
$M_1 =$ *на входе* $\langle G, s, t\rangle$, *где* G *– ориентированный граф,* s, t *– его вершины* :

1. *Пометить вершину* s.
2. *Пометить все вершины, которые можно за один шаг достичь из уже помеченных.*
3. *Повторять шаг 2, пока не перестанут появляться новые помеченные вершины.*
4. *Если вершина* t *помечена, то допустить, иначе отвергнуть.*

Шаги 1 и 4 имеют $O(n)$ временную сложность. В шаге 2 для каждой помеченной вершины надо проверить все остальные вершины на предмет того, можно ли их достичь из данной. Т.е. временная сложность шага 2 составляет $O(n^2)$. При этом он повторяется в рамках шага 3 не более $O(n)$ раз. Тогда итоговая временная сложность M_1 составляет $O(n^3)$. □

Определение. Числа $x, y \in \mathbb{N}$ называются *взаимно простыми*, если их наибольший общий делитель равен 1.

Через $RELPRIME$ обозначим задачу проверки, являются ли два натуральных числа взаимнопростыми:

$$RELPRIME = \{\langle x, y\rangle : x, y \in \mathbb{N} \text{ – взаимно простые числа}\}.$$

Теорема 7. *Справедливо включение* $RELPRIME \in \mathrm{P}$.

Доказательство. Построим вспомогательную машину Тьюринга E:
$E =$ *на входе* $\langle x, y\rangle$, *где* $x, y \in \mathbb{N}$:

1. *Положить* x *равным* $x \mod y$.
2. *Поменять значения* x *и* y.
3. *Повторять шаги 1 и 2 до тех пор, пока* $y \neq 0$.
4. *Вывести на ленту* x.

Поскольку за каждые два повтора шагов 1–3 значения чисел x и y уменьшаются вдвое, то число повторений займёт не более $\max\{2\log_2 x; 2\log_2 y\}$ итераций, что пропорционально числу символов в бинарной записи x и y. С учётом того, что операции вычисления остатка от деления и обмена значениями переменных имеют полиномиальную временную сложность, верно, что E также имеет полиномиальную временную сложность. При этом E реализует алгоритм Евклида вычисления наибольшего общего делителя двух натуральных чисел.

Построим машину Тьюринга M_2:
$M_2 =$ *на входе* $\langle x, y\rangle$, *где* $x, y \in \mathbb{N}$:

1. *Запустить* E *на* $\langle x, y\rangle$.
2. *Если на ленте 1, то допустить, иначе отвергнуть.*

Машина Тьюринга M_2 решает язык $RELPRIME$, при этом её временная сложность совпадает со сложностью E. Тогда $RELPRIME \in \mathrm{P}$. □

Определение. *Контекстно-свободной (КС) грамматикой* называется четвёрка $G = (V, \Sigma, R, S)$, где

1. V – конечное непустое множество переменных;
2. Σ – алфавит, такой что $\Sigma \cap V = \varnothing$;
3. $R \subset V \times (\Sigma \cup V)^*$ – конечное множество правил;
4. $S \in V$ – начальная переменная.

Определение. Пусть $u, v, w \in (\Sigma \cup V)^*$, $A \in V$. Говорят, что uAv *порождает* uwv в грамматике $G = (V, \Sigma, R, S)$:

$$uAv \Rightarrow uwv,$$

если $(A \to w) \in R$. Говорят, что u *из* u *выводится* v в грамматике G:

$$u \overset{*}{\Rightarrow} v,$$

если существуют $u_1, \ldots, u_k \in (\Sigma \cup V)^*$, $k \geqslant 0$, такие что

$$u \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

Определение. *Языком грамматики* $G = (V, \Sigma, R, S)$ называется множество строк, которые выводятся из начальной переменной:

$$L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}.$$

Язык A называется *контекстно-свободным (КС)*, если существует КС-грамматика G такая, что $A = L(G)$.

Определение. Говорят, что грамматика $G = (V, \Sigma, R, S)$находится в *нормальной форме Хомского*, если каждое правило имеет один из следующих двух видов:

$$A \to BC, \quad A \to a,$$

где $A, B, C \in V$, $a \in \Sigma$, $B, C \neq S$; также допустимо правило $S \to \varepsilon$.

Теорема 8. *Для любого КС-языка A существует КС-грамматика G в нормальной форме Хомского такая, что $A = L(G)$.*

Теорема 9. *Пусть A – КС-язык. Тогда $A \in$ P.*

Доказательство. Пусть G – КС-грамматика в нормальной форме Хомского, порождающая язык A, S – начальная переменная G. Построим следующую машину Тьюринга:
$M_3 =$ *на входе $w = w_1 w_2 \dots w_n$:*

1. *Если $w = \varepsilon$ и $S \to \varepsilon$ – правило G, то допустить.*
2. *Перебрать все $i = \overline{1, n}$.*
3. *Для каждой переменной A*
4. *проверить, является ли правило $A \to b$, где $b = w_i$, правилом G.*
5. *Если это так, то включить A в множество $table(i, i)$.*
6. *Перебрать все $l = \overline{2, n}$.*
7. *Перебрать все $i = \overline{1, n - l + 1}$.*
8. *Положить $j = i + l - 1$. Перебрать все $k = \overline{i, j - 1}$.*
9. *Для каждого правила $A \to BC$, если $B \in table(i, k)$, $C \in table(k + 1, j)$, то включить A в множество $table(i, j)$.*
10. *Если $S \in table(1, n)$, то допустить, иначе отвергнуть.*

Множества $table(i, j)$ состоят из тех переменных, из которых в грамматике G можно вывести подстроку $w_i w_{i+1} \dots w_j$. В шагах 2–5 рассматриваются подстроки длины 1. В шагах 6–9 рекурсивно рассматриваются подстроки большей длины. Здесь l – длина подстроки в строке w, i – номер символа, с которого начинается подстрока, j – символ конца подстроки, k – место разделения подстроки.

Сожность M_3 определяется шагами 6–9 и составляет $O(n^3)$. □

4 Класс NP

Через $HAMPATH$ обозначим задачу поиска гамильтонова пути в ориентированном графе, т.е. такого пути, который проходит через все вершины графа ровно один раз:

$$HAMPATH = \{\langle G, s, t\rangle : G \text{ – ориентированный граф,}$$

$$\text{в котором есть гамильтонов путь из вершины } s \text{ в вершину } t\}.$$

Определение временной сложности МТ не позволяет конструктивно построить нижнюю оценку сложности. Например, для языка $HAMPATH$ неизвестен алгоритм, решающий его за полиномиальное время, но данный факт не означает, что такого алгоритма не существует. Поэтому расширим класс разрешимых языков с помощью аппарата верификации.

Определение. *Верификатором* языка A называется машина Тьюринга V такая, что $w \in A$ тогда и только тогда, когда существует $c \in \Sigma^*$, для которой V допускает $\langle w, c\rangle$. Строка c называется *сертификатом* строки w.

Замечание. Предполагается, что временная сложность верификатора V является функцией только от $|w|$. Длина сертификата не учитывается.

Пример 9. Для языка $HAMPATH$ в качестве сертификата строки $\langle G, s, t\rangle$ можно рассматривать гамильтонов путь из вершины s в вершину t. Тогда верификатор V будет иметь следующий вид:
$V =$ *на входе* $\langle G, s, t, c\rangle$:

1. *Проверить, начинается ли путь c из s и заканчивается ли в t.*
2. *Проверить, состоит ли путь c из всех вершин графа G.*
3. *Для каждых двух смежных вершин в c проверить наличие связывающего их ребра G.*
4. *Если все проверки пройдены, то допустить, иначе отвергнуть.*

Можно заметить, что каждая проверка в описании V занимает не более $O(n^2)$ шагов. Т.е. $HAMPATH$ может быть проверен за полиномиальное время.

Определение. *Классом* NP называется класс всех языков, верифицируемых за полиномиальное время на детерминированной машине Тьюринга.

Замечание. Справедливы включения $\mathrm{P} \subset \mathrm{NP}$, $HAMPATH \in \mathrm{NP}$.

Пример 10. Следующая НМТ N_1 решает $HAMPATH$ за полиномиальное время.
$N_1 =$ *на входе* $\langle G, s, t\rangle$, *где G – ориентированный граф, s, t – его вершины* :

1. *Недетерминировано записать в некотором порядке $p_1, \ldots, p_n$ все вершины G.*
2. *Если в списке есть повторения, то отвергнуть.*
3. *Если $s \neq p_1$ или $t \neq p_n$, то отвергнуть.*
4. *Если для некоторого $i = \overline{1, n-1}$ пара вершин (p_i, p_{i+1}) не является ребром G, то отвергнуть, иначе допустить.*

Теорема 10. *$A \in \mathrm{NP}$ тогда и только тогда, когда существует недетерминированная машина Тьюринга, которая решает A за полиномиальное время.*

Доказательство. Пусть $A \in \mathrm{NP}$. Тогда по определению существует полиномиальный верификатор V для A. Предположим, что V работает за n^k время. Тогда следующая недетерминированная машина Тьюринга N решает A за полиномиальное время:
$N =$ *на входе* w :

1. *Недетерминировано дописать на ленту произвольную строку c длины не более n^k.*
2. *Запустить V на $\langle w, c\rangle$.*
3. *Ответить то же, что и V.*

Предположим, что существует N – недетерминированная машина Тюринга, решающая A за полиномиальное время. Построим верификатор V, который в качестве сертификата c использует историю недетерминированных вычислений N на w:
$V =$ *на входе* $\langle w, c\rangle$:

1. *Симулировать N на w, используя поочерёдно каждый символ c для выбора ветви недетерминированных вычислений.*
2. *Если полученная ветвь вычислений допускает, то допустить, иначе отвергнуть.*

□

Определение. Пусть $t\colon \mathbb{N} \to [0; +\infty)$. Тогда

$$\mathrm{NTIME}(t(n)) = \{A \subset \Sigma^* \colon \text{существует недетерминированная машина Тьюринга,}$$

$$\text{которая решает } A \text{ за } O(t(n)) \text{ время}\}.$$

Замечание. Фактически теорема 10 утверждает, что

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k).$$

Определение. *Кликой* называется подмножество вершин неоринетированного графа, в котором каждая вершина соединена ребрами со всеми остальными вершинами из данного подмножества.

Через $CLIQUE$ обозначим задачу обнаружения клики заданного размера в неоринетированном графе G:

$$CLIQUE = \{\langle G, k\rangle : G \text{ – неориентированный граф, содержащий } k\text{-клику}\}.$$

Теорема 11. *Справедливо включение* $CLIQUE \in \mathrm{NP}$.

Доказательство. Построим полиномиальный верификатор V для $CLIQUE$, где в качестве сертификата используется список вершин k-клики:
$V =$ *на входе* $\langle G, k, c\rangle$, *где* G *– неориентированный граф,* $k \in \mathbb{N}$:
1. *Проверить, является ли* c *подмножеством вершин* G.
2. *Проверить, не повторяются ли вершины в* c.
3. *Проверить, соединены ли все вершины из* c *рёбрами.*
4. *Если все проверки пройдены, то допустить, иначе отвергнуть.*

□

Через $SUBSET - SUM$ обозначим задачу представления натурального числа t в виде суммы заданного набора натуральных чисел $S = \{x_1, \ldots, x_k\}$:

$$SUBSET - SUM = \left\{ \langle S, t\rangle : S = \{x_1, \ldots, x_k\},\ \sum_{i=1}^{l} y_i = t,\ \{y_1, \ldots, y_l\} \subset S \right\}.$$

Замечание. В $SUBSET - SUM$ в множестве S допустимы повторы, т.е. различные x_i могут иметь одинаковое значение.

Теорема 12. *Справедливо включение* $SUBSET - SUM \in \mathrm{NP}$.

Доказательство. Построим полиномиальный недетерминированный решатель N для $SUBSET - SUM$:
$N =$ *на входе* $\langle S, t\rangle$, *где* S *– множество натуральных чисел,* $t \in \mathbb{N}$:
1. *Недетерминировано выбрать подмножество* $c \subset S$ *размером не более* t.
2. *Проверить, равна ли сумма чисел в* c *числу* t.
3. *Если да, то допустить, иначе отвергнуть.*

□

Замечание. На данный момент неизвестно, совпадают ли классы NP и *co* NP или классы P и NP. Тем не менее, справедливы включения $\mathrm{P} \subset \mathrm{NP}$ и $\mathrm{P} \subset \mathit{co}\,\mathrm{NP}$.

Также можно условно принять, что P – класс языков, которые можно «быстро» решить, а NP – класс языков, которые можно «быстро» проверить.

5 NP-полные языки

Изучение соотношения классов P и NP является ключевым в теории сложности. Одним из инструментов исследования принадлежности некоторого языка тому или иному классу является понятие сводимости: возможность построения алгоритма, который преобразует элементы одного языка в элементы другого языка, сложность которого известна. Также важным является выделение внутри класса сложности полного подмножества – множества тех языков, к которым можно свести любой другой из данного класса сложности.

Определение. Переменная, которая может принимать одно из двух значений $\{TRUE, FALSE\}$, называется *булевой переменной*.

Определение. *Булевой формулой* называется выражение, состоящее из булевых переменных и операций, таких как дезъюнкция, конъюнкция и отрицание.

Определение. Формула называется *выполнимой*, если существует набор значений переменных, которые обращают формулу в истину.

Через SAT обозначим задачу проверки, является ли булева формула φ выполнимой:

$$SAT = \{\langle\varphi\rangle \colon \varphi \text{ – выполнимая булева формула}\}.$$

Теорема 13 (Кука-Левина)**.** *Включение* $SAT \in \mathrm{P}$ *справедливо тогда и только тогда, когда* $\mathrm{P} = \mathrm{NP}$.

Определение. Язык A называется *сводимым к* B *за полиномиальное время*:

$$A \leqslant_P B,$$

если существует функция вычислимая за полиномиальная время $f \colon \Sigma^* \to \Sigma^*$ такая, что $w \in A$ тогда и только тогда, когда $f(w) \in B$. При этом функция f называется *полиномиальной редукцией от* A *к* B.

Теорема 14. *Пусть* $A \leqslant_P B$, $B \in \mathrm{P}$. *Тогда* $A \in \mathrm{P}$.

Доказательство. Пусть f – полиномиальная редукция от A к B. Построим решатель A:
$M =$ *на входе* w :
1. *Вычислить* $f(w)$.
2. *Проверить* $f(w) \in B$ *и ответить то же.*

Оба шага M занимают полиномиальное время, откуда следует, что $A = L(M) \in \mathrm{P}$. $\square$

Определение. Булева формула φ от переменных $x_1, \ldots x_n$ находится в *конъюнктивной нормальной форме*, если она допускает следующее представление:

$$\varphi = \varphi_1 \wedge \ldots \wedge \varphi_k,$$

$$\varphi_i = (a_{i,1} \vee \ldots \vee a_{i,l_i}),\ k, l_1, \ldots, l_k \in \mathbb{N}, i = \overline{1,k},$$

где $a_{i,j} \in \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$ называется *термом*. Булева формула называется *3КНФ-формулой*, если $l_1 = \ldots = l_k = 3$.

Пример 11. Следующая булева формула φ является 3КНФ-формулой:

$$\varphi = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6).$$

Через $3SAT$ обозначим задачу проверки, является ли булева 3КНФ-формула φ выполнимой:

$$3SAT = \{\langle\varphi\rangle \colon \varphi \text{ – выполнимая булева 3КНФ-формула}\}.$$

Теорема 15. *Справедлива оценка*

$$3SAT \leqslant_P CLIQUE.$$

Доказательство. Пусть φ – 3КНФ-формула, состоящая из k скобок:

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge \ldots \wedge (a_k \vee b_k \vee c_k).$$

Опишем алгоритм, который преобразует формулу φ в $\langle G, k\rangle$, где граф G имеет k-клику тогда и только тогда, когда φ выполнима. Граф будет состоять из $3k$ вершин, которые будут организованы в k троек $t_1, \ldots, t_k$, где каждая тройка соответствует одной из скобок φ, а каждая вершина в тройке – одному из термов в скобке. Рёбра в графе проведём исходя из следующих принципов:

1. не соединены вершины в одной тройке;
2. не соединены вершины, соответствующие взаимнопротивоположным термам: x_i и $\overline{x_i}$;
3. остальные вершины соединены.

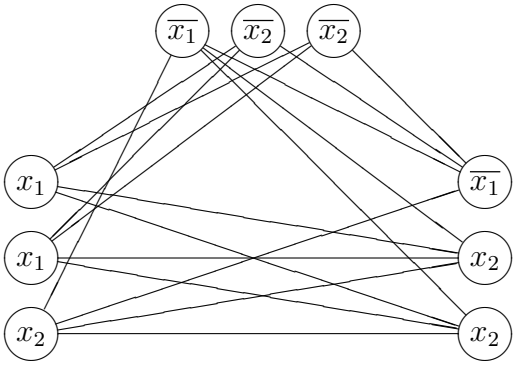
Описанный алгоритм имеет полиномиальную сложность. Покажем, что он действительно является редукцией от $3SAT$ к $CLIQUE$.

Предположим, что φ выполнима. Тогда существует такой набор значений булевых переменных, что хотя бы один терм в каждой скобке является истинным. Тогда данные термы не могут быть взаимнопротивоположными, а значит, они соединены между собой рёбрами в графе G, т.е. образуют k-клику.

Предположим, что в построенном графе G существует k-клика. Тогда по построению все вершины в k-клике принадлежат различным тройкам и ни одна из них не является взаимнопротивоположной какой-либо другой. Тогда каждую из них можно положить истинной, т.е. формула φ выполнима. □

Пример 12. Продемонстрируем алгоритм редукции из доказательства теоремы 15 на примере преобразования 3КНФ-формулы φ в граф G.

$$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$$



Вершины $x_2, \overline{x_1}, x_2$ образуют 3-клику, откуда следует выполнимость φ.

Определение. Язык B называется NP-*трудным*, если для любого $A \in$ NP справедлива оценка сложности $A \leqslant_P B$. Если также $B \in$ NP, то B называется NP-*полным*.

Теорема 16. *Пусть* $B \in$ P *и* B – NP-*полный язык. Тогда* P = NP.

Доказательство. Доказательство следует непосредственно из определения. □

Теорема 17. *Пусть* B – NP-*полный язык,* $C \in$ NP *и* $B \leqslant_P C$. *Тогда* C – NP-*полный язык.*

Доказательство. Доказательство следует непосредственно из определения. □

Теорема 18 (Кука-Левина)**.** *SAT* – NP-*полный язык.*

Доказательство. Для доказательства включения $SAT \in \mathrm{NP}$ достаточно построить недетерминированную машину Тьюринга, которая параллельно подставляет в формулу φ все возможные значения булевых переменных. Если хотя бы один такой набор обращает формулу в истину, то допустить. Иначе отвергнуть.

Теперь покажем, что любой $A \in \mathrm{NP}$ полиномиально сводим к SAT. Согласно теореме 10 существует $k \in \mathbb{N}$ такой, что некоторая недетерминированная машина Тьюринга N решает A за $n^k - 3$ шага. Таблицей для N на входной строке $w \in \Sigma^*$ назовём таблицу размера $n^k \times n^k$, строки которой являются конфигурациями одной из веток вычислений N на w. Каждая строка начинается и заканчивается вспомогательным символом $\#$. Первая строка всегда является начальной конфигурацией N на w. Таблица называется допускающей, если в ней есть строка с допускающей конфигурацией. Таким образом, если существует допускающая таблица, то N допускает w.

Теперь построим полиномиальную редукцию от A к SAT. Т.е. требуется описать алгоритм, который преобразует описание машины Тьюринга N и входную строку w в булеву формулу φ, которая выполнима в том и только в том случае, когда существует допускающая таблица для N и w. Пусть Q – множество состояний N, Γ – рабочий алфавит N. Тогда $C = Q \cup \Gamma \cup \{\#\}$. Для всех $i, j = \overline{1, n^k}$ и $s \in C$ введём переменную $x_{i,j,s}$:

$$x_{i,j,s} = \begin{cases} 1, & \text{если } (i,j)\text{-я клетка таблицы содержит } s, \\ 0, & \text{иначе.} \end{cases}$$

Представим искомую φ в следующем виде:

$$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}.$$

Здесь φ_{cell} гарантирует, что в каждой клетке таблицы записан ровно один символ:

$$\varphi_{cell} = \bigwedge_{1 \leqslant i,j \leqslant n^k} \left(\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right).$$

φ_{start} гарантирует, что первая строка таблицы является начальной конфигурацией:

$$\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \ldots \wedge x_{1,n+3,w_n} \wedge x_{1,n+4,\square} \wedge \ldots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#}.$$

φ_{aceept} гарантирует, что в таблице встречается допускающая конфигурация:

$$\varphi_{accept} = \bigvee_{1 \leqslant i,j, \leqslant n^k} x_{i,j,q_{accept}}.$$

φ_{move} гарантирует, что каждая следующая строка таблицы может быть получена из предыдущей согласно переходной функции. Для этого будет использовано «окно» – подтаблица размером 2×3. Для каждого правила переходной функции

$$(q_j, b, R) \in \delta(q_i, a)$$

и для всех символов $c_1, c_2, c_3, c_4 \in \Gamma \cup \{\#\}$ определим следующие допустимые окна:

c_1	c_2	q_i
c_1	c_2	b

c_2	q_i	a
c_2	b	q_j

q_i	a	c_3
b	q_j	c_3

a	c_3	c_4
q_j	c_3	c_4

Для каждого правила переходной функции

$$(q_j, b, L) \in \delta(q_i, a)$$

и для всех символов $c_1, c_2, c_3, c_4 \in \Gamma \cup \{\#\}$ определим следующие допустимые окна:

c_1	c_2	c_3
c_1	c_2	q_j

c_2	c_3	q_i
c_2	q_j	c_3

c_3	q_i	a
q_j	c_3	b

q_i	a	c_4
c_3	b	c_4

для всех символов $c_1, c_2, c_3 \in \Gamma \cup \{\#\}$ определим следующее допустимое окно:

c_1	c_2	c_3
c_1	c_2	c_3

Все прочие окна будем считать недопустимыми.

Тогда определим φ_{move} следующим образом:

$$\varphi_{move} = \bigwedge_{\substack{1 \leqslant i \leqslant n^k - 1 \\ 2 \leqslant j \leqslant n^k - 1}} \varphi_{i,j,window},$$

$$\varphi_{i,j,window} = x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6},$$

где $a_1, a_2, a_3, a_4, a_5, a_6 \in C$ соответствуют допустимым окнам, описанным выше.

Оценим сложность редукции. Время построения прямопропорциональна количеству переменных, из которых состоит формула φ. Всего существует ln^{2k} различных переменных, где l – число элементов в C. Размер φ_{cell} составляет $O(n^{2k})$, размер φ_{start} составляет $O(n^k)$, размер φ_{accept} составляет $O(n^{2k})$, размер φ_{move} составляет $O(n^{2k})$. Таким образом, φ может быть составлена за полиномиальное время. □

Следствие 3. $3SAT$ – NP-*полный язык.*

Доказательство. Для доказательства в силу теоремы 18 достаточно показать, что любая формула в КНФ может быть переведена в 3КНФ-формулу за полиномиальное время. Для этого каждую скобку КНФ-формулы φ вида

$$(x_1 \vee \ldots \vee x_l),\ l > 3$$

заменим на эквивалентные скобки

$$(x_1 \vee x_2 \vee z_1) \wedge (\overline{z_1} \vee x_3 \vee z_2) \wedge (\overline{z_2} \vee x_4 \vee z_3) \wedge \ldots \wedge (\overline{z_{l-3}} \vee x_{l-1} \vee x_l).$$

Данное действие имеет временную сложность $O(l)$. Также скобки вида (x_1) следует заменить на скобки вида $(x_1 \vee x_1 \vee x_1)$, $(x_1 \vee x_2)$ следует заменить на скобки вида $(x_1 \vee x_2 \vee x_2)$. Всего скобок может быть не более n, где n – число термов в исходной формуле, а величина каждой скобки также не превосходит n. Отсюда следует, что сложность редукции составляет $O(n^2)$. □

6 Дополнительные NP-полные языки

С точки зрения проблемы перебора при построении алгоритма решения той или иной задачи может быть полезно проверить не является ли данная задача NP-полной. Так как построение полиномиального алгоритма решения NP-полной задачи эквивалентно решению проблемы перебора. По этой причине рассмотрим ещё несколько NP-полных задач.

Следствие 4. $CLIQUE$ – NP-*полный язык.*

Доказательство. Данный факт следует непосредственно из теорем 15, 17 и следствия 3. □

Определение. *Покрытием* неориентированного графа G называется множество его вершин такое, что каждое ребро графа соприкасается хотя бы с одной вершиной из покрытия.

Через $VERTEX - COVER$ обозначим задачу проверки, существует ли в неориентированном графе G покрытие размера k:

$$VERTEX - COVER = \{\langle G, k\rangle : G \text{ – неориентированный граф,}$$

$$\text{содержащий покрытие размера } k\}.$$

Теорема 19. $VERTEX - COVER$ – NP-*полный язык.*

Доказательство. Верно включение $VERTEX - COVER \in$ NP, т.к. в качестве сертификата для верификатора можно использовать список вершин из искомого покрытия размера k. Тогда в силу теоремы 17 и следствия 3 достаточно показать, что существует полиномиальный алгоритм сведения $3SAT$ к $VERTEX - COVER$.

Для некоторой произвольной булевой 3КНФ-формулы φ, содержащей m различных переменных и l скобок, построим неориентированный граф следующим образом. Для каждой переменной x_i, $i = \overline{1, m}$, будет определена пара вершин, соединенных между собой рёбрами, где одна вершина соответствует x_i, а вторая соответствует $\overline{x_i}$. Для каждой из l скобок будет определена тройка вершин, соединённых между собой рёбрами, где каждая вершина из тройки соответствует одному терму x_i или $\overline{x_i}$. Также каждая вершина из тройки соединена ребром с вершиной из пары, соответствующей тому же самому терму. Тогда полученный граф будет иметь $2m + 3l$ вершин и $m + 6l$ рёбер. Положим $k = m + 2l$.

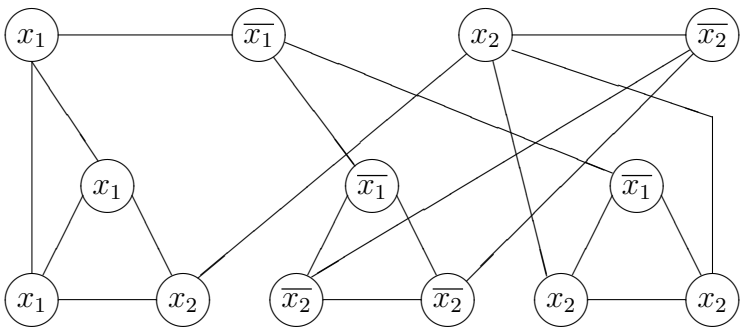
Предположим, что φ выполнима. Тогда в каждой скобке содержится хотя бы один истинный терм. Добавим в покрытие две оставшиеся вершины из каждой тройки, кроме терма, соответствующего истинному значению (тем самым все рёбра в тройках покрыты). Также добавим в покрытие те вершины из пар, которые соответствуют истинному значению терма (тем самым все рёбра в парах покрыты). Рёбра, соединяющие пары и тройки, покрыты, поскольку либо вершина из пары состоит в покрытии (если терм истинный), либо вершина из тройки (если терм ложный).

Пусть в G существует покрытие размера $k = m + 2l$. Тогда по построению данное покрытие должно содержать по одной вершине из каждой пары и по две вершины из каждой тройки. Положим истинным каждый терм, соответствующий вершине в покрытии из пары. По определению покрытия, каждая тройка должна быть соединена с вершиной в покрытии из пары. Откуда следует, что в каждой скобке φ содержится один истинный терм, т.е. φ выполнима. □

Продемонстрируем редукцию из доказательства теоремы 20 на примере построения графа G для 3КНФ-формулы φ:

$$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$$

Здесь $m = 2$, $l = 3$. Тогда граф будет иметь 13 вершин, а размер покрытия составит $k = 8$.



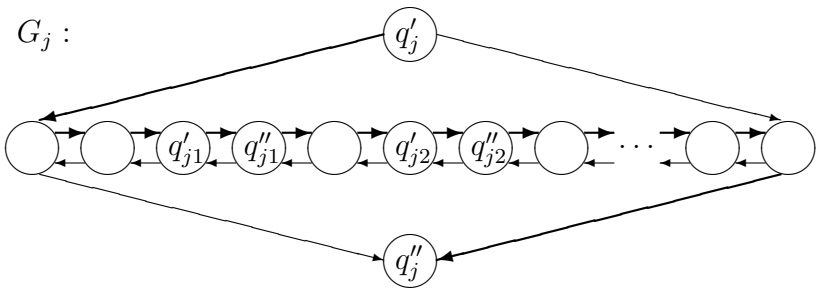
Теорема 20. $HAMPATH$ – NP-*полный язык.*

Доказательство. В силу теоремы 17 и следствия 3 достаточно показать, что существует полиномиальный алгоритм сведения $3SAT$ к $HAMPATH$.

Пусть φ – 3КНФ-формула:

$$\varphi(x_1, \ldots, x_l) = (a_1 \vee b_1 \vee c_1) \wedge \ldots \wedge (a_k \vee b_k \vee c_k),$$

где $a_i, b_i, c_i \in \{x_1, \overline{x_1}, \ldots, x_l, \overline{x_l}\}$. Построим для каждого $j = \overline{1,l}$ подграф G_j:



Для всех $j = \overline{1, l-1}$ соединим подграфы G_j, предполагая, что $q''_j = q'_{j+1}$. Также для каждой скобки $(a_i \vee b_i \vee c_i)$ введём вершину r_i, $i = \overline{1,k}$. Если для некоторых $j = \overline{1,l}$, $i = \overline{1,k}$ верно, что $x_j = a_i$, $x_j = b_i$ или $x_j = c_i$, то добавить в граф G дугу из q'_{ji} в r_i и дугу из r_i в q''_{ji}. Если верно, что $\overline{x_j} = a_i$, $\overline{x_j} = b_i$ или $\overline{x_j} = c_i$, то добавить в граф G дугу из q''_{ji} в r_i и дугу из r_i в q'_{ji}. Положим $s = q'_1$, $t = q''_l$.

Пусть φ выполнима. Гамильтонов путь из s в t в графе G по построению должен обходить по порядку все подграфы G_j. Если соответствующая переменная x_j истинная, то подграф G_j обходится в порядке обозначенном сплошными стрелками на рисунке. Если соответствующая переменная x_j ложная, то подграф G_j обходится в порядке обозначенном пунктирными стрелками на рисунке. При этом для каждой истинной переменной x_j путь внутри подграфа G_j проходит через вершину r_i, если x_j входит в i-ю скобку, а для каждой ложной переменной x_j путь внутри подграфа G_j проходит через вершину r_i, если $\overline{x_j}$ входит в i-ю скобку (но не более одного раза за весь путь для каждого $i = \overline{1,k}$.

Пусть в G есть гамильтонов путь. Тогда по построению данный путь должен обходить последовательно все подграфы G_j в одном из двух направлений, указанных на рисунке толстыми или тонкими стрелками. При этом для k пар вершин вида q'_{ji}, q''_{ji} путь должен также проходить через вершину r_i. Для обращения φ в истину достаточно положить истинными те переменные x_j, для которых подграф G_j обходится в направлении, указанном сплошными стрелками. Все остальные переменные следует положить ложными. □

Теорема 21. $SUBSET - SUM$ – NP-*полный язык.*

Доказательство. В силу теорем 12, 17 и следствия 3 достаточно показать, что существует полиномиальный алгоритм сведения $3SAT$ к $SUBSET - SUM$.

Пусть φ – 3КНФ-формула:

$$\varphi(x_1, \ldots, x_l) = (a_1 \vee b_1 \vee c_1) \wedge \ldots \wedge (a_k \vee b_k \vee c_k),$$

где $a_i, b_i, c_i \in \{x_1, \overline{x_1}, \ldots, x_l, \overline{x_l}\}$. Предполагается, что ни в одной скобке не содержится одновременно переменная и её отрицание, каждая переменная входит по крайней мере в одну скобку.

Составим множество $S \subset \mathbb{N}$. Для каждой переменной x_i, $i = \overline{1,l}$ добавим в S два числа:

$$v_i = \alpha_1^i \cdot 10^0 + \alpha_2^i \cdot 10^1 + \ldots + \alpha_l^i \cdot 10^{l-1} + \beta_1^i \cdot 10^l + \ldots + \beta_k^i \cdot 10^{l+k-1},$$

$$u_i = \gamma_1^i \cdot 10^0 + \gamma_2^i \cdot 10^1 + \ldots + \gamma_l^i \cdot 10^{l-1} + \delta_1^i \cdot 10^l + \ldots + \delta_k^i \cdot 10^{l+k-1},$$

где $\alpha_i^i = \gamma_i^i = 1$, $\alpha_j^i = \gamma_j^i = 0$, $j \neq i$. Также $\beta_j^i = 1$, если в j-ю скобку входит терм x_i, иначе $\beta_j^i = 0$. В свою очередь $\delta_j^i = 1$, если в j-ю скобку входит терм $\overline{x_i}$, иначе $\delta_j^i = 0$. Для каждой скобки $(a_i \vee b_i \vee c_i)$ в формуле φ, $i = \overline{1,k}$ добавим в S два числа:

$$d_i = 10^{l-1+i}, \quad e_i = 2 \cdot 10^{l-1+i}.$$

Также пусть

$$t = 10^0 + 10^1 + \ldots + 10^{l-1} + 4 \cdot 10^l + \ldots + 4 \cdot 10^{l+k-1}.$$

Пусть формула φ выполнима. То есть существует набор значений переменных $x_1, \ldots, x_l$, который обращает φ в истину. Выберем из S те v_i, которые соответствуют истинным значениями x_i, и те u_i, которые соответствуют ложным значениями x_i. Сумма выбранных чисел имеет вид:

$$\hat{t} = 10^0 + 10^1 + \ldots + 10^{l-1} + t_1 \cdot 10^l + \ldots + t_k \cdot 10^{l+k-1},\ t_i \in \{1,2,3\},\ i = \overline{1,k}.$$

Тогда можно дополнить выбранный набор числами d_i и e_i так, чтобы итоговая сумма совпала с t, т.е. $\langle S, t\rangle \in SUBSET - SUM$.

Пусть $\langle S, t\rangle \in SUBSET - SUM$. Тогда в разложении t на слагаемые из S по построению не может присутствовать одновременно числа v_i и u_i для всех $i = \overline{1,l}$. Положим x_i истиной, если в разложение входит v_i, и ложью, если если в разложение входит u_i. Поскольку разложение совпадает с t, у который в k старших разрядах стоит 4, то хотя бы одно истинное значение переменной входит в каждую скобку. Т.е. φ обращается в истину. $\square$

7 Теорема Сэвича

Рассмотрим сложность алгоритмов и различных вычислительных задач с точки зрения количества используемой памяти.

Определение. Пусть M – детерминированная МТ, которая останавливается на любом входе. Тогда *пространственной сложностью* алгоритма M называется функция $f\colon \mathbb{N} \to \mathbb{N}$, где $f(n)$ – максимальное число ячеек ленты машины Тьюринга M, с которых она производит чтение на входе $w \in \Sigma^*$, где $|w| = n$.

Если M – недетерминированная МТ, которая останавливается на всех ветках вычислений на любом входе, то *пространственная сложность* f определяется как максимальное число ячеек ленты машины Тьюринга M, с которых она производит чтение в любой ветке вычислений на входе $w \in \Sigma^*$, где $|w| = n$.

Определение. Пусть $f\colon \mathbb{N} \to [0; +\infty)$. Тогда

$$\mathrm{SPACE}(f(n)) = \{A \subset \Sigma^*\colon \text{существует детерминированная машина Тьюринга,}$$
$$\text{которая решает } A \text{ за } O(f(n)) \text{ памяти}\}.$$
$$\mathrm{NSPACE}(f(n)) = \{A \subset \Sigma^*\colon \text{существует недетерминированная машина Тьюринга,}$$
$$\text{которая решает } A \text{ за } O(f(n)) \text{ памяти}\}.$$

Пример 13. Рассмотрим язык SAT, который согласно теореме 18 является NP-полным. Продемонстрируем, что данный язык может быть решён с линейными затратами памяти. Построим машину Тьюринга M_1, которая решает SAT:
$M_1 =$ *на входе булева формула* φ :

1. *Перебрать последовательно все наборы значений переменных* $x_1, \ldots, x_m$.
2. *Вычисить значение* φ *на данном наборе переменных.*

3. *Если φ обратилась в истину, то допустить.*
4. *Если ни на одном наборе φ не обратилось в истину, то отвергнуть.*

Так как на каждом шаге цикла машина Тьюринга M_1 повторно использует те же ячейки памяти, то её пространственная сложность составляет $O(n)$.

Замечание. Пространственная сложность МТ, как правило, связана с временной сложностью. Хотя в прикладных задачах зачастую интерес представляет именно временная сложность, изучение пространственной сложности позволяет расширить иерархию классов сложности. Это связно с тем, что любая МТ, которая использует в вычислениях $f(n)$ ячеек памяти, должна совершить по крайней мере $f(n)$ тактов, чтобы обратиться к данным ячейкам. Т.е. временная сложность всегда не меньше пространственной.

Пример 14. Рассмотрим пространственную сложность для недетерминированных машин Тьюринга. Пусть ALL_{NFA} – язык, состоящий из описаний тех недетерминированных конечных автоматов, которые допускают все строки:

$$ALL_{NFA} = \{\langle M\rangle \colon M \text{ – НКА, } L(M) = \Sigma^*\}.$$

Построим недетерминированную машину Тьюринга N, которая решает $\overline{ALL_{NFA}}$ за линейное время. Заметим, что вообще говоря неизвестно, является ли $\overline{ALL_{NFA}}$ элементом NP или *co*NP.

$N =$ *на входе НКА M :*
1. *Пометить начальное состояние M.*
2. *Повторить следующее действие 2^q раз, где q – число состояний M.*
3. *Недетерминированно выбрать символ из Σ на роль входного, просимулировать один шаг A и пометить новое текущее состояние.*
4. *Если хотя бы раз было помечено допускающее состояния, то отвергнуть, иначе допустить.*

Машина Тьюринга N допустит вход $\langle M\rangle$, если одна из веток недетерминированных вычислений оказалась допускающей, т.е. нашлась строка длины не более 2^q, которое НКА M отвергает. Если таких строк не найдено, то в силу леммы о накачке все строки большей длины также будут допущены M. Таким образом N решает $\overline{ALL_{NFA}}$. При это затраты памяти определяются только необходимостью хранения помеченных состояний, что имеет сложность $O(n)$.

Рассмотрим, как именно меняется пространственная сложность при переходе с детерминированной на недетерминированную вычислительную модель.

Теорема 22 (Сэвича)**.** *Пусть $f \colon \mathbb{N} \to [0; +\infty)$. Тогда*

$$\mathrm{NSPACE}(f(n)) \subset \mathrm{SPACE}(f^2(n)).$$

Доказательство. Пусть N – недетерминированная машина Тьюринга, которая решает язык $A \subset \Sigma^*$ за $f(n)$ памяти. Построим детерминированную машину Тьюринга M, которая решает A. Для этого используем вспомогательную процедуру $CANYIELD(c_1, c_2, t)$, которая допускает вход, если конфигурация c_2 недетерминированно выводится из конфигурации c_1 машиной Тьюринга N не более чем за t шагов. Без ограничения общности будем предполагать, что допустимо представление $t = 2^k$.

$CANYIELD =$ *на входе c_1, c_2, t :*
1. *Если $t = 1$, проверить равенство $c_1 = c_2$ либо проверить, выводится ли c_2 из c_1 за 1 шаг.*
2. *Если $t > 2$, то для каждой конфигурации $\tilde{c}$, использующей не более $f(n)$ памяти выполнить шаги 3 и 4.*

3. *Выполнить* $CANYIELD\left(c_1, \tilde{c}, \frac{t}{2}\right)$.
4. *Выполнить* $CANYIELD\left(\tilde{c}, c_2, \frac{t}{2}\right)$.
5. *Если оба шага 3 и 4 допускают, то допустить, иначе отвергнуть.*

Перед построением M модифицируем N так, чтобы она перед тем, как допустить, очищала содержимое ленты и перемещала пишущую головку в крайнее левое положение. После чего N переходила бы в новую допускающую конфигурацию c_{accept}. Обозначим через c_{start} начальную конфигурацию N на строке w. Выберем $d \in \mathbb{N}$ так, чтобы N имела не более $2^{df(n)}$ различных конфигураций, использующих $f(n)$ памяти, где $|w| = n$. Тогда $2^{df(n)}$ является ограничением сверху на время работы N на w. Определим M следующим образом:

$M =$ *на входе* w :

1. *Выполнить* $CANYIELD(c_{start}, c_{accept}, 2^{df(n)})$ *и ответить то же самое.*

С учётом построения алгоритма $CANYIELD$ машина Тьюринга M симулирует N. При этом каждый вызов $CANYIELD$ требует для записи конфигурации c_1 и c_2, т.е. задействует $O(f(n))$ памяти. Поскольку число шагов t при каждом рекурсивном вызове $CANYIELD$ уменьшается в 2 раза, то глубина вложенности рекурсивных вызовов составляет $O(\log_2 2^{df(n)}) = O(f(n))$. То есть M расходует $O(f^2(n))$ памяти.

Если $f(n)$ при запуске M неизвестна, то можно по очереди перебрать все варианты $f(n) = 1, 2, 3, \ldots$. □

Определение. *Классом* PSPACE называется класс всех языков разрешимых с полиномиальной пространственной сложностью на детерминированной машине Тьюринга:

$$\text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k).$$

Если определить аналагичным образом класс NPSPACE для недетерминированной вычислительной модели, то в силу теоремы 22 будет справедливо равенство

$$\text{PSPACE} = \text{NPSPACE}\,.$$

Также любая машина Тьюринга, которая совершает на входе длины n не более $t(n)$ шагов, может просканировать не более $t(n)$ ячеек памяти. Отсюда следует, что $\text{TIME}(t(n)) \subset \text{SPACE}(t(n))$ и $\text{NTIME}(t(n)) \subset \text{NSPACE}(t(n))$. С другой стороны, история вычислений любой машины Тьюринга, задействующей не более $f(n)$ ячеек памяти, может состоять не более чем из $f(n)2^{O(f(n))}$ различных конфигураций, что приводит к включению $\text{SPACE}(f(n)) \subset \text{TIME}\left(f(n)2^{O(f(n))}\right)$. Окончательно с учетом теорем 10 и 22 получаем следующую цепочку включений:

$$\text{P} \subset \text{NP} \subset \text{PSPACE} = \text{NPSPACE} \subset \text{EXPTIME} = \bigcup_{k=1}^{\infty} \text{TIME}\left(2^{n^k}\right).$$

8 PSPACE-полные языки

Аналогично классу NP рассмотрим существование подмножества полного подмножества в классе PSPACE.

Определение. Язык B называется PSPACE-*трудным*, если для любого $A \in$ PSPACE справедлива оценка сложности $A \leqslant_P B$. Если $B \in$ PSPACE, то язык B называется PSPACE-*полным*.

Использование временной сложности для описания сведения языка A к B вместо пространственной сложности связано с тем, что алгоритм сведения должен быть проще, чем типичная задача из выбранного класса сложности.

Определение. *Квантифицированной булевой формулой* φ называется булева формула, у которой часть переменных связана кванторами общности $\forall$ и существования $\exists$. Если каждая переменная формулы φ связана каким-либо квантором, то формула φ называется *полностью квантифицированной.*

Пример 15. Следующие выражения являются полностью квантифицированными булевыми формулами:

$$\varphi_1 = \forall x \exists y[(x \vee y) \wedge (\overline{x} \vee \overline{y})],$$

$$\varphi_2 = \exists y \forall x[(x \vee y) \wedge (\overline{x} \vee \overline{y})].$$

При этом $\varphi_1 = TRUE$, $\varphi_2 = FALSE$.

Через $TQBF$ обозначим задачу проверки, является ли полностью квантифицированная булева формула φ истинной или ложной:

$$TQBF = \{\langle\varphi\rangle : \varphi \text{ – полностью квантифицированная истинная булева формула}\}.$$

Теорема 23. *$TQBF$ является* PSPACE*-полным языком.*

Доказательство. Построим алгоритм T с пространственной полиномиальной сложностью, который решает $TQBF$:
$T =$ *на входе* $\langle\varphi\rangle$, *где* φ *– полностью квантифицированная булева формула* :

1. *Если в* φ *нет кванторов, то вычислить её как булево выражение и вернуть результат.*
2. *Если* $\varphi = \exists x\psi$, *то вызывать* T *на* ψ, *подставив вместо* x *поочередно 1 и 0.*
3. *Если хотя бы один вызов* T *на* ψ *допускает, то допустить* φ. *Иначе отвергнуть.*
4. *Если* $\varphi = \forall x\psi$, *то вызывать* T *на* ψ, *подставив вместо* x *поочередно 1 и 0.*
5. *Если хотя оба вызова* T *на* ψ *допускают, то допустить* φ. *Иначе отвергнуть.*

Алгоритм T решает $TQBF$. При каждом рекурсивном вызове T необходимо хранить на ленте значение одной переменной. Поэтому общий объём используемой памяти составляет $O(m)$, где m – число переменных формулы φ. Таким образом, $TQBF \in$ SPACE$(n) \subset$ PSPACE.

Покажем, что $TQBF$ – PSPACE-трудный язык. Пусть $A \in$ PSPACE. Тогда существует машина Тьюринга M, которая решает A с использованием n^k памяти для некоторого $k \in \mathbb{N}$. Пусть c_1 и c_2 – некоторые конфигурации M, $t \in \mathbb{N}$. Обозначим через $\varphi_{c_1,c_2,t}$ полностью квантифицированную булеву формулу, которая является истинной тогда и только тогда, когда M может перейти из c_1 в c_2 не более чем за t шагов. Таким образом, требуемая редукция сводится к построению формулы $\varphi_{c_{start},c_{accept},2^{dn^k}}$, где число d выбирается так, что M не может иметь более 2^{dn^k} различных конфигураций для входного строки длины n. Без ограничения общности будем также полагать, что t является натуральной степенью 2.

Все переменные искомой формулы кодируют содержимое ленты M аналогично тому, как это было представлено в доказательстве теоремы 13. С учётом пространственной сложности M каждая её конфигурация занимает не более n^k ячеек памяти, а значит может быть описана $O(n^k)$ переменными. Если $t = 1$, то формула $\varphi_{c_1,c_2,t}$ сводится к проверке равенства $c_1 = c_2$ или к проверке выполнимости формулы аналогичной φ_{move} из доказательства теоремы 13, где вместо переменных используются их значения, описывающие конфигуарации c_1 и c_2.

Обозначим для некоторой конфигурации c машины Тьюринга M через $\exists c$ выражение $\exists x_1 \dots \exists x_l$, где $l = O(n^k)$, и $x_1, \dots, x_l$ – переменные, описывающие конфигурацию c. Тогда искомая формула имеет следующий вид:

$$\varphi_{c_1,c_2,t} = \exists c \forall (c_3, c_4) \in \{(c_1, c), (c, c_2)\} \left[\varphi_{c_3,c_4,\frac{t}{2}}\right].$$

Здесь выражение $\forall (c_3, c_4) \in \{(c_1, c), (c, c_2)\}$ предполагает, что переменные, описывающие c_3 и c_4, принимают те же значения, что и переменные описывающие c_1 и c или c и c_2 соответственно, а результирующая формула $\varphi_{c_3,c_4,\frac{t}{2}}$ является истинной в любом случае. Заметим, что для корректной записи выражения $\forall x \in \{y, z\}$ в терминах квантифицированных булевых формул можно использовать выражение $\forall x[(x = y) \vee (x = z)]$.

Для оценки сложности сведения A к $TQBF$ оценим размер формулы $\varphi_{c_{start},c_{accept},2^{dn^k}}$. На каждом уровне рекурсии строится формула, имеющая размер $O(n^k)$. Число уровней рекурсии представляет собой $\log_2 2^{dn^k} = O(n^k)$. Таким образом размер итоговой формулы, а значит, и сложность сведения представляют собой $O(n^{2k})$. □

Рассмотрим возможности приложения алгоритмов для поиска оптимальных стратегий в различных играх. Для этого введём понятие *игры*. Пусть φ – полностью квантифицированная булева формула, в которой кванторы, связывающие переменные, чередуются:

$$\varphi = \exists x_1 \forall x_2 \exists x_3 \dots [\psi].$$

Игрок E и игрок A по очереди выбирают значения тех переменных, которые связаны соответствующими кванторами: $\exists$ для игрока E и $\forall$ для игрока A. Считается, что игрок E победил, если при подстановке всех выбранных значений переменных формула ψ обращается в истину. Если формула ψ обращается в ложь, то победителем считается игрок A.

Пример 16. Рассмотрим формулу φ_1:

$$\varphi_1 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})].$$

Например, если игроки E и A по очереди выберут значения переменных $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, то формула

$$(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})$$

обратиться в истину и в игре победит игрок E. Более того в данной игре у игрока E существует выигрышная стратегия: достаточно принять $x_1 = 1$ и $x_3 = \overline{x_2}$.

Рассмотрим формулу φ_2, для которой у игрока A существует выигрышная стратегия:

$$\varphi_2 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})].$$

Для победы игроку A достаточно всегда принимать $x_2 = 0$.

Далее будем предполагать, что кванторы в формуле φ не обязательно чередуются, но при этом игрок E всегда определяет значения переменных, связанных кванторами $\exists$, а игрок A – переменных, связанных кванторами $\forall$.

Через $FORMULA-GAMES$ обозначим задачу проверки, существует ли выигрышная стратегия в игре, связанной с заданной полностью квантифицированной булевой формулой:

$$FORMULA - GAMES = \{\langle\varphi\rangle : \text{Игрок E имеет выигрышную стратегию в игре,}$$

$$\text{связанной с полностью квантифицированной булевой формулой } \varphi\}.$$

Теорема 24. *$FORMULA - GAMES$ является* PSPACE-*полным языком.*

Доказательство. Обратим внимание, что в полностью квантифицированной булевой формуле φ игрок E имеет выигрышную стратегию тогда и только тогда, когда для «своих» переменных он может подобрать такие значения, что для любых значений «чужих» переменных внутренняя формула ψ обращается в истину. Данный факт равносилен тому, что $\varphi = TRUE$. Таким образом, $FORMULA - GAMES = TQBF$. Откуда с учётом теоремы 23 следует утверждение теоремы 24. □

Рассмотрим ещё одну игровую задачу: игру в «города» или игру на графе. По правилам два игрока по очереди называют названия городов так, чтобы название последующего города начиналось на ту же букву, на которую закончилось название предыдущего города. Повторы названий не допустимы. Проигравшим считается тот игрок, который не может назвать город, название которого до сих пор не использовалось в игре.

Математически данную игру можно описать в виде перемещения по ориентированному графу, в котором вершины обозначают города, а дуги определяют допустимую последовательность названий городов. Игроки совершают перемещение между вершинами графа поочерёдно. При этом вершины не могут повторяться. Проигравшим считается тот игрок, который не может сделать следующий ход.

Предполагая, что игрок I ходит первым, а игрок II – вторым, обозначим через GG задачу проверки, существует ли у игрока I выигрышная стратегия в игре на заданном ориентированном графе:

$$GG = \{\langle G, b\rangle : \text{Игрок I имеет выигрышную стратегию}$$

$$\text{в игре на ориентированном графе } G, \text{ начиная из вершины } b\}.$$

Теорема 25. *GG является* PSPACE-*полным языком.*

Доказательство. Продемонстрируем, что $GG \in$ PSPACE. Построим решатель GG:
$M =$ *на входе* $\langle G, b\rangle$, *где* G *– ориентированный граф,* b *– его вершина* :

1. *Если из* b *нет выходящих дуг, то отвергнуть.*
2. *Удалить вершину* b *и все связанные с ней дуги, чтобы получить новый граф* G_1.
3. *Для каждой вершины* $b_1, \ldots, b_k$ *графа* G_1 *рекурсивно вызвать* M *на* $\langle G_1, b_i\rangle$.
4. *Если каждый рекурсивный вызов в шаге 3 допустил, то отвергнуть.*
5. *Иначе допустить.*

Глубина рекурсии в M равна числу вершин. При этом каждый рекурсивный вызов расходует память только на построение нового подграфа G_1. Таким образом $GG \in \text{SPACE}(n^2)$.

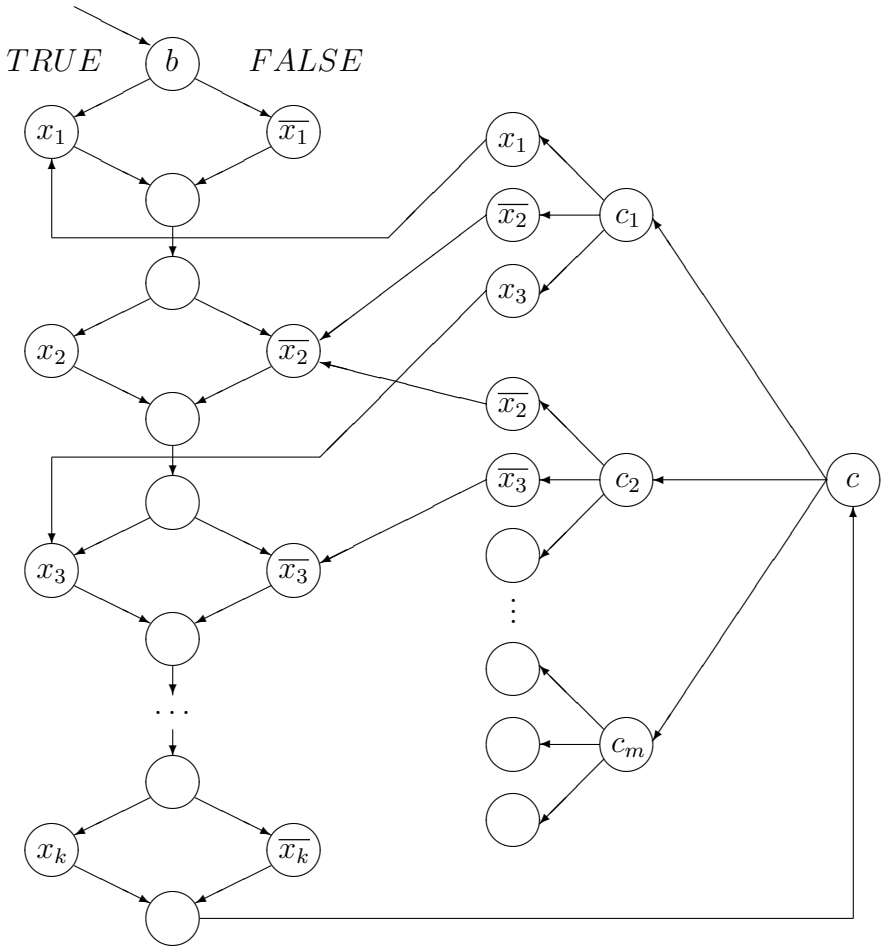
Для доказательства того, что GG является $PSPACE$-трудным языком, построим полиномиальную по времени редукцию от $FORMULA - GAMES$ к GG. Пусть φ – полностью квантифицированная булева формула от k переменных $x_1, \ldots, x_k$:

$$\varphi = \exists x_1 \forall x_2 \exists x_3 \ldots [\psi].$$

Здесь без ограничения общности предполагается, что кванторы перед переменными чередуются, начиная с $\exists$, число переменных нечётное, а ψ находится в КНФ. На основе φ построим граф G, игра на котором иммитирует игру на формуле φ. Структура графа G на примере формулы

$$\varphi = \exists x_1 \forall x_2 \exists x_3 \ldots \exists x_k [(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee \ldots) \wedge \ldots \wedge (\ldots)]$$

представлена на следующем рисунке.



Вершины отвечающие термам внутри скобок формулы φ соединяются дугами с вершинами, которые отождествляются либо с переменной, либо с её отрицанием. Выбор значений переменных в игре на формуле φ иммитируется проходом через вершины x_i и $\overline{x_i}$. При этом переход в вершину c совершает игрок I, так как число переменных нечётное. Тогда выбор вершины c_j, соответствующей какой-либо из скобок формулы φ производит игрок II. Если выбранные значения переменных обращают φ в истину (т.е. победил игрок E), то в каждой скобке есть истинный терм, и игрок I может выбрать вершину, соответствующую ему, что завершит игру на графе G победой игрока I. И, наоборот, если выбранные значения переменных обращают φ в ложь (т.е. победил игрок A), то существует скобка, в которой все термы ложные, и игрок II может выбрать вершину, соответствующую данной скобке. Какой бы ход не совершил игрок II впоследствии, игрок I сможет совершить ещё один ход, который завершит игру в «города» на графе G победой игрока II.

Таким образом, существование выигрышной стратегии для игрока E на формуле φ эквивалентно существованию выигрышной стратегии для игрока I на графе G, начиная с вершины b.

Временная сложность построенной редукции пропорциональна квадрату числа вершин графа G, которое линейно зависит от длины формулы φ, т.е. $FORMULA - GAMES \leqslant_P GG$. □

Замечание. Вообще говоря, из теоремы 25 следует, что существание полиномиальных алгоритмов, позволяющих построить оптимальную стратегию в играх, подобных игре на ориентированном графе, эквивалентно равенству классов P и PSPACE. Одной из таких игр являются, например, шахматы.

9 Классы L и NL

В данном разделе обсудим возможность решить различные вычислительные задачи с использованием менее чем линейного объёма памяти. То есть машина Тьюринга должна использовать количество памяти меньшее, чем длина входной строки. Для построения соответствующего класса сложности модифицируем вычислительную модель.

Далее в этом разделе будем предполагать, что рассматриваемые машины Тьюринга являются двухленточными. Первая лента используется в режиме «только для чтения», и на неё изначально записано входная строка. Вторая лента машины Тьюринга используется в обычном режиме: и для чтения, и для записи. При этом пространственную сложность такой машины Тьюринга определяет только число использованных ячеек на второй ленте.

Определение. *Классом* L называется класс всех языков, разрешимых с логарифмическими затратами памяти на детерминированной машине Тьюринга:

$$\mathrm{L} = \mathrm{SPACE}(\log n).$$

Классом NL называется класс всех языков, разрешимых с логарифмическими затратами памяти на недетерминированной машине Тьюринга:

$$\mathrm{NL} = \mathrm{NSPACE}(\log n).$$

Пример 17. Рассмотрим язык $A = \{0^k 1^k \colon k \geqslant 0\}$. Покажем, что $A \in \mathrm{L}$. Построим двухленточную машину Тьюринга M, которая решает A:
$M =$ *на входе* w :

1. *Проверить, что* $w = 0^i 1^k$. *Если это не так, то отвергнуть.*
2. *Считывая по очереди нули с первой ленты, через инкремент вычислить их количество на второй ленте.*
3. *Считывая по очереди единицы с первой ленты, через дикремент вычислить разность числа нулей и единиц на второй ленте.*
4. *Если счётчик на второй ленте равен 0, то допустить. Иначе отвергнуть.*

Поскольку для записи любого числа достаточно памяти равной логарифму от его значения, то M решает A с пространственной сложностью $O(\log n)$. При этом содержимое первой ленты не меняется.

Пример 18. Рассмотрим язык $PATH$:

$$PATH = \{\langle G, s, t\rangle \colon G \text{ – ориентированный граф,}$$

$$\text{в котором есть путь из вершины } s \text{ в вершину } t\}.$$

Как продемонстрировано в теореме 6, верно включение $PATH \in \mathrm{P}$. Но детерминированный алгоритм, предложенный при доказательстве теоремы, использовал $O(n)$ памяти. Вообще говоря неизвестно, возможно ли решить $PATH$ с менее чем линейными затратами памяти на детерминированной машине Тьюринга, но можно продемонстрировать включение $PATH \in \mathrm{NL}$.
$N =$ *на входе* $\langle G, s, t\rangle$, *где* G *– ориентированный граф,* s, t *– его вершины* :

1. *Если* $s = t$, *то допустить.*
2. *Записать на второй ленте* m *– число вершин графа* G.
3. *Записать на второй ленте также в качестве текущей вершины* s.
4. *Повторить следующее действие* m *раз.*
5. *Недетерминировано заменить текущую вершину на каждую вершину, достижимую из неё за один шаг.*
6. *Если текущая вершина совпадает с* t, *то допустить.*

7. *Если t не было достигнуто, то отвергнуть.*

Поскольку в ходе работы N требуется хранить в памяти только текущую вершину и значение счётчика для шага 4, а не весь путь, то $PATH \in \mathrm{NL}$.

Прежняя оценка временной сложности в виде $2^{O(f(n))}$ шагов для машины Тьюринга, использующей не более $O(f(n))$ памяти некорректна для новой вычислительной модели. Например, машина Тьюринга, не использующая вторую ленту вообще, что эквивалентно пространственной сложности $O(1)$, может работать с линейной временной сложностью. Для уточнения оценки временной сложности посредством пространственной уточним понятие конфигурации машины Тьюринга.

Определение. Пусть M – двухленточная машина Тьюринга, первая лента которой работает только для чтения. Тогда *конфигурацией* M на входной строке $w \in \Sigma^*$ называется совокупность текущего состояния машины Тьюринга, местоположения рабочих головок на обеих лентах и содержимое второй ленты памяти.

Входная строка w, которое перманентно записано на первой ленте, не предполагается частью конфигурации.

Замечание. *Для временной сложности $g(n)$ двухленточной машины Тьюринга справедлива следующая оценка $g(n) = n2^{O(f(n))}$, где $f(n)$ – пространственная сложность данной машины Тьюринга. Если $f(n) \geqslant \log n$, то $n2^{O(f(n))} = 2^{O(f(n))}$. Данный факт позволяет усилить теорему Сэвича на случай, когда $f(n) \geqslant \log n$.*

Рассмотрим вопросы соотношения классов L и NL. Как будет показано далее, верно включение $\mathrm{NL} \subset \mathrm{P}$, и любые два языка из NL могут быть сведены друг к другу за полиномиальное время. Тогда для введения понятия NL-полных языков необходимо использовать другой тип сводимости.

Определение. *Преобразователем с логарифмической пространственной сложностью* называется трёхленточная машина Тьюринга, у которой первая лента работает в режиме чтения, вторая работает в режиме записи, а третья является обычной рабочей лентой, но может использовать не более $O(\log n)$ ячеек памяти, где n – длина слова, записанная на 1-й ленте.

Вычислимой функцией с логарифмической пространственной сложностью наызывается функция $f\colon \Sigma^* \to \Sigma^*$, для которой существует преобразователь с логарифмической пространственной сложностью M, который, начиная вычисления с $w \in \Sigma^*$ на первой ленте, завершает их с $f(w)$ на второй ленте.

Говорят, что язык A *сводим с логарифмической пространственной сложностью* к языку B, обозначается как $A \leqslant_L B$, если существует вычислимая функция с логарифмической пространственной сложностью f такая, что $w \in A$ тогда и только тогда, когда $f(w) \in B$.

Определение. Язык B называется NL-*трудным*, если для любого $A \in \mathrm{NL}$ справедлива оценка сложности $A \leqslant_L B$. Если также $B \in \mathrm{NL}$, то язык B называется NL-*полным*.

Теорема 26. *Пусть $A \leqslant_L B$ и $B \in \mathrm{L}$. Тогда $A \in \mathrm{L}$.*

Доказательство. Пусть M_B – машина Тьюринга, которая решает B с логарифмической пространственной сложностью, T – преобразователь с логарифмической пространственной сложностью, обеспечивающий редукцию от A к B. Построим для языка A решатель M_A:
$M_A =$ *на входе w :*

1. *Положить $i = 1$.*
2. *Вычилсить i-й символ $f(w)$, симулируя T.*
3. *Симулировать M_B до тех пор, пока для симуляции не потребуется передвинуть головку на первой ленте на j-ю позицию, $j \neq i$.*

4. Вычилсить j-й символ $f(w)$, симулируя T, положить $i = j$ и перейти к шагу 2.

5. Ответить то же, что и M_B.

Для работы M_A, кроме логарифмического объёма памяти, необходимого для симуляции M_B, требуется хранить на ленте только один символ строки $f(w)$ и его порядковый номер i в строке. Поскольку T является преобразователем с логарифмической пространственной сложностью, время его работы, а значит и $|f(w)|$, не более чем полиномиальная, по отношению к $|w|$. Тогда для записи i требуется логарифмическое количество памяти. □

Теорема 27. $PATH$ – NL-*полный язык.*

Доказательство. Как было ранее продемоснтрировано, $PATH \in$ NL. Поэтому для доказательства теоремы достаточно показать, что для произвольного $A \in$ NL верно, что $A \leqslant_L PATH$.

Пусть M – недетерминированная машина Тьюринга, которая решает A с логарифмической пространственной сложностью. Предполагается, что M построена так, что перед тем как допустить строку, она полностью очищает содержимое рабочей ленты и переводит обе рабочие головки в крайнее левое положение. Редукция строит для произвольного $w \in \Sigma^*$ граф ориентированный граф G, где каждая вершина G является конфигурацией M на w. Если из некоторой конфигурации c_1 за один шаг может быть выведена конфигурация c_2, то в G добавляется дуга из вершины c_1 в вершину c_2. В качестве вершины s предполагается начальная конфигурация, в качестве вершины t – допускающая. Тогда по построению $\langle G, s, t\rangle \in PATH$ тогда и только тогда, когда $w \in A$.

Для построения графа G требуется построить множества его вершин и дуг. Список вершин может быть построен посредством полного перебора всех строк длины не более $d \log n$ и проверкой их на то, являются ли они конфигурациями M на w. Здесь d выбрано так, чтобы пространственная сложность M не превосходила $d \log n$. Для построения множества дуг требуется перебрать все пары конфигураций и проверить выводится ли одна из другой согласно переходной функции машины Тьюринга M. Таким образом данная редукция реализуема посредством вычислимой функции с логарифмической пространственной сложностью. □

Следствие 5. *Верно включение* NL $\subset$ P.

Доказательство. Согласно теореме 27 для любого $A \in$ NL следует, что $A \leqslant_L PATH$. Откуда получаем оценку $A \leqslant_P PATH$. Поскольку в силу теоремы 6 $PATH \in$ P, то с учётом теоремы 14 верно также, что $A \in$ P. □

Замечание. *Окончательно, можно представить следующую иерархию классов сложности:*

$$\mathrm{L} \subset \mathrm{NL} \subset \mathrm{P} \subset \mathrm{NP} \subset \mathrm{PSPACE} = \mathrm{NPSPACE} \subset \mathrm{EXPTIME}\,.$$