Глава 7

СЛОЖНОСТЬ ВЫЧИСЛЕНИЙ

Make things as simple as possible but not simpler.

A. Einstein

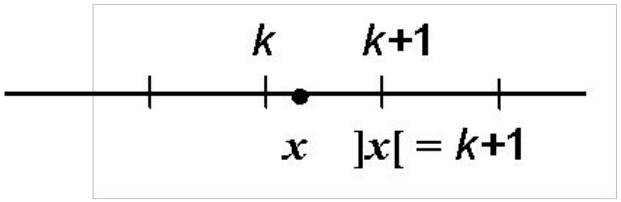
ПОНЯТИЕ О СЛОЖНОСТИ

Сложность описания алгоритма **зависит от способа его задания.**

$g(n) = 0(f(n))$, если $\exists c$, что $g(n) \le c\, f(n)$ для $\forall n$, кроме конечного множества значений n, $n \in \{0, 1, 2, 3, \ldots\}$.

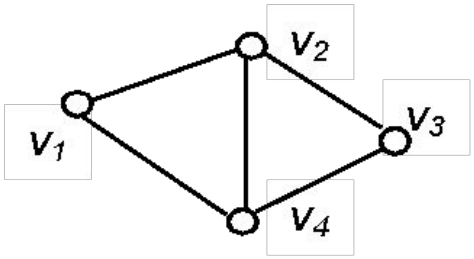
Сложность исходных данных **понимается как длина их записи.**

Число символов, для записи числа n равно $\lceil \log_A n \rceil$.



$\log_A n = (\log_2 n) / (\log_2 A)$, тогда для n требуется $0(\log_2 n)$ символов.

Граф $G = (V, X)$ можно задать матрицей смежностей $A_G = (a_{ij})$ размера $|V| \times |V|$ или с помощью *списков смежностей*.



$A(v_1) = \{v_2, v_4\}$;
$A(v_2) = \{v_1, v_3, v_4\}$;
$A(v_3) = \{v_2, v_4\}$;
$A(v_4) = \{v_1, v_2, v_3\}$.

При этом потребуется $0(|X| \log_2 |V|)$ символов.

КОЛИЧЕСТВО СИМВОЛОВ ДЛЯ ЗАПИСИ ЧИСЕЛ

$n = 100 \leftrightarrow \underbrace{111\ldots111}_{101}$

$n \leftrightarrow]\log_A n[\,; \quad A \geq 2:$

		кол-во симв.
$A = 2,\ n = 100 \leftrightarrow$	1100100	$(\log_2 100 = 6{,}65);$ 7
$A = 4,\ n = 100 \leftrightarrow$	1210	$(\log_4 100 = 3{,}32);$ 4
$A = 8,\ n = 100 \leftrightarrow$	144	$(\log_8 100 = 2{,}22);$ 3
$A = 16,\ n = 100 \leftrightarrow$	64	$(\log_{16} 100 = 1{,}67);$ 2
$A = 20,\ n = 100 \leftrightarrow$	50	$(\log_{20} 100 = 1{,}53);$ 2

ВРЕМЕННАЯ СЛОЖНОСТЬ ВЫЧИСЛЕНИЙ (АЛГОРИТМА)

Временная сложность алгоритма **- число операций в худшем случае по всем входам размера** ***n*****.**

Временная сложность алгоритма A в среднем **на входе размера** ***n*****:**

$$M_A(n) = \sum_{a_i \in P_n} p(a_I)\,\mu(A(a_I)),$$

$p(a_I)$ – вероятность появления задачи a_I; $\mu(A(a_I))$ - число операций, на решение индивидуальной задачи a_I; P_n – множество задач размера n, $P_n = \{a_I\}$.

Сложность алгоритма оценивается асимптотической сложностью, т.е. порядком роста числа операций при неограниченном росте размера входа.

Под *временной сложностью задачи* понимается временная сложность наилучшего алгоритма, известного для ее решения.

ПОЛИНОМИАЛЬНЫЕ АЛГОРИТМЫ И ЗАДАЧИ. КЛАСС *P*

Алгоритм *полиномиальный* или имеет *полиномиальную временную сложность*, если ∃ полином $p(x)$ такой, что на ∀ входе длины n алгоритм завершает вычисления после выполнения не более чем $p(n)$ операций.

Задача разрешима за полиномиальное время или полиномиально разрешима, если для неё ∃ полиномиальный алгоритм.

Класс P - множество всех полиномиально разрешимых задач.

$$\begin{array}{l} a_1\, a_2 \ldots a_n \\ b_1\, b_2 \ldots b_n \\ \hline \end{array}$$

Школьный алг. сложения - сложность – $0(n)$.
Школьный алг. умножения - сложность $0(n^2)$.

Алг. Шёнхаге-Штрассена умножения - сложность $0(n \log_2 n \log_2 \log_2 n)$.

$x =$	a	b

$y =$	c	d

Считаем, что число цифр в числе есть степень числа 2. Тогда

$$x\,y = (a\,2^{n/2} + b)(c\,2^{n/2} + d) = a\,c\,2^{n} + (a\,d + b\,c)\,2^{n/2} + b\,d.$$

Тогда нужно $0(n^{\log_2(3)}) \approx 0(n^{1,59})$ битовых операций.

Обычный метод умножения матриц - сложности $0(n^3)$. Алгоритм Штрассена имеет сложность $0(n^{\log_2 7})$.

Полиномиальный алгоритм $\leftrightarrow$ **Эффективный алгоритм**

P – класс задач решаемых за полиномиальное время (класс эффективно решаемых задач)

НАХОЖДЕНИЕ НАИБОЛЬШЕГО И НАИМЕНЬШЕГО ЭЛЕМЕНТОВ

Множество *S* имеет *n* чисел. Наибольший элемент можно найти, проводя *n - 1* сравнений. Для выбора наименьшего элемента тоже требуется *n - 1* сравнений. В итоге потребуется *2 n - 2* сравнений.

Принцип «разделяй и властвуй» (стратегия дублирования):

```
            procedure MAXMIN(S):
1.    if |S|=2 then   begin
2.          пусть S={a,b};
3.          return(MAX(a,b),MIN(a,b))
                        end
      else
         begin
4.          разбить S на два подмнож. S1 и S2  с одинак. числом
            элементов;
5.          (max1, min1)←MAXMIN(S1);
6.          (max2, min2)←MAXMIN(S2);
7.          return(MAX(max1, max2). MIN(min1, min2))
         end
```

Сравнения происходят на 3-м шаге и на шаге 7, где сравниваются *max1* с *max2* и *min1* с *min2*. Пусть *T (n)* – число сравнений. Тогда:

$$T(n) = \begin{cases} 1, & \text{если } n = 2, \\ 2\,T(n/2) + 2, & \text{если } n > 2. \end{cases}$$

$$T(n) = (3/2)\,n - 2.$$

ЗАВИСИМОСТИ ОТ СЛОЖНОСТИ АЛГОРИТМА И ОТ БЫСТОДЕЙСТВИЯ ЭВМ

Зависимость от сложности алгоритма				
Алгоритм	Временная сложность алгоритма	Максимальный размер задачи, решаемой за указанное время		
		1 с	1 мин	1 час
A1	*N*	*1 000*	*60 000*	*3 600 000*
A2	N^2	*31*	*244*	*1 897*
A3	N^3	*10*	*39*	*153*
A4	2^N	*9*	*15*	*21*

Зависимость от быстродействия ЭВМ			
Алгоритм	Временная сложность алгоритма	Максимальный размер задачи	
		до ускорения	после ускорения в 10 раз
A1	*N*	*S1*	*10 S1*
A2	N^2	*S2*	*3,16 S2*
A3	N^3	*S3*	*2,15 S3*
A4	2^N	*S4*	*S5+3,3*

NP КЛАСС

***NP* – класс задач, решение которых может быть быстро проверено.**

NP - класс задач распознавания, для которых существуют проверяющие алгоритмы, работающие полиномиальное время, причем длина сертификата ограничена полиномиально от размера задачи.

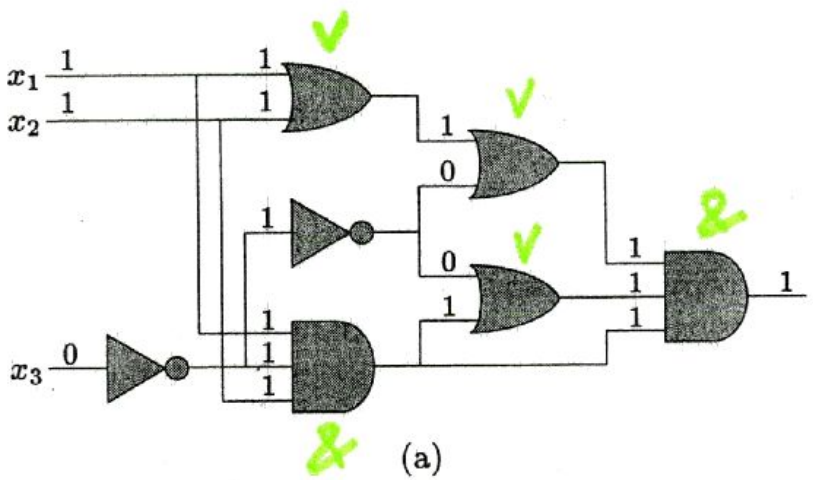
Например, сертификатом задачи МОД является дерево на *V*. Проверяющий алгоритм выясняет, является ли предъявленное решение остовным деревом и будет ли общая длина всех его ребер не превосходить число *L*.

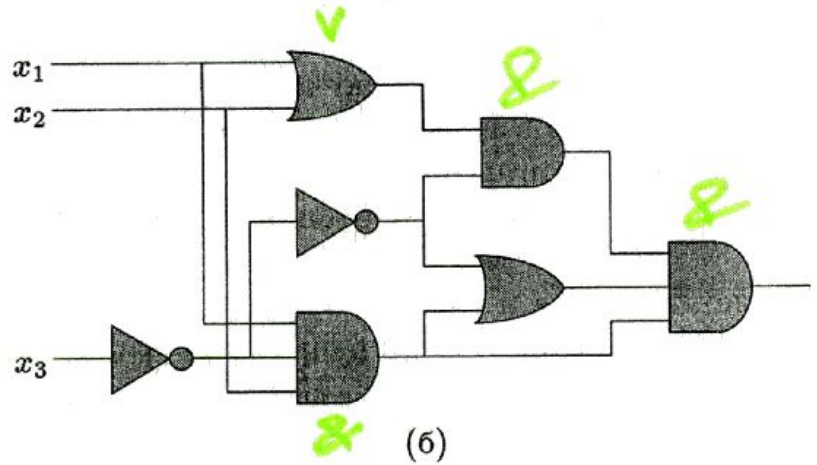
NP - класс задач «разрешимых на Недетерминированной машине Тьюринга за Полиномиальное время». Примеры задач из класса ***NP***:

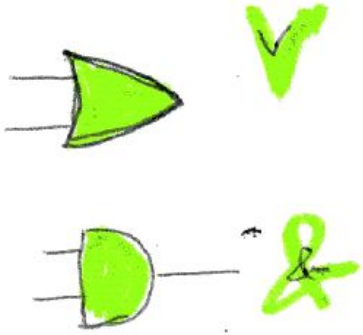
1) выяснение выполнимости формулы логики высказываний;

ПРИМЕРЫ ЗАДАЧ ИЗ *NP* КЛАССА

1. Выполнимость формулы логики высказываний ↔ Выполнимость схемы

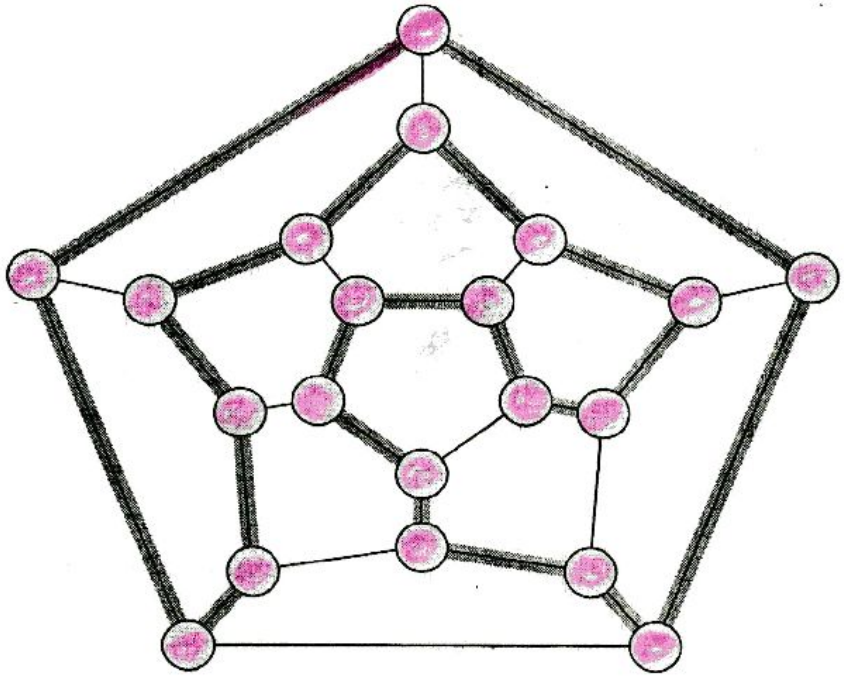




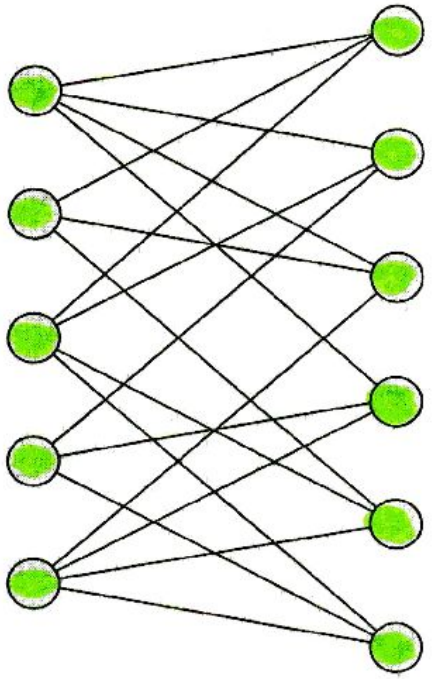


ПРИМЕРЫ ЗАДАЧ ИЗ *NP* КЛАССА

2) выяснение гамильтоновости графа



(а)



(б)

ПРИМЕРЫ ЗАДАЧ ИЗ *NP* КЛАССА

1) выяснение выполнимости формулы логики высказываний;
2) выяснение гамильтоновости графа;
3) задача распознавания простого числа;
4) задача коммивояжёра;
5) нахождение целоч. решений системы линейных уравнений;
6) размещение обслуживающих центров.
7) оптимальный раскрой;
8) составление расписаний, учитывающих определённые условия;
9) оптимальная загрузка ёмкости (рюкзак, поезд, корабль, самолёт);
10) динамическое распределение памяти;
11) организация памяти в виде корневого дерева;
12) выяснение достаточности числа регистров для реализации циклов.

P = NP ?

~

Если на какой то вопрос есть положительный ответ и его можно проверить быстро (полиномиально), то верно ли, что и ответ можно найти так же быстро?

NP – ПОЛНЫЕ И *NP* - ТРУДНЫЕ ЗАДАЧИ

Задача Z_1 сводится к задаче Z_2, если метод решения Z_2 можно преобразовать в метод решения Z_1.

Задача называется *NP - трудной,* если каждая задача из *NP* полиноминально сводится к ней.

Задача называется *NP - полной*, если она входит в *NP* и каждая задача из *NP* полиноминально сводится к ней (т.е. является *NP*-трудной).

***NP* - полные задачи понимаются как самые трудные задачи из класса *NP*. Класс *NP* - полных задач обладает сл. свойствами.**

1. **Никакую *NP* - полную задачу нельзя решить никаким известным полиномиальным алгоритмом.**
2. **Если бы удалось построить полиномиальный алгоритм для какой-нибудь *NP* - полной задачи, то это бы означало полиномиальную разрешимость каждой *NP* - полной задачи.**

Предполагают, что $P \neq NP$.

***NP* - полные задачи по существу труднорешаемы и нет возможности решить на практике такие задачи, за исключением очень малого числа индивидуальных задач.**

ПРИМЕРЫ NP - ПОЛНЫХ ЗАДАЧ

Теорема (теорема Кука). Задача выяснения выполнимости формулы логики высказываний, является *NP*-полной.

+ все примеры из предыдущего списка:

1) **выяснение выполнимости формулы логики высказываний;**
2) **выяснение гамильтоновости графа;**
3) **задача распознавания простого числа;**
4) **задача коммивояжёра;**
5) **нахождение целоч. решений системы линейных уравнений;**

...

КЛАСС *E*

***Первое определение*: Алгоритм имеет полиномиальную временную сложность, если ∃ полином *p(x)* такой, что на ∀ входном слове длины *n* алгоритм завершает вычисления после выполнения не более чем *p(n)* операций. Если такого полинома не существует, *временная сложность считается экспоненциальной*.**

***Второе определение*: алгоритм имеет *экспоненциальную временную сложность*, если его временная сложность имеет порядок не меньше, чем $c\,a^{n}$, где c ($c > 0$), a ($a > 1$) – постоянные:**

$c\,a^{n} \le f(n)$ **для** $\forall\, n$ **, кроме конечного числа значений** n.

При первом определении, задача, имеющая сложность порядка $0(n^{\log_2 n})$ [$0(n^{\log_2 n}) = 0(2^{(\log_2 n)^2})$] **будет отнесена к задаче с экспоненциальной временной сложностью, а по второму определению – нет. Отметим, что функция** $n^{\log_2 n}$, **хотя и растет быстрее любого полинома, но медленнее, чем** 2^{n}.

ПРИМЕР ЭКСПОНЕНЦИАЛЬНОГО АЛГОРИТМА

Рассмотрим задачу линейного программирования:

$$z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \rightarrow min\ (max) \qquad (1)$$

при ограничениях:

$$a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \geq b_1,$$

$$a_{21} x_1 + a_{21} x_2 + \ldots + a_{2n} x_n \geq b_2, \qquad (2)$$

$$\ldots$$

$$a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \geq b_m,$$

$$x_i \geq 0,\ 1 \leq i \leq n.$$

Для решения этой задачи известен **симплекс-метод, который имеет экспоненциальную временную сложность**, но метод широко используется и является результативным.

ЭКСПОНЕНЦИАЛЬНЫЙ АЛГОРИТМ

Для решения задачи линейного программирования существует симплекс-метод, который хотя и экспоненциален в худшем случае, но уверенно работает для задач достаточно большого размера.

Для этой задачи Л. Г. Хачиян в 1979 г. предложил алгоритм эллипсоидов, который имеет полиномиальную временную сложность.

Метод ветвей и границ успешно решает задачу о рюкзаке, хотя этот алгоритм имеет экспоненциальную временную сложность.

Экспоненциальная сложность задачи имеет следующие аспекты:

-для отыскания решения требуется экспоненциальное время;

-искомое решение может быть настолько велико, что не может быть представлено выражением, длина которого была бы ограничена полиномом от длины входа.

Вторая ситуация возникает, например, если в задаче коммивояжера требуется найти все маршруты длины, не превосходящей заданной величины *L*. Может оказаться, что имеется экспоненциальное число маршрутов длины, не превосходящей *L*.

ЁМКОСТНАЯ (ЛЕНТОЧНАЯ) СЛОЖНОСТЬ АЛГОРИТМА

Ёмкостная, или ленточная, сложность алгоритма характеризует необходимую для вычисления память для хранения исходных данных, промежуточных результатов и окончательного результата.

Определим $S_f(x)$ как длину активной зоны при работе машины T на числе x, если $f(x)$ определено. В противном случае значение $S_f(x)$ будем считать неопределённым. Функцию $S_f(x)$ назовём *ленточной сложностью*.

Теорема. Для ленточной сложности вычисления функции f имеет место:

$$S_f(x) \le x + 1 + t_f(x),$$

где $t_f(x)$ -временная сложность вычисления $f(x)$.

ЗАДАЧА РАСПОЗНАВАНИЯ

Дан граф G. Существует ли в G цикл, содержащий каждую вершину графа G в точности по одному разу?

Эта задача распознавания, ибо ответ должен быть «да» либо «нет». Доказано, что эта задача является *NP*-полной.

Рассмотрим версию той же задачи, которая считается дополнительной к ЗГЦ.

Дан граф G. Является ли G негамильтоновым?

Здесь уже не существует общего проверяющего алгоритма кроме перебора всех возможных перестановок вершин. Список всех перестановок имеет экспоненциальную длину.

ЗАДАЧИ РАЗРЕШЕНИЯ – ОПТИМИЗАЦИОННЫЕ ЗАДАЧИ

В теории *NP*- полноты рассматриваются только *задачи разрешения* – задачи, в которых имеется ответ “да” или “нет”.

Многие задачи можно преобразовать к такому виду. Например, рассмотрим задачу поиска кратчайшего пути в графе - задачу ПУТЬ – “по заданному графу $G=(V,X)$, паре вершин $u,v \in V$ и натуральному числу k определить, существует ли в G путь между вершинами u и v, длина которого не превосходит k”.

Тогда ПУТЬ(i)=1, если длина кратчайшего пути между вершинами u и v не превосходит k, и ПУТЬ(i)=0 в противном случае.

В задачах оптимизации требуется минимизировать или максимизировать значение некоторой величины. Прежде чем ставится вопрос о *NP*-полноте таких задач, их следует преобразовать в задачу разрешения (распознавания). Обычно в качестве такой задачи разрешения рассматривают задачу проверки, является ли некоторое число верхней (или нижней) границей для оптимизируемой величины. Если после этого получается *NP*- полная задача, то тем самым установлена трудность исходной задачи.

Оптимизационные задачи обычно не являются задачами распознавания. Оптимизационная задача считается *NP*-трудной, если соответствующая ей задача распознавания является *NP*-полной.

СВЕДЕНИЕ ОПТИМИЗАЦИОННОЙ ЗАДАЧИ К ЗАДАЧЕ РАСПОЗНАВАНИЯ - 1

Дан связный граф $G=(V, E)$, $|V|=n$ и $n \times n$ симметричная матрица расстояний (d_{ij}) с целыми d_{ij}, $0 < d_{ij} < \infty$, $(v_i, v_j) \in E$ и $d_{ij} = \infty$ если $(v_i, v_j) \notin E$. Требуется найти остовный связный подграф (дерево) которое имеет минимальную общую длину всех ее ребер.

Можно переформулировать эту задачу как задачу распознавания:

Дан связный граф $G=(V, E)$ с матрицей расстояний, заданной в примере 1.25 и целое L. Существует ли остовное дерево для G общая длина ребер которого не превосходит L?

СВЕДЕНИЕ ОПТИМИЗАЦИОННОЙ ЗАДАЧИ К ЗАДАЧЕ РАСПОЗНАВАНИЯ - 2

Задача коммивояжера (ЗК). Дан полный граф $G=(V,E)$ с $n\times n$-симметричной матрицей расстояний $D=(d_{ij})$. ЗК состоит в нахождении тура с минимальной общей длиной. Пусть $\pi(j)$ номер вершины, которая посещается после вершины j, $j=1,\ldots,n$. Тогда оптимизационная задача состоит в следующем:

$$min \sum_{j=1}^{n} d_{j\pi(j)}$$

Представление ЗК в виде задачи распознавания.

Дан полный граф $G=(V, E)$, $|V|=n$ с симметричной $n\times n$ матрицей расстояний $D=(d_{ij})$ и дано целое число L. Существует ли тур π такой, что

$$\sum_{j=1}^{n} d_{j\pi(j)} \le L?$$

Дополнение ЗК. Дан полный граф $G = (V, E)$, $|V| = n$ с симметричной матрицей расстояний (d_{ij}) и задано целое L. Выяснить будет ли для всех туров выполняться условие

$$\sum_{j=1}^{n} d_{j\pi(j)} > L.$$