Введение

Каждый алгоритм A характеризуется тем, что на его вход могут поступать различные входные данные x, которые он преобразует в некоторые выходные данные y. При этом процесс работы A на входных данных x можно охарактеризовать некоторыми сложностными характеристиками $L_A(x)$ (число шагов алгоритма, объем используемой памяти и др.). Однако дать явное представление функции $L_A(x)$ для всех x обычно не представляется возможным. Даже поведение $L_A(x)$ как функции от x обычно трудно описать. Поэтому при анализе сложности алгоритмов часто рассматривают более грубые характеристики. Наиболее распространенным является следующий подход. Входные данные характеризуются некоторым натуральным параметром n их сложности (чаще всего n — длина представления входных данных некоторым заданным способом). Далее изучается функция $L_A(n)$, определяемая как максимум $L_A(x)$ по всем x с параметром n (сложность в худшем случае) или как некоторое среднее $L_A(x)$ по всем x с параметром n (средняя сложность). В этих случаях уже удается получать интересные результаты. В данном пособии мы будем рассматривать только одну сложностную характеристику алгоритмов — время, или число шагов, работы алгоритма. При этом мы должны четко определять, что такое шаг алгоритма. Если же мы хотим получать утверждения типа "для любого алгоритма", то мы также должны четко описать весь класс алгоритмов, которые мы рассматриваем. Мы поясним это вначале примерами.

Поиск в упорядоченном массиве.

Пусть имеется упорядоченный массив элементов из некоторого линейно упорядоченного множества $a_1 < a_2 < \ldots < a_n$. На вход алгоритма будет поступать некоторый элемент a, совпадающий с одним из элементов $a_1, a_2, \ldots, a_n$. Один шаг алгоритма состоит в сравнении a с некоторым a_i, получении одного из двух ответов $a \leqslant a_i$ или $a > a_i$ и анализе этого ответа. Алгоритм должен выдать номер j того элемента a_j, для которого $a = a_j$. Рассмотрим, например, алгоритм, который сравнивает a по очереди со всеми элементами от a_1 до a_n. Тогда если $a = a_1$, он может выдать ответ уже после 1-го шага. Однако, если $a = a_{n-1}$ или $a = a_n$, то алгоритм будет делать $n-1$ шагов. В среднем, если считать, что a совпадает с любым a_i с вероятностью $\frac{1}{n}$, число шагов будет $\frac{(1+2+\ldots+n-1)+n-1}{n} = \frac{n+1}{2} - \frac{1}{n}$.

В дальнейшем мы будем алгоритмы считать детерминированными. Так, например, для любого алгоритма поиска элемента в упорядоченном массиве на первом шаге однозначно определяется номер i элемента, с которым сравнивается a. Этот номер не зависит от входа a. В зависимости от ответа ($a \leqslant a_i$ или $a > a_i$) однозначно определяется следующий номер элемента, с которым сравнивается a, и т.д. Таким образом всякий алгоритм поиска (из указанного выше класса) можно представить корневым бинарным деревом, в котором каждой вершине, отличной от листьев, приписан некоторый номер элемента с которым сравнивается a, а каждому листу приписан номер элемента, равного a.

Определение. *Сложностью (в худшем случае) $L_A(n)$ алгоритма поиска в упорядоченном массиве из n элементов называется максимальное число сравнений элемента a с элементами массива до получения ответа. Средней сложностью $L_A^{cp}(n)$ алгоритма поиска A в упорядоченном массиве из n элементов называется величина $L_A^{cp}(n) = \frac{1}{n}\sum_{i=1}^{n} L_A(a_i)$, где $L_A(a_i)$ — число шагов алгоритма, если вход $a = a_i$.*

Если α - действительное число, то через $\lfloor\alpha\rfloor$ и $\lceil\alpha\rceil$ мы будем обозначать наибольшее (соответственно, наименьшее) целое число, не большее (соответственно, не меньшее) чем α. Часто $\lfloor\alpha\rfloor$ обозначают $[\alpha]$ и называют целой частью числа α.

Теорема. *Существует алгоритм A поиска в упорядоченном массиве, для которого $L_A(n) = \lfloor\log_2 n\rfloor$.*

Доказательство. Доказывать существование алгоритма с нужными свойствами обычно легко — достаточно явно предъявить такой алгоритм. Требуемому в теореме условию удовлетворяет следующий алгоритм, называемый "бинарным поиском", и описываемый рекуррентно.

Если $n = 1$, то выдать ответ $a = a_1$.

Если $n \geqslant 2$, то вычислить $k = \lfloor\frac{n}{2}\rfloor$ и сравнить a с a_k. Если $a \leqslant a_k$, то рекуррентно (тем же алгоритмам) осуществить поиск a в массиве $a_1 < a_2 < \ldots < a_k$. Если $a > a_k$, то осуществить (рекуррентно) поиск a в массиве $a_{k+1} < a_{k+2} < \ldots < a_n$.

В любом случае длина получаемого массива не превосходит $n - \lfloor\frac{n}{2}\rfloor = \lceil\frac{n}{2}\rceil$, и, следовательно, $L_A(n) = 1 + L_A(\lceil\frac{n}{2}\rceil)$. Кроме того $L_A(1) = 0$. Докажем индукцией по m, что для всех натуральных n, таких, что $2^{m-1} < n \leqslant 2^m$, выполняется $L_A(n) = m$. При $m = 0$ получаем $n = 1$ и $L_A(1) = 0 = m$. Пусть утверждение верно для $m = p$ и $2^p < n \leqslant 2^{p+1}$. Тогда $2^{p-1} < \lceil\frac{n}{2}\rceil \leqslant 2^p$ и по предположению индукции $L_A(\lceil\frac{n}{2}\rceil) = p$.

Отсюда $L_A(n) = 1 + L_A(\lceil \frac{n}{2} \rceil) = p + 1$, то есть утверждение верно для $m = p + 1$. По индукции получаем, что утверждение верно для всех n, то есть $L_A(n) = m = \lceil \log_2 n \rceil$. Теорема доказана.

Следствие. *Для алгоритма A бинарного поиска* $L_A^{cp}(n) \leqslant \lceil \log_2 n \rceil$.

Доказать утверждение типа "для любого алгоритма" обычно существенно труднее, чем утверждение типа "существует алгоритм". В этом случае мы должны четко описать весь класс рассматриваемых алгоритмов. Выше было указано, что любой алгоритм поиска в упорядоченном массиве из n элементов можно представить в виде бинарного дерева. Поэтому далее мы рассмотрим некоторые свойства бинарных деревьев.

Определение. *Глубиной $h(x)$ листа x в корневом дереве D будем называть число ребер в (единственном) пути из корня дерева в лист x. Высотой $h(D)$ дерева D будем называть* $\max h(x)$, *где максимум берется по всем листьям дерева D. Средней высотой $h_{cp}(D)$ дерева D будем называть среднее арифметическое величин $h(X)$ по всем листьям дерева D.*

Лемма. *Для любого бинарного дерева с n листьями выполняются неравенства: 1)$h(D) \geqslant \lceil \log_2 n \rceil$, 2)$h_{cp}(D) \geqslant \log_2 n$.*

Доказательство. 1)Любое бинарное дерево высоты h можно достроить до полного бинарного дерева высоты h (в котором все пути от корня до листьев содержат по h ребер). Для этого достаточно к каждому листу x высоты $h(x)$ подклеить полное бинарное дерево высоты $h - h(x)$. При этом число листьев не уменьшится. Поскольку в полном бинарном дереве высоты h число листьев равно 2^h, то для числа n листьев в исходном дереве выполняется неравенство $n \leqslant 2^h$, или $h \geqslant \log_2 n$. Так как h — натуральное число, то $h \geqslant \lceil \log_2 n \rceil$.

2)Опять достроим дерево d высоты h до полного бинарного дерева. Поскольку к листу x подклеивается полное бинарное дерево высоты $h - h(x)$, то вместо листа x образуется $2^{h-h(x)}$ листьев. Так как общее число листьев в полном бинарном дереве высоты h равно 2^h, то получаем равенство $\sum_x 2^{h-h(x)} = 2^h$ где суммирование ведется по всем листьям дерева D. Сокращая на 2^h, получаем следующее равенство, верное для любого бинарного дерева:

$$\sum_x \frac{1}{2^{h(x)}} = 1, ()$$

где суммирование ведется по всем листьям дерева D. Пусть число листьев в дереве D равно n. По теореме о среднем арифметическом и

среднем геометрическом n положительных чисел имеем

$$\frac{1}{n} = \frac{1}{n}\sum_x \frac{1}{2^{h(x)}} \geqslant \sqrt[n]{\prod_x \frac{1}{2^{h(x)}}} = \sqrt[n]{\frac{1}{2^{\sum_x h(x)}}}.$$

Отсюда
$2^{\sum_x h(x)} \geqslant n^n$.
и
$\frac{1}{n}\sum_x h(x) \geqslant \log_2 n$.
Лемма доказана.

Теперь уже легко доказать следующее утверждение.

Теорема. *Для любого алгоритма A поиска в упорядоченном массиве из n элементов справедливы оценки*

$$L_A(n) \geqslant \lceil \log_2 n \rceil, L_A^{cp}(n) \geqslant \log_2 n.$$

Доказательство. Представим алгоритм A в виде бинарного дерева D. Так как результатом алгоритма может оказаться любой номер j от 1 до n (такой, что $a_j = a$), то в дереве D не менее n листьев. Поэтому утверждение теоремы следует из определения величин $L_A(n)$ и $L_A^{\mathrm{cp}}(n)$ и леммы.

Сортировка

В качестве еще одного примера рассмотрим задачу сортировки на линейно упорядоченном множестве, которая обычно ставится следующим образом.

Вход: последовательность элементов $a_1, a_2, \ldots, a_n$ некоторого линейно упорядоченного множества (для простоты будем считать, что $a_i \neq a_j$ при $i \neq j$).

Выход: перестановка $(i_1, i_2, \ldots, i_n)$ элементов $1, 2, \ldots, n$ такая, что $a_{i_1} < a_{i_2} < \ldots < a_{i_n}$.

Один шаг алгоритма: сравнение любой пары элементов a_i и a_j и любое использование полученного ответа $a_i < a_j$ или $a_i > a_j$. Алгоритм считаем детерминированным, то есть для данного n однозначно определена пара номеров (i, j) тех элементов, которые сравниваются на первом шаге. В зависимости от одного из двух ответов однозначно определяется пара номеров тех элементов, которые сравниваются на втором шаге и т.д. Таким образом, алгоритм можно представить в виде бинарного корневого дерева, в котором каждой вершине, отличной от листьев, приписана пара номеров сравниваемых элементов, а листьям приписаны ответы в виде перестановок $(i_1, i_2, \ldots, i_n)$.

Определение. *Сложностью $L_A(n)$ алгоритма сортировки A называется максимальное число вопросов от начала работы до ответа, где максимум берется по всем возможным входным последовательностям длины n. Сложностью сортировки n элементов $L_{copm}(n)$ называется* $\min L_A(n)$*, где минимум берется по всем алгоритмам, сортирующим правильно n элементов.*

Теорема. *Для любого алгоритма A, сортирующего n элементов, выполняется неравенство $L_A(n) \geqslant \log_2 n!$.*

Доказательство. Алгоритм A можно представить в виде бинарного дерева D. Любая перестановка $(i_1, i_2, \ldots, i_n)$ элементов $1, 2, \ldots, n$ может быть ответом в алгоритме и, следовательно, должна быть приписана хотя бы одному листу. Поэтому в дереве D не менее $n!$ листьев. Отсюда по лемме получаем, что высота дерева $h(D) \geqslant \log_2 n!$. Но, по определению $L_A(n) = h(D)$. Теорема доказана.

Следствие. $L_{copm}(n) \geqslant \log_2 n!$.

Используя формулу стирлинга для $n!$, получаем

Следствие. $L_{copm}(n) \geqslant (1 - o(1)) n \log_2 n$ *(или* $L_{copm}(n) \gtrsim n \log_2 n$*).*

Рассмотрим далее 2 алгоритма сортировки, сложность которых близка к полученной нижней оценке.

Сортировка вставками

Последовательно решаем подзадачи: отсортировать $a_1, \ldots, a_n$ при $k = 1, 2, \ldots, n$. При $k = 1$ (базис) ответ тривиален, при $k = n$ получаем ответ всей задачи. Переход от подзадачи с параметром $k-1$ к k происходит путем вставки в уже упорядоченную последовательность $a_{i_1} < a_{i_2} < \ldots < a_{i_{k-1}}$ элемента a_k на соответствующее место. При этом для a_k имеется k возможных положений: перед a_{i_1}, между a_{i_1} и a_{i_2},..., после $a_{i_{k-1}}$.Вставка a_k на нужное место осуществляется бинарным поиском.

Теорема. *Сложность алгоритма сортировки вставками* $L_{вст}(n)$ *удовлетворяет неравенству* $L_{вст}(n) \leqslant \log_2 n! + n - 1$.

Доказательство. Так как при вставке элемента a_k вначале имеется k возможных положений: перед a_{i_1},между a_{i_1} и a_{i_2},..., после $a_{i_{k-1}}$, то для вставки a_k бинарным поиском нужно сделать не более $\lceil \log_2 k \rceil$ сравнений. Весь алгоритм требует сравнений не более $\lceil \log_2 2 \rceil + \lceil \log_2 3 \rceil + \lceil \log_2 4 \rceil + \ldots + \lceil \log_2 n \rceil \leqslant \log_2 2 + \log_2 3 + \ldots + \log_2 n + (n-1) = \log_2 n! + n - 1$.

Следствие. $L_{вст}(n) \leqslant (1 + o(1)) n \log_2 n$ *при* $n \to \infty$.

Следствие. $L_{сорт}(n) \sim n \log_2 n$ *при* $n \to \infty$.

Последнее следствие вытекает из следствий из теорем 1 и 2.

Сортировка слиянием

Сортировка слиянием n элементов описывается рекурсивно. Если $n = 1$, то задача тривиальна. Для $n \geqslant 2$ делим последовательность $a_1, a_2, \ldots, a_n$ на 2 последовательности $a_1, a_2, \ldots, a_{\lfloor \frac{n}{2} \rfloor}$, сортируем тем же алгоритмом сортировки слиянием каждую из подпоследовательностей и затем сливаем 2 полученные отсортированные последовательности $A = (a_{i_1} < a_{i_2} < \ldots < a_{i_{\lfloor \frac{n}{2} \rfloor}})$ и $B = (a_{j_1} < a_{j_2} < \ldots < a_{j_{n-\lfloor \frac{n}{2} \rfloor}})$, формируя отсортированную последовательность C. На каждом шаге слияния мы сравниваем первые элементы из A и B и переносим меньший из них очередным элементом в C (если A или B становится пустым, то переносим оставшиеся элементы в C по порядку). Пусть $L_{\text{СЛ}}(n)$ — сложность (число сравнений) алгоритма сортировки слиянием для n элементов в худшем случае. Тогда $L_{\text{СЛ}}(1) = 0$ и $L_{\text{СЛ}}(n) = L_{\text{СЛ}}(\lfloor \frac{n}{2} \rfloor) + L_{\text{СЛ}}(\lceil \frac{n}{2} \rceil) + n - 1$ при $n \geqslant 2$, поскольку на слияние в худшем случае может потребоваться $n - 1$ сравнений.

Лемма. $L_{сл}(n) = n \log_2 n - n + 1$ *для* $n = 2^k$, *где* k - *любое натуральное число или* $k = 0$.

Доказательство (индукцией по k). При $k = 0$ получаем верное равенство $L_{\text{СЛ}}(1) = 0$. Пусть утверждение леммы верно при всех

$0 \leqslant k \leqslant m-1$, где m - натуральное число. Тогда для $k = m$ имеем $L_{\text{СЛ}}(2^m) = 2L_{\text{СЛ}}(2^{m-1}) + 2^m - 1 = 2(2^{m-1} \cdot (m-1) - 2^{m-1} + 1) + 2^m - 1 = m2^m - 2^m + 1$, то есть для $k = m$ утверждение леммы также верно. Следовательно, оно верно для всех натуральных k.

Теорема. $L_{сл}(n) < 2n \log_2 n + 1$ *для всех натуральных* n.

Доказательство. Утверждение теоремы справедливо при $n = 1$. Для любого натурального $n \geqslant 2$ найдется натуральное k такое, что $2^{k-1} < n \leqslant 2^k$. Функция $L_{\text{СЛ}}(n)$, очевидно, не убывает с ростом n. Поэтому $L_{\text{СЛ}}(n) \leqslant L_{\text{СЛ}}(2^k) = 2^k \cdot k - 2^k + 1 = 2^k(k-1) + 1 < 2n \log_2 n + 1$. Теорема доказана.

Рекуррентные методы построения алгоритмов. Метод динамического программирования

Одно из важных направлений в построении быстрых алгоритмов — это рекуррентные методы. При этом решение задачи сводится к решению более простых подзадач такого же типа, которые, в свою очередь, сводятся к еще более простым подзадачам и т.д. Естественно при этом должен быть некоторый базисный уровень, задачи которого решаются уже не рекуррентно, а непосредственно. Можно выделить 2 основных рекуррентных метода, которые используются для построения быстрых алгоритмов: метод динамического программирования и метод "разделяй и властвуй".

В самом широком виде идея динамического программирования состоит в выделении в данной задаче с параметром n (характеризующим длину входа) подзадач с меньшими параметрами и решении подзадач в соответствии с увеличением параметра, начиная с самого меньшего (обычно 0 или 1). При этом задача с параметром k решается, когда уже решены все подзадачи с параметром $k-1$ и меньше (иногда не $k-1$, а $k-c$, где c — константа). При этом большого числа подзадач удается часто избежать за счет того, что решение разных подзадач сводится к решению одних и тех же подзадач. Рассмотрим примеры.

Задача об оптимальном порядке умножения матриц.

Мы будем рассматривать здесь только обычный способ умножения двух матриц ("строчка на столбец") и будем учитывать только число умножений элементов. При этом если матрицы A и B имеют размеры $m \times n$ и $n \times p$, то для вычисления $A \cdot B$ требуется, очевидно, mnp умножений элементов. Известно, что для любых трех матриц $(AB)C = A(BC)$, то есть произведение матриц не зависит от расстановки скобок. Однако число операций умножения элементов может при этом оказаться разным.

Пример. Пусть матрицы A, B, C имеют размеры $n \times 1, 1 \times n, n \times n$. Тогда матрица AB имеет размеры $n \times n$ и при вычислении $(AB)C$ используется $n^2 + n^3$ умножений элементов. Матрица BC имеет размеры $1 \times n$, поэтому при вычислении $A(BC)$ используется $n^2 + n^2 = 2n^2$ умножений элементов, что примерно в $\frac{n}{2}$ раз меньше, чем для $(AB)C$. Таким образом, имеет смысл следующая задача.

Задача. Вход: набор натуральных чисел $(m_0, m_1, \ldots, m_n)$ (который задает размеры матриц в произведении $A_1 A_2 \cdot \ldots \cdot A_n$, где A_i имеет размеры $m_{i-1} \times m_i$).

Требуется: Расставить скобки в произведении $A_1 \cdot A_2 \cdot \ldots \cdot A_n$ так, чтобы общее число умножений элементов было минимальным, и вычислить это минимальное число.

Посмотрим сначала, какова сложность тривиального алгоритма, который перебирает все способы расстановки скобок. Пусть a_n — число способов правильной расстановки скобок в произведении $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

Теорема. $a_n = \frac{1}{n} C_{2n-2}^{n-1} = \frac{(2n-2)!}{n!(n-1)!}$ *при* $n \geqslant 2$.

Доказательство. Очевидно, что $a_1 = 1$, $a_2 = 1$, $a_3 = 2$. Операция, которую мы сделаем последней в $A_1 \cdot A_2 \cdot \ldots \cdot A_n$, сводит задачу к 2 подзадачам $A_1 \cdot \ldots \cdot A_k$ и $A_{k+1} \cdot \ldots \cdot A_n$, где $1 \leqslant k \leqslant n-1$. Поэтому

$$a_n = a_1 a_{n-1} + a_2 a_{n-2} + \ldots + a_{n-1} a_1.$$

Рассмотрим производящую функцию

$$f(x) = a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n + \ldots ()$$

Тогда

$$f^2(x) = (a_1 a_1) x^2 + (a_1 a_2 + a_2 a_1) x^3 + (a_1 a_3 + a_2 a_2 + a_3 a_1) x^4 + \ldots = \tag{1}$$

$$a_2 x^2 + a_3 x^3 + a_4 x^4 + \ldots = f(x) - a_1 x = f(x) - x. \tag{2}$$

Таким образом, $f^2(x) - f(x) + x = 0$. Решая квадратное уравнение, получаем $f(x) = \frac{1 \pm \sqrt{1-4x}}{2}$. Поскольку $f(0) = 0$, то $f(x) = \frac{1-\sqrt{1-4x}}{2}$. Раскладывая $f(x)$ в ряд Тейлора и сравнивая с ( ), получаем (проверьте):

$$a_n = \frac{1}{2} \cdot \frac{3}{2} \cdot \frac{5}{2} \cdot \ldots \cdot \frac{2n-3}{2} \cdot \frac{4^{n-1}}{n!} = \tag{3}$$

$$\frac{2^{n-1}}{n!} \cdot (1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2n-3)) = \frac{2^{n-1}(2n-2)!}{n!(2 \cdot 4 \cdot 6 \cdot \cdots \cdot (2n-2))} = \tag{4}$$

$$\frac{2^{n-1}(2n-2)!}{n! 2^{n-1}(n-1)!} = \frac{1}{n} C_{2n-2}^{n-1}. \tag{5}$$

Теорема доказана.

Замечание. Для полной строгости в доказательстве нужно обсудить существование функции $f(x)$, заданной равенством (). Можно показать, что ряд справа сходится, например, при $0 \leqslant x \leqslant \frac{1}{4}$.

Раскрывая факториалы по формуле Стирлинга, легко получить, что $C_{2m}^{m} \sim \sqrt{\frac{2}{\pi}} \frac{2^{2m}}{\sqrt{2m}} = \frac{4^m}{\sqrt{\pi m}}$, то есть a_n растет экспоненциально с ростом n. Следовательно переборный алгоритм имеет экспоненциальную сложность.

Теорема. *Для нахождения оптимального порядка умножения n матриц существует алгоритм (типа динамического программирования) с числом операций (арифметических и сравнений чисел) $O(n^3)$.*

Доказательство. Пусть на вход поступает набор чисел $(m_0, m_1, \ldots, m_n)$. Введем такие подзадачи B_{ij}: найти оптимальный порядок вычислений и наименьшее число k_{ij} умножений элементов для произведения $A_i \times A_{i+1} \times \ldots \times A_j$, $(1 \leqslant i \leqslant j \leqslant n)$. Очевидно, $k_{ii} = 0$ для всех i, и общее число подзадач B_{ij} есть $O(n^2)$.

Утверждение. *Если* $1 \leqslant i < j \leqslant n$, *то*

$$k_{ij} = \min\{k_{i,i+s} + k_{i+s+1,j} + m_{i-1}m_{i+s}m_j\}, () \tag{}$$

где минимум берется по всем s таким, что $0 \leqslant s \leqslant j - i - 1$.

Доказательство. Если последняя операция умножения делит произведение $A_i \cdot A_2 \cdot \ldots \cdot A_j$ на 2 произведения $(A_i \cdot \ldots A_{i+s}) \cdot (A_{i+s+1} \cdot \ldots \cdot A_j)$, то для получения минимального числа операций надо использовать минимальное число операций в обеих скобках, то есть всего $k_{i,i+s} + k_{i+s+1,j}$ операций. После вычисления этих произведений надо еще перемножить 2 матрицы размеров $m_{i-1} \times m_{i+s}$ и $m_{i+s} \times m_j$. Получаем общее число операций, стоящее в ( ) в фигурных скобках. Теперь остается заметить, что для минимизации общего числа умножений достаточно перебором выбрать оптимальное место для последней операции. Утверждение доказано.

Будем решать подзадачи B_{ij} ярусами, относя к ярусу t все подзадачи с $j - i = t$. Рассмотрим последовательно $t = 0, 1, 2, \ldots, n-1$. При $t = 0$ получим подзадачи B_{ij}, для которых $k_{ii} = 0$ и скобки вообще не надо расставлять. При решении очередной задачи Bij с $j - i = t$ воспользуемся утверждением. При этом легко видеть, что справа в ( ) будут использоваться результаты подзадач ярусов $t_1 < t$, которые уже решены. Тогда для вычисления k_{ij} по формуле ( ) достаточно сделать $2(j - i)$ умножений, $2(j - i)$ сложений и $j - i - 1$ сравнений. Общее число операций для вычисления одного k_{ij} не превосходит $O(n)$, а для вычисления всех k_{ij} — не превосходит $O(n^3)$ (поскольку общее число подзадач B_{ij} есть $O(n^2)$). При вычислении k_{ij} указанным способом мы находим и то s, для которого достигается минимум в ( ). Если мы для всех (i, j) будем фиксировать это s, то сможем быстро оптимально расставить скобки в задаче B_{1n} (в исходной задаче), разбивая каждое произведение последовательно оптимальным образом на 2 произведения. Теорема доказана.

Задача о кратчайших путях

Пусть G —полный ориентированный граф с n вершинами $v_1, v_2, \ldots, v_n$. Пусть каждой дуге (v_i, v_j) сопоставлено действительное число $d_{ij} \geqslant 0$, либо $d_{ij} = +\infty$. Число d_{ij} трактуется как расстояние из v_i в v_j "напрямую". Длина ориентированного пути из v_i в v_j определяется как сумма длин всех дуг этого пути (она равна $+\infty$, если хотя бы одно слагаемое равно $+\infty$). Кратчайшее расстояние d_{ij} из v_i в v_j определим как минимум длин по всем ориентированным путям из v_i в v_j. Соответствующий путь будем называть кратчайшим. Рассмотрим следующую задачу о кратчайших путях.

Вход: матрица $D = \|d_{ij}\|$ порядка n (считаем, что $d_{ii} = 0$ для всех i).

Требуется: построить матрицу $\bar{D} = \|\bar{d}_{ij}\|$ кратчайших расстояний.

Отметим, что аналогичную задачу для неполного графа можно свести к задаче о полном графе, положив $d_{ij} = +\infty$ для несуществующих дуг. Если $d_{ij} = d_{ji}$ для всех i, j, то граф G можно считать неориентированным.

Алгоритм для указанной задачи, основанный на переборе всех возможных путей, имеет не менее, чем экспоненциальную сложность, поскольку из v_i в v_j существует более $(n-2)!$ путей без повторяющихся вершин.

Теорема. *Существует алгоритм для задачи о кратчайших путях, строящий по матрице D матрицу $\bar{D}$, с числом операций (арифметических и сравнений чисел) $O(n^3)$, где n — число вершин в графе.*

Доказательство. Введем следующие подзадачи: для каждой пары i, j и натурального $k \geqslant 0$ вычислить $d_{ij}^{(k)}$ — минимальную длину среди всех ориентированных путей из v_i в v_j, проходящих, кроме v_i и v_j, только по вершинам $v_1, v_2, \ldots, v_k$ (возможно только по части или напрямую из v_i в v_j). Если $k = 0$, то разрешается только переход из v_i в v_j "напрямую". Пусть $D^{(k)} = \|d_{ij}^{(k)}\|$. Тогда $D^{(0)} = D$ и $D^{(n)} = \bar{D}$. Будем последовательно вычислять $D^{(1)}, D^{(2)}, \ldots, D^{(n)}$.

Утверждение. *Для любых i, j и $k > 0$*

$$d_{ij}^{(k)} = min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}).$$

Доказательство. Все пути из v_i в V_j, использующие только вершины $v_1, v_2, \ldots, v_k$, распадаются на 2 множества A и B — не проходящих через v_k и проходящих через v_k. Минимальная длина путей в A равна $d_{ij}^{(k-1)}$ по определению. Каждый путь из B разбивается на 2 части: путь B_1 из v_i в v_k по вершинам $v_1, v_2, \ldots, v_{k-1}$ и путь B_2 из v_k в v_j по вершинам $v_1, v_2, \ldots, v_{k-1}$. Минимальная длина пути в B_1 равна

$d_{ik}^{(k-1)}$, а в B_2 — $d_{kj}^{(k-1)}$. Сравнивая $d_{ij}^{(k-1)}$ и $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$, получаем $d_{ij}^{(k)}$. Утверждение доказано.

Замечание. Вычисляя $d_{ij}^{(k)}$ описанным способом, мы, в частности, узнаем, использовать v_k или нет.

Таким образом, для вычисления $D^{(k)}$ по $D^{(k-1)}$ достаточно n^2 сложений и n^2 сравнений чисел, а для вычисления $D^{(1)}, D^{(2)}, \ldots, D^{(n)} = \bar{D}$ по заданной матрице $D = D^{(0)}$ достаточно n^3 сложений и n^3 сравнений. Теорема доказана.

Метод "разделяй и властвуй". Теорема о рекуррентном неравенстве.

Другой рекуррентный метод построения быстрых алгоритмов — это метод, который называют "разделяй и властвуй". В нем также решение задачи сводится к решению подзадач, но, в отличие от метода динамического программирования, размерность подзадач отличается от размерности задачи не **на** константу, а **в** константу раз и подзадачи решаются независимо друг от друга. Для получения оценок сложности таких алгоритмов используется следующая теорема.

Теорема (о рекуррентном неравенстве). *Пусть $L(n)$ — монотонно неубывающая функция натурального параметра n. Пусть c — натуральное число, $c \geqslant 2$, и a, b, α - действительные константы, причем $a > 0$, и пусть для всех $n = c^k$, где k - любое натуральное число ($k = 1, 2, 3, \dots$), выполняется неравенство*

$$L(n) \leqslant aL(\frac{n}{c}) + bn^{\alpha}. \tag{6}$$

Тогда при стремлении n к бесконечности для всех n выполняется

$$L(n) = \begin{cases} O(n^{\alpha}), & \text{если } \alpha > \log_c a, \\ O(n^{\log_c a}), & \text{если } \alpha < \log_c a, \\ O(n^{\alpha} \log n), & \text{если } \alpha = \log_c a. \end{cases} \tag{7}$$

Доказательство. Пусть $n = c^k$, где $k = 1, 2, 3, \dots$. Тогда, применяя несколько раз (6), получаем

$$L(n) \leqslant aL(\frac{n}{c}) + bn^{\alpha} \leqslant a(aL(\frac{n}{c^2}) + b\left(\frac{n}{c}\right)^{\alpha}) + bn^{\alpha} =$$

$$= bn^{\alpha} + ab\left(\frac{n}{c}\right)^{\alpha} + a^2 L\left(\frac{n}{c^2}\right) \leqslant$$

$$\leqslant bn^{\alpha} + b\left(\frac{a}{c^{\alpha}}\right) n^{\alpha} + a^2(aL\left(\frac{n}{c^3}\right) + b\left(\frac{n}{c^2}\right)^{\alpha}) =$$

$$= bn^{\alpha} + bn^{\alpha}\left(\frac{a}{c^{\alpha}}\right) + bn^{\alpha}\left(\frac{a}{c^{\alpha}}\right)^2 + a^3 L\left(\frac{n}{c^3}\right) \leqslant$$

$$\leqslant \dots \leqslant bn^{\alpha} + bn^{\alpha}\left(\frac{a}{c^{\alpha}}\right) + \dots + bn^{\alpha}\left(\frac{a}{c^{\alpha}}\right)^{k-1} + a^k L\left(\frac{n}{c^k}\right).$$

Пусть $d = \max(b, L(1))$. Так как $\frac{n}{c^k} = 1$, то

$$L(n) \leqslant dn^{\alpha}\left(1 + \frac{a}{c^{\alpha}} + \left(\frac{a}{c^{\alpha}}\right)^2 + \dots + \left(\frac{a}{c^{\alpha}}\right)^{k-1}\right) + da^k =$$

$$= dn^\alpha \left(1 + \frac{a}{c^\alpha} + \left(\frac{a}{c^\alpha}\right)^2 + \cdots + \left(\frac{a}{c^\alpha}\right)^k\right).$$

Рассмотрим 3 случая:

$$1) \quad \log_c a < \alpha \Longrightarrow \frac{a}{c^\alpha} < 1 \Longrightarrow L(n) \leqslant dn^\alpha const = O(n^\alpha);$$

$$2) \quad \log_c a > \alpha \Longrightarrow \frac{a}{c^\alpha} > 1 \Longrightarrow$$

$$\Longrightarrow L(n) \leqslant dn^\alpha \left(\frac{a}{c^\alpha}\right)^k \underbrace{\left(1 + \frac{c^\alpha}{a} + \left(\frac{c^\alpha}{a}\right)^2 + \cdots + \left(\frac{c^\alpha}{a}\right)^k\right)}_{\leqslant const} \Longrightarrow \tag{8}$$

$$\Longrightarrow L(n) \leqslant da^k const = O(a^{\log_c n}) = O(n^{\log_c a}) \text{ (так как } a^{\log_c n} = n^{\log_c a});$$

$$3) \quad \log_c a = \alpha \Longrightarrow a = c^\alpha \Longrightarrow L(n) \leqslant dn^\alpha(k+1) = dn^\alpha(1 + \log_c n) =$$

$$= O(n^\alpha \log n).$$

Пусть теперь n — любое. Тогда существует натуральное k, такое, что $c^k < n \leqslant c^{k+1}$. Рассмотрим 3 случая, учитывая, что по условию $L(n)$ — неубывающая функция:

$$1) \quad \alpha > \log_c a \Longrightarrow L(n) \leqslant L(c^{k+1}) \leqslant p(c^{k+1})^\alpha = pc^\alpha (c^k)^\alpha \leqslant pc^\alpha n^\alpha =$$

$$= O(n^\alpha);$$

$$2) \quad \alpha < \log_c a \Longrightarrow L(n) \leqslant L(c^{k+1}) \leqslant p(c^{k+1})^{\log_c a} = pc^{\log_c a}(c^k)^{\log_c a} \leqslant \tag{9}$$

$$\leqslant pan^{\log_c a} = O(n^{\log_c a});$$

$$3) \quad \alpha = \log_c a \Longrightarrow L(n) \leqslant L(c^{k+1}) \leqslant pc^\alpha 2k(c^k)^\alpha \leqslant pc^\alpha 2n^\alpha \log_c n =$$

$$= O(n^\alpha) \log n.$$

Теорема доказана.

Часть 2. Схемы

§ 6. Реализация некоторых "арифметических" систем ФАЛ в классе СФЭ

Рассмотрим теперь некоторые "арифметические" системы ФАЛ и построим реализующие их СФЭ.

Системы S_n, M_n и W_n, состоящие из $(n+1)$, $2n$ и n ФАЛ от БП $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n)$ соответственно, такие, что

$$|S_n(x,y)| = |x| + |y|,\ |M_n(x,y)| = |x| \cdot |y|,$$

и, если $|x| \geqslant |y|$, то

$$|W_n(x,y)| = |x| - |y|,$$

называются (функциональным) *сумматором, умножителем* и *вычитателем порядка* n соответственно.

Система C_n, состоящая из $(n+1)$ ФАЛ от БП $x = (x_1, \ldots, x_{2^n})$, такая, что значение $|C_n(x)|$ равно числу единиц в наборе x, называется (функциональным) *счетчиком порядка* n.

В [8] приведен сумматор порядка n, имеющий сложность $9n - 5$. Мы построим такой же сумматор несколько иначе.

Теорема 6.1. *Существует схемный сумматор порядка* n, *имеющий сложость* $9n - 5$.

Доказательство. Для $n = 1$ сумматор Σ_1 показан на рис. 4.4. На рис. 6.1.а показана СФЭ Σ' сложности 9, которая реализует систему ФАЛ S' от БП x, y и q' такую, что

$$|S'(x,y,q')| = x + y + q',$$

т. е. реализует сложение трех одноразрядных чисел. Действительно, на выходе z_2 СФЭ Σ' реализуется ФАЛ $x \oplus y \oplus q'$, а на выходе z_1 единица появляется только тогда, когда либо $x = y = 1$, либо $x \oplus y = q' = 1$, т. е. на выходе z_1 в СФЭ Σ' реализуется ФАЛ

$$xy \vee (x \oplus y)q' = xy \vee xq' \vee yq'.$$

Схемный сумматор Σ_n порядка n с входными БП $x_1, \ldots, x_n, y_1, \ldots, y_n$ и выходными БП $z_0, z_1, \ldots, z_n$ можно построить из сумматора Σ_{n-1} порядка $(n-1)$ с входными БП $x_2, \ldots, x_n, y_2, \ldots, y_n$ и выходными БП $z_1', z_2, \ldots, z_n$, а также одной СФЭ Σ' так, как это показано на рис. 6.2. Заметим, что переход от сумматора Σ_{n-1} к сумматору Σ_n моделирует

сложение n-разрядных чисел в два этапа: на первом этапе складываются числа, образуемые $(n-1)$ младшими разрядами, а на втором этапе складываются старшие разряды и перенос, возникший на первом этапе. Очевидно, что

$$L(\Sigma_n) = 9n - 5.$$

Теорема доказана.

Следствие. $L(S_n) \leqslant 9n - 5$.

Теорема 6.2. *Существует схемный вычитатель порядка n, имеющий сложность не больше, чем* $11n - 5$.

Доказательство. Заметим, что

$$|\overline{x}| = 2^n - 1 - |x|,$$

где $x = (x_1, \ldots, x_n)$, $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$, и поэтому

$$W_n(x, y) = |x| - |y| = 2^n - 1 - (2^n - 1 - |x| + |y|) = \overline{S}_n(\overline{x}, y)$$

Следовательно, схемный вычитатель порядка n может быть получен из схемного сумматора порядка n в результате инвертирования входов его первого слагаемого, а также всех его выходов, и имеет сложность не больше, чем $11n - 5$.

Теорема доказана.

Рассмотрим теперь сложность умножителя M_n для умножения двух неотрицательных n-разрядных двоичных чисел $X = |(x_1, x_2, \ldots, x_n)|$ и $Y = |(y_1, y_2, \ldots, y_n)|$. Так как $X < 2^n$ и $Y < 2^n$, то $XY < 2^{2n}$ и для представления результата требуется не более 2n выходов. Обозначим через $L_M(n)$ наименьшее возможное число элементов в умножителе M_n. Очевидно, что $L_M(1) = 1$, так как умножение 1-разрядных чисел осуществляет элемент конъюнкция.

Утверждение. *Существует СФЭ для умножения n-разрядного числа X на 1-разрядное число y с числом элементов n.*

Действительно, если $X = |(x_1, x_2, \ldots, x_n)|$ и $Xy = Z = |(z_1, z_2, \ldots, z_n)|$, то $z_i{=}x_i y$ для всех $i = 1, 2, \ldots, n$.

При умножении двух n-разрядных чисел X и Y "в столбик" надо n раз умножить X на 1-разрядное число (всего n^2 конъюнкций) и затем $n - 1$ раз сложить числа длиной не более 2n. Такой алгоритм (схема) имеет сложность по порядку n^2. Следующая теорема показывает, что алгоритм умножения "в столбик" не оптимален по порядку.

Теорема 6.4 (Карацуба А. А. [4]). *Существует такая константа c, что $L_M(n) \leqslant cn^{\log_2 3}$ для всех n.*

Докажем сначала несколько вспомогательных лемм.

Лемма 6.1. *Существует константа c_1 такая, что $L_M(n+1) \leqslant L_M(n) + c_1 n$ для всех n.*

Доказательство. Пусть требуется перемножить два $(n+1)$-разрядных числа $X_1 = |(x_0, x_1, \ldots, x_n)|$ и $Y_1 = |(y_0, y_1, \ldots, y_n)|$. Тогда обозначим $|(x_1, x_2, \ldots, x_n)| = X$ и $|(y_1, y_2, \ldots, y_n)| = Y$. При этом $X_1 = x_0 2^n + X$, $Y_1 = y_0 2^n + Y$ и

$$X_1 Y_1 = x_0 y_0 2^{2n} + (x_0 Y + y_0 X) 2^n + XY.$$

Поэтому для вычисления $X_1 Y_1$ достаточно использовать умножитель M_n для вычисления XY, $2n$ элементов для вычисления $x_0 Y$ и $y_0 X$, 1 элемент для вычисления $x_0 y_0$ и 3 сумматора порядка не более $2n+2$, так как $X_1 Y_1 < 2^{2n+2}$. Отметим, что числа $x_0 y_0$ и $x_0 Y + y_0 X$ надо подавать на сумматоры со сдвигом, одновременно подавая на младшие разряды 0. При этом 0 можно предварительно получить подсхемой с 2 элементами, реализующей $x_0 \bar{x}_0 = 0$.Так как сложность каждого сумматора можно сделать не более $9(2n+2)$, а сложность M_n равной $L_M(n)$, то сложность полученной схемы будет не больше чем $L_M(n) + c_1 n$ для некоторой константы c_1. Лемма доказана.

Лемма 6.2 (основная). *Существует константа c_2 такая, что $L_M(2n) \leqslant 3L_M(n) + c_2 n$ для всех n.*

Доказательство. Пусть нужно перемножить два $2n$-разрядных числа X и Y. Разобьем их на части, содержащие по n разрядов. Тогда получим $X = X_1 2^n + X_2$, $Y = Y_1 2^n + Y_2$. Отсюда

$$XY = X_1 Y_1 2^{2n} + (X_1 Y_2 + X_2 Y_1) 2^n + X_2 Y_2 =$$

$$= X_1 Y_1 2^{2n} + [(X_1 + X_2)(Y_1 + Y_2) - X_1 Y_1 - X_2 Y_2] 2^n + X_2 Y_2.$$

Так как $X_1 Y_2 + X_2 Y_1 \geqslant 0$, то при вычитании в квадратной скобке не возникает отрицательных чисел. Таким образом, схему для умножения XY можно построить, используя 2 оптимальных умножителя M_n с числом элементов $L_M(n)$ в каждом для вычисления $X_1 Y_1$ и $X_2 Y_2$, умножитель M_{n+1} с числом элементов $L_M(n+1)$ для вычисления $(X_1 + X_2)(Y_1 + Y_2)$, 4 сумматора порядка не более $4n$ (так как $XY < 2^{4n}$) и 2 вычитателя порядка $2n+2$. В некоторых сумматорах опять на младшие разряды надо подавать 0, который реализуем подсхемой с 2 элементами: $0 = x\bar{x}$, где x - любая входная переменная. Для построенной схемы M_{2n} с учетом леммы 6.1 получим для некоторых констант c и c_2:

$$L(M_{2n}) \leqslant 2L_M(n) + L_M(n+1) + cn \leqslant 3L_M(n) + c_1 n + cn = 3L_M(n) + c_2 n$$

.

Лемма 6.3. *Существует такая константа c_3, что для любого натурального k выполняется $L_M(2^k) \leqslant c_3 3^k$.*

Доказательство. Положим $f(k) = \frac{L_M(2^k)}{3^k}$. Тогда из предыдущей леммы имеем

$$\frac{L_M(2^k)}{3^k} \leqslant \frac{L_M(2^{k-1})}{3^{k-1}} + \frac{c_2}{3}(\frac{2}{3})^{k-1}$$

и

$$f(k) \leqslant f(k-1) + \frac{c_2}{3}(\frac{2}{3})^{k-1} \leqslant f(k-2) + \frac{c_2}{3}(\frac{2}{3})^{k-2} + \frac{c_2}{3}(\frac{2}{3})^{k-1} \leqslant$$

$$\leqslant \ldots \leqslant f(1) + \frac{c_2}{3}[\frac{2}{3} + (\frac{2}{3})^2 + \cdots + (\frac{2}{3})^{k-1}] \leqslant c_3$$

для некоторой константы c_3, поскольку сумма в квадратных скобках не превосходит сумму 2 бесконечно убывающей геометрической прогрессии с первым членом $\frac{2}{3}$ и знаменателем $\frac{2}{3}$. Таким образом $\frac{L_M(2^k)}{3^k} \leqslant c_3$ и $L_M(2^k) \leqslant c_3 3^k$.

Доказательство теоремы Карацубы. Пусть n — любое натуральное число и $n > 1$. Тогда существует натуральное k такое, что $2^{k-1} < n \leqslant 2^k$. Для умножения n-разрядных чисел будем использовать схему M_{2^k} с числом элементов $L_M(2^k)$, подавая на старшие $2^k - n$ входных разрядов обоих сомножителей 0, предварительно реализованный подсхемой из 2 элементов. Тогда имеем

$$L_M(n) \leqslant L_M(2^k) + 2 \leqslant c_3 3^k + 2 = 3c_3 3^{k-1} + 2 =$$

$$= 3c_3 2^{(k-1)\log_2 3} + 2 < 3c_3 n^{\log_2 3} + 2 \leqslant cn^{\log_2 3}$$

для некоторой константы c.

В заключение отметим, что существует алгоритм Шенхаге и Штрассена для умножения двух n-разрядных чисел, дающий оценку $L_M(n) \leqslant cn \log n \log\log n$, где c — некоторая константа и логарифмы можно считать двоичными.

Алгоритм Тоома

Нам потребуется следующий известный факт о многочленах.

Утверждение (интерполяционная формула Лагранжа). *Пусть* $P(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$ — *произвольный полином степени* n, *значения которого* $P(d_m)$ *известны в* $n+1$ *различных точках* $d_1, d_2, \ldots, d_{n+1}$. *Тогда существуют такие константы* α_{km}, *зависящие только от* $d_1, d_2, \ldots, d_{n+1}$, *что*

$$c_q = \sum_{i=1}^{n+1} \alpha_{qm} P(d_m), q = 0, 1, \ldots, n.$$

При этом, если все d_m *рациональны, то и все* α_{km} *рациональны.*

Теорема. *Для любого фиксированного* $\varepsilon > 0$ *выполняется* $M(n) = O(n^{1+\varepsilon})$.

Доказательство. Зафиксируем натуральное $k \geqslant 2$ и рассмотрим следующий алгоритм Тоома для умножения n-разрядных двоичных чисел A и B. Если $k^{m-1} < n \leqslant k^m$, то увеличим разрядность до k^m, приписывая спереди нули. Для $n = k^m$ поступаем следующим образом. Режем A и B на k кусков длины k^{m-1}. Пусть $A = (A_{k-1}A_{k-2}\ldots A_1 A_0)_2$ и $B = (B_{k-1}B_{k-2}\ldots B_1 B_0)_2$. Рассмотрим многочлены $f(x) = A_{k-1}x^{k-1} + A_{k-2}x^{k-2} + \ldots + A_1 x + A_0$ и $g(x) = B_{k-1}x^{k-1} + B_{k-2}x^{k-2} + \ldots + B_1 x + B_0$. Тогда $A = f(2^{k^{m-1}})$, $B = g(2^{k^{m-1}})$ и искомое $C = A \cdot B = f(2^{k^{m-1}}) \cdot g(2^{k^{m-1}}) = h(2^{k^{m-1}})$, где $h(x) = f(x) \cdot g(x)$. Заметим, что $h(x)$ — многочлен степени $2k-2$. Алгоритм состоит из следующих шагов.

1.Вычисляем $f(x_m)$ и $g(x_m)$, где $x_1, x_2, , \ldots, x_{2k-1}$ —любые фиксированные целые точки (например, $x_m = 0, \pm 1, \pm 2, \ldots, \pm(k-1)$).

2.Вычисляем $h(x_m) = f(x_m)g(x_m)$ тем же алгоритмом для $n = k^{m-1}$ (мы уточним это ниже).

3. По формуле ( ) вычисляем коэффициенты c_q $(q = 0, 1, \ldots, 2k-1)$ многочлена $h(x)$.

4. Вычисляем $h(2^{k^{m-1}}) = C = AB$.

Оценим теперь сложность каждого шага. Отметим, что $k^m = n$, $k^{m-1} = \frac{n}{k}$ и k — константа.

Шаг 1. На этом шаге вычисляем $f(x_m)$ и $g(x_m)$ непосредственно по формулам многочленов, выполняя все операции "в столбик". При этом, так как все x_m — константы и k — константа, вычисление всех $x_m^l (m = 1, 2, \ldots, 2k-1; l = 2, 3, \ldots, k-1)$ требует константного числа битовых операций и длины всех получаемых чисел ограничены константой (зависящей от k, но не зависящей от n). Поэтому вычисление всех одночленов $A_l x_m^l$ требует $O(n)$ битовых операций и длины

получаемых чисел не превосходят $\frac{n}{k} + const$. Аналогично для $B_l x_m^l$. Складывая эти одночлены (k —константа), получаем, что вычисление всех значений $f(x_m)$ и $g(x_m)$ требует $O(n)$ битовых операций и длина всех этих значений не превосходит $\frac{n}{k} + const$.

Шаг 2. На этом шаге нам надо $2k-1$ раз перемножить числа длины не более $\frac{n}{k}+const$. Пусть C и D — 2 таких числа, и $C = (C_1C_0)_2$, $D = (D_1D_0)_2$, где длина чисел C_0 и D_0 равна $\frac{n}{k}$. Тогда $C \cdot D = (C_1 \cdot 2^{\frac{n}{k}} + C_0) \cdot (D_1 \cdot 2^{\frac{n}{k}} + D_0) = C_1D_1 \cdot 2^{\frac{2n}{k}} + (C_1D_0 + C_0D_1) \cdot 2^{\frac{n}{k}} + C_0D_0$. Будем вычислять C_1D_1, C_1D_0, C_0D_1 "в столбик", а C_0D_0 рекурсивно тем же алгоритмом, если длина C_0 и D_0, равная k^{m-1}, больше 1. Если же $k^{m-1} = 1$, то C_0D_0 также вычисляем "в столбик". Пусть $L_T(n)$ — битовая сложность алгоритма Тоома (в худшем случае) для умножения чисел длины n. Тогда число операций на шаге 2 не превосходит $(2k-1)L_T(\frac{n}{k}) + O(n)$, и получающиеся числа имеют длину $O(n)$.

Шаг 3. Так как все x_m — целые, то все α_{qm} в формуле ( ) рациональные. Пусть β — их общий знаменатель и $\alpha_{qm} = \frac{\beta_{qm}}{\beta}$. Тогда все β_{qm} — целые и $c_q = \frac{1}{\beta}\sum_{m=1}^{2k-1} \beta_{qm}h(x_{km})$. Так как k — константа, все β_{qm} — константы и длина всех чисел $h(x_{km})$ есть $O(n)$, то для вычисления всех сумм $\sum_{m=1}^{2k-1} \beta_{qm}h(x_{km}), q = 0, 1, \ldots, 2k-1$, требуется $O(n)$ битовых операций и при этом получаются числа длины $O(n)$. Так как β — константа, то вычисление всех c_q (которые заведомо должны быть целыми, как коэффициенты многочлена $h(x) = f(x)g(x)$) требует $O(n)$ битовых операций (делим "в столбик"), и все c_q имеют длину $O(n)$.

Шаг 4. Вычисление $h(2^{k^{m-1}})$ сводится к сложению чисел C_q, сдвинутых влево не более, чем на n разрядов. Так как чисел C_q константное количество, то вычисление $h(2^{k^{m-1}}) = C = AB$ требует $O(n)$ битовых операций.

Для общего числа $L_T(n)$ битовых операций в описанном алгоритме (при $n = k^m$) имеем

$$L_T(n) \leqslant (2k-1)L_T(\frac{n}{k}) + O(n).$$

Тогда по теореме о рекуррентном неравенстве для всех n получаем

$$L_T(n) = O(n^{\log_k(2k-1)}).$$

С ростом k имеем

$$\log_k(2k-1) = 1 + \log_k(2 - \frac{1}{k}) \longrightarrow 1.$$

Поэтому для любого $\varepsilon > 0$ можно выбрать k так, что $\log_k(2k-1) < 1+\varepsilon$ и $L_T(n) = O(n^{1+\varepsilon})$. Теорема доказана.

Замечание. Еще более быстрым является алгоритм умножения чисел Шенхаге и Штрассена, битовая сложность которого равна $O(n \log n \log\log n)$.

Алгоритм Штрассена для умножения матриц

Рассмотрим задачу умножения двух квадратных матриц $A = \|a_{ij}\|$ и $B = \|b_{kl}\|$ порядка n. Пусть $A \cdot B = C = \|c_{rs}\|$. Тогда по определению $c_{rs} = \sum_{p=1}^{n} a_{rp} b_{ps}$. В качестве входа мы будем рассматривать все значения a_{ij} и b_{kl}, считая их "неделимыми", то есть мы воспринимаем их как единое целое и не можем работать с какими-либо их частями. В качестве операций будем рассматривать 4 арифметические операции, которые могут применяться как к исходным переменным a_{ij}, b_{kl}, так и к уже построенным выражениям. Наша задача состоит в получении всех выражений для c_{rs}. Сложностью алгоритма будет считаться число арифметических операций.

Обычный алгоритм умножения матриц ("строчка на столбец") требует n^3 умножений и $n^2(n-1)$ сложений, то есть порядка n^3 операций.

Теорема (Штрассен). *Для умножения двух матриц порядка n существует алгоритм с числом арифметических операций $O(n^{\log_2 7})$.*

Доказательство. Опишем такой алгоритм. Если n— не степень двойки и ближайшая к n сверху степень двойки есть 2^k, то расширим данные матрицы A и B до матриц A' и B' порядка 2^k так, чтобы в левых верхних углах матриц A' и B' стояли, соответственно, A и B, а остальные элементы были равны 0. Тогда, если $A' \cdot B' = C'$, то легко видеть, что в C' в левом верхнем углу стоит матрица $C = A \cdot B$, а остальные элементы равны 0. Поэтому для вычисления $C = A \cdot B$ достаточно перемножить матрицы A' и B' порядка 2^k. Пусть далее $n = 2^k$ — степень двойки и A, B — матрицы порядка $n = 2^k$. Разрежем каждую из матриц A и B, а также искомую матрицу $C = A \cdot B$, на 4 квадратных блока размера $\frac{n}{2} \times \frac{n}{2}$:

$$A = \begin{Vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{Vmatrix}, B = \begin{Vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{Vmatrix}, C = \begin{Vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{Vmatrix}.$$

Из алгебры известно, что в этом случае

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}, \quad C_{12} = A_{11}B_{12} + A_{12}B_{22},$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}, \quad C_{22} = A_{21}B_{12} + A_{22}B_{22}.$$

Таким образом, вычисление матрицы C сводится к 8 умножениям матриц порядка $\frac{n}{2}$ (и нескольким сложениям). Идея Штрассена состоит в замене 8 умножений на 7 (сравните с алгоритмом Карацубы). Рас-

смотрим следующие 7 произведений:

$$
\begin{aligned}
D_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), & D_5 &= (A_{11} + A_{12})B_{22},\\
D_2 &= (-A_{11} + A_{21})(B_{11} + B_{12}), & D_6 &= A_{22}(-B_{11} + B_{21}),\\
D_3 &= (A_{12} - A_{22})(B_{21} + B_{22}), & D_7 &= (A_{21} + A_{22})B_{11}.\\
D_4 &= A_{11}(B_{12} - B_{22}),
\end{aligned}
$$

Раскрывая скобки и приводя подобные члены, можно проверить, что

$$
\begin{aligned}
C_{11} &= D_1 + D_3 - D_5 + D_6, & C_{12} &= D_4 + D_5,\\
C_{21} &= D_6 + D_7, & C_{22} &= D_1 + D_2 + D_4 - D_7.
\end{aligned}
$$

Таким образом, умножение матриц порядка n сводится к 7 умножениям матриц порядка $\frac{n}{2}$ и нескольким сложениям матриц порядка $\frac{n}{2}$. Если $n = 2^k$, то этот процесс можно продолжить рекурсивно. Если же $n = 1$, то для умножения матриц порядка 1 требуется всего 1 умножение элементов. Пусть $L(n)$ — число арифметических операций в описанном алгоритме. Так как сложение двух матриц порядка $\frac{n}{2}$ требует $O(n^2)$ операций, то для $n = 2^k (k = 1, 2, 3, \dots)$ получаем рекуррентное неравенство

$$L(n) \leqslant 7L(\frac{n}{2}) + O(n^2).$$

По теореме о рекуррентном неравенстве отсюда получаем $L(n) = O(n^{\log_2 7})$. Теорема доказана.

Замечание. Ожидается, что для умножения матриц порядка n существует алгоритм с числом арифметических операций $O(n^{2+\varepsilon})$ для любого фиксированного $\varepsilon > 0$ (сравните с алгоритмом Тоома), однако пока (середина 2001 года) наилучшей оценкой является $O(n^{2.38})$ [ ].

Алгоритмы умножения 0-1-матриц

Пусть матрицы A и B состоят только из 0 и 1 и требуется вычислить $C = A \cdot B$, где все элементы c_{rs} должны быть представлены в двоичной системе. В качестве операций разрешим только любые битовые операции над двумя переменными.

Теорема. *Для вычисления (обычного) произведения двух 0-1-матриц порядка n существует алгоритм с числом битовых операций $O(n^{\log_2 7} \log^2 n)$.*

Лемма. *Если в исходных матрицах A и B порядка n все элементы имеют двоичную длину не более k (включая знак), то в алгоритме Штрассена для вычисления AB все возникающие числа имеют двоичную длину не более $2k + 4\lceil \log_2 n \rceil$.*

Доказательство леммы. При формировании подзадач вычисления $D_1 - D_7$ в алгоритме Штрассена происходит сложение (или вычитание) не более чем двух матриц. Поэтому модули всех чисел не более чем удваиваются, то есть добавляется не более одного разряда. При переходе от размерности n к размерности 1 подзадачи формируются $\lceil \log_2 n \rceil$ раз. Следовательно, в подзадачах размерности 1 все числа имеют длину не более $k + \lceil \log_2 n \rceil$. Для подзадач размерности 1 алгоритм Штрассена производит обычное умножение. При этом длина получающихся чисел не превосходит $2k+2\lceil \log_2 n \rceil$. При вычислении C_{rs} по результатам подзадач $D_1 - D_7$ складываются (вычитаются) не более чем по 4 матрицы. При этом максимальные модули чисел возрастают не более чем в 4 раза, то есть добавляется не более чем по 2 разряда. Поскольку обратных шагов также $\lceil \log_2 n \rceil$, то все получаемые числа имеют длину не более $2k + 2\lceil \log_2 n \rceil + 2\lceil \log_2 n \rceil = 2k + 4\lceil \log_2 n \rceil$. Лемма доказана.

Доказательство теоремы. Применим для вычисления AB алгоритм Штрассена. По условию в исходных матрицах A и B все элементы имеют длину 2 (включая знак). Тогда по лемме все возникающие в алгоритме числа будут иметь длину не более $4 + 4\lceil \log_2 n \rceil = O(\log n)$. Так как в алгоритме Штрассена используются только сложение, вычитание и умножение, то любая арифметическая операция в алгоритме Штрассена требует $O(\log^2 n)$ битовых операций. Поскольку алгоритм Штрассена использует $O(n^{\log_2 7})$ арифметических операций, то все они потребуют $O(n^{\log_2 7} \log^2 n)$ битовых операций. Теорема доказана.

Замечание 1. Оценку можно улучшить, если использовать быстрые алгоритмы для умножения чисел.

Замечание 2. В этой теореме можно получить оценку $O(n^{2.38})$,

если использовать известный более быстрый алгоритм умножения матриц.

Рассмотрим теперь операцию булевского умножения 0-1-матриц.

Определение. *Пусть* $A = \|a_{ij}\|$ *и* $B = \|b_{kl}\|$ *—две 0-1-матрицы порядка* n. *Булевским произведением* $A \circ B$ *называется матрица* $D = \|d_{rs}\|$ *такая, что*

$$d_{rs} = \bigvee_{p=1}^{n} a_{rp} \cdot b_{ps}$$

для всех r *и* s.

Для булевского умножения матриц нельзя непосредственно применить идею Штрассена, так как в алгоритме Штрассена есть вычитание, а у дизъюнкции нет обратной операции. Несмотря на это, справедлива следующая теорема.

Теорема. *Булевское произведение* $D = A \circ B$ *двух 0-1-матриц* A *и* B *порядка* n *можно вычислить с числом битовых операций* $O(n^{\log_2 7} \log^2 n)$.

Доказательство. Мы опишем соответствующий алгоритм, который основан на идее "расширения модели" (см. алгоритм Тоома). Вместо вычисления $D = A \circ B$ мы вычислим сначала обычное произведение $C = AB$. При этом отметим следующую связь между D и C:

$$d_{rs} = 1 \Longleftrightarrow c_{rs} > 0.$$

По предыдущей теореме для вычисления $C = \|c_{rs}\|$ существует алгоритм с числом битовых операций $O(n^{\log_2 7} \log^2 n)$. После этого в каждом c_{rs} достаточно взять дизъюнкцию всех разрядов (исключая знак), чтобы вычислить d_{rs}. Поскольку $0 \leqslant c_{rs} \leqslant n$, то длина каждого c_{rs} не превосходит $O(\log n)$ и на вычисление всех d_{rs} из c_{rs} потребуется $O(n^2 \log n)$ битовых операций. Общее число битовых операций будет $O(n^{\log_2 7} \log^2 n) + O(n^2 \log n) = O(n^{\log_2 7} \log^2 n)$. Теорема доказана.

Замечание. См. замечания к предыдущей теореме.

Транзитивное замыкание графов

Дано: ориентированный граф G в виде матрицы $A = \|a_{ij}\|$, где $a_{ij} = 1$, если в G есть дуга из v_i в v_j, и $a_{ij} = 0$, если такой дуги нет ($a_{ii} = 0$ для всех i).

Требуется: построить матрицу $B = \|b_{ij}\|$, такую, что $b_{ij} = 1$, если есть ориентированный путь из v_i в v_j, и $b_{ij} = 0$, если такого пути нет (в частности, $b_{ii} = 1$ для всех i).

Определение. *Ориентированный граф с матрицей смежности B называется транзитивным замыканием графа G.*

Теорема. *Транзитивное замыкание ориентированного графа с n вершинами можно построить, используя $O(n^{\log_2 7} \log^3 n)$ битовых операций.*

Доказательство. Пусть A — матрица смежности орграфа G и матрица $\bar{A} = \|\bar{a}_{ij}\|$ получается из A заменой всех диагональных элементов на 1. Тогда $\bar{a}_{ij} = 1$ в том и только в том случае, если из v_i в v_j существует ориентированный путь длины (т.е. с числом дуг) не более 1. Пусть $\bar{A}^{\circ k} = \bar{A} \circ \bar{A} \circ \ldots \bar{A}$, где число сомножителей равно k и умножение матриц булевское.

Лемма. *Если $\bar{A}^{\circ k} = \|a^k_{ij}\|$, то $a^k_{ij} = 1$ в том и только в том случае, если в G существует ориентированный путь из v_i в v_j длины не более k.*

Доказательство (индукцией по k). При $k = 1$ утверждение верно. Пусть оно верно при $k = p$, то есть $a^p_{ij} = 1$ тогда и только тогда, когда существует путь из v_i в v_j длины не более p. По определению получаем $A^{\circ(p+1)} = A^{\circ p} \circ \bar{A}$ и $a^{p+1}_{ij} = \bigvee_q a^p_{iq} \circ \bar{a}_{qj}$. Отсюда $a^{p+1}_{ij} = 1$ тогда и только тогда, когда существует вершина v_q такая, что из v_i в v_q существует путь длины не более p, и из v_q в v_j существует путь длины не более 1. Но это условие равносильно тому, что из v_i в v_j существует путь длины не более $p+1$. Таким образом, утверждение леммы верно и при $k = p + 1$. Лемма доказана.

Если в орграфе G из v_i в v_j существует хотя бы один ориентированный путь, то существует такой путь без повторения вершин и, следовательно, длины не более $n-1$, где n — число вершин в G. Поэтому из леммы следует, что $\bar{A}^{\circ k} = B$ при любом $k \geqslant n-1$. Будем вычислять последовательно $\bar{A}, \bar{A}^{\circ 2}, \bar{A}^{\circ 4}, \bar{A}^{\circ 8}, \ldots \bar{A}^{\circ 2^m}$, где $m = \lceil \log_2(n-1) \rceil$. Так как $2^m \geqslant n - 1$, то $\bar{A}^{\circ 2^m} = B$. По теореме существует алгоритм для вычисления всех этих матриц и, в частности B, с числом битовых операций $m \cdot O(n^{\log_2 7} \cdot \log^2 n) = O(n^{\log_2 7} \log^3 n)$. Теорема доказана.

Замечание. См. замечания к теореме.

Распознавание принадлежности булевых функций предполным классам Поста

Рассмотрим задачу распознавания свойств булевых функций, причем сейчас будем считать, что булевы функции поступают на вход алгоритма в векторном представлении. А именно, пусть все наборы длины n из 0 и 1 упорядочены естественным образом (как соответствующие им двоичные числа). Тогда булевскую функцию $f(x_1, \dots, x_n)$ от n переменных можно задать вектором $(a_0, a_1, \dots, a_{2^n-1})$ ее значений на всех 2^n наборах. В качестве алгоритмов мы рассмотрим алгоритмы с битовыми операциями. Любой такой алгоритм можно рассматривать как схему из функциональных элементов (СФЭ), элементами в которой могут быть любые функции от 2 переменных (или 1). Если ответ в задаче для данного входа "да", то на выходе должна быть 1, иначе 0. Под сложностью алгоритма будем понимать число битовых операций (число элементов в СФЭ).

Теорема Поста о полноте системы булевых функций сводит вопрос о полноте к вопросу о принадлежности функций 5 предполным классам T_0, T_1, S, L, M (см.[ ]). Мы рассмотрим вопрос о сложности распознавания этих свойств. Напомним, что

$f \in T_0 \iff f(0, \dots, 0) = 0, \quad f \in T_1 \iff f(1, \dots, 1) = 1,$

$f \in S \iff f(\bar{x}_1, \dots, \bar{x}_n) = \bar{f}(x_1, \dots, x_n),$

$f \in L \iff f(x_1, \dots, x_n) = a_0 \oplus a_1 x_1 \oplus \dots \oplus a_n x_n,$

$f \in M \iff$ для всех $\tilde{\alpha} = (\alpha_1, \dots, \alpha_n)$ и $\tilde{\beta} = (\beta_1, \dots, \beta_n)$ таких, что $\forall i(\alpha_i \leqslant \beta_i)$, выполняется $f(\tilde{\alpha}) \leqslant f(\tilde{\beta})$.

Утверждение 1. *При векторном представлении функций для распознавния свойства "$f(x_1, \dots, x_n) \in T_0$?" существует алгоритм (СФЭ) со сложностью 1.*

В этом случае выход z задается формулой $z = \bar{\alpha}_0$.

Утверждение 2. *При векторном представлении функций для распознавния свойства "$f(x_1, \dots, x_n) \in T_1$?" существует алгоритм (СФЭ) со сложностью 0.*

В этом случае выход z задается формулой $z = \alpha_{2^n-1}$.

Утверждение 3. *При векторном представлении функций для распознавния свойства "$f(x_1, \dots, x_n) \in S$?" существует алгоритм (СФЭ) со сложностью $O(N)$, где $N = 2^n$ — длина входа.*

Доказательство. По определению самодвойственных функций $f(x_1, \dots, x_n) \in S$ тогда и только тогда, когда для любого $\tilde{\alpha} = (\alpha_1, \dots, \alpha_n)$ выполняется $f(\alpha_1, \dots, \alpha_n) = f(\bar{\alpha}_1, \dots, \bar{\alpha}_n)$, то есть когда для всех $i = 0, 1, \dots, 2^{n-1} - 1$ выполняется $a_i = a_{i+2^{n-1}}$. Таким образом,

для распознавания свойства "$f \in S$?" достаточно использовать 2^{n-1} булевых операций $a_i \sim a_{i+2^{n-1}}$ и затем взять конъюнкцию полученных значений. Общее число битовых операций будет $2^{n-1}+2^{n-1}-1 = N-1$.

Утверждение 4. *При векторном представлении функций для распознавания свойства "$f(x_1, \ldots, x_n) \in L$?" существует алгоритм (СФЭ) со сложностью $O(N)$, где $N = 2^n$ — длина входа.*

Лемма.

$$f(x_1, \ldots, x_n) \in L \Longleftrightarrow \begin{cases} f(0, x_2, \ldots, x_n) \in L, \\ f(1, x_2, \ldots, x_n) \equiv f(0, x_2, \ldots, x_n) \oplus c, c \in \{0, 1\}. \end{cases}$$

Доказательство. Если $f(x_1, \ldots, x_n) = a_1 x_1 \oplus a_2 x_2 \oplus \ldots \oplus a_n x_n + a_0$, то, очевидно, $f(0, x_2, \ldots, x_n) \in L$ и $f(1, x_2, \ldots, x_n) \equiv f(0, x_2, \ldots, x_n) \oplus a_1$. Для доказательства обратного перехода заметим, что для любой булевой функции справедливо представление

$f(x_1, \ldots, x_n) = \bar{x}_1 \cdot f(0, x_2, \ldots, x_n) \oplus x_1 \cdot f(1, x_2, \ldots, x_n) = (x_1 \oplus 1) \cdot f(0, x_2, \ldots, x_n) \oplus f(1, x_2, \ldots, x_n) = x_1 \cdot (f(0, x_2, \ldots, x_n) \oplus f(1, x_2, \ldots, x_n)) \oplus f(0, x_2, \ldots, x_n)$. Поэтому, если $f(1, x_2, \ldots, x_n) \equiv f(0, x_2, \ldots, x_n) \oplus c$, то есть $f(0, x_2, \ldots, x_n) \oplus f(1, x_2, \ldots, x_n) \equiv c$, и $f(0, x_2, \ldots, x_n) \in L$, то и $f(x_1, x_2, \ldots, x_n) \in L$.

Лемма доказана.

Будем строить алгоритм (СФЭ) для распознавания свойства "$f(x_1, \ldots, x_n) \in L$?" рекурсивно в соответствии с леммой. Для проверки условия $f(1, x_2, \ldots, x_n) \equiv f(0, x_2, \ldots, x_n)$ достаточно $2^n - 1$ бинарных битовых операций (2^{n-1} сравнений $\alpha_i = \alpha_{i+2^{n-1}}$ и $2^{n-1} - 1$ конъюнкций полученных значений). Аналогично $2^n - 1$ бинарных операций достаточно для проверки условия $f(1, x_2, \ldots, x_n) \equiv f(0, x_2, \ldots, x_n) \oplus 1$. Для проверки условия $f(1, x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n) \oplus c$ достаточно взять дизъюнкцию двух полученных результатов сравнений. При этом общее число битовых операций $2(2^n - 1) < 2^{n+1}$. Пусть $L(n)$ — минимальное число битовых операций для ответа на вопрос "$f(x_1, \ldots, x_n) \in L$?" Тогда $L(1) = 1$ (т.к. выход $z \equiv 1$) и в соответствии с леммой $L(n) < L(n-1) + 2^{n+1} < L(n-2) + 2^n + 2^{n+1} < L(n-3) + 2^{n-1} + 2^n + 2^{n+1} < \ldots < L(1) + 2^3 + 2^4 + \ldots + 2^{n+1} < 2^{n+2} = 4N$. Теорема доказана.

Утверждение 5. *При векторном представлении функций для распознавния свойства "$f(x_1, \ldots, x_n) \in M$?" существует алгоритм (СФЭ) со сложностью $O(N \log N)$, где $N = 2^n$ — длина входа.*

Доказательство. Известно, что $f(x_1, \ldots, x_n) \in M$ тогда и только тогда, когда для любых двух наборов $\tilde{\alpha}$ и $\tilde{\beta}$ таких, что $\tilde{\alpha} = (\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n)$ и $\tilde{\beta} = (\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n)$ (где

i — любое), выполняется $f(\tilde{\alpha}) \leqslant f(\tilde{\beta})$. Число указанных пар наборов $(\tilde{\alpha}, \tilde{\beta})$ равно $n \cdot 2^{n-1}$. Таким образом, для распознавания свойства "$f \in M$?" достаточно использовать $n \cdot 2^{n-1}$ битовых операций "$x \leqslant y$" и затем взять конъюнкцию полученных значений. Общее число битовых операций будет $n \cdot 2^{n-1} + n \cdot 2^{n-1} - 1 = N \log_2 N - 1$.

Из утверждений 1-5 и теоремы Поста о полноте получаем следующее утверждение.

Теорема. *При векторном представлении функций для распознавния полноты системы функций существует алгоритм (СФЭ) со сложностью $O(N \log N)$, где N — длина входа.*

Замечание. Вороненко А. А. [] нашел более быстрый алгоритм со сложностью $O(N\sqrt{\log N} \log\log N)$ для распознавания свойства монотонности (класс M), откуда следует, что и в теореме оценку $O(N \log N)$ можно понизить до $O(N\sqrt{\log N} \log\log N)$.

Распознавание сохранения 2-х местных предикатов.

Пусть $E_k = \{0, 1, \dots, k-1\}$. Через P_k будем обозначать множество всех k-значных функций, то есть функций, отображающих E_k^n в E_k при всех n.

Определение. *Предикатом* (m-местным) на конечном множестве D называется любая функция $R(y_1 \dots y_m) : D^m \longrightarrow \{\text{истина, ложь}\}$.

Определение. Пусть $f(x_1 \dots x_n) \in P_k$ и $R(y_1, y_2)$ — 2-местный предикат на множестве E_k. Будем говорить, что $f(x_1 \dots x_n)$ *сохраняет предикат* R, если для любых наборов $\tilde{\alpha} = (\alpha_1 \dots \alpha_n)$ и $\tilde{\beta} = (\beta_1 \dots \beta_n)$ выполняется импликация:

$$(\forall j R(\alpha_j, \beta_j)) \to R(f(\tilde{\alpha}), f(\tilde{\beta})).$$

Обозначим $U(R)$ — класс всех функций,сохраняющих предикат R. Пусть $\tilde{\alpha} = (\alpha_1, \dots, \alpha_n)$, $\tilde{\beta} = (\beta_1, \dots, \beta_n)$ — наборы с элементами из D и R — 2-местный предикат на D. Тогда определим предикат R^n на D^n следующим образом:

$$R^n(\tilde{\alpha}, \tilde{\beta}) \equiv \forall j R(\alpha_j, \beta_j).$$

Если $\tilde{\gamma} = (\alpha_1, \dots, \alpha_{n-1})$, $\tilde{\delta} = (\delta_1, \dots, \delta_{n-1})$, то легко видеть, что

$$R^n(\tilde{\alpha}, \tilde{\beta}) \equiv R^{n-1}(\tilde{\gamma}, \tilde{\delta}) \& R(\alpha_n, \beta_n).$$

Задача. Фиксируем E_k. Задан предикат $R(y_1, y_2)$ на E_k. Требуется построить алгоритм (СФЭ) для ответа на вопрос "$f(x_1 \dots x_n) \in U(R)$?". При этом считается, что наборы из E_k^n упорядочиваются лексикографически и функция f задается вектором $f = (\alpha_0, \alpha_1, \dots, \alpha_{k^n-1})$ ее значений на этих наборах; $N = k^n$ — длина входа.

Тривиальный алгоритм для этой задачи имеет сложность $O(N^2)$ (проверка пар). Если предикат R истинен на d парах, то существует алгоритм со сложностью $O(d^n) = O(N^{\log_k d})$. Однако верен следующий более сильный результат.

Теорема. *Для любого предиката $R(y_1, y_2)$ на E_k существует алгоритм (СФЭ), распознающий "$f(x_1 \dots x_n) \in U(R)$?" со сложностью $O(N \log N)$, где $N = k^n$ — длина входа.*

Доказательство.

$$f(x_1 \dots x_n) \notin U(R) \Longleftrightarrow \exists \tilde{\alpha}, \tilde{\beta} (R^n(\tilde{\alpha}, \tilde{\beta}) \& \bar{R}(f(\alpha), f(\beta))) \Longleftrightarrow$$

$$\Longleftrightarrow \bigvee_{\tilde{\alpha}, \tilde{\beta}} (R^n(\tilde{\alpha}, \tilde{\beta}) \& \bar{R}(f(\tilde{\alpha}), f(\tilde{\beta}))) \Longleftrightarrow$$

$$\Longleftrightarrow \bigvee_{c,d\in E_k} \bigvee_{\tilde\alpha,\tilde\beta} (R^n(\tilde\alpha,\tilde\beta)\&(f(\tilde\alpha)=c)\&(f(\tilde\beta)=d)\&\bar R(c,d)) =$$

$$= \bigvee_{(c,d):\bar R(c,d)} \bigvee_{(\tilde\alpha,\tilde\beta)} (R^n(\tilde\alpha,\tilde\beta)\&(f(\tilde\alpha)=c)\&(f(\tilde\beta)=d)).$$

При фиксированных c, d вычисление логических переменных $x_{\tilde\alpha} \equiv (f(\tilde\alpha)=c)$ и $y_{\tilde\beta}\equiv(f(\tilde\beta)=d)$ требует $O(N)$ операций.

Пусть $L(N)=L'(n)$ — минимальная битовая сложность вычисления выражения $\bigvee_{\tilde\alpha,\tilde\beta} R^n(\tilde\alpha,\tilde\beta)x_{\tilde\alpha}y_{\tilde\beta}$ при заданных $x_{\tilde\alpha}, y_{\tilde\beta}$. Докажем, что $L(N)=O(N\log N)$.

Пусть $\tilde\alpha=(\tilde\gamma,\alpha_n), \tilde\beta=(\tilde\delta,\beta_n)$. Тогда

$$\bigvee_{\tilde\alpha,\tilde\beta} R^n(\tilde\alpha,\tilde\beta)x_{\tilde\alpha}y_{\tilde\beta} = \bigvee_{\tilde\alpha=(\tilde\gamma,\alpha_n),\tilde\beta=(\tilde\delta,\beta_n)} R^{n-1}(\tilde\gamma,\tilde\delta)R(\alpha_n,\beta_n)x_{\tilde\gamma,\alpha_n}y_{\tilde\delta,\beta_n} =$$

$$= \bigvee_{\tilde\gamma,\tilde\delta}\bigvee_{\alpha_n,\beta_n} R^{n-1}(\tilde\gamma,\tilde\delta)R(\alpha_n,\beta_n)x_{(\tilde\gamma,\alpha_n)}y_{(\tilde\delta,\beta_n)} =$$

$$= \bigvee_{\tilde\gamma,\tilde\delta} R^{n-1}(\tilde\gamma,\tilde\delta)\bigvee_{\alpha_n,\beta_n} R(\alpha_n,\beta_n)x_{(\tilde\gamma,\alpha_n)}y_{(\tilde\delta,\beta_n)} =$$

$$= \bigvee_{\tilde\gamma,\tilde\delta} R^{n-1}(\tilde\gamma,\tilde\delta)\bigvee_{\alpha_n} x_{(\tilde\gamma,\alpha_n)}\bigvee_{\beta_n:R(\alpha_n,\beta_n)} y_{(\tilde\delta,\beta_n)} =$$

$$= \bigvee_{\alpha_n}(\bigvee_{\tilde\gamma,\tilde\delta} R^{n-1}(\tilde\gamma,\tilde\delta)x_{(\tilde\gamma,\alpha_n)}z_{(\tilde\delta,\alpha_n)}),$$

где $z_{(\tilde\delta,\alpha_n)}=\bigvee_{\beta_n:R(\alpha_n,\beta_n)} y_{(\tilde\delta,\beta_n)}$.

Отсюда $L'(n)\leqslant kL'(n-1)+k^n(k-1)+k-1$, поскольку задача для n сводится к k таким же задачам для $n-1$, при этом для вычисления каждой из k^n переменных z требуется не более $k-1$ дизъюнкций и после решения всех k подзадач требуется $k-1$ дизъюнкций для вычисления дизъюнкции по α_n. Переходя к $L(N)$, получаем $L(N)\leqslant kL(\frac{N}{k})+O(N)$, откуда, по теореме о рекуррентном неравенстве, $L(N)=O(N\log N)$. Теорема доказана.

Классы F^m.

Пусть теперь $R(y_1, \dots, y_m)$ — m-местный предикат на множестве $E_k = \{0, 1, \dots, k-1\}$. Если $\tilde{\alpha}_j = (\alpha_1^j, \alpha_2^j, \dots, \alpha_n^j)$, $j = 1, 2, \dots, m$ — наборы с элементами из E_k, то определим

$$R^n(\tilde{\alpha}_1, \tilde{\alpha}_2, \dots, \tilde{\alpha}_m) \equiv \forall i R(\alpha_i^1, \alpha_i^2, \dots, \alpha_i^m).$$

Определение. Будем говорить, что функция $f(x_1, \dots, x_n)$ из P_k *сохраняет* R, если для любых m наборов $\tilde{\alpha}_1, \tilde{\alpha}_2, \dots, \tilde{\alpha}_m$ выполняется импликация

$$R^n(\tilde{\alpha}_1, \tilde{\alpha}_2, \dots, \tilde{\alpha}_m) \Longrightarrow R(f(\tilde{\alpha}_1), f(\tilde{\alpha}_2), \dots, f(\tilde{\alpha}_m)).$$

Рассмотрим следующий предикат F^m на $E_2 = \{0, 1\}$:

$$R_m(y_1, \dots, y_m) = \begin{cases} \text{истина} & \Longleftrightarrow \exists i (y_i = 0), \\ \text{ложь} & \Longleftrightarrow \tilde{y} = (1, \dots, 1). \end{cases} \tag{10}$$

Класс всех функций алгебры логики, сохраняющих предикат R_m, обозначим F^m. Классы F^m при $m = 2, 3, 4, \dots$ образуют одну из 8 бесконечных цепочек замкнутых классов в алгебре логики. Рассмотрим следующую задачу.

Задача. Пусть $m \geqslant 2$ — фиксированное натуральное число. Требуется построить алгоритм для ответа на вопрос "$f(x_1, \dots, x_n) \in F^m$?" при условии, что функция поступает на вход в виде вектора значений $f(x_1, \dots, x_n) = (a_0, a_1, \dots, a_{2^n-1})$ длины $N = 2^n$.

Заметим, что тривиальный алгоритм, основанный на просмотре всех выборок по m значений функции и проверке импликации ( ) требует по порядку не менее N^m операций.

Теорема. *Для любого фиксированного $m \geqslant 2$ существует алгоритм для ответа на вопрос "$f(x_1 \dots x_n) \in U(R_m)$?" с битовой сложностью $O(N \log^3 N)$.*

Замечание. Константа зависит от m (растет с ростом m).

Доказательство. Пусть $\tilde{\alpha}_1, \tilde{\alpha}_2, \dots, \tilde{\alpha}_m$ — произвольные наборы, где $\tilde{\alpha}_j = (\alpha_1^j, \alpha_2^j, \dots, \alpha_n^j)$. Тогда по определению

$$f(x_1, \dots, x_m) \notin F_m \Longleftrightarrow$$

$$\Longleftrightarrow \exists \tilde{\alpha}_1, \dots, \tilde{\alpha}_m (R_m^n(\tilde{\alpha}_1, \dots, \tilde{\alpha}_m) \& (f(\tilde{\alpha}_1) = 1) \& \dots \& (f(\tilde{\alpha}_m) = 1)) \Longleftrightarrow$$

$$\Longleftrightarrow \bigvee_{\tilde{\alpha}_1, \dots, \tilde{\alpha}_m} R_m^n(\tilde{\alpha}_1, \dots, \tilde{\alpha}_m) x_{\tilde{\alpha}_1} \cdot \dots \cdot x_{\tilde{\alpha}_m} =$$

$$= \bigvee_{\tilde{\alpha}_1,\dots,\tilde{\alpha}_m} R_m(\alpha_1^1 \dots \alpha_1^m) \cdot R_m(\alpha_2^1 \dots \alpha_2^m) \cdot \dots \cdot R_m(\alpha_n^1 \dots \alpha_n^m) x_{\tilde{\alpha}_1} \cdot \dots \cdot x_{\tilde{\alpha}_m},$$

где $x_{\tilde{\alpha}} \equiv (f(\tilde{\alpha}) = 1)$, т.е. $x_{\tilde{\alpha}}$ — соответсвующая координата в векторе функции. Определим функцию $t_m(\alpha_1, \dots, \alpha_m)$, где $\alpha_j \in \{0,1\}$, следующим образом:

$$t_m(\alpha_1, \dots, \alpha_m) = \begin{cases} 0, & \text{если } \bar{R}_m(\alpha_1, \dots, \alpha_m), \\ 1, & \text{если } R_m(\alpha_1, \dots, \alpha_m). \end{cases} \tag{11}$$

Легко видеть, что

$$\bigvee_{\tilde{\alpha}_1,\dots,\tilde{\alpha}_m} R_m(\alpha_1^1, \dots, \alpha_1^m) \cdot \dots \cdot R_m(\alpha_n^1, \dots, \alpha_n^m) x_{\tilde{\alpha}_1} \cdot \dots \cdot x_{\tilde{\alpha}_m} = \text{И} \iff$$

$$T_n = \sum_{\tilde{\alpha}_1,\dots,\tilde{\alpha}_m} t_m(\alpha_1^1, \dots, \alpha_1^m) \cdot \dots \cdot t_m(\alpha_n^1, \dots, \alpha_n^m) x_{\tilde{\alpha}_1} \cdot \dots \cdot x_{\tilde{\alpha}_m} > 0.$$

Пусть $L'_{ap}(n) = L_{ap}(N)$ —наименьшее число арифметических операций, необходимых для вычисления T_n. Обозначим $\tilde{\beta}_j = (\alpha_1^j, \alpha_2^j, \dots, \alpha_{n-1}^j), \gamma_j = \alpha_n^j$. Тогда

$$T_n = \sum_{\tilde{\beta}_1,\dots,\tilde{\beta}_m} \sum_{\gamma_1,\dots,\gamma_m} t_m(\alpha_1^1, \dots, \alpha_1^m) \cdot \dots \cdot t_m(\alpha_n^1, \dots, \alpha_n^m) \cdot x_{\tilde{\beta}_1,\gamma_1} \cdot \dots \cdot x_{\tilde{\beta}_m,\gamma_m} =$$

$$= \sum_{\tilde{\beta}_1,\dots,\tilde{\beta}_m} t_m^{n-1}(\tilde{\beta}_1, \dots, \tilde{\beta}_m) \sum_{(\gamma_1,\dots,\gamma_m) \in E_2^m} t_m(\gamma_1, \dots, \gamma_m) x_{\tilde{\beta}_1,\gamma_1} \cdot \dots \cdot x_{\tilde{\beta}_m,\gamma_m} =$$

$$= \sum_{\tilde{\beta}_1,\dots,\tilde{\beta}_m} t_m^{n-1}(\tilde{\beta}_1, \dots, \tilde{\beta}_m) \sum_{(\gamma_1,\dots,\gamma_m) \neq (1,\dots,1)} x_{\tilde{\beta}_1,\gamma_1} \cdot \dots \cdot x_{\tilde{\beta}_m,\gamma_m} =$$

$$= \sum_{\tilde{\beta}_1,\dots,\tilde{\beta}_m} t_m^{n-1}(\tilde{\beta}_1, \dots, \tilde{\beta}_m)((x_{\tilde{\beta}_1,0} + x_{\tilde{\beta}_1,1})(x_{\tilde{\beta}_2,0} + x_{\tilde{\beta}_2,1}) \cdot \dots \cdot (x_{\tilde{\beta}_m,0} + x_{\tilde{\beta}_m,1}) -$$

$$- x_{\tilde{\beta}_1,1} x_{\tilde{\beta}_2,1} \cdot \dots \cdot x_{\tilde{\beta}_m,1}) = \sum_{\tilde{\beta}_1,\dots,\tilde{\beta}_m} t_m^{n-1}(\tilde{\beta}_1, \dots, \tilde{\beta}_m) y_{\tilde{\beta}_1} y_{\tilde{\beta}_2} \cdot \dots \cdot y_{\tilde{\beta}_m} -$$

$$- \sum_{\tilde{\beta}_1,\dots,\tilde{\beta}_m} t_m^{n-1}(\tilde{\beta}_1, \dots, \tilde{\beta}_m) z_{\tilde{\beta}_1} z_{\tilde{\beta}_2} \cdot \dots \cdot z_{\tilde{\beta}_m},$$

где $y_{\tilde{\beta}} = x_{\tilde{\beta},0} + x_{\tilde{\beta},1}$, а $z_{\tilde{\beta}} = x_{\tilde{\beta},1}$

Свели задачу с параметром n к 2 подзадачам с параметром $n-1$. Отсюда $L'_{\text{ap}}(n) \leqslant 2L'_{\text{ap}}(n-1) + 2^{n-1} + 1$, поскольку для вычисления всех $y_{\tilde{\beta}}$ достаточно 2^{n-1} сложений и одного вычитания достаточно для вычисления T_n после решения двух подзадач. Переходя к N, имеем $L'_{ap}(N) = 2L_{ap}(\frac{N}{2}) + O(N)$. Отсюда по теореме о рекуррентном неравенстве получаем $L_{ap}(N) = O(N \log N)$ (в этой теореме имеем $a = 2, c =$

$2, \alpha = 1$). Отметим, что исходная задача была для $x_{\tilde{\alpha}} \in \{0, 1\}$. Однако в подзадачах переменные могут быть произвольными натуральными числами. Переход к подзадачам делается $n-1$ раз. При каждом переходе разрядность переменных увеличивается не более, чем на 1, поэтому в подзадачах все числа имеют длину не более $n+1$. При $n = 0$ получаются подзадачи вычисления T_0 вида $T_0 = z \cdot z \cdot z \cdot z \cdot \ldots \cdot z = z^m$, в которых образуются числа длины $\leqslant m(n+1)$. При переходе к задаче из подзадач длина чисел увеличивается не более, чем на 1, поэтому все числа в алгоритме имеют длину не более $m(n+1)+n \leqslant const \cdot n$. Следовательно, каждая арифметическая операция требует $O(n^2) = O(\log^2 N)$ битовых операций, откуда $L(N) = L_{ap}(N) \cdot O(\log^2 N) = O(N \log^3 N)$, ч.т.д.

Машины Тьюринга

В рассмотренных выше примерах класс алгоритмов был ограничен. Так, в задачах сортировки и поиска мы не обсуждали способ представления элементов a_i и не разрешали изучать и преобразовывать отдельные части этих представлений. Если мы хотим получать утверждения типа "для любых алгоритмов" вообще, то мы должны принять какую-нибудь универсальную модель алгоритмов. Одной из таких моделей является *машина Тьюринга.*

Машина Тьюринга M имеет потенциально бесконечную ленту, разделенную на ячейки, и головку, которая в каждый (дискретный) момент времени $t = 0, 1, 2, \ldots$ обозревает ровно одну ячейку. Задано некоторое конечное множество состояний $Q = \{q_0, q_1, \ldots, q_l\}$, и машина в каждый момент времени находится ровно в 1 из этих состояний. Задан конечный ленточный (рабочий) алфавит $C = \{c_0, c_1, \ldots, c_m\}$, где $c_0 = \Lambda$ — пустой символ, и в каждый момент времени в каждой ячейке находится ровно 1 символ из алфавита C, причем будем считать, что символы, отличные от Λ, находятся лишь в конечном числе ячеек. Машина M характеризуется ее программой, которая представляет собой конечный набор команд вида: $c_i q_j \to c_k q_r T$, где $c_i \in C$, $c_k \in C$, $q_j \in Q$, $q_r \in Q$, $T \in L, R, S$. На каждом такте машина M работает следующим образом. Если головка обозревает ячейку, в которой находится символ c_j, M находится в состоянии q_j и в программе машины M есть команда $c_i q_j \to c_k q_r T$, то символ в обозреваемой ячейке заменяется на c_k, машина переходит в состояние q_r и головка остается на месте, если $T = S$, или сдвигается на 1 ячейку вправо или влево, если $T = R$ или $T = L$. Далее машина переходит к следующему такту и повторяет этот процесс. Если же в программе машины нет команды, в левой части которой стоит пара $c_i q_j$, то машина останавливается.

Мы будем рассматривать только детерминированные машины Тьюринга, то есть такие, у которых в программе каждая пара $c_i q_j$ встречается в левых частях команд не более одного раза. В этом случае каждый шаг машины определяется однозначно.

Определение. Если A - алфавит, то через A^* будем обозначать множество всех (конечных) слов в алфавите A. Пусть C - ленточный алфавит машины M, $C_0 = C \setminus \{\Lambda\}$ и пусть задан некоторый входной алфавит A, такой, что $A \subseteq C_0$. Тогда мы будем считать, что машина M осуществляет преобразование $\varphi_M : A^* \to C_0^* \cup \{*\}$, которое определяется следующим образом (здесь $*$ в $\{*\}$ трактуется как неопределенность). Пусть задано некоторое слово $\bar{a} = a_1 a_2 \ldots a_n \in A^*$.

Разместим символы этого слова в последовательные ячейки ленты, а в остальные ячейки поместим Λ, поместим головку на ячейку, в которой стоит a_1, установим машину в начальное состояние q_1 и начнем работу машины. Если после этого машина M будет работать бесконечно долго, то считаем, что $\varphi_M(a_1a_2\dots a_n) = *$. Если машина M остановится и головка будет обозревать Λ, то также считаем, что $\varphi_M(\bar{a}) = *$. Если же M остановится и обозреваемый символ отличен от Λ, то рассмотрим максимальную связную (состоящую из последовательных ячеек) область ленты, которая содержит обозреваемую ячейку и не содержит Λ. Пусть в ячейках этой области записано слово $\bar{b} = b_1b_2\dots b_k$ (эта область, очевидно, конечна). Тогда считаем, что $\varphi_M(\bar{a}) = \bar{b}$.

Тезис Тьюринга. *Для любого алгоритмического преобразования $\varphi : A^* \to C_0^* \cup \{*\}$ существует машина Тьюринга M, осуществляющая преобразование φ.*

Распознавание симметрии

Пусть $A = \{0,1\}$ и слово $\bar{a} = a_1a_2\dots a_n \in A^*$. Будем говорить, что слово $\bar{a}$ симметрично, если $a_1 = a_n, a_2 = a_{n-1}$ и т.д. Пусть машина Тьюринга M имеет ленточный алфавит C и множество состояний Q, причем $A \subseteq C$ и $q' \in Q$, $q'' \in Q$. Будем говорить, что M распознает симметрию, если для любого входного слова $\bar{a} \in A^*$ машина M всегда останавливается и при этом находится в состоянии q', если $\bar{a}$ симметрично, или q'', если $\bar{a}$ не симметрично.

Утверждение. *Существует машина Тьюринга M, которая распознает симметрию и делает при любом входном слове длины n не более cn^2 шагов, где c - некоторая константа.*

Доказательство. Достаточно построить машину M, которая запоминает и стирает первый символ, перегоняет головку в конец слова и сравнивает символ в памяти с последним символом слова. Если они не совпадают, то M переходит в состояние q'' и останавливается. Если совпадают, то она стирает последний символ, возвращается в начало слова и повторяет процесс. Если слово полностью стерто, то M переходит в состояние q' и останавливается. При этом головка не более n раз пробегает по слову длины не более n. Поэтому общее число шагов есть $O(n^2)$.

Общие утверждения о сложности задач

Рассмотрим все машины Тьюринга, имеющие в ленточном алфавите символы Λ и $|$. Пусть $Z^+ = \{0\} \cup N$, где N - множество натуральных чисел. Будем представлять число $k \in Z^+$ на ленте машины в виде кода $\Lambda \; || \; \ldots \; | \; \Lambda$, где количество $|$ равно $k+1$ (остальные символы на ленте Λ). Набор $(k_1, k_2, \ldots, k_n)$ будем представлять в виде кода $\Lambda \; || \; \ldots \; | \; \Lambda \; || \; \ldots \; | \; \Lambda \ldots \Lambda \; || \; \ldots \; | \; \Lambda$,где в первом массиве k_1+1 символов $|$, во втором k_2+1 и т.д. Применяя машину M к коду числа или набора, будем помещать головку на самый первый символ $|$ и устанавливать машину M в ее начальное состояние q_1.

Определение. Для любой машины Тьюринга M и любого натурального числа n будем считать, что машина M вычисляет функцию $f(x_1, x_2, \ldots, x_n) : (Z^+)^n \to Z^+ \cup \{*\}$ которая определяется следующим образом. Применим машину M к коду набора $(a_1, a_2, \ldots, a_n)$. Если M остановится и после остановки на ленте будет только код некоторого числа $b \in Z^+$, то $f(a_1, a_2, \ldots, a_n) = b$. Во всех остальных случаях $f(a_1, a_2, \ldots, a_n) = *$ (неопределенность).

Определение. Функция $f(x_1, x_2, \ldots, x_n) : (Z^+)^n \to Z^+ \cup \{*\}$ называется вычислимой, если существует машина Тьюринга M, которая ее вычисляет.

Определение. Говорят, что машина M правильно вычисляет функцию $f(x_1, x_2, \ldots, x_n)$, если начиная работу с кода набора $(a_1, \ldots, a_n)$ машина M в том случае, когда $f(a_1, a_2, \ldots, a_n)$ не определено, обязательно работает бесконечно долго, а в том случае, когда $f(a_1, a_2, \ldots, a_n) = b$, машина останавливается, на ленте остается код b и головка машины обозревает самый левый символ $|$.

Утверждение. *Если $f(x_1, x_2, \ldots, x_n)$ вычислима, то существует машина Тьюринга M, которая ее вычисляет правильно.*

Доказательство см., например, в [ ].

Определение. Всюду определенные вычислимые функции мы будем называть общерекурсивными функциями. (Обычно общерекурсивные функции определяют иначе, но данное определение эквивалентно обычному; см., например, [ ]).

Функция, вычисляемая машиной M, не изменится, если произвольно переименовать все символы ленточного алфавита, кроме Λ и $|$, и все состояния, оставляя начальное состояние начальным (конечно, разные символы должны переименовываться в разные). Поэтому мы будем считать, что есть бесконечный фиксированный алфавит $\{c_1 = \Lambda, c_2 = |, c_3, c_4, \ldots\}$, из которого берется ленточный алфавит

машины M, и бесконечный фиксированный алфавит $\{q_1, q_2, \dots\}$, из которого берутся состояния машины M. Будем записывать индексы в строку после c или q, представляя их в двоичной системе счисления (например, $c_6 = c110$). Программу машины M будем записывать в виде последовательности всех ее команд $c_i q_j \longrightarrow c_k q_r T$, разделенных точкой с запятой. Тогда программа любой машины будет представлять собой слово в алфавите $D = \{\Lambda, |, c, q, 0, 1, \longrightarrow, R, L, S, ;\}$.

Теорема. *Существует алгоритм нумерации всех машин Тьюринга из указанного выше семейства такой, что для восстановления программы по ее номеру также существует алгоритм.*

Доказательство. Будем считать, что символы алфавита D упорядочены (например, так, как это сделано выше). Тогда все слова одной длины k можно упорядочить лексиграфически (как в словаре). Будем теперь просматривать все слова в алфавите D в соответствии с их длиной: сначала длины 1, затем длины 2 и т.д. Слова одной длины k просматриваем в лексикографическом порядке. Для каждого слова применяем алгоритм, который проверяет, является ли это слово правильно построенной программой некоторой детерминированной машины Тьюринга. Если да, то приписываем этой программе очередной номер (начиная с 0). При этом любой машине Тьюринга (из рассматриваемого семейства) по ее программе будет (алгоритмично) сопоставляться некоторый номер. Тот же перебор осуществляем, если задан номер и требуется найти соответствующую этому номеру программу.

Зафиксируем далее некоторую нумерацию машин Тьюринга $i \longleftrightarrow M_i$, удовлетворяющую теореме. Так как машина M_i вычисляет некоторую функцию $f(x)$, то мы получаем также некоторую нумерацию всех вычислимых функций одной переменной $i \longrightarrow \varphi_i(x)$. Заметим, что при этом может быть $\varphi_i(x) \equiv \varphi_j(x)$ при $i \neq j$, поскольку разные машины Тьюринга могут вычислять одну и ту же функцию $f(x)$.

Докажем теперь теоремы о том, что существуют сколь угодно сложно вычислимые функции.

Теорема. *Для любой общерекурсивной функции $T(x)$ существует общерекурсивная функция $f(x)$, принимающая только 2 значения 0 и 1 и такая, что для любой машины Тьюринга M_i, вычисляющей $f(x)$, хотя бы при одном x выполняется неравенство $t_i(x) > T(x)$, где $t_i(x)$ — время работы машины M_i на входе x (точнее, на коде числа x).*

Замечание. Отметим, что функция $T(x)$ может расти очень быстро. Например, функции $g_1(n) = n$, $g_2(n) = n^n$, $g_3(n) = n^{g_2(n)}$, $\dots$, $g_{m+1}(n) = n^{g_m(n)}$, $\dots$ общерекурсивны. Также общерекурсивна и

функция $h(n) = g_n(n)$, которая растет с астрономической скоростью.

Доказательство. Для всех $i \in Z^+$ и $x \in Z^+$ пусть $t_i(x)$ обозначает время работы машины с номером i, если входом является код числа x ($t_i(x)$ может быть и бесконечным), и пусть $\varphi_i(x)$ обозначает функцию, вычисляемую машиной M_i. Определим функцию $f(x)$ следующим образом:

$$f(x) = \begin{cases} 1, & \text{если } t_x(x) \leqslant T(x) \text{ и } \varphi_x(x) = 0, \\ 0, & \text{иначе.} \end{cases} \tag{12}$$

Утверждение. *Функция $f(x)$ — вычислимая, а следовательно, общерекурсивная.*

Доказательство. Опишем алгоритм вычисления $f(x)$. По заданному $x \in Z^+$ находим программу машины M_x (см. теорему). Вычисляем $T(x)$ (так как $T(x)$ - общерекурсивна, то для этого существует алгоритм). Имея программу машины M_x, моделируем ее работу в течение $T(x)$ тактов, взяв в качестве входного слова код числа x. Если за $T(x)$ тактов машина остановится и результатом будет код числа 0, то выдаем ответ 1, иначе выдаем ответ 0. Моделируя работу машины, мы можем работать только с той частью ленты, на которой записывается входное слово, а также которая посещается головкой во время работы. Тогда на каждом шаге нам достаточно хранить лишь конечный кусок ленты, что позволяет определить содержимое ленты и после останова машины. Следовательно, весь процесс вычисления $f(x)$ алгоритмичен. В соответствии с тезисом Тьюринга существует машина Тьюринга, которая вычисляет $f(x)$. Мы примем здесь это утверждение, хотя для описанной функции $f(x)$ можно и явно построить вычисляющую ее машину Тьюринга (правда, долго и громоздко).

Пусть машина M_i вычисляет $f(x)$, то есть $f(x) = \varphi_i(x)$. В частности $\varphi_i(i) = f(i)$ и значит определено. Допустим, что $t_i(i) \leqslant T(i)$. Тогда по определению $f(x)$ получаем: если $\varphi_i(i) = 0$, то $f(i) = 1$, а если $\varphi_i(i) \neq 0$, то $f(i) = 0$. В любом случае $f(i) \neq \varphi_i(i)$ — противоречие. Следовательно (от противного) $t_i(i) > T(i)$. Теорема доказана.

Теорема. *Для любой общерекурсивной функции $T(x)$ существует общерекурсивная функция $f(x)$, принимающая только 2 значения 0 и 1 и такая, что для любой машины Тьюринга M_i, вычисляющей $f(x)$, существует бесконечное число значений x, для которых выполняется неравенство $t_i(x) > T(x)$.*

Доказательство. Пусть $g(x) = x - (\lfloor\sqrt{x}\rfloor)^2$. Тогда функция $g(x)$ вычислима и всюду определена (то есть общерекурсивна). При $x = 0, 1, 2, 3, \ldots$ функция $g(x)$ принимает значения

$0, 0, 1, 2, 0, 1, 2, 3, 4, 0, 1, \ldots$. Легко доказать, что функция $g(x)$ принимает каждое значение из Z^+ бесконечное число раз. Определим функцию $f(x)$ следующим образом:

$$f(x) = \begin{cases} 1, & \text{если } t_{g(x)}(x) \leqslant T(x) \text{ и } \varphi_{g(x)}(x) = 0, \\ 0, & \text{иначе.} \end{cases} \tag{13}$$

Тогда функция $f(x)$ общерекурсивна (доказывается так же, как в предыдущей теореме). Пусть машина M_i вычисляет $f(x)$, то есть $f(x) = \varphi_i(x)$. Пусть j - любое число, такое, что $g(j) = i$ (таких j бесконечно много). Допустим, что $t_i(j) \leqslant T(j)$. Тогда по определению $f(x)$ получаем: если $\varphi_i(j) = 0$, то $f(j) = 1$, а если $\varphi_i(j) \neq 0$, то $f(j) = 0$. В любом случае $f(j) \neq \varphi_i(j)$ - противоречие. Следовательно, $t_i(j) > T(j)$. Теорема доказана.

Справедливо еще более сильное утверждение, которое мы приведем без доказательства.

Теорема. *Для любой общерекурсивной функции $T(x)$ существует общерекурсивная функция $f(x)$, принимающая только 2 значения 0 и 1 и такая, что для любой машины Тьюринга M_i, вычисляющей $f(x)$, множество тех x, для которых $t_i(x) \leqslant T(x)$, конечно.*

Теоремы показывают, что существуют сколь угодно сложно вычислимые общерекурсивные функции с двумя значениями (или, что эквивалентно, сколь угодно сложно распознаваемые языки). Возникает вопрос: а какой вообще может быть сложность задач (языков)? Существенный ответ на этот вопрос дает следующая теорема, которую мы приводим без доказательства.

Теорема. *Пусть общерекурсивные функции $t(n)$ и $T(n)$ таковы, что $\frac{T(n)}{t(n)\log_2 t(n)} \to \infty$ при $n \to \infty$. Тогда существует язык L, который распознается некоторой машиной Тьюринга с числом шагов не более $T(n)$ (для всех входных слов любой длины n) и не распознается никакой машиной Тьюринга с числом шагов $t(n)$.*

Эта теорема показывает, что возможные функции сложности языков образуют довольно плотное множество. Можно ли получить результат о большей плотности в общем случае неизвестно. Однако для одного важного интервала мы сейчас получим отрицательный ответ. А именно, мы покажем, что не существует языков со сложностью распознавания по порядку между n и $n \log n$.

Регулярные языки

Регулярные языки — это языки, распознаваемые автоматами. В этом контексте автомат можно определить как машину Тьюринга со следующими ограничениями: головка машины на каждом шаге движется вправо или машина останавливается; машина останавливается тогда и только тогда, когда головка обозревает символ Λ; машина останавливается в одном из двух состояний q' ("принять") или q'' ("отвергнуть").

Определение. Пусть C - ленточный алфавит автомата M и $A = C\backslash\{\Lambda\}$. Пусть $L \subseteq A^*$. Будем говорить, что автомат M распознает язык L, если для любого слова $\bar{a} \in A^*$ работа M при входном слове $\bar{a}$ (в стандартной начальной конфигурации) заканчивается состоянием q', если $\bar{a} \in L$, и заканчивается состоянием q'', если $\bar{a} \notin L$.

Определение. Пусть A - некоторый алфавит и $L \subseteq A^*$ - некоторый язык в алфавите A. Для каждого слова $\bar{a} \in A^*$ остаточный язык $L_{\bar{b}}$ определим следующим образом

$$\bar{b} \in L_{\bar{a}} \Longleftrightarrow \bar{a}\bar{b} \in L.$$

Язык называется регулярным, если у него лишь конечное число различных остаточных языков. (Здесь рассматривается и $\bar{b} = \Lambda$ — пустое слово; при этом $\Lambda \in L_{\bar{a}} \Longleftrightarrow \bar{a} \in L$).

В теории автоматов и языков доказывается следующая теорема (см., например, [ ]).

Теорема. *1) Язык, распознаваемый любым автоматом, регулярен. 2) Для любого регулярного языка существует распознающий его автомат.*

Следствие. *Если язык L регулярен, то для него существует распознающая его машина Тьюринга, время работы которой (число шагов) на каждом входном слове длины n равно $n + 1$.*

Оказывается, что не существует языков, для распознавания которых на машинах Тьюринга достаточно времени существенно меньшего, чем $n \log_2 n$ (n - длина входного слова) и не достаточно времени $n + 1$. Более точно это выражается в приводимых ниже теоремах.

Определение. Рассмотрим точку на ленте машины Тьюринга между ячейками с номерами i и $i + 1$. *Следом* в этой точке при работе машины на некотором входном слове будем называть последовательность всех состояний, в которые переходит машина, когда ее головка смещается из ячейки i в ячейку $i + 1$ или наоборот (то есть проходит над этой точкой).

Теорема. *Пусть машина Тьюринга M распознает язык $L \subseteq A^*$ и пусть существует константа $c > 0$ такая, что при работе M на любом входном слове $\bar{a} \in A^*$ длина следа в любой точке не превосходит c. Тогда L — регулярный язык. (Следовательно, существует автомат, распознающий L с $c = 1$).*

Доказательство. Пусть $\bar{a} \in A^*$. Построим множество $D_{\bar{a}}$ всех следов, которые могут получиться при работе M на словах вида $\bar{a}\bar{x} \in A^*$ в точке i, разделяющей $\bar{a}$ и $\bar{x}$. Пусть след $q_{i_1}q_{i_2}q_{i_3}\dots q_{i_s} \in D_{\bar{a}}$. Рассмотрим работу машины M слева от разделяющей точки. Она однозначно определяется словом $\bar{a}$ и теми состояниями $q_{i_2}, q_{i_4}, \dots$, в которых находится машина, когда головка возвращается на левую зону через точку i. По условию $s \leqslant c$. Если s — четно, то машина останавливается слева от точки i. В этом случае к последовательности $q_{i_1}q_{i_2}q_{i_3}$ припишем $+$, если M останавливается в состоянии "принять", и припишем $-$, если M останавливается в состоянии "отвергнуть". Так как $s \leqslant c$, то возможных следов конечное число и разных возможных множеств $D_{\bar{a}}$ также конечное число. Тогда утверждение теоремы вытекает из следующей леммы.

Лемма. *Если $D_{\bar{a}} = D_{\bar{b}}$, то остаточные языки $L_{\bar{a}} = L_{\bar{b}}$ совпадают.*

Доказательство. Пусть $\bar{x}$ - любое слово из A^*. Рассмотрим работу M на словах $\bar{a}\bar{x}$ и $\bar{b}\bar{x}$. Пусть $q_{i_1}q_{i_2}q_{i_3}\dots q_{i_s}$ и $q_{j_1}q_{j_2}\dots q_{j_m}$ — следы в точках, разделяющих $\bar{a}$ и $\bar{x}$, $\bar{b}$ и $\bar{x}$. Заметим, что работа справа от разделяющих точек однозначно определяется словом $\bar{x}$ и состояниями $q_{i_1}q_{i_3}q_{i_5}\dots$ и $q_{j_1}q_{j_3}q_{j_5}\dots$, в которых головка переходит через разделяющие точки вправо. При этом q_{i_1} и q_{j_1} однозначно определяются по $\bar{a}$ и $\bar{b}$. Так как $D_{\bar{a}} = D_{\bar{b}}$, то $q_{i_1} = q_{j_1}$ и работа справа после первого перехода через разделяющие точки происходит одинаково. Тогда $q_{i_2} = q_{j_2}$. Опять, так как $D_{\bar{a}} = D_{\bar{b}}$, то $q_{i_3} = q_{j_3}$ и опять работа справа происходит одинаково. Последовательно получаем, что $s = m$ и $q_{i_r} = q_{j_r}$ для всех $r = 1, 2, \dots, s$. Если s нечетно, то после перехода вправо в состоянии q_{i_s} в обоих случаях работа справа будет одинаковой и, следовательно, M остановится в одном и том же состоянии. Если s четно, то машина в обоих случаях остановится слева от разделяющей точки, причем в одном и том же состоянии, поскольку $D_{\bar{a}} = D_{\bar{b}}$ и след $q_{i_1}q_{i_2}\dots q_{i_s}$ в обоих множествах дополнен одним и тем же знаком $+$ или $-$. Таким образом, M принимает слово $\bar{a}\bar{x}$ тогда и только тогда, когда она принимает слово $\bar{b}\bar{x}$, то есть либо оба слова входят в L, либо оба не входят в L. Поэтому $L_{\bar{a}} = L_{\bar{b}}$. Этим доказана лемма, а вместе с ней и теорема.

Докажем теперь несколько лемм, которые используем в следующей теореме.

Лемма. *Пусть* $A = \{a_1, a_2, \dots, a_r\}$ — *алфавит,* $\bar{b}_1, \bar{b}_2, \dots, \bar{b}_n$ — *разные слова в этом алфавите,* l_i — *длина слова* $\bar{b}_i$ *и* $l_{max} = \max_i l_i$. *Тогда* $l_{max} \geqslant c \log_2 n$, *где* $c > 0$ — *некоторая константа, зависящая только от* r.

Доказательство. Все n слов имеют длину, не превосходящую l_{max}. Но всего таких слов меньше, чем $r^{l_{max}+1}$ (см. лемму). Поэтому $n \leqslant r^{l_{max}+1}$, $l_{max} \geqslant \log_r n - 1 \geqslant c_1 \log_r n = \frac{c_1}{\log_2 r} \log_2 n$. Лемма доказана.

Лемма. *При условиях леммы существует константа* $c > 0$ *такая, что* $\sum_{i=1}^{n} l_i \geqslant cn \log_2 n$.

Доказательство. Число всех слов длины меньшей, чем $\lfloor \frac{1}{2} \log_r n \rfloor$ не превосходит $r^{\lfloor \frac{1}{2} \log_r n \rfloor} \leqslant r^{\frac{1}{2} \log_r n} = \sqrt{n}$. Поэтому среди слов $\bar{b}_i$ не менее, чем $n - \sqrt{n}$ слов, имеют длину не меньше чем $\lfloor \frac{1}{2} \log_r n \rfloor$. Поэтому

$$\sum_{i=1}^{n} l_i \geqslant (n - \sqrt{n}) \lfloor \frac{1}{2} \log_r n \rfloor \geqslant cn \log_2 n$$

для некоторой константы c.

Лемма. *Пусть последовательность* $n_1, n_2, \dots, n_k, \dots$ *не ограничена сверху. Тогда из нее можно выделить подпоследовательность* $n_{i_1}, n_{i_2}, \dots$ *такую, что для любого* s *и всех* $1 \leqslant j < i_s$ *выполняется* $n_j < n_{i_s}$.

Доказательство. Первым элементом подпоследовательности возьмем n_1. Пусть уже выбраны $n_{i_1}, n_{i_2}, \dots, n_{i_r}$. Тогда, просматривая элементы по порядку после n_{i_r}, в качестве очередного элемента выбираем первый элемент $n_{i_{r+1}}$, больший, чем n_{i_r}. Если все элементы n_j исходной последовательности с $j = 1, 2, \dots, i_2 - 1$ меньше, чем n_{i_r}, то, поскольку $n_{i_{r+1}} > n_{i_r}$, а все элементы между n_{i_r} и n_{i_r+1} не превосходят n_{i_r}, то все элементы n_j с $j = 1, 2, \dots, i_{r+1} - 1$ меньше, чем n_{i_r+1}. Таким образом по индукции проверяется требуемое свойство. То, что n_{i_r+1} существует, следует из неограниченности исходной последовательности.

Обозначим через $\xi_M(\bar{a}\bar{b})$ след при работе машины Тьюринга M на слове $\bar{a}\bar{b}$ в точке, разделяющей $\bar{a}$ и $\bar{b}$.

Лемма. *Пусть* $\bar{a}, \bar{b}, \bar{c}$ — *некоторые слова и пусть* $\xi_M(\bar{a}|\bar{b}\bar{c}) = \xi_M(\bar{a}\bar{b}|\bar{c})$. *Тогда при работе* M *на слове* $\bar{a}\bar{c}$ *слева и справа от точки, разделяющей* $\bar{a}$ *и* $\bar{c}$, *машина работает так же, как на соответствующих частях при работе на* $\bar{a}\bar{b}\bar{c}$.

Доказательство. Рассмотрим работу машины M на слове $\bar{a}\bar{b}\bar{c}$ только на 2 частях ленты: слева и справа от $\bar{b}$. Поскольку $\xi_M(\bar{a}|\bar{b}\bar{c}) =$

$\xi_M(\bar{a}\bar{b}|\bar{c})$, то при выбрасывании части ленты, на которой записано $\bar{b}$, и склеивании оставшихся частей корректно "склеиваются" и процессы работы M на этих частях.

Теорема. *Пусть машина Тьюринга M распознает язык $L \subseteq A^*$. Пусть $d_M(n)$ — максимальная длина следов в точках $1, 2, \dots, n$ при работе машины M на всех словах $\bar{a} \subseteq A^*$ длины n, а $T_M(n)$ — максимальное время вычисления (число шагов) машины M на словах длины n из A^*. Тогда если выполняется хотя бы одно из двух условий: а) $d_M(n) = o(\log n)$; б) $T_M(n) = o(n \log n)$, то L — регулярный язык.*

Доказательство теоремы (от противного). Допустим, что L — не регулярный язык. Тогда по теореме $d_M(n)$ неограниченная последовательность. По лемме из нее можно выделить подпоследовательность $n_1, n_2, \dots$ такую, что

$$d_M(n) < d_M(n_i) \tag{14}$$

для всех $n < n_i$ $(i = 1, 2, \dots)$.

Лемма. *Пусть $n_1, n_2, \dots$ удовлетворяют ( ) и $\bar{a}_i$ — слово длины n_i, на котором достигается $d_M(n_i)$. Тогда при работе M на слове $\bar{a}_i$ один и тот же след в точках $1, 2, \dots, n_i$ не может повторяться более чем 2 раза.*

Доказательство. Предположим, что $\bar{a}_i = \bar{a}\bar{b}\bar{c}\bar{d}$ и $\xi_M(\bar{a}|\bar{b}\bar{c}\bar{d}) = \xi_M(\bar{a}\bar{b}|\bar{c}\bar{d}) = \xi_M(\bar{a}\bar{b}\bar{c}|\bar{d})$, где $\bar{a}$, $\bar{b}$, $\bar{c}$ — не пустые слова. При работе M на слове $\bar{a}_i$ есть следы длины $d_M(n_i)$. По крайней мере один такой след либо не лежит внутри $\bar{b}$, либо не лежит внутри $\bar{c}$. Тогда по лемме он сохранится при работе M либо на слове $\bar{a}\bar{c}\bar{d}$, либо на слове $\bar{a}\bar{b}\bar{d}$, но это противоречит тому, что $d_M(n) < d_M(n_i)$ для всех $n < n_i$. Лемма доказана.

Из этой леммы получаем, что при работе M на слове $\bar{a}_i$ в точках $1, 2, \dots, n_i$ имеется не менее $\frac{n_i}{2}$ разных следов. Тогда по леммам $d_M(n_i) \geqslant c \log_2 \frac{n_i}{2}$, и сумма длин этих разных следов, а значит и время работы машины M, не меньше, чем $cn_i \log_2 \frac{n_i}{2}$, где c - некоторая константа. Если выполнено условие а) или б) из теоремы, то получаем противоречие. Следовательно, от противного, получаем, что при выполнении условия а) или б) язык L регулярен. Теорема доказана.

Следствие. *Если $d_M(n) = o(\log n)$ или $T_M(n) = o(n \log n)$, то существует машина Тьюринга (автомат) N, распознающая тот же язык L, для которой $d_N(n) = 1$, $T_N(n) = n + 1$.*

Классы P и NP

Определение. Пусть алгоритм осуществляет преобразование $\varphi : A^* \to B^*$ слов в алфавите A в слова в алфавите B. Тогда этот алгоритм называется полиномиальным (или имеющим полиномиальную сложность), если существует полином $p(n)$ такой, что для любого натурального n время работы алгоритма на любом входном слове длины n не превосходит $p(n)$. (При этом можно считать, что все коэффициенты в $p(n)$ неотрицательны, то есть $p(n)$ возрастающая функция.)

Определение. Задачей распознавания называется любое отображение $\varphi : A^* \to \{\text{“да”}, \text{“нет”}\}$.

С любой задачей распознавания φ можно связать язык $L_\varphi \subseteq A^*$ следующим образом: $\bar{a} \in L_\varphi \iff \varphi : \bar{a} \to$ “да”. И обратно, любой язык можно рассматривать как задачу распознавания.

Определение. Класс P —это класс всех языков (задач распознавания), для каждого из которых существует распознающий алгоритм с полиномиальной сложностью.

Определение. Будем говорить, что язык $L_1 \subseteq A^*$ полиномиально сводится к языку $L_2 \subseteq B^*$, если существует полиномиальный алгоритм (например, машина Тьюринга) $\varphi : A^* \to B^*$, такой что $\varphi(\bar{a}) \in L_2 \iff \bar{a} \in L_1$.

Теорема. *Пусть $L_1 \subseteq A^*$, $L_2 \subseteq B^*$, $L_2 \in P$ и L_1 полиномиально сводится к L_2. Тогда $L_1 \in P$.*

Доказательство. По условию существуют машины Тьюринга M_1 и M_2 такие, что M_1 полиномиально сводит L_1 к L_2, а M_2 с полиномиальной сложностью распознает L_2. Рассмотрим машину Тьюринга $M = M_2(M_1)$. Тогда $M : A^* \to \{\text{“да”}, \text{“нет”}\}$, причем для любого слова $\bar{a} \in A^*$ имеем

$$M(\bar{a}) = \text{“да”} \iff M_2(M_1(\bar{a})) = \text{“да”} \iff M_1(\bar{a}) \in L_2 \iff \bar{a} \in L_1,$$

то есть M распознает язык L_1. По условию время работы (число шагов) машин M_1 и M_2 на входных словах длины n не превосходит $p_1(n)$ и $p_2(n)$, где p_1, p_2 — полиномы. Тогда время работы M на слове $\bar{a}$ длины n не превосходит $p_1(n) + p_2(|M_1(\bar{a})|)$, где $|M_1(\bar{a})|$ — длина слова $M_1(\bar{a})$. Так как машина Тьюринга M_1 на каждом шаге может увеличивать длину слова не более чем на 1, то $|M_1(\bar{a})| \leqslant n + p_1(n)$ и время работы M на $\bar{a}$ не превосходит $p_1(n) + p_2(n + p_1(n)) = p_3(n)$, где p_3 — полином. (Здесь считается, что все коэффициенты в p_2 неотрицательны и, следовательно, $p_2(n)$ — неубывающая функция). Таким образом M распознает язык L_1 с полиномиальной сложностью. Теорема доказана.

Эта теорема позволяет получать полиномиальные алгоритмы для одних задач распознавания из имеющихся полиномиальных алгоритмов для других задач просто путем полиномиального сведения одних задач к другим.

К сожалению, для большинства задач, возникающих на практике, пока не известно, входят ли они в класс P, но почти все такие задачи оказываются в другом классе, который обозначают NP.

Определение. Язык $L \subseteq A^*$ (задача распознавания) входит в класс NP, если и только если существуют алфавит B, полином $q(n)$ и предикат $Q(x, y) : A^* \times B^* \to \{\text{и, л}\}$ такие, что $Q(x, y) \in P$ и для любого слова $\bar{a} \in A^*$ выполняется:

$$\bar{a} \in L \Longleftrightarrow \exists \bar{b} \in B^*(|\bar{b}| \leqslant q(|\bar{a}|) \& Q(\bar{a}, \bar{b}))$$

(здесь $|\bar{a}|$ и $|\bar{b}|$ —длина слов $\bar{a}$ и $\bar{b}$).

Слово $\bar{b}$ называют сертификатом для слова $\bar{a}$, а алгоритм, распознающий предикат $Q(\bar{a}, \bar{b})$, — алгоритмом проверки сертификата. Таким образом, если $\bar{a} \in L$ (в задаче распознавания для входа $\bar{a}$ ответ "да"), то должно существовать быстрое подтверждение для этого, то есть должен существовать подтверждающий это сертификат $\bar{b}$ (небольшой длины) и быстрый способ подтвердить, что это действительно подходящий сертификат. Если же $\bar{a} \notin L$, то по определению в этом случае ничего не требуется. Таким образом, ответы "да" и "нет" здесь не симметричны. Заметим также, что для случая $\bar{a} \in L$ лишь утверждается существование сертификата $\bar{b}$, но ничего не говорится о сложности его нахождения (если в B имеется r букв и $|\bar{a}| = n$, то $|\bar{b}| \leqslant q(n)$ и число таких слов $\bar{b}$ не меньше, чем $r^{q(n)}$, то есть экспоненциально зависит от n).

Рассмотрим примеры языков из NP.

КЛИКА. *Вход:* любой неориентированный граф G и натуральное число k.

Вопрос: Существует ли в графе G клика размера k, то есть k вершин таких, что любая пара из них соединена ребром?

Более строго, мы должны задать входной алфавит A и способ представления графов в этом алфавите. Можно, например, считать, что $A = \{0, 1, ;\}$ и граф задается матрицей смежности (из 0 и 1), которая затем выписывается в одно слово подряд по строкам матрицы с разделителем ; между строками матрицы.

Утверждение. *КЛИКА* $\in NP$.

Доказательство. В качестве сертификата $\bar{b}$ для входа $\bar{a}$ будем брать список из k вершин, составляющих клику. Очевидно, $|\bar{b}| \leqslant |\bar{a}|$.

Предикат Q будет обозначать, что данные вершины задают клику в данном графе и этих вершин ровно k. Для распознавания справедливости такого свойства Q легко построить алгоритм со сложностью, не превосходящей полинома от суммарной длины кода графа $\bar{a}$ и сертификата $\bar{b}$.

ГАМИЛЬТОНОВ ЦИКЛ (ГЦ). *Вход:* любой неориентированный граф G.

Вопрос: Существует ли в графе G гамильтонов цикл, то есть цикл, проходящий через каждую вершину ровно 1 раз?

Утверждение. *ГЦ* $\in NP$.

Доказательство. Сертификатом здесь является последовательность из вершин $v_1, v_2, \ldots, v_m$. Предикат Q выражает утверждение, что в этой последовательности все вершины графа встречаются ровно 1 раз и в графе есть ребра (v_i, v_{i+1}) для всех $i = 1, 2, \ldots, m-1$, а также ребро (v_m, v_1). Для распознавания справедливости такого свойства Q легко построить алгоритм со сложностью, не превосходящей полинома от суммарной длины кода графа $\bar{a}$ и сертификата $\bar{b}$.

Определение. Конъюнктивной нормальной формой (КНФ) называется булева формула вида $F(x_1, \ldots, x_m) = D_1 \& D_2 \& \ldots \& D_k$, где для каждого $j : D_j = t_{j,1} \vee t_{j,2} \vee \ldots \vee t_{j,n_j}$ и все $t_{j,k}$ — либо переменные, либо отрицания переменных. Выражения D_j называют дизъюнктами, а составляющие их $t_{j,k}$ литералами.

ВЫПОЛНИМОСТЬ (ВЫП). *Вход:* любая формула F в виде КНФ.

Вопрос: существует ли набор переменных $(\alpha_1, \ldots, \alpha_m)$, на котором $F(\alpha_1, \ldots, \alpha_m) = 1$ (выполнима ли F)?

Утверждение. *ВЫП* $\in NP$.

Доказательство. Сертификатом для входа F является набор $(\alpha_1, \ldots, \alpha_m)$, на котором $F(\alpha_1, \ldots, \alpha_m) = 1$. Предикат Q выражает тот факт, что данная формула F на данном наборе $(\alpha_1, \ldots, \alpha_m)$ действительно принимает значение 1. Для распознавания справедливости такого свойства Q легко построить алгоритм со сложностью, не превосходящей полинома от суммарной длины кода формулы F и кода набора $(\alpha_1, \ldots, \alpha_m)$.

Еще раз обсудим вопрос о представлении входных данных. Мы не можем, например, включить в алфавит A произвольные переменные, так как их бесконечное число, а любая машина Тьюринга работает лишь с конечными алфавитами. Однако достаточно взять алфавит $A = \{x, 0, 1, \&, \vee, \neg, (,)\}$ и переменную x_i записывать как x с идущим

далее числом i, представленным в двоичной системе счисления. Обратим также внимание на то, что в определении задач распознавания на вход может поступить любое слово в заданном алфавите A. В задаче ВЫП многие такие слова не представляют КНФ. Предполагается, что ответом для таких входных слов является "нет". Аналогично понимаются и другие задачи (например, КЛИКА или ГЦ).

Теорема. $P \subseteq NP$.

Доказательство. Пусть $L \in P$, и $L \subseteq A^*$. Возьмем любой алфавит B и $q(n) = 1$. Предикат $Q(\bar{a}, \bar{b})$ пусть выражает тот факт, что $\bar{a} \in L$ (независимо от $\bar{b}$). Так как $L \in P$, то предикат Q распознается за время, полиномиально зависящее от $|\bar{a}|$, а значит и от $|\bar{a}| + |\bar{b}|$, то есть $Q \in P$. При этом, очевидно, что для любого $\bar{a} \in A^*$ справедливо соотношение

$$\bar{a} \in L \Longleftrightarrow \exists \bar{b} \in B^*(|\bar{b}| \leqslant 1 \& Q(\bar{a}, \bar{b}))$$

(сертификат $\bar{b}$ здесь не зависит от $\bar{a}$). Таким образом, $L \in NP$. Теорема доказана.

Теорема Кука

Определение. Язык L (задача распознавания) называется NP-трудным, если любой язык L_1 из NP полиномиально сводится к L.

В соответствии с теоремой ( ), если язык L является NP-трудным и $L \in P$, то $NP \subseteq P$ и с учетом теоремы ( ) $P = NP$. И обратно, если $P \neq NP$, то $L \notin P$. Таким образом, NP-трудность языка является косвенным свидетельством того, что $L \notin P$ (косвенным потому, что вероятно $P \neq NP$, но это пока не доказано и не опровергнуто).

Определение. Язык L (задача распознавания) называется NP-полным, если $L \in NP$ и L является NP-трудным.

Естественно возникает вопрос о том, существуют ли такие "универсальные" задачи в классе NP, к которым полиномиально сводятся все задачи из NP. Оказывается, что существуют. Первый результат такого рода был установлен С. Куком.

Теорема (С. Кук). *Задача ВЫП (о выполнимости КНФ) является NP-полной.*

Доказательство. Выше уже доказано, что ВЫП $\in NP$. Поэтому надо доказать, что любой язык L из NP полиномиально сводится к ВЫП. Пусть $L \in NP$ и $L \subseteq A^*$. Это означает, что существуют полином $q(n)$, алфавит B и предикат $Q(x, y) : A^* \times B^* \to \{$"и", "л"$\}$ такие, что $Q(x, y) \in P$ и для любого слова $\bar{a} \in A^*$ справедливо

$$\bar{a} \in L \Longleftrightarrow \exists \bar{b}(|\bar{b}| \leqslant q(|\bar{a}|) \& Q(\bar{a}, \bar{b})).$$

Нам надо показать, что L полиномиально сводится к ВЫП. Это означает, что надо построить такое преобразование с полиномиальной сложностью $\varphi : A^* \to C^*$, где C — алфавит задачи ВЫП, что $\bar{a} \in L \Longleftrightarrow \varphi(\bar{a}) = F_{\bar{a}}$ — выполнимая КНФ от некоторых переменных.

Так как $Q(x, y) \in P$, то существует машина Тьюринга M, которая распознает предикат $Q(x, y)$ за время (число шагов), не превосходящее некоторого полинома p_0 от длины входа. Будем считать, что начальная конфигурация машины M_1 является стандартной, то есть пара $(\bar{a}, \bar{b})$ представляется на ленте двумя словами $\bar{a}$ и $\bar{b}$ с одной разделяющей ячейкой, в которой ставится пустой символ Λ, головка обозревает самый левый символ слова $\bar{a}$ и машина находится в начальном состоянии q_1. Тогда время работы M_1 на произвольной паре $(\bar{a}, \bar{b})$ не превышает $p_0(|\bar{a}| + |\bar{b}| + 1)$. Будем считать, что машина M_1 останавливается только в одном из двух заключительных состояний, причем заключительное

состояние q_0 машины M_1 соответствует ответу "да" (какое состояние соответствует ответу "нет" для нас будет не важно).

Пусть дано $\bar{a} \in A^*$ и $|\bar{a}| = n$. Тот факт, что $\bar{a} \in L$, равносилен в соответствии с () тому, что найдется слово $\bar{b} \in B^*$ с длиной $|\bar{b}| \leqslant q(n)$ такое, что машина M_1 начав работу на паре $(\bar{a}, \bar{b})$ придет в состояние $q_0 =$ "да". При этом время работы M_1 на паре $(\bar{a}, \bar{b})$ не превосходит $p_0(n + q(n) + 1) = p(n)$, где p — некоторый полином. Отметим, что можно считать, что во всех полиномах все коэффициенты неотрицательны. Тогда $p(n) \geqslant n + 1 + q(n)$.

Несколько модифицируем программу машины M_1. А именно, если машина M_1 находится в некотором заключительном состоянии и головка обозревает некоторый символ, то пусть машина M_1 оставляет в ячейке тот же символ, головка никуда не сдвигается и машина остается в том же состоянии. То есть реально ничего не происходит, но формально машина продолжает работать бесконечно. Полученную машину обозначим M. Тогда для слова $\bar{a} \in A^*$ длины $|\bar{a}| = n$ имеем: $\bar{a} \in L \Longleftrightarrow \exists \bar{b} \in B^*(|\bar{b}| \leqslant q(n)$ и машина M, запущенная на паре $(\bar{a}, \bar{b})$, в момент времени $p(n)$ будет находиться в состоянии $q_0 =$ "да").

Наша дальнейшая цель — записать правую часть в этой равносильности в виде КНФ $F_{\bar{a}}(x_1, \ldots, x_m)$ от некоторых переменных, так чтобы формула $F_{\bar{a}}$ была выполнима тогда и только тогда, когда эта правая часть истинна. Перепишем ( ) более подробно:

$\bar{a} \in L \Longleftrightarrow \exists \bar{b} \in B^* \exists K_0, K_1, \ldots, K_{p(n)}$ ($|\bar{b}| \leqslant q(n)$ и $K_0, K_1, \ldots, K_{p(n)}$ — конфигурации машины M такие, что K_0 — начальная конфигурация для пары $(\bar{a}, \bar{b})$, состояние в $K_{p(n)}$ есть q_0 и для каждого $j = 0, 1, \ldots, p(n) - 1$ конфигурация K_{j+1} получается из K_j по программе машины M).

Пусть ячейки ленты в M занумерованы целыми числами слева направо и ячейка, с которой головка начинает работать (и с которой начинается слово $\bar{a}$), имеет номер 0. Тогда за $p(n)$ тактов головка не может попасть в ячейки с номерами меньше $-p(n)$ и больше $p(n)$. Поэтому можно считать, что конфигурации $K_0, K_1, \ldots, K_{p(n)}$ определены только на зоне $[-p(n), p(n)]$ ленты.

Пусть машина M имеет ленточный алфавит $D = \{d_0, d_1, \ldots, d_m\}$, где $d_0 = \Lambda$, при этом $A \subseteq D$ и $B \subseteq D$. Пусть $Q = \{q_0, q_1, \ldots, q_l\}$ — множество состояний машины M, причем q_1 — начальное состояние и $q_0 =$ "да". Введем булевские переменные $x_{i,j}^t, y_i^t, z_k^t$, где $i = -p(n), -p(n) + 1, \ldots, p(n); j = 0, 1, \ldots, m; k = 0, 1, \ldots, l; t = 0, 1, \ldots, p(n)$ и придадим им следующий смысл:

$x^t_{i,j}$ = "и" $\Longleftrightarrow$ в i-й ячейке в конфигурации K_t находится символ d_j;

y^t_i = "и" $\Longleftrightarrow$ в конфигурации K_t головка обозревает ячейку с номером i;

z^t_k = "и" $\Longleftrightarrow$ в конфигурации K_t состояние q_k.

Искомую формулу $F_{\bar a}$ мы будем строить как КНФ от всех этих переменных $F_{\bar a}(\{x^t_{i,j}\},\{y^t_i\},\{z^t_k\})$ причем так, чтобы она была выполнима тогда и только тогда, когда правая часть в ( ) истинна. Для этого достаточно, чтобы КНФ $F_{\bar a}$ была истинна на некотором наборе тогда и только тогда, когда: 1) этот набор корректно задает набор конфигураций $K_0, K_1, \ldots, K_{p(n)}$ машины M; 2) при этом конфигурация K_0 является правильной начальной конфигурацией для пары $(\bar a, \bar b)$, где $\bar a$ — заданное слово и $\bar b \in B^*$ — какое-нибудь слово длины не более $q(n)$; 3) в конфигурации $K_{p(n)}$ состояние q_0 = "да"; 4) для каждого $j = 0, 1, \ldots, p(n)-1$ конфигурация K_{j+1} получается из K_j по программе машины M.

Рассмотрим свойство 1). Если задана конфигурация K_t, то по ней однозначно определяются значения переменных $x^t_{i,j}, y^t_i, z^t_k$ при данном t. Обратное неверно, поскольку, например, если сразу 2 переменные x_{i,j_1} и x_{i,j_2} истинны, то это означает, что в конфигурации K_t в ячейке i должны находиться и символ d_{j_1} и символ d_{j_2}. Легко понять, что условие корректного задания конфигураций выражается следующим образом: при каждом t для каждого i ровно одна из переменных $x^t_{i,j}$ = "и"; при каждом t ровно одна из переменных y^t_i = "и" и при каждом t ровно одна из переменных z^t_k = "и".

Пусть $H(v_1, \ldots, v_s)$ — функция алгебры логики, равная 1 тогда и только тогда, когда среди $v_1, \ldots, v_s$ ровно 1 единице.

Лемма. *Функцию $H(v_1, \ldots, v_s)$ можно представить в виде КНФ с длиной (числом литералов) не более s^2.*

Доказательство. Легко проверить, что

$$H(v_1, \ldots, v_s) = (v_1 \vee v_2 \vee \ldots \vee v_s)\&(\&_{i\neq j}(\bar v_i \vee \bar v_j)).$$

Длина этой КНФ равна $s + \frac{s(s-1)}{2} \cdot 2 = s^2$.

Лемма. *Тот факт, что набор переменных $x^t_{i,j}, y^t_i, z^t_k$ корректно задает конфигурации $K_0, K_1, \ldots, K_{p(n)}$, можно выразить в виде КНФ F_1 длины не более $p_1(n)$, где p_1 — некоторый полином.*

Доказательство. Этот факт выражается формулой

$$F_1' = \&^{p(n)}_{t=0}\&^{p(n)}_{i=-p(n)} H(x^t_{i,0}, x^t_{i,1}, \ldots, x^t_{i,m}) \;\&$$
$$\&\; \&^{p(n)}_{t=0} H(y^t_{-p(n)}, y^t_{-p(n)+1}, \ldots, y^t_{p(n)}) \;\&\; \&^{p(n)}_{t=0} H(z^t_0, z^t_1, \ldots, z^t_l). \quad (15)$$

Представляя каждую функцию H с помощью КНФ в соответствии с леммой ( ), получим КНФ F_1 длины

$$(p(n)(2p(n)+1)(m+1)^2+(p(n)+1)(2p(n)+1)^2+(p(n)+1)(l+1)^2 \leqslant p_1(n),$$

где $p_1(n)$ — некоторый полином.

Лемма. *При условии, что набор переменных $x^0_{i,j}, y^0_i, z^0_k$ корректно задает конфигурацию K_0, тот факт, что K_0 является правильной начальной конфигурацией для пары $(\bar a, \bar b)$, где $\bar a$ — заданное слово и $\bar b \in B^*$ — какое-нибудь слово длины не более $q(n)$, можно выразить в виде КНФ F_2 длины не более $p_2(n)$, где p_2 — некоторый полином.*

Доказательство. Пусть $\bar a = d_{j_1}d_{j_2}\dots d_{j_n}$ и $\bar b \in B^*$, где $B = \{d_{r_1}, d_{r_2}, \dots, d_{r_w}\}$. Тогда указанный в лемме факт выражается формулой

$$\begin{aligned} F_2 = x^0_{0,j_1}\&x^0_{1,j_2}\&\dots\&x^0_{n-1,j_n}\&x^0_{n,0}\ \& \\ \&\ (\&^{-1}_{i=-p(n)}x^0_{i,0})\ \&\ \&^{n+q(n)}_{i=n+1}(x^0_{i,r_1}\vee x^0_{i,r_2}\vee\dots\vee x^0_{i,r_w})\ \& \\ \&\ (\&^{p(n)}_{i=n+q(n)+1}x^0_{i,0})\ \&\ \&^{n+q(n)-1}_{i=n+1}(\bar x^0_{i,0}\vee x^0_{i+1,0}). \end{aligned} \quad (16)$$

Последняя скобка $\bar x^0_{i,0}\vee x^0_{i+1,0} = \bar x^0_{i,0} \to x^0_{i+1,0}$ означает, что если в ячейке i стоит пустой символ, то и в следующей ячейке должен стоять пустой символ, то есть слово $\bar b$ не может иметь разрывов. Формула F_2 является КНФ с длиной

$$n+1+p(n)+q(n)\cdot w+p(n)-n-q(n)-1+(q(n)-1)\cdot 2 \leqslant p_2(n),$$

где p_2 — некоторый полином.

Следующее утверждение очевидно.

Лемма. *Тот факт, что в $K_{p(n)}$ состояние q_0, выражается в виде КНФ $F_3 = z_0^{p(n)}$.*

Рассмотрим теперь более подробно программу машины M. Представим ее команды в виде $a_jq_k \to a_{\varphi(j,k)}q_{\psi(j,k)}R(j,k)$, где $R(j,k) = -1, 0$ или 1 соответственно, для сдвига влево, оставления головки на месте и сдвига вправо.

Лемма. *При условии, что набор переменных $x^t_{i,j}, y^t_i, z^t_k$ корректно задает конфигурации $K_0, K_1, \dots, K_{p(n)}$, тот факт, что каждая конфигурация K_{j+1} получается из K_j по программе машины M, можно выразить в виде КНФ F_4 длины не более $p_4(n)$, где p_4 — некоторый полином.*

Доказательство. Этот факт выражается формулой

$$F_4' = \&_{t=0}^{p(n)-1}\&_{i=-p(n)}^{p(n)}\&_{j=0}^{m}\&_{k=0}^{l}(y_i^t\&x_{i,j}^t\&z_k^t \to \\ \to x_{i,\varphi(j,k)}^{t+1}\&z_{\psi(j,k)}^{t+1}\&y_{i+R(j,k)}^{t+1}) \& \\ \& \&_{t=0}^{p(n)-1}\&_{i=-p(n)}^{p(n)}(\bar{y}_i^t \to \&_{j=0}^{m}(x_{i,j}^{t+1} \equiv x_{i,j}^t)). \quad (17)$$

Первая часть этой формулы выражает изменение информации в обозреваемой ячейке, изменение состояния и сдвиг головки, вторая часть выражает тот факт, что символы во всех ячейках, кроме обозреваемой, не изменяются. Выражение в первой скобке в F_4' — это функция от 6 переменных и ее (как любую функцию алгебры логики, отличную от константы 1) можно представить в виде КНФ некоторой длины L_1. Аналогично можно представить в виде КНФ константной длины L_2 функцию от $2m+1$ переменных, стоящую во второй скобке. При этом F_4' преобразуется в КНФ K_4 длины $p(n)\cdot(2p(n)+1)\cdot m\cdot l\cdot L_1 + p(n)\cdot(2p(n)+1)\cdot L_2 \leqslant p_4(n)$, где p_4 — некоторый полином.

Положим $F_{\bar{a}} = F_1\cdot F_2\cdot F_3\cdot F_4$. Тогда $F_{\bar{a}}$ — КНФ и по леммам на наборе переменных $x_{i,j}^t, y_i^t, z_k^t$ она истинна тогда и только тогда, когда переменные корректно задают некоторое вычисление машины M, приводящее в состояние q_0 = "да", для входной пары $(\bar{a},\bar{b})$, где $\bar{b}$ — какое-нибудь слово из B^* такое, что $|\bar{b}| \leqslant q(|\bar{a}|)$. Таким образом $F_{\bar{a}}$ истинна хотя бы на одном наборе (т.е. выполнима) в том и только в том случае, если истинна правая часть в (1) и, следовательно, если $\bar{a}\in L$. Получаем, что $F_{\bar{a}}$ — искомая КНФ. Ее длина не превосходит некоторого полинома от n. При этом нетрудно понять, что по данному слову $\bar{a}$ (и фиксированной программе машины M) эта КНФ $F_{\bar{a}} = F_1\cdot F_2\cdot F_3\cdot F_4$ выписывается за время, ограниченное полиномом от ее длины, и, следовательно, ограниченное полиномом от длины слова $\bar{a}$. Таким образом, отображение $\bar{a}\to F_{\bar{a}}$ является полиномиальным сведением языка L к языку ВЫП. Поскольку L — произвольный язык из NP, то получаем, что ВЫП — NP-трудная задача, а так как ВЫП $\in NP$, то ВЫП — NP-полная задача. Теорема Кука доказана.

NP-полные задачи

Следующая теорема позволяет выводить NP-полноту одних задач из NP-полноты других задач.

Теорема. *Если L_1 — NP-трудный язык и L_1 полиномиально сводится к языку L_2, то L_2 — NP-трудный язык. Если при этом $L_2 \in NP$, то L_2 — NP-полный язык.*

Доказательство. Пусть L — любой язык из NP. Так как L_1 — NP-трудный язык, то L полиномиально сводится к L_1. Так как по условию L_1 полиномиально сводится к L_2, то и L полиномиально сводится к L_2. Так как L — произвольный язык из NP, то L_2 — NP-трудный язык по определению.

Определение. КНФ, у которой в каждом дизъюнкте ровно 3 литерала, будем называть 3-КНФ.

Задача 3-выполнимость (**3-ВЫП**).

Входной алфавит тот же, что и в задаче ВЫП.

Вопрос: верно ли, что входное слово — это 3-КНФ, которая выполнима.

Утверждение. *3-ВЫП $\in NP$.*

Доказательство. Задача 3-ВЫП удовлетворяет определению задач из класса NP. При этом в качестве сертификата достаточно взять набор $\tilde{\alpha}$, на котором данная 3-КНФ выполнима (если такой существует), а алгоритм проверки сертификата будет проверять, действительно ли входное слово есть 3-КНФ и верно ли, что эта КНФ на наборе $\tilde{\alpha}$ равна 1. Все это можно осуществить за полиномиальное (от длины входа) время.

Теорема. *Задача 3-ВЫП NP-полна.*

Доказательство. Сведем задачу ВЫП полиномиально к задаче 3-ВЫП. Пусть A — алфавит обеих задач. Нам надо для каждого слова $\bar{a} \in A^*$ за полиномиальное (от длины слова $\bar{a}$) время построить слово $\varphi(\bar{a})$ так, чтобы $\varphi(\bar{a})$ было выполнимой 3-КНФ тогда и только тогда, когда $\bar{a}$ — выполнимая КНФ. Если $\bar{a} \in A^*$ не КНФ, то положим $\varphi(\bar{a}) = \bar{a}$. Если же $\bar{a}$ — КНФ $D_1 \cdot D_2 \cdot \ldots \cdot D_s$ от переменных $x_1, x_2, \ldots, x_n$, то преобразуем ее в 3-КНФ $\varphi(\bar{a}) = F_1 \cdot F_2 \cdot \ldots \cdot F_s$ следующим образом. Пусть $Y = \{y_1, y_2, \ldots\}$ — некоторые переменные, которые не встречаются в КНФ $\bar{a}$. Рассмотрим 4 случая.

1) Если $D_i = t_{i,1} \vee t_{i,2} \vee t_{i,3}$, то положим $F_i = D_i$.

2) Если $D_i = t_{i,1} \vee t_{i,2}$, то положим $F_i = (t_{i,1} \vee t_{i,2} \vee y_j) \cdot (t_{i,1} \vee t_{i,2} \vee \bar{y}_j)$, где $y_j \in Y$. Заметим, что $F_i = 1 \Longleftrightarrow D_i = 1$.

3) Если $D_i = t_i$, то положим

$$F_i = (t_i \vee y_k \vee y_l)(t_i \vee y_k \vee \bar{y}_l) \cdot (t_i \vee \bar{y}_k \vee y_l)(t_i \vee \bar{y}_k \vee \bar{y}_l),$$

где $y_k \neq y_l$. Опять $F_i = 1 \Longleftrightarrow D_i = 1$.

4) Пусть $D_i = t_1 \vee t_2 \vee \ldots \vee t_m$ и $m \geqslant 4$.Положим

$$F_i = (t_1 \vee t_2 \vee y_1)(\bar{y}_1 \vee t_3 \vee y_2)(\bar{y}_2 \vee t_4 \vee y_3) \cdot \ldots$$
$$\ldots \cdot (\bar{y}_{m-4} \vee t_{m-2} \vee y_{m-3})(\bar{y}_{m-3} \vee t_{m-1} \vee t_m), \quad (18)$$

где все y_j различны.

Лемма. *Если $F_i = 1$, то и $D_i = 1$. Если $D_i = 1$, то существует набор значений переменных $y_1, y_2, \ldots, y_{m-3}$ такой, что $F_i = 1$.*

Доказательство. Пусть $\tilde{\alpha} = (\alpha_1, \ldots, \alpha_n)$—набор значений переменных $x_1, \ldots, x_n$ из D_i и $\tilde{\beta} = (\beta_1, \ldots, \beta_{m-3})$—набор значений переменных $y_1, \ldots, y_{m-3}$. Пусть $F_i(\tilde{\alpha}, \tilde{\beta}) = 1$, то есть все скобки в F_i на наборе $(\tilde{\alpha}, \tilde{\beta})$ равны 1. Если $\beta_1 = 0$, то $t_1(\tilde{\alpha}) \vee t_2(\tilde{\alpha}) = 1$ и $D_i(\tilde{\alpha}) = 1$. Если $\beta_{m-3} = 1$, то $t_{m-1}(\tilde{\alpha}) \vee t_m(\tilde{\alpha}) = 1$ и $D_i(\tilde{\alpha}) = 1$. Если же $\beta_1 = 1$ и $\beta_{m-3} = 0$, то найдется k такое, что $\beta_k = 1$, $\beta_{k+1} = 0$. Так как $\bar{\beta}_k \vee t_{k+2} \vee \beta_{k+1} = 1$, то в этом случае $t_{k+2}(\tilde{\alpha}) = 1$ и $D_i(\tilde{\alpha}) = 1$. Следовательно, если $F_i = 1$, то и $D_i = 1$. Обратно пусть $D_i(\tilde{\alpha}) = 1$. Тогда существует t_k такое, что $t_k(\tilde{\alpha}) = 1$. Если $k = 1$ или $k = 2$, то выберем $\beta_1 = \beta_2 = \ldots = \beta_{m-3} = 0$. При этом $F_i(\tilde{\alpha}, \tilde{\beta}) = 1$. Если $k = m - 1$ или $k = m$, то выберем $\beta_1 = \beta_2 = \ldots = \beta_{m-3} = 1$. При этом опять $F_i(\tilde{\alpha}, \tilde{\beta}) = 1$. В остальных случаях положим $\beta_1 = \beta_2 = \ldots = \beta_{k-2} = 1$, $\beta_{k-1} = \beta_k = \ldots = \beta_{m-3} = 0$. Снова получим $F_i(\tilde{\alpha}, \tilde{\beta}) = 1$. Лемма доказана.

Проделаем указанные выше в пунктах 2)-4) замены D_i на F_i, используя для разных D_i разные переменные y_j. Тогда по лемме получаем, что если 3-КНФ $F_1 \cdot F_2 \cdot \ldots \cdot F_s$ равна 1 на каком-то наборе, то на том же наборе равна 1 и КНФ $D_1 \cdot D_2 \cdot \ldots \cdot D_s$, и обратно, если КНФ $D_1 \cdot D_2 \cdot \ldots \cdot D_s$ равна 1 на некотором наборе, то существует набор, на котором 3-КНФ $F_1 \cdot F_2 \cdot \ldots \cdot F_s$ также равна 1. Таким образом 3-КНФ $F_1 \cdot F_2 \cdot \ldots \cdot F_s$ выполнима тогда и только тогда, когда выполнима КНФ $D_1 \cdot D_2 \cdot \ldots \cdot D_s$, то есть наше преобразование является сведением задачи ВЫП к задаче 3-ВЫП.

Распознать, является ли входное слово $\bar{a} \in A^*$ КНФ можно за полиномиальное от длины $\bar{a}$ время. Преобразовать все D_i в F_i также можно за полиномиальное время. Поэтому мы имеем полиномиальное сведение задачи ВЫП к задаче 3-ВЫП. Поскольку задача ВЫП является NP-полной и 3-ВЫП $\in NP$, то по теореме получаем, что задача 3-ВЫП является NP-полной.

Посмотрим, нельзя ли в последней теореме заменить 3 на 2.

Определение. КНФ, у которой в каждом дизъюнкте не более 2 литералов, будем называть 2-КНФ.

Задача 2-выполнимость (**2-ВЫП**).

Входной алфавит тот же, что и в задаче ВЫП.

Вопрос: верно ли, что входное слово — это 2-КНФ, которая выполнима.

Теорема. *Для задачи 2-ВЫП существует алгоритм с полиномиальной сложностью (то есть 2-ВЫП $\in P$).*

Доказательство. Проверить, является ли входное слово 2-КНФ, можно за полиномиальное время. Поэтому будем считать, что нам уже дана 2-КНФ $D_1 \cdot D_2 \cdot \ldots \cdot D_s$ и требуется выяснить, выполнима ли она. Пусть дизъюнкты $D' = x_i \vee t_1$ и $D'' = \bar{x}_i \vee t_2$ имеют противоположные литералы x_i и $\bar{x}_i$ (при этом может быть $t_1 = \varnothing$ или $t_2 = \varnothing$). Тогда резольвентой дизъюнктов D' и D'' по x_i будем называть дизъюнкт $D = t_1 \vee t_2$ (если $t_1 = t_2$, то $t_1 \vee t_2 = t_1$). Если $t_1 = \varnothing$ и $t_2 = \varnothing$, то положим $D \equiv 0$.

Лемма. *Для любых формул A и B выполняется равенство*

$$(x_i \vee A)(\bar{x}_i \vee B) = (x_i \vee A)(\bar{x}_i \vee B)(A \vee B).$$

Доказательство. Если правая часть равна 1, то, очевидно, и левая часть равна 1. Если правая часть равна 0, то либо $x_i \vee A = 0$, либо $\bar{x}_i \vee B = 0$, либо $A \vee B = 0$. В первых двух случаях левая часть равна 0. В последнем случае $A = 0$ и $B = 0$. Тогда левая часть равна $(x_i \vee 0)(\bar{x}_i \vee 0) = x_i\bar{x}_i = 0$. Лемма доказана.

Лемма ( ) показывает, что добавление к 2-КНФ резольвенты любой пары дизъюнктов не меняет функцию, реализуемую 2-КНФ. Будем просматривать все пары дизъюнктов текущей 2-КНФ и строить их резольвенты. Если окажется, что некоторая резольвента отсутствует в текущей 2-КНФ, то добавим ее и начнем просмотр сначала. Так будем делать до тех пор, пока не окажется, что все резольвенты текущей 2-КНФ уже содержатся в ней.Если при этом будет порождена резольвента 0, то выдаем ответ: “Исходная 2-КНФ невыполнима”, иначе выдаем ответ: “Исходная 2-КНФ выполнима”.

Лемма. *Алгоритм обязательно остановится и работает полиномиальное время.*

Доказательство. Если длина исходной 2-КНФ равна n, то в ней не более n переменных, из которых можно построить не более $(n+1)^2$ дизъюнктов с одной или двумя переменными. Поэтому поиск новых резольвент будет повторяться не более $(n+1)^2$ раз, при этом число

просматриваемых пар дизъюнктов не превосходит $(n+1)^4$. Отсюда следует утверждение леммы.

Лемма. *Алгоритм правильно решает задачу 2-ВЫП.*

Доказательство. 1) Пусть $K = D_1 D_2 \cdot \ldots \cdot D_s$— исходная 2-КНФ и пусть в конечной КНФ K' есть сомножитель 0. По лемме $K = K'$ и, следовательно, $K \equiv 0$, то есть K невыполнима. 2) Пусть в конечной 2-КНФ K' нет 0. По построению K' замкнута относительно взятия резольвент, то есть резольвента любых двух дизъюнктов из K' снова содержится в K'. Докажем, что любая 2-КНФ с таким свойством выполнима. Доказательство проведем индукцией по числу переменных n в 2-КНФ K'.

Базис индукции: $n = 1$. Тогда $K' = x_i$ или $K' = x_i'$. В любом случае K' выполнима.

Индуктивный переход. Пусть утверждение верно для 2-КНФ с $n < m$ переменными и пусть в K' имеется m переменных $x_1, x_2, \ldots, x_m$. Тогда

$$K' = (x_m \vee t_1)(x_m \vee t_2) \cdot \ldots \cdot (x_m \vee t_k)(\bar{x}_m \vee t_1')(\bar{x}_m \vee t_2') \cdot \ldots \cdot (\bar{x}_m \vee t_l') \cdot C_1 \cdot C_2 \cdot \ldots \cdot C_r, \quad (19)$$

где t_i, t_j' — литералы, либо 0, и $C_1 \cdot C_2 \cdot \ldots \cdot C_r$ — 2-КНФ с переменными $x_1, x_2, \ldots, x_{m-1}$, замкнутая относительно взятия резольвент. По предположению индукции существует набор $\tilde{\alpha} = (\alpha_1, \ldots, \alpha_{m-1})$, на котором $C_1 \cdot C_2 \cdot \ldots \cdot C_r$ равно 1. Если бы существовали t_i и t_j' такие, что $t_i(\tilde{\alpha}) = 0$ и $t_j'(\tilde{\alpha}) = 0$, то было бы и $t_i(\tilde{\alpha}) \vee t_j'(\tilde{\alpha}) = 0$. Но $t_i \vee t_j'$ — резольвента дизъюнктов $x_m \vee t_i$ и $\bar{x}_m \vee t_j'$. Так как 2-КНФ K' замкнута относительно взятия револьвент, то $t_i \vee t_j'$ содержится среди $C_1, C_2, \ldots, C_r$. Но это противоречит тому, что $C_v(\tilde{\alpha}) = 1$ при всех v. Следовательно, либо все $t_i(\tilde{\alpha}) = 1$, либо все $t_j'(\tilde{\alpha}) = 1$. В первом случае K' выполнима на наборе $(\tilde{\alpha}, 0)$, во втором — на наборе $(\tilde{\alpha}, 1)$.

Теорема полностью доказана.

Теорема. *Задача КЛИКА является NP-полной.*

Доказательство. Покажем, что задача ВЫП полиномиально сводится к задаче КЛИКА. Для этого каждому слову $\bar{a}$ в алфавите языка ВЫП сопоставим пару $\varphi(\bar{a}) = (G, k)$, где G — некоторый граф и k — натуральное число, так, чтобы в G существовала клика с k вершинами тогда и только тогда, когда $\bar{a}$ — выполнимая КНФ. Если $\bar{a}$ — не КНФ, то положим $\varphi(\bar{a}) = (G_2, 2)$, где G_2 — граф с 2 вершинами и без ребер (очевидно, в G_2 нет клики с 2 вершинами). Пусть теперь $\bar{a}$ — КНФ и $\bar{a} = D_1 \cdot D_2 \cdot \ldots \cdot D_s$, где $D_i = t_{i,1} \vee t_{i,2} \vee \ldots \vee t_{i,m_i}$ — дизъюнкты. Построим граф $\varphi(\bar{a}) = G_{\bar{a}} = (V, E)$ с множеством вершин V и множеством ребер E следующим образом. Каждому литералу $t_{i,j}$ из $\bar{a}$ сопоставим вершину $v_{i,j}$ и будем считать, что

$$(v_{i_1,j_1}, v_{i_2,j_2}) \in E \Longleftrightarrow (i_1 \neq i_2) \text{ и } (t_{i_2,j_2} \neq \bar{t}_{i_1,j_1}).$$

Положим $k = s$, где s — число дизъюнктов в $\bar{a}$.

Лемма. *В $G_{\bar{a}}$ есть клика с s вершинами тогда и только тогда, когда КНФ $\bar{a}$ выполнима.*

Доказательство. Пусть КНФ $\bar{a}$ принимает значение 1 на наборе $\tilde{\alpha}$. Тогда $D_i(\tilde{\alpha}) = 1$ для всех i. Следовательно, для каждого i существует j_i такое, что $t_{i,j_i} = 1$. Тогда среди $t_{1,j_1}, t_{2,j_2}, \ldots, t_{s,j_s}$ нет противоположных литералов. Поэтому любые вершины из $v_{1,j_1}, v_{2,j_2}, \ldots, v_{s,j_s}$ соединяются в $G_{\bar{a}}$ ребром, то есть образуют клику с s вершинами.

Обратно, пусть в графе $G_{\bar{a}}$ есть клика с s вершинами $v_{i_1,j_1}, v_{i_2,j_2}, \ldots, v_{i_s,j_s}$. Тогда $i_1, i_2, \ldots, i_s$ все различны, то есть литералы из семейства $M = \{t_{i_1,j_1}, t_{i_2,j_2}, \ldots, t_{i_s,j_s}\}$ входят по одному в каждый дизъюнкт КНФ $\bar{a}$, причем среди этих литералов нет противоположных. Пусть $x_1, x_2 \ldots, x_n$ — все переменные из $\bar{a}$. Для каждого k положим $x_k = 1$, если литерал x_k встречается в M, $x_k = 0$, если $\bar{x}_k$ встречается в M, и x_k — любое, если ни x_k, ни $\bar{x}_k$ нет в M. Тогда на построенном наборе $\tilde{\alpha}$ все литералы из M обращаются в 1 и, следовательно, все дизъюнкты и вся КНФ $\bar{a}$ принимают значение 1, то есть КНФ $\bar{a}$ выполнима. Лемма доказана.

Таким образом переход $\bar{a} \to \varphi(\bar{a})$ является сведением задачи ВЫП к задаче КЛИКА. Распознать, является ли $\bar{a}$ КНФ, и построить по $\bar{a}$ граф $G_{\bar{a}}$ и число k можно за полиномиальное время. Поэтому это полиномиальное сведение. Так как задача ВЫП NP-полна и КЛИКА $\in NP$, то по теореме получаем, что и задача КЛИКА NP-полна.

Задача о независимом множестве вершин (**НМ**).

Вход: пара (G, k), где G — граф, k — натуральное число.

Вопрос: существуют ли в G k вершин, образующих независимое множество, то есть множество, в котором никакие вершины не соединены ребром в G?

Лемма. *$НМ \in NP$.*

Доказательство. Сертификатом является само независимое подмножество M с k вершинами (если такое есть). Проверить, что M содержит ровно k вершин и является независимым, можно за полиномиальное время.

Задача о вершинном покрытии (**ВП**).

Вход: пара (G, k), где G — граф, k — натуральное число.

Вопрос: существует ли в G множество M из k вершин, образующих вершинное покрытие, то есть такое, что любое ребро из G имеет хотя бы один конец в M?

Лемма. *$ВП \in NP$.*

Доказательство. Сертификатом является само вершинное покрытие M с k вершинами (если такое есть). Проверить, что M содержит ровно k вершин и является вершинным покрытием, можно за полиномиальное время.

Теорема. *Задачи НМ и ВП NP-полны.*

Доказательство. Сопоставим паре (G, k) пару $(\bar{G}, k)$, где $\bar{G}$—граф, являющийся дополнением к G (то есть в $\bar{G}$ то же множество вершин V, что и в G, и две вершины соединяются ребром в $\bar{G}$ тогда и только тогда, когда они не соединяются ребром в G). При этом подмножество $M \subseteq V$ является кликой с k вершинами в G тогда и только тогда, когда M является независимым множеством с k вершинами в $\bar{G}$. Получаем сведение (очевидно, полиномиальное) задачи КЛИКА к задаче НМ. Так как задача КЛИКА NP-полна и $НМ \in NP$, то задача НМ NP-полная. Сопоставим паре (G, k) пару $(G, n - k)$, где $n = |V|$—число вершин в графе G. При этом подмножество $M \subseteq V$ является независимым множеством с k вершинами в G тогда и только тогда, когда $V \setminus M$ является вершинным покрытием с $n - k$ вершинами в G. Получаем полиномиальное сведение задачи НМ к задаче ВП. Так как задача НМ NP-полная и $ВП \in NP$, то и задача ВП NP-полная.

Определение. Цикл в графе, проходящий через каждую вершину ровно 1 раз, называется гамильтоновым. Гамильтоновой цепью называется незамкнутая цепь, проходящая через каждую вершину ровно 1 раз.

Задача о гамильтоновом цикле (**ГЦ**).

Вход: произвольный граф G.

Вопрос: есть ли в G гамильтонов цикл?

Лемма. *ГЦ* $\in NP$.

Доказательство. Для данного графа G с n вершинами сертификатом является последовательность вершин $(v_1, v_2, \ldots, v_n)$. Алгоритм проверки сертификата должен убедиться, что в этом списке столько же вершин, сколько в графе G, что все они различны и что для всех $j = 1, 2, \ldots, n-1$ в графе G есть ребро (v_j, v_{j+1}), а также есть ребро (v_n, v_1). Все это можно выполнить за полиномиальное время.

Теорема. *Задача ГЦ NP-полна.*

Доказательство. Построим полиномиальное сведение задачи 3-ВЫП к задаче ГЦ. Рассмотрим 2 вспомогательных графа: α-граф и β-граф, изображенные на рис. Пусть α-граф является подграфом некоторого графа G, причем с другими вершинами графа могут соединяться только вершины A_1, A_2, B_1, B_2, и пусть в G существует гамильтонов цикл. Нетрудно проверить, что если этот цикл входит внутрь α-графа, то он обязан сразу обойти все внутренние вершины α-графа. При этом, если он входит через вершину A_1, то выходит обязательно через B_1 и наоборот. Аналогично, если он входит через A_2, то выходит через B_2 и наоборот. Поэтому условно можно считать, что α-граф имеет всего 2 ребра A_1B_1 и A_2B_2 и требуется, чтобы цикл проходил ровно по одному из них.

Пусть β-граф (рис.) является подграфом некоторого графа G, причем с другими вершинами графа G могут соединяться только вершины P и S. Если гамильтонов цикл входит в β-граф через P или S, то он должен сразу обойти все вершины этого β-графа (и выйти через другую вершину пары P, S).

В β-графе (рис.) 3 правых ребра PQ, QR и RS будем называть основными, а вершины P и S — граничными.

Лемма. *Не существует гамильтоновой цепи в β-графе из вершины P в вершину S, проходящей по всем трем основным ребрам PQ, QR, RS. Для любого другого подмножества основных ребер (включая пустое подмножество) существует гамильтонова цепь из P в S, проходящая в точности по этому подмножеству основных ребер.*

Первая часть этой леммы очевидна, для доказательства второй части достаточно рассмотреть все возможные случаи и в каждом построить соответствующую гамильтонову цепь (постройте).

Пусть A — алфавит задачи 3-ВЫП. Каждому слову $\bar{a} \in A^*$ мы сопоставим граф G так, чтобы в G существовал гамильтонов цикл

тогда и только тогда, когда $\bar{a}$ — выполнимая 3-КНФ. Если $\bar{a} \in A^*$ и $\bar{a}$ — не 3-КНФ, то сопоставим $\bar{a}$ граф G с двумя вершинами и 1 ребром. Очевидно, в нем нет гамильтонова цикла. Пусть теперь $\bar{a} = K = D_1 \cdot D_2 \cdot \ldots \cdot D_s$ — произвольная 3-КНФ с переменными $x_1, x_2, \ldots, x_n$. Пусть $H_1, H_2, \ldots, H_s$ — β-графы с граничными вершинами P_j, S_j. Соединим ребрами вершины S_j и P_{j+1} для $j = 1, 2, \ldots, s-1$. Полученный граф обозначим G_1. Через G_2 обозначим граф с вершинами $C_0, C_1, \ldots, C_n$, в котором для каждого i есть 2 ребра (C_{i-1}, C_i) и нет других ребер. Вершину P_1 графа G_1 соединим ребром с C_0, а S_s соединим ребром с C_n. Из двух ребер (C_{i-1}, C_i) одно сопоставим переменной x_i, а другое — $\bar{x}_i$. Первое обозначим e_i^1, второе e_i^0. В каждом подграфе H_j основные ребра P_jQ_j, Q_jR_j, R_jS_j сопоставим литералам $t_{j,1}, t_{j,2}, t_{j,3}$ дизъюнкта D_j. Пусть литерал x_i встречается в k дизъюнктах D_j и в подграфах H_j литералу x_i соответствуют k ребер $e_1, e_2, \ldots, e_k$. Между ребром $e_i^1 = (C_{i-1}, C_i)$, соответствующим x_i, и каждым из ребер $e_1, e_2, \ldots, e_k$ вставим соединительные ребра так, чтобы образовалось k α-графов. Так поступим для всех x_i. Аналогично поступим для всех $\bar{x}_i$ с заменой e_i^1 на e_i^0. Полученный граф обозначим G_k.

Лемма. *В G_k есть гамильтонов цикл тогда и только тогда, когда КНФ K выполнима.*

Доказательство. Пусть в G_k существует гамильтонов цикл W. По свойствам α-графа (см.выше) можно условно считать, что гамильтонов цикл проходит ровно по одному из ребер A_1B_1 или A_2B_2 α-графа. Следовательно, можно считать, что цикл W сначала проходит по всем вершинам подграфа G_1, потом по всем вершинам подграфа G_2, при этом выполняется требуемое свойство для каждого α-графа. Для каждой пары ребер e_i^0, e_i^1 в G_2 цикл W, очевидно, должен проходить ровно по одному из этих ребер. Для каждого i положим $x_i = 0$, если W проходит по e_i^0, и $x_i = 1$, если W проходит по e_i^1. Полученный набор обозначим $\tilde{\alpha} = (\alpha_1, \ldots, \alpha_n)$. Рассмотрим один из подграфов H_j в G_1. По лемме гамильтонов цикл W не проходит по крайней мере по одному из основных ребер подграфа H_j. Пусть этому ребру соспоставлен литерал t с переменной x_i. Если $t = x_i$, то это ребро соединено α-графом с e_i^1, если $t = \bar{x}_i$, то с e_i^0. Так как по данному ребру цикл W не проходит, он должен проходить по e_i^1, если $t = x_i$, и по e_i^0, если $t = \bar{x}_i$. Из выбора α_i получаем, что в любом случае $t(\tilde{\alpha}) = 1$ и, следовательно, $D_j(\tilde{\alpha}) = 1$. Поскольку это верно для всех j, то $K(\tilde{\alpha}) = 1$, то есть КНФ K выполнима.

Обратно, пусть КНФ K выполнима и $K(\tilde{\alpha}) = 1$, где $\tilde{\alpha} =$

$(\alpha_1, \ldots, \alpha_n)$. Проведем цикл W по подграфу G_2 так, чтобы для каждого $i = 1, 2, \ldots, n$ он проходил по e_i^0, если $\alpha_i = 0$, и по e_i^1, если $\alpha_i = 1$. Тогда по свойствам α-графов цикл W не может проходить по основному ребру подграфа G_1, которому приписан литерал t, такой, что $t(\tilde{\alpha}) = 1$, и обязан проходить по основному ребру, если ему приписан литерал t, такой, что $t(\tilde{\alpha}) = 0$. Так как $D_j(\tilde{\alpha}) = 1$ для всех j, то в каждом подграфе H_j существует хотя бы одно ребро, такое, что для соответствующего ему литерала t_j выполняется $t_j(\tilde{\alpha}) = 1$. Следовательно в каждом подграфе H_j существует одно, два или три основных ребра, таких, что цикл W не должен по ним проходить, и должен проходить по остальным основным ребрам. По лемме в каждом H_j можно построить гамильтонову цепь, удовлетворяющую этим требованиям, и в целом в графе G_k можно построить гамильтонов цикл. Лемма доказана.

Проверить, является ли слово $\bar{a} \in A^*$ 3-КНФ, и построить граф G_k, если $\bar{a} = K$ — это 3-КНФ, можно за полиномиальное (от длины $\bar{a}$) время. Поэтому мы получаем полиномиальное сведение задачи 3-ВЫП к задаче ГЦ. Так как задача 3-ВЫП NP-полна и ГЦ $\in NP$, то и задача ГЦ NP-полна.

Задачи оптимизации

В практических приложениях часто возникают задачи оптимизации, которые имеют следующую структуру. Каждому входу x сопоставляется некоторое множество Y_x допустимых решений. Задан функционал $F : Y_x \to R$, где R — множество действительных чисел. Требуется найти $\min_{y \in Y_x} F(y)$, или $\max_{y \in Y_x} F(y)$, или то допустимое решение y_0, на котором достигается оптимальное значение функционала. Если функционал F вычисляется быстро, то найдя оптимальное допустимое решение, мы можем легко получить и оптимальное значение функционала $F_{\text{опт}}$. Обратное, вообще говоря, не ясно: может существовать быстрый алгоритм, который находит $F_{\text{опт}}$, не находя оптимального решения.

С каждой задачей оптимизации можно связать задачу распознавания. При этом на вход кроме x подается число k и спрашивается, верно ли, что $F_{\text{опт}} \leqslant k$ (или $F_{\text{опт}} \geqslant k$).

На практике для решения задач оптимизации часто используются алгоритмы, называемые жадными или градиентными. В таких алгоритмах допустимое решение строится постепенно по шагам, причем на каждом шаге делается выбор, оптимальный для данного шага. Как мы увидим ниже, такой подход не всегда приводит к оптимальному решению в целом. Однако для следующей задачи он всегда дает оптимальное решение. Напомним, что деревом называется любой неориентированный связный граф без циклов. Подграф G_1 графа G называется остовным, если G_1 содержит все вершины графа G. Через K_n обозначается полный граф на n вершинах, то есть граф, в котором каждая пара вершин соединена ребром.

Задача о кратчайшем остовном дереве.

Вход: неориентированный полный граф K_n, в котором для любого ребра e задан вес $w(e) \geqslant 0$.

Требуется: выделить в K_n остовное дерево с минимальной суммой весов ребер.

Замечание. На практике это означает требование построить сеть минимальной стоимости, связывающую n данных объектов.

Напомним некоторые факты из теории графов.

Утверждение 1. *Если в графе с n вершинами число ребер $q < n - 1$, то граф не связный.*

Утверждение 2. *Если в графе G нет циклов и $q = n - 1$ (q, n—число ребер и вершин), то G—дерево.*

Утверждение 3. *В любом дереве с n вершинами число ребер $q = n - 1$.*

Утверждение 4. *Если к дереву добавить новое ребро на тех же вершинах, то образуется ровно 1 цикл.*

Рассмотрим следующий алгоритм для задачи о кратчайшем остовном дереве.

1. Взять любое ребро e_1 минимального веса.

2. Рекурсивный шаг: пусть уже выбраны ребра $e_1, e_2, \ldots, e_m$. Если $m = n - 1$, то остановиться. Иначе, среди всех ребер, не образующих циклов с $e_1, e_2, \ldots, e_m$, взять ребро e_{m+1} минимального веса и повторить рекурсивный шаг.

Алгоритм делает меньше, чем n итераций и на каждой просматривает менее, чем n^2 ребер. При этом, если из ребер $e_1, e_2, \ldots, e_m$ сформировать связные компоненты, то тот факт, что e_{m+1} не образует с ними циклов, эквивалентен тому, что концы ребра e_{m+1} не лежат в одной связной компоненте. Это свойство легко проверяется. Таким образом, алгоритм может быть реализован с полиномиальным от n числом операций, включающих поиск информации и сравнение весов.

Теорема. *Описанный алгоритм корректно строит минимальное остовное дерево.*

Доказательство. 1) Докажем, что если $m < n-1$, то существуют ребра, не образующие циклов с $e_1, e_2, \ldots, e_m$. Если $m < n - 1$, то подграф, состоящий из всех вершин и ребер $e_1, e_2, \ldots, e_m$, не связный (по утв. 1). Если взять любое ребро, соединяющее две вершины из разных компонент этого подграфа, то циклы не образуются. Таким образом, алгоритм проработает до $m = n - 1$.

2) При остановке $m = n - 1$ и ребра $e_1, e_2, \ldots, e_m$ не образуют циклов. Тогда (по утв. 2) они образуют остовное дерево.

3) Пусть алгоритм строит остовное дерево D. Докажем, что D — минимальное остовное дерево. Рассмотрим все минимальные остовные деревья, и пусть T — минимальное остовное дерево, имеющее с D наибольшее число общих ребер. Докажем (от противного), что $D = T$. Допустим, что $T \neq D$. Так как и в T и в D $n - 1$ ребро (утв. 3), то в D есть ребра, не входящие в T. Пусть в алгоритме ребра дерева D появлялись в порядке: $e_1, e_2, \ldots, e_{n-1}$ и пусть ребра $e_1, e_2, \ldots, e_k$ принадлежат дереву T, а $e_{k+1} \notin T$. Рассмотрим граф $H = T \cup \{e_{k+1}\}$. В H имеется единственный цикл C (утв.4), содержащий e_{k+1}. Так как D не содержит циклов, то в C есть хотя бы одно ребро e такое, что $e \notin D$. При этом $e \in T$. Рассмотрим $H_1 = H \setminus \{e\}$. Граф H_1 — связный и без

циклов, то есть H_1— остовное дерево. Пусть $w(H_1)$ и $w(T)$ — суммы весов ребер в H_1 и T_1. Так как T — минимальное остовное дерево, то $w(H_1) \geqslant w(T)$ и

$$w(H_1) = w(T) + w(e_{k+1}) - w(e) \geqslant w(T).$$

Отсюда $w(e) \leqslant w(e_{k+1})$. Поскольку $e \in T$ и $e_1, e_2, \ldots, e_k$ принадлежат дереву T, то e не образует циклов с $e_1, e_2, \ldots, e_k$. Если бы было $w(e) < w(e_{k+1})$, то на $k+1$-м шаге алгоритма не могло бы выбираться ребро e_{k+1}. Значит $w(e) = w(e_{k+1})$ и $w(H_1) = w(T)$. Получаем, что H_1 — также минимальное остовное дерево, но имеющее с D на 1 общее ребро больше, чем T с D. Это противоречит выбору дерева T. Из полученного противоречия следует, что должно быть $D = T$, то есть D — минимальное остовное дерево. Теорема доказана.

Приближенные алгоритмы.

Задача о минимальном вершинном покрытии (**МВП**).

Будем говорить, что вершина v покрывает ребро e, если e является одним из концов ребра e. подмножество $A \subseteq V$ вершин графа $G = (V, E)$ называется вершинным покрытием, если вершины из A покрывают все ребра из E.

Вход: неориентированный граф $G = (V, E)$.

Требуется: найти вершинное покрытие (ВП) минимальной мощности.

"Жадный" алгоритм для МВП.

На каждом шаге выбирается любая вершина, покрывающая наибольшее число еще не покрытых ребер. Алгоритм останавливается, когда все ребра покрыты.

Легко показать, что "жадный" алгоритм для МВП имеет полиномиальную сложность.

Теорема. *Для "жадного" алгоритма для задачи МВП для любого натурального n существует граф G_n такой, что при входе G_n выполняется неравенство:*

$$F_{алг} \geqslant F_{опт}(\ln n - \ln 2 - 1).$$

Доказательство. Включим в граф G_n сначала вершины $u_1, u_2, \ldots, u_n$, между которыми не будет ребер. Далее выделим из вершин $u_1, u_2, \ldots, u_n$ $\left[\frac{n}{2}\right]$ непересекающихся пар (одна вершина может не

участвовать в парах) и каждую пару соединим с новой вершиной, при этом получим новые вершины $v_1, v_2, \ldots, v_{[\frac{n}{2}]}$ степени 2. Затем выделим из вершин $u_1, u_2, \ldots, u_n$ $\left[\frac{n}{3}\right]$ непересекающихся троек и каждую тройку соединим с новой вершиной (степени 3). Далее аналогично выделим непересекающиеся четверки, пятерки вершин и т.д. На последнем этапе выделим из $u_1, u_2, \ldots, u_n$ группу из $n-1$ вершин и соединим ее с новой вершиной (степени $n-1$). Заметим, что после добавления новых вершин степени k вершины $u_1, u_2, \ldots, u_n$ имеют степень не более $k-1$, в частности, в заключительном графе G_n они имеют степень не более $n-2$. Поэтому "жадный" алгоритм, примененный к G_n, сначала выберет добавленную вершину степени $n-1$, затем (после удаления этой вершины и покрываемых ею ребер) выберет все добавленные вершины степени $n-2$, затем все добавленные вершины степени $n-3$ и т.д. На последнем этапе он выберет все добавленные вершины степени 2. Таким образом

$$F_{\text{алг}} = \left[\frac{n}{2}\right] + \left[\frac{n}{3}\right] + \ldots + \left[\frac{n}{n-1}\right] >$$

$$> \left(\frac{n}{2} - 1\right) + \left(\frac{n}{3} - 1\right) + \ldots + \left(\frac{n}{n-1} - 1\right) =$$

$$= n\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-1}\right) - (n-2) >$$

$$> n \int_2^n \frac{1}{x} dx - n = n(\ln n - \ln 2) - n \quad (20)$$

С другой стороны множество $\{u_1, u_2, \ldots, u_n\}$ является вершинным покрытием в G_n. Поэтому $F_{\text{опт}} \leqslant n$ и

$$F_{\text{алг}} = n(\ln n - \ln 2 - 1) \geqslant F_{\text{опт}}(\ln n - \ln 2 - 1).$$

Определение. Пусть дана задача оптимизации с функционалом F. Алгоритм для этой задачи называется ε- приближенным, если всегда

$$\left|\frac{F_{\text{алг}} - F_{\text{опт}}}{F_{\text{опт}}}\right| < \varepsilon,$$

где $F_{\text{алг}}$ и $F_{\text{опт}}$ —значение функционала, выдаваемое алгоритмом, и оптимальное значение.

Если дана задача минимизации и $F_{\text{опт}} > 0$, то указанное неравенство эквивалентно неравенству:$F_{\text{алг}} \leqslant (1+\varepsilon)F_{\text{опт}}$.

Следствие. *Жадный алгоритм для МВП не является ε-приближенным ни при каком фиксированном ε.*

Следующая теорема показывает, что теоретически "жадная" стратегия для задачи МВП не является хорошей.

Теорема. *Для задачи МВП существует 1-приближенный алгоритм с полиномиальной сложностью.*

Доказательство. Рассмотрим следующий алгоритм. Пусть дан граф $G = (V, E)$. Будем формировать вершиное покрытие A. Возьмем любое ребро $e_1 = (v_1, v_2)$ и включим v_1 и v_2 в A. Выбросим из графа G вершины v_1 и v_2 и все ребра, которые ими покрываются. В полученном графе G_1 опять возьмем любое ребро $e_2 = (v_3, v_4)$, добавим v_3 и v_4 в A и удалим из G_1 вершин v_3 и v_4 и все покрываемые ими ребра. Процесс закончим, когда будут удалены все ребра. Легко понять, что этот алгоритм можно реализовать с полиномиальной сложностью. Также по построению очевидно, что полученное множество вершин A покрывает все ребра. Пусть в процессе алгоритма выбирались ребра $e_1, e_2, \ldots, e_k$. Тогда $|A| = 2k$. С другой стороны ребра $e_1, e_2, \ldots, e_k$ не имеют общих вершин и, следовательно, любое вершинное покрытие должно содержать не менее k вершин (чтобы покрыть $e_1, e_2, \ldots, e_k$). Таким образом $F_{\text{опт}} \geqslant k$ и $F_{\text{алг}} = |A| \geqslant 2F_{\text{опт}}$. Теорема доказана.

Возникает вопрос, а нельзя ли для задачи МВП построить не приближенный, а точный алгоритм с полиномиальной сложностью. Выше была доказана NP-полнота задачи о вершинном покрытии (ВП), где по заданному графу G и числу k требуется выяснить, есть ли в графе G вершинное покрытие мощности не более k.

Теорема. *Если для задачи МВП существует алгоритм с полиномиальной сложностью, то и для задачи ВП существует алгоритм с полиномиальной сложностью.*

Доказательство. Пусть алгоритм H решает задачу МВП за полиномиальное время и пусть в задаче ВП заданы граф G и число k. Применяем к графу G алгоритм H и получаем m — минимальную мощность вершинного покрытия в G. Если $m \leqslant k$, то ответ в задаче ВП "да", иначе ответ "нет". Получаем полиномиальный алгоритм для задачи ВП.

Замечание. Если для задачи ВП существует алгоритм H с полиномиальной сложностью и в графе G n вершин, то, применяя алгоритм H к парам $(G, 0), (G, 1), \ldots, (G, n-1)$, можно за полиномиальное время определить мощность минимального вершинного покрытия, однако не ясно, как найти само минимальное вершинное покрытие.

Определение. Задачу оптимизации будем называть NP-трудной, если из существования алгоритма полиномиальной

сложности для нее следует существование алгоритма полиномиальной сложности для некоторой NP-полной задачи (и,следовательно, для всех задач из NP).

Следствие. *Задача МВП является NP-трудной.*

Следствие. *Если $P \neq NP$, то для задачи МВП не существует алгоритма с полиномиальной сложностью.*

Задача коммивояжера

Выше мы получили условный результат о трудности нахождения точного решения задачи МВП. Здесь мы покажем, что такие условные отрицательные результаты можно получать и для нахождения приближенных решений.

Напомним, что цикл в графе называется гамильтоновым, если он проходит через каждую вершину ровно 1 раз. Выше было показано, что задача о существовании в графе гамильтонова цикла (ГЦ) является NP-полной.

Задача коммивояжера (**ЗК**).

Вход: полный граф K_n, в котором каждому ребру $e = (v_i, v_j)$ сопоставлен вес $d(e) = d(v_i, v_j) \geqslant 0$. При этом будем считать, что все $d(e)$— целые числа и длина входа включает в себя суммарную длину двоичного представления всех $d(e)$.

Требуется: найти гамильтонов цикл в K_n с минимальной суммой весов ребер.

Теорема. *ЗК является NP-трудной.*

Доказательство. Пусть существует алгоритм H для ЗК со сложностью, полиномиально зависящей от длины входа. Пусть дан граф $G = (V, E)$ с n вершинами и спрашивается есть ли в G гамильтонов цикл. Пусть $V = \{v_1, v_2, \ldots, v_n\}$. Построим полный граф K_n на множестве вершин V и зададим веса следующим образом:

Применим алгоритм H для ЗК к графу K_n с этими весами. Если получим для ЗК, что $F_{\min} = n$, то в G существует гамильтонов цикл, иначе в G не существует гамильтонова цикла. Таким образом, получаем алгоритм для задачи о гамильтоновом цикле (ГЦ). Поскольку в G меньше, чем n^2 ребер, то суммарная длина двоичной записи всех весов не превосходит cn^2, где c—некоторая контсанта, то есть длина входа для H не превосходит полинома от n. Так как H—полиномиальный (от длины входа) алгоритм, то построенный нами алгоритм для ГЦ имеет полиномиальную от n сложность. Таким образом из существования алгоритма с полиномиальной сложностью для ЗК вытекает существование агоритма с полиномиальной сложнотью для ГЦ. Поскольку задача ГЦ является NP-полной, то получаем, что задача ЗК является NP-трудной.

Теорема. *Если $P \neq NP$, то ни для какого сколь угодно большого постоянного числа ε-приближенного алгоритма для ЗК с полиномиальной сложностью.*

Доказательство. Допустим, что существует ε и существует ε-приближенный алгоритм H с полиномиальной сложностью для ЗК. Построим тогда алгоритм с полиномиальной сложностью для ГЦ. Пусть дан граф $G = (V, E)$ с n вершинами. Построим полный граф $K_n = (V, E')$ и для всех $e \in E'$ положим

Применим к K_n с весами d алгоритм H. Пусть алгоритм H находит гамильтонов цикл с суммарной длиной F_H.

Лемма. *Если в гарфе G есть гамильтонов цикл, то $F_H \leqslant n(1+\varepsilon)$. Если в графе G нет гамильтонова цикла, то $F_H \geqslant n(1+\varepsilon)+1$.*

Доказательство. Если в G есть гамильтонов цикл, то в ЗК для K_n с dtcfvb d будет $F_{\text{опт}} = n$. Так как H является ε-приближенным алгоритмом для ЗК, то $F_H \leqslant F_{\text{опт}}(1+\varepsilon) = n(1+\varepsilon)$. Если в G нет гамильтонова цикла, то любой гамильтонов цикл содержит хотя бы одно ребро с весом $[3+\varepsilon n]$ и $n-1$ ребер с весом не менее 1. Таким образом, суммарный вес любого гамильтонова цикла не меньше чем $n-1+2+\varepsilon n = n(1+\varepsilon)+1$.

Лемма показывает, что по результату работы алгоритма H можно определить, есть ли в G гамильтонов цикл. Таким образом, мы получаем алгоритм H_1 для задачи ГЦ. Оценим время его работы. Длина двоичного представления каждого веса $d(e)$ не превосходит $c\log_2 n$, где c—некоторая константа, и количество весов меньше, чем n^2. Поэтому длина входа для алгоритма H не превосходит полинома от n. Поскольку время работы H зависит полиномиально от длины входа, то общее время работы алгоритма H_1 не превосходит полинома от n. В результате мы получаем, что если существует ε-приближенный алгоритм H с полиномиальной сложностью для ЗК, то существует алгоритм с полиномиальной сложностью для задачи ГЦ. Но задача ГЦ NP-полна. Тогда получаем, что $P = NP$. Теорема доказана.

Во многих практических задачах веса удовлетворяют естественному ограничению, называемому неравенством треугольника:

$$d(v_i, v_j) \leqslant d(v_i, v_k) + d(v_k, v_j)$$

для всех различных i, j, k. Будем говорить, что дана задача коммивояжера с неравенством треугольника (ЗКНТ), если на вход поступают только веса, удовлетворяющие неравеству треугольника.

Теорема. *ЗКНТ NP-полна.*

Для доказательства этой теоремы полностью проходит доказательство теоремы ( ). Достаточно только отметить, что набор весов, который строится в этом доказательстве, удовлетворяет неравенству треугольника.

Теорема. *Для ЗКНТ существует 1-приближенный алгоритм с полиномиальной сложностью.*

Доказательство. Мы должны построить алгоритм H для ЗКНТ такой, что всегда $F_H \leqslant 2F_{\text{опт}}$. Применим к заданному графу K_n с весами d алгоритм с полиномиальной сложностью для построения кратчайшего остовного дерева. Пусть он строит кратчайшее остовное дерево D с суммарным весом ребер $d(D)$. Пусть C — любой гамильтонов цикл. Если выбросить любое ребро из C, то получим дерево T. При этом

$$d(D) \leqslant d(T) \leqslant d(C).$$

Поэтому $d(D) \leqslant F_{\text{опт}}$. Рассмотрим дерево D и заменим каждое ребро $e = (v_i, v_j)$ в D двумя ребрами $e' = (v_i, v_j)$ и $e" = (v_i, v_j)$. Тогда получим мультиграф K (граф с кратными ребрами), в котором степень каждой вершины четна. Так как D— остовное дерево, то мультиграф K связный. Выше доказано (см. теорему ( )), что в любом связном мультиграфе, в котором степени всех вершин четны, существует эйлеров цикл. Применим к K алгоритм с полиномиальной сложностью для построения в K эйлерова цикла C_1 (существование такого алгоритма доказано в теореме( )). Поскольку цикл C_1 проходит по каждому ребру в K ровно 1 раз, то вес $d(C_1) = 2d(D) \leqslant 2F_{\text{опт}}$. Выберем любую вершину v_1 в C_1 в качестве начальной и пусть вершины в C_1 встречаются в порядке $v_1, v_2, v_3, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_1$. Пусть выделенное значение v_i встречается в цикле раньше. Тогда заменим последовательность ребер $(v_{i-1}, v_i), (v_i, v_{i+1})$ на ребро $(v_{i-1}, v_{i+1}$ в исходном графе. При этом получим опять цикл, проходящий по всем вершинам. Поскольку веса удовлетворяют неравенству треугольника, то суммарный вес цикла при этом не возрастет. Если в полученном цикле снова есть повторяющиеся вершину, то опять выбросим одну вершину, осуществив "спрямление". Этот процесс будем повторять до тех пор, пока не получится цикл C_2 без повторяющихся вершин. Тогда цикл C_2 будет гамильтоновым и $d(C_2) \leqslant d(C_1) \leqslant 2F_{\text{опт}}$. В результате мы получаем алгоритм для ЗКНТ с полиномиальной сложностью, который является 1-приближенным.

Задача о максимальной клике

Выше было доказано, что задача КЛИКА является NP-полной.

Рассмотрим теперь следующую задачу **МК**.

Вход: неориентированный граф G.

Требуется: найти какую-нибудь максимальную по числу вершин клику.

Теорема. *Задача о максимальной клике (МК) является NP-трудной.*

Доказательство. Пусть A — алгоритм с полиномиальной сложностью для МК, и пусть пара (G, k) — вход для задачи КЛИКА. Применим к G алгоритм A и найдем мощность m полученной максимальной клики в G. Если $m \geqslant k$, то для пары (G, k) в задаче КЛИКА ответ "да", иначе ответ "нет". Получаем полиномиальный алгоритм для задачи КЛИКА. Так как КЛИКА NP-полна, то из существования полиномиального алгоритма для МК следует существование полиномиального алгоритма для всех задач из NP, то есть МК NP-трудна.

Прежде, чем исследовать приближенные алгоритмы для МК, докажем лемму.

Определение. Пусть $G = (V, E)$ — неориентированный граф. Определим граф $G^2 = (V^2, E^2)$ как граф с множеством вершин $V^2 = V \times V = \{(u, v) | u \in V, v \in V\}$ и множеством ребер $E^2 = \{(u_1, v_1), (u_2, v_2)\}$, где либо $u_1 = u_2$ и $(v_1, v_2) \in E$, либо $(u_1, u_2) \in E$.

Лемма. *Если в G есть клика размера k, то в G^2 есть клика размера k^2. Если в G^2 есть клика размера m, где $(k-1)^2 < m \leqslant k^2$, то в G есть клика размера k, и в G^2 есть клика размера k^2.*

Доказательство. Пусть в G есть клика $C = \{v_1, v_2, \ldots, v_k\}$. Тогда из определения легко проверить, что $C^2 = \{(u, v) | u \in C, v \in C\}$ — клика в G^2 размера k^2. Обратно, пусть в G^2 есть клика D размера m. Вершинами в D являются пары $(u, v) \in V^2$. Пусть $V = \{v_1, v_2, \ldots, v_n\}$ и пусть D_i—множество вершин (u, v) из D, у которых $u = v_i$. По определению графа G^2 вершины (u, v') и $(u, v")$ смежны в G^2 тогда и только тогда, когда $(v', v") \in E$. Поэтому вторые координаты всех вершин из D_i образуют клику в G. Если $|D_i| \geqslant k$ хотя бы для одного i, то получаем в G клику размера k. В противном случае $|D_i| \leqslant k-1$ для всех i и, следовательно, число непустых D_i не менее k, так как $m > (k-1)^2$. Выберем в каждом непустом D_i любую вершину (v_i, v_{d_i}). Так как все эти вершины принадлежат одной клике D, то все они смежны. Так как $v_i \neq v_j$ при $i \neq i$, то $(v_i, v_j) \in E$ для всех первых координат выбранных вершин по определению графа G^2. Поскольку

число выбранных вершин $m \geqslant k$, то получаем клику в G размера k. Последнее утверждение леммы следует из первого.

Следствие 1. *Мощность максимальной клики в G^2 имеет вид k^2 для некоторого натуарльного k.*

Следствие 2. *Существует полиномиальный алгоритм, который по заданной клике D мощности m в графе G^2, где $(k-1)^2 < m \leqslant k^2$, строит клику C мощности k в графе G.*

Пусть $m_{\text{алг}}$ и $m_{\max}$ — мощность клики, которая строится некоторым алгоритмом, и мощность максимальной клики для данного входа G. Тогда $m_{\text{алг}} \leqslant m_{\max}$ и

$$\varepsilon = \frac{|m_{\text{алг}} - m_{\max}|}{m_{\max}} \leqslant 1.$$

Теорема. *Если для задачи МК существует полиномиальный ε-приближенный алгоритм для некоторого $0 < \varepsilon < 1$, то для МК существует полиномиальный ε-приближенный алгоритм для всех $0 < \varepsilon < 1$.*

Доказательство. Пусть для задачи МК для некоторого $0 < \varepsilon < 1$ имеется полиномиальный ε—приближенный алгоритм A_ε, и пусть $0 < \delta < 1$. Выберем натуральное r так, что $(1-\delta)^{2^r} < 1-\varepsilon$. Такое r существует, так как $1 - \delta < 1$. Рассмотрим следующий алгоритм B. Пусть на вход поступает граф G. Строим последовательно $G^2, G^4, \ldots, G^{2^r}$. Применяем к G^{2^r} алгоритм A_ε. Получаем клику D_r в G^{2^r}. По клике D_r строим клику D_{r-1} в $G^{2^{r-1}}$ так, как в доказательстве леммы. По D_{r-1} аналогично строим клику D_{r-2} в $G^{2^{r-2}}$ и т.д. до клики D_0 в G. Клику D_0 выдаем в ответ. Так как r—фиксировано, то алгоритм B полиномиален (см. следствие 2). Докажем, что он является δ-приближенным.

Пусть мощность максимальной клики в G равна k. Тогда мощность максимальной клики в G^{2^r} равна k^{2^r} по лемме (см. следствие 1). Так как алгоритм A_ε является ε-приближенным, то $|D_r| \geqslant k^{2^r}(1-\varepsilon)$. Поскольку $|D_{i-1}| \geqslant \sqrt{|D_i|}$ для всех i, то

$$|D_0| \geqslant \sqrt[2^r]{1-\varepsilon} > k(1-\delta).$$

Следовательно, алгоритм B является δ-приближенным. Теорема доказана.

Классы $PSPACE$ и $DLOG$

Классы P и NP определялись через используемое алгоритмом время работы. Другие классы мы можем получить, если будем рассматривать используемую память.

Определение. Класс **PSPACE** определяется как класс всех задач распознавания (языков), для которых существует алгоритм, использующий память (например, число ячеек машины Тьюринга), не превосходящую $p(n)$, где n —длина входа и p— произвольный (фиксированный для данной задачи) полином.

Очевидно, что $P \subseteq PSPACE$.

Теорема. $NP \subseteq PSPACE$.

Доказательство. Пусть задача распознавания $R(x) \in NP$. По определению класса NP $R(x)$ представимо в виде:

$$R(x) = \exists y(|y| \leqslant p_1(|x|) \& Q(x, y)),$$

где $|x|$ и $|y|$ —длина слов x и y, p_1— некоторый полином и предикат $Q(x, y) \in P$. Покажем, что для вычисления $R(x)$ существует алгоритм с полиномиальной памятью. Пусть дан вход x. Вычисляем длину n слова x. Вычисляем $p_1(n)$ и отмечаем в памяти зону $p_1(n)$, на которой перебираем по очереди все слова y длины $\leqslant p_1(n)$. Для каждого y вычисляем $Q(x, y)$. Если при вычислении $Q(x, y)$ хотя бы один раз ответ $Q(x, y) =$ "и"(истина), то выдаем ответ "да", иначе выдаем ответ "нет". Так как $|x| + |y| \leqslant n + p_1(n)$ и $Q \in P$, то время вычисления $Q(x, y)$ для одного y не превосходит некоторого полинома от n. но тогда и используемая память не превосходит полинома от n. Теорема доказана.

Лемма. *Если зона работы машины Тьюринга на входах длины n содержит не более $p_1(n)$ ячеек, где $p_1(n)$—некоторый полином, в ленточном алфавите машины r символов, у машины k состояний и машина останавливается на любом входе, то максимальное время работы $t(n)$ машины на словах длины n удовлетворяет неравенству:*

$$t(n) \leqslant r^{p_1(n)} p_1(n) \cdot k \leqslant 2^{p(n)},$$

где $p(n)$— некоторый полином.

Доказательство. Если зона работы машины содержит не более $p_1(n)$ ячеек, то при работе машины может порождаться не более $r^{p_1(n)} p_1(n) \cdot k$ различных конфигураций, поскольку на ленте можно записать не более $r^{p_1(n)}$ различных слов, головка может обозревать любую из не более $p_1(n)$ ячеек и машина может находиться в любом из k

состояний. Поскольку по условию при любом входе машина останавливается, то она не может "зациклиться", то есть конфигурация не может повториться. поэтому время работы при любом входе не превосходит числа различных конфигураций. При этом

$$\log_2(r^{p_1(n)}p_1(n)\cdot k) = p_1(n)\cdot\log_2 r + \log_2 p_1(n) + \log_2 k \leqslant p(n),$$

где $p(n)$— некоторый полином. Теорема доказана.

Следствие. *Для любой задачи из* $PSPACE$ $t(n) \leqslant 2^{p(n)}$, *где* $p(n)$— *некоторый полином.*

Определение. Задача распознавания (язык) L называется $PSPACE$-полной, если:

1)$L \subseteq PSPACE$,

2)к L полиномиально сводятся все задачи из $PSPACE$.

Утверждение. *Если для некоторой* $PSPACE$*-полной задачи существует алгоритм с полиномиальной сложностью, то* $PSPACE = P$.

Задача о квантифицированных булевских формулах (**QBF**).

Вход: формула вида

$$(Q_1x_1)(Q_2x_2)\dots(Q_mx_m)[F(x_1,\dots,x_m)],$$

где $x_1,\dots,x_m$—булевские переменные, F—булевская формула в базисе {конъюнкция,дизъюнкция,отрицание}, $Q-i \in \{\exists,\forall\}$ для всех i.

Требуется: выяснить, истинна ли данная формула.

Лемма. $QBF \in PSPACE$.

Доказательство. Пусть на вход поступила формула

$$(Q_1x_1)(Q_2x_2)\dots(Q_mx_m)[F(x_1,\dots,x_m)],$$

длины n. Тогда длина формулы $F(x_1,\dots,x_m)$ не более n, и для любого заданного набора $(\alpha_1,\dots,\alpha_m$ вычисление $F(\alpha_1,\dots,\alpha_m)$ можно выполнить за время, а значит и с использованием памяти, не более $p_1(n)$, где p_1 —некоторый полином. Если зафиксированы только значения $\alpha_1,\dots,\alpha_k$ переменных $x_1,\dots,x_k$, то мы получаем подзадачу: найти истинностное значение формулы

$$(Q_{k+1}x_{k+1})\dots(Q_mx_m)[F(\alpha_1,\dots,\alpha_k)(x_{k+1},\dots,x_m)].$$

Применим для решения исходной задачи (и всех ее подзадач) следующий рекурсивный алгоритм:

1.Вычислить $(Q_2x_2)\dots(Q_mx_m)F(0,x_2,\dots,x_m)$ этим же рекурсивным алгоритмом. Запомнить полученное значение (1 бит) в дополнительной ячейке.

2.Вычислить на той же зоне этим же алгоритмом $(Q_2x_2)\dots(Q_mx_m)F(0,x_2,\dots,x_m)$.

3.Если $Q_1=\forall$ и оба значения, вычисленные в 1 и 2, равны 1,то выдать ответ 1, иначе выдать 0.Если $Q_1=\exists$ и оба значения в 1 и 2 равно 0, то выдать 0, иначе выдать 1.

Из описания алгоритма видно, что для решения задачи каждого уровня нужно на 1 ячейку больше, чем на решение любой ее подзадачи. Так как на вычисление $F(\alpha_1,\dots,\alpha_m)$ требуется памяти не более $p_1(n)$, то для вычисления истинностного значения исходной формулы требуется память не более $p_1(n)+n$ ячеек. Для управления процессом перехода от одинх подзадач к другим в описанном алгоритме достаточно помнить, какая подзадача решается в данный момент, то есть помнить значения $\alpha_1,\dots,\alpha_k$, определяющие эту подзадачу. Таким образом, в целом оприсанный алгоритм требует не более $p(n)$ ячеек памяти, где p—некоторый полином.

Теорема. *Задача QBF является $PSPACE$—полной.*

Доказательство. Нам надло доказать, что любая задача L из $PSPACE$ полиномиально сводится к QBF. Если $L\in PSPACE$, то существует машина Тьюринга M, которая решает задачу L с памятью не более $p_1(n)$ и временем не более $p(n)$ (см. лемму), где n—длина входа. Пусть x—вход длины n для задачи L. Нам надо за полиномиальное время построить квантифицированную формулу $F^(x)$ так, что $F^(x)$ истинна тогда и только тогда, когда $x\in L$. Тот факт, что $x\in L$, равносилен утверждению: для входа x существует принимающее (с ответом "да") вычисление машины M. Это последнее утверждение мы и выразим в виде формулы $F^(x)$. Так же, как при доказательстве NP-полноты задачи ВЫП, введем два множества переменных V и V', описывающих 2 произвольные конфигурации на зоне $p_1(n)$, и запишем формулу $F_0(V,V')$ выражающую тот факт, что V и V' правильно задают конфигурации и либо $V=V'$, либо из конфигурации V мы за один шаг машины M переходим в конфигурацию V'. Как показано при доказательстве NP-полноты задачи ВЫП длина формулы $F_0(V,V')$ может быть ограничена некоторым полиномом $p_2(n)$ и ее можно построить за полиномиальное от n время. Формула $F_0(V,V')$ выражает тот факт, что из конфигурации V в конфигурацию V' можно перейти за не более чем 1 шаг. Построим теперь индуктивно формулы $F_1,F_2,\dots,F_s$, где $s=p(n)$, следующим образом. Пусть W—еще одно множество переменных, описывающих конфигурацию на зоне $p_1(n)$.

Тогда положим

$$F_k(V,V') = \exists W[F_{k-1}(V,W)\&F_{k-1}(W,V')].$$

Формула F_k выражает тот факт, что V, V'—правильные конфигурации и из V в V' можно перейти за не более, чем 2^k шагов. Формула в квадратных скобках равносильна следующей формуле:

$$(\forall Y)(\forall Z)[(Y = V)\&(Z = W) \vee (Y = W)\&(Z = V') \to F_{k-1}(Y,Z)]$$

где Y, Z—два множества переменных, описывающих 2 произвольные конфигурации. Поэтому формулу F_k можно записать в виде:

$$F_k(V,V') = (\exists W)(\forall Y)(\forall Z)[(Y = V)\&(Z = W)\vee(Y = W)\&(Z = V') \to F_{k-1}(Y,Z)].$$

Таким образом, длина $F_k(V,V')$ отличается от длины $F_{k-1}(V,V')$ не более чем на некоторый полином $p_3(n)$ и длина $F_k(V,V')$ не превосходит $p_2(n) + kp_3(n)$. Пусть время работы машины M не превосходит $2^{p(n)}$ и $s+p(n)$. Тогда $F_s(V,V')$ имеет длину не более $p_2(n)+p(n)\cdot p_3(n) = p_4(n)$, где p_4— полином, и выражает тот факт, что V и V' правильные конфигурации и из V в V' можно перейти за не более $2^{p(n)}$ шагов машины M. Пусть формула $G_x(V)$ выражает тот факт, что конфигурация V является правильной начальной конфигурацией для входа x (yf pjyt $p_1(n)$), а формула $H(V)$ выражает тот факт, что в конфигурации V состояние "да". Тогда тот факт, что для входа x существует принимающее вычисление машины M можно представить формулой

$$F^{(x)} = (\exists V)(\exists V')[G_x(V)\&H(V')\&F_s(V,V')].$$

Поскольку длина формул $G_x(V)$ и $H(V)$ может быть ограничена полиномом от n и они могут быть построены за полиномиальное от n время (см. доказательство NP-полноты задачи ВЫП), то получаем, что длина $F^{(x)}$ не превосходит полинома от n и $F^{(x)}$ может быть построена за полиномиальное от n время. Теорема доказана.

При определении класса $DLOG$ обычно используют модель многоленточной машины Тьюринга. Пусть у машины Тьюринга имеется несколько лент, одна из которых выделена как входная, и на каждой ленте имеется одна головка. Один шаг работы такой машины состоит в одновременном выполнении обычных действий каждой головкой (у каждой головки свои действия), причем весь набор действий однозначно определяется теми символами, которые обозреваются всеми головками и состоянием машины. Входное слово записывается на входной ленте в стандартной конфигурации и головка на входной ленте может только читать символы, но не может записывать новые.

Определение. Класс $DLOG$ определяется как класс всех задач распознавания (языков), для которых существует распознающая их многоленточная машина Тьюринга, использующая на всех лентах, кроме входной, не более $c \log_2 n$ ячеек, где n —длина входа и c—некоторая константа (для данной задачи).

Используя лемму, нетрудно показать, что $DLOG \subseteq P$. Таким образом

$$DLOG \subseteq P \subseteq NP \subseteq PSPACE.$$

Можно доказать, что $DLOG \neq PSPACE$, поэтому хотя бы одно из указанных включений должно быть строгим. Однако какое (или какие) именно, пока (2001 год) не известно.