ЛЕКЦІЯ 2

Міри складності алгоритмів. Класи задач P і NP

Міри складності алгоритмів

Існує множинність алгоритмів для розв'язування однієї задачі

Одну і ту ж задачу можна розв’язувати багатьма алгоритмами. Ефективність їх роботи залежить від різних факторів:

- кваліфікації програміста,
- застосованої системи програмування,
- потужності обчислювальних засобів,
- застосовуваних методів обчислень.

Критерії оцінки алгоритмів.

Основний критерій — кількість "часу", необхідного для виконання алгоритму.

Час вимірюється не реальним числом секунд роботи комп'ютера, а опосередковано через *число операцій*, виконуваних алгоритмом.

Чому не можна користуватися реальним часом для оцінки алгоритмів?

Фактична кількість секунд, необхідна для виконання алгоритму на комп'ютері, непридатна для аналізу, оскільки нас цікавить **тільки відносна ефективність алгоритму**, що вирішує конкретну задачу.

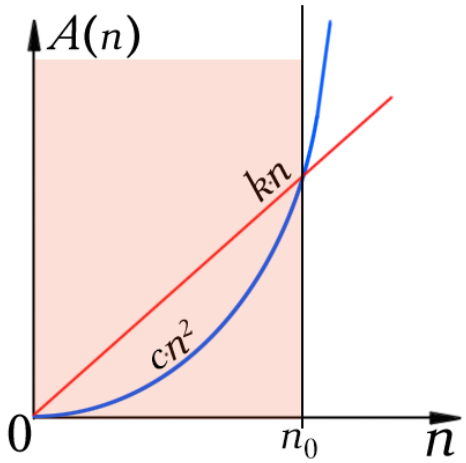
Дійсно, алгоритм не стає кращим, якщо його перенести на більш швидкий комп'ютер, або не стає гіршим, якщо його виконувати на більш повільному комп'ютері.

Чому не можна оцінювати алгоритм без обліку вхідних даних?

Фактична кількість операцій алгоритму на одному наборі даних не представляє великого інтересу й мало його характеризує.

Трудомісткість розв'язування задачі суттєво й дуже часто нелінійно залежить від обсягу вхідних даних.

Тому порівнювані алгоритми на різних обсягах вхідних даних можуть мати різні характеристики.



Може виникнути ситуація, що перший алгоритм *на малих обсягах може характеризуватися однаковою або навіть більшою кількістю операцій*, ніж другий.

Однак на більших обсягах він буде вимагати суттєво меншої кількості операцій, ніж другий.

$n -$ кількість байт вхідних даних, $A\left(n\right) -$ кількість операцій

Критерії оцінки алгоритму

Основний критерій оцінки алгоритмів – залежність числа операцій конкретного алгоритму від обсягу вхідних даних.

Ми можемо порівняти два алгоритми **за швидкістю зростання числа операцій від зростання обсягу вхідних даних**.

Саме швидкість зростання відіграє ключову роль.

Золоте правило для алгоритмів

Програємо в кількості операцій – виграємо в обсязі використовуваної пам'яті і навпаки, виграємо в кількості операцій – програємо в обсязі використовуваної пам'яті (*пам'яті потрібно **більше***).

Перекоси в золотому правилі

Раніше комп'ютери мали невеликий обсяг пам'яті.

Intel 8088 (5МГц)/
RAM128(256) КБ/
HDD 10(20) МБ

Доводилося вибирати більш повільний алгоритм, якщо він вимагав менше пам'яті.

Сьогодні. Розробники сучасних програм не відчувають потреби в економії пам'яті.

Intel Core i9-7960X (2.8 ГГц), 16 ядер /
RAM 128 ГБ, 4x32ГБ /
SSD 480 ГБ + HDD 2 ТБ /
Asus PCI-Ex GeForce GTX 1080 Ti ROG Strix OC 11GB

$$\frac{128000\text{МБ}}{0.256\text{МБ}} = 500000, \frac{2800\text{МГц}}{5\text{МГц}} = 560, \frac{500000}{560} \approx 893$$

Обчислювальна складність

В теорії алгоритмів:
обчислювальна складність алгоритму — це *функція*, що визначає *залежність обсягу роботи*, виконуваної деяким алгоритмом, *від обсягу вхідних даних*.

$$\text{Обсяг роботи} = F\left(\text{Обсяг даних}\right)$$

Обсяг роботи зазвичай вимірюють категоріями часу.

Обсяг даних визначають представленими **обчислювальними ресурсами.**

Час визначається кількістю елементарних кроків, необхідних для розв'язування задачі.

Простір визначається обсягом пам'яті (біти, байти) або місцем на носієві даних (доріжки, сектори).

Обчислювальну складність також називають часовою складність

Обчислювальна (часова) складність алгоритму — це функція від обсягу вхідних даних, яка дорівнює

максимальній, середній або мінімальній

кількості елементарних операцій, виконуваних алгоритмом для розв'язування задачі.

Нехай $V_i = f_i(n)$ – кількість операцій, виконуваних i–м алгоритмом при розв'язуванні задачі.

Розглядають обчислювальні складності W_1, W_2 та W_3 :

- у найгіршому випадку, $W_1 = \max_i V_i$
- середній час роботи алгоритму $W_2 = M(V_i)$,
- у найкращому випадку $W_3 = \min_i V_i$

Швидкість росту алгоритму

Швидкістю росту алгоритму називається швидкість зростання числа операцій при зростанні обсягу вхідних даних.

Нас цікавить **тільки загальний характер поведінки**

$$an^k \Rightarrow n^k, \qquad an^k + c \Rightarrow n^k, \qquad an^k + cn^{k-1} \Rightarrow n^k$$

алгоритму, тобто **клас функції швидкості росту**,

$$n, \quad n\log_2 n, \quad n^2, \quad n^3 \quad \text{та} \quad 2^n, \quad n!$$

до якого належить алгоритм.

Основні класи функцій наведені в таблиці.

Таблиця класів росту функцій

n	$log_2 n$	n^2	n^3	2^n	$n!$
1	0	1	1	2	1
2	1	4	8	4	2
5	2.3	25	125	32	120
10	3.3	100	1000	1 024	362880
15	3.9	225	3375	32 768	1.4×10^{12}
20	4.3	400	8000	1 048 576	2.5×10^{18}
30	4.9	900	27000	1 073 741 824	2.7×10^{32}

При **невеликих обсягах** вхідних даних значення функцій мало відрізняються, однак при зростанні обсягів різниця суттєво зростає.

Принципи вибору класів росту функцій

З ростом обсягу даних **швидкозростаючі функції домінують** над функціями з повільним ростом.

Правило. Якщо складність алгоритму залежить від суми функцій, що ростуть із різною швидкістю, то відкидають усі функції з меншим ростом**. Залишають тільки ті** функції, які **ростуть найшвидше**.

$$an^k \Rightarrow n^k, \quad an^k + c \Rightarrow n^k, \quad an^k + cn^{k-1} \Rightarrow n^k$$

Приклад. Нехай установлено, що алгоритму потрібно n^3–$30n$ операцій, де n – обсяг вхідних даних.

Будемо вважати, що складність алгоритму росте як n^3. Причина цього в тому, що вже при 100 вхідних даних різниця між n^3 і n^3 - 30*x* становить лише 0,3%.

Причина. $n = 6: \; n^3 - 30n = 6^3 - 30 \cdot 6 = 216 - 180 = 36 \;\; x^3 = 6^3 = 180$

$n = 100: \; n^3 - 30n = 100^3 - 30 \cdot 100 = 997000 \;\; x^3 = 100^3 = 1000000$

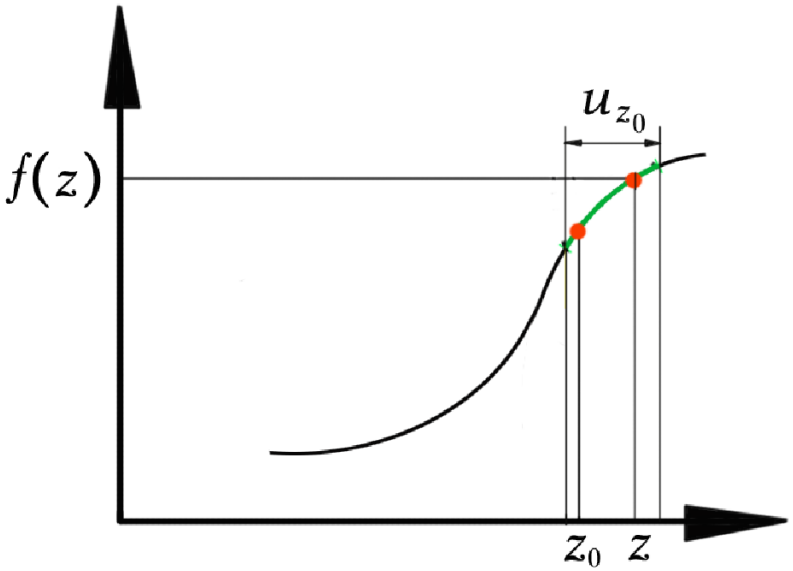
$$\frac{180 - 36}{180} \cdot 100\% = 80\% \qquad \frac{1000000 - 997000}{1000000} \cdot 100\% = 0.3\%$$

Асимптотична складність

При використанні вхідних даних великого обсягу й при оцінці **порядку зростання** часу роботи алгоритму, приходять до поняття **асимптотичної складності** алгоритму.

Порядком **зростання** функції f в точці z_0 називається деяке число $a \geq 0$ таке, що для деякого околу u_{z_0} існує таке число $M > 0$, що для довільної точки $z \in u_{z_0}$ виконується нерівність

$$|f(z)| \leq \frac{M}{|z - z_0|^a}.$$



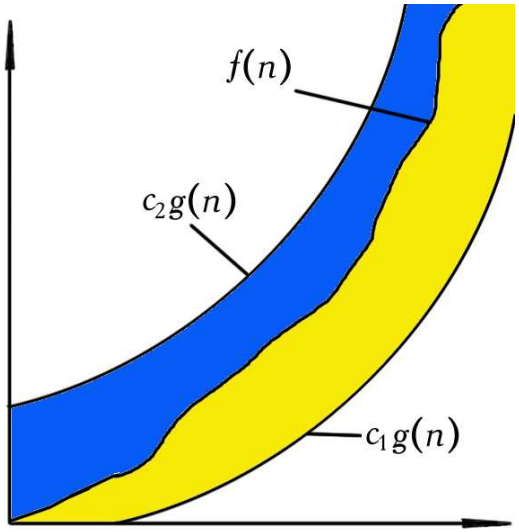
Асимптотична обчислювальна складність Θ

Якщо $f(n)$ і $g(n)$ – деякі функції, то запис $f(n) = \Theta(g(n))$

означає, що найдуться такі $c_1, c_2 > 0$ й таке n_0, що

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$

для всіх $n \ge n_0$.



У цьому випадку говорять, що $g(n)$ є **асимптотично точною оцінкою** для $f(n)$.

Скорочений запис:

$$\exists(c_1, c_2 \ge 0), n_0 : \forall(n \ge n_0) \Rightarrow |c_1 g(n)| \le |f(n)| \le |c_2 g(n)|$$

Приклад. $f(n) = n^2 - 3n$, $g(n) = n^2$. $n_0 = 3$

При $n = 1 \to f(n) = 1^2 - 3 \cdot 1 = -2$; $n = 3 \to f(n) = 3^2 - 3 \cdot 3 = 0$

Властивості асимптотичної оцінки Θ

Однак це позначення слід вживати з обережністю:

З того, що $f_1\left(n\right) = \Theta\left(g\left(n\right)\right)$ і $f_2\left(n\right) = \Theta\left(g\left(n\right)\right)$, не слід заключати, що $f_1\left(n\right) = f_{\ 2}\left(n\right)$!

$$f_1(n) = n^2 - 3n,\ f_2\left(n\right) = 2n^2 + 5n,\ n^2 - 3n \neq 2n^2 + 5n$$

1. Визначення $\Theta\left(g\left(n\right)\right)$ припускає, що функції $f\left(n\right)$ і $g\left(n\right)$ **асимптотично невід'ємі**, тобто **невід'ємі** для достатньо великих значень n.

2. Якщо f й g **строго додатні**, то можна виключити n_0 з визначення (змінивши константи c_1 й c_2 так, щоб для малих n нерівність також виконувалася).

Приклад 1. Перевіримо, що $\frac{n^2}{2} - 3n = \Theta\left(n^2\right)$. Згідно з визначенням, треба вказати додатні константи c_1, c_2 і число n_0 так, щоб нерівність $c_1 n^2 \le \frac{n^2}{2} - 3n \le c_2 n^2$ виконувалася для всіх $n \ge n_0$.

1. Розділимо дану нерівність на n^2: $c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$
2. Розглянемо окремо випадки $c_1 \le \frac{1}{2} - \frac{3}{n}$ і $c_2 \ge \frac{1}{2} - \frac{3}{n}$

Нехай $n_0 = 7$, Тоді $c_1 \le \frac{1}{2} - \frac{3}{7} = \frac{7-6}{14} = \frac{1}{14} \Rightarrow c_1 = \frac{1}{14}$

1) При $n_0 = 7$: $\frac{1}{2} - \frac{3}{n} = \frac{1}{14} > 0$ Нехай $n \ge 7$ тоді $1/2 - 3/n \ge 1/14$

При $n \to \infty$: $1/2 - 3/n < 1/2 \Rightarrow c_2 = \frac{1}{2}$

Отже $\frac{1}{14} n^2 \le \frac{n^2}{2} - 3n \le \frac{1}{2} n^2$ при $n \ge 7$.

Доведено, що $\frac{n^2}{2} - 3n = \Theta\left(n^2\right)$

Приклад 2. Покажемо, що $6n^3 \neq \Theta\left(n^2\right)$.

Насправді, припустимо, що знайдуться такі c_1, c_2 і n_0, що $c_1 n^2 \leq 6n^3 \leq c_2 n^2$ для всіх $n \geq n_0$.

1. Розділимо нерівність n^2: $c_1 \leq 6n \leq c_2$,
2. Розглянемо окремо випадки: $c_1 \leq 6n$ і $c_2 \geq 6n$
3. Нехай $n_0 = 1$, тоді $c_1 = 6n_0 = 6$. Якщо $n < 1$ то $c_1 = 6 > 6n$
4. Розглянемо нерівність $6n \leq c_2$. Для довільного c_2 можемо підібрати n при якому $6n \leq c_2$ не справджується.
5. Ситуація з п. 4 справедлива при довільному додатному n_0

Отже $6n^3 \neq \Theta\left(n^2\right)$ оскільки неможливо підібрати c_2

Асимптотичні обчислювальні складності O та Ω.

Запис $f(n) = \Theta(g(n))$ містить у собі дві оцінки: верхню й нижню. Їх можна розділити.

Якщо $f(n)$ *та* $g(n)$ *– деякі функції, то*

- *запис* $f(n) = O(g(n))$ *означає, що знайдеться така константа* $c > 0$ *й таке* n_0*, що* $f(n) \le cg(n)$ *для всіх* $n \ge n_0$. Скорочено:

$$\exists c, n_0 : \forall (n \ge n_0) \Rightarrow |f(n)| \le |cg(n)|$$

- *запис* $f(n) = \Omega(g(n))$ *означає, що знайдеться така константа* $c > 0$ *й таке* n_0*, що* $0 \le cg(n) \le f(n)$ *для всіх* $n \ge n_0$.

$$\exists c, n_0 : \forall (n \ge n_0) \Rightarrow |cg(n)| \le |f(n)|$$

Теорема 1.

1. Для будь-яких двох функцій $f\left(n\right)$ і $g\left(n\right)$ властивість $f\left(n\right)=\Theta\left(g\left(n\right)\right)$ справджується тоді і тільки тоді, коли $f\left(n\right)=O\left(g\left(n\right)\right)$ та $f\left(n\right)=\Omega\left(g\left(n\right)\right)$.

2. Для будь-яких двох функцій $f\left(n\right)$і $g\left(n\right)$ властивість $f\left(n\right)=O\left(g\left(n\right)\right)$ справджується тоді й тільки тоді, коли $g\left(n\right)=\Omega\left(f\left(n\right)\right)$.

Приклад.

Оскільки $an^2+bn+c=\Theta\left(n^2\right)$ (при $a>0$).

Тому $an^2+bn+c=O\left(n^2\right)$.

Асимптотичні оцінки O, Θ і Ω – це *однобічні* **рівності**.

Функції O, Θ і Ω
стоять тільки **справа** від знака рівності.

$$f(n) = \Theta(g(n)),$$

$$f(n) = O(g(n)),$$

$$f(n) = \Omega(g(n))$$

Приклад.

Нехай $f(n) = an^2 + b$ **. Тоді** $f(n) = \Theta(n^2)$ **або** $an^2 + b = \Theta(n^2)$

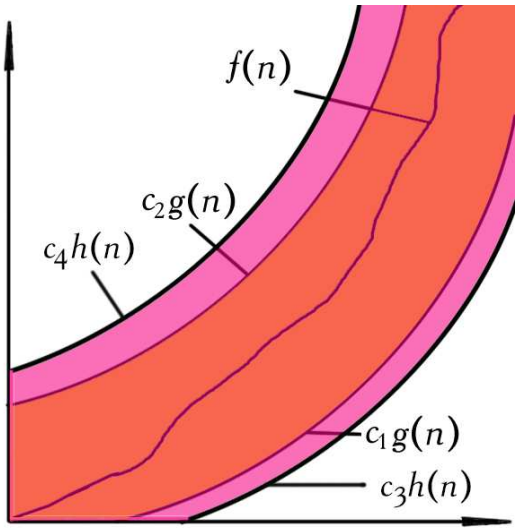
Властивості асимптотичних обчислювальних складностей

Транзитивність

Нехай існує асимптотична складність: $f\left(n\right) = \Theta\left(g\left(n\right)\right)$

Нехай також існує асимпт. складність: $g\left(n\right) = \Theta\left(h\left(n\right)\right)$.

Тоді існує асимптотична складність: $f\left(n\right) = \Theta\left(h\left(n\right)\right)$

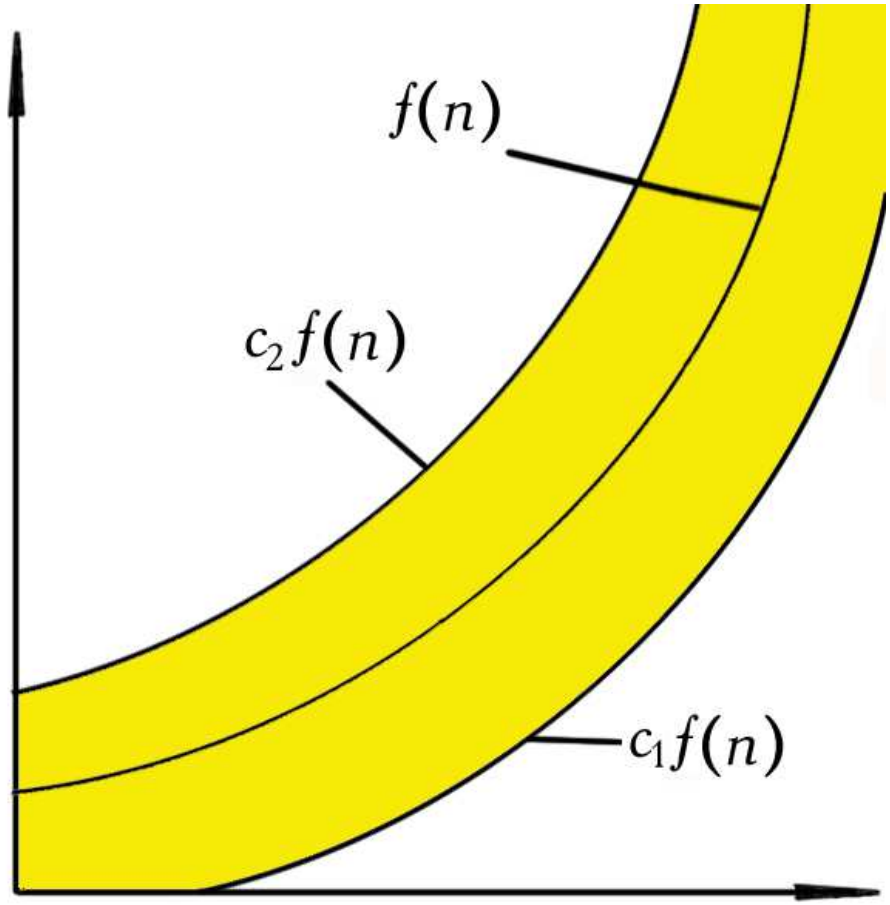


$f\left(n\right) = O\left(g\left(n\right)\right)$ і $g\left(n\right) = O\left(h\left(n\right)\right)$ спричиняє $f\left(n\right) = O\left(h\left(n\right)\right)$

$f\left(n\right) = \Omega\left(g\left(n\right)\right)$ і $g\left(n\right) = \Omega\left(h\left(n\right)\right)$ спричиняє $f\left(n\right) = \Omega\left(h\left(n\right)\right)$

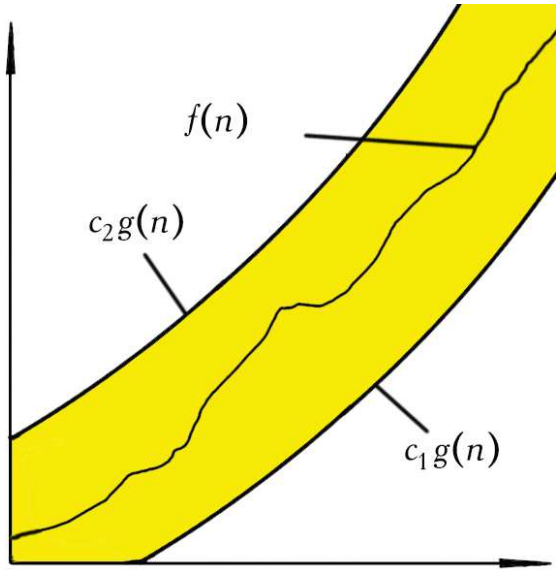
Рефлексивність

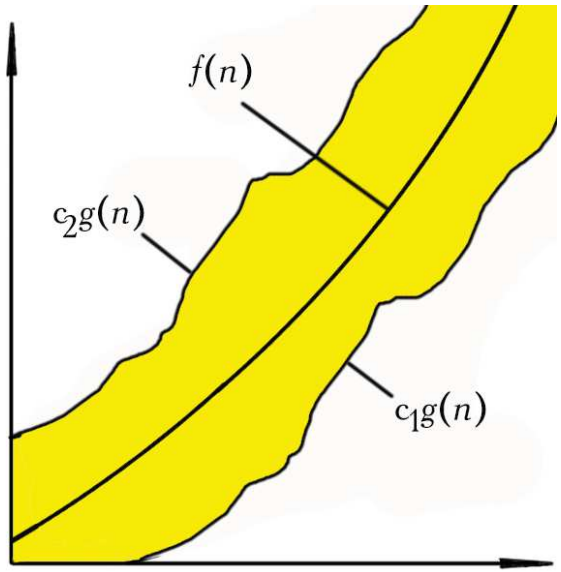
$$f(n) = \Theta(f(n)),\ f(n) = O(f(n)),\ f(n) = \Omega(f(n))$$



Симетричність

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$





Алгоритми поліноміальної складності (клас P)

Більшість алгоритмів мають поліноміальний порядок складності, тобто $O(n)$, $O(n^2)$, $O(n^3)$,...,$O(n^k)$, де $k = const$

Поліном: $f(x) = c_0 + c_1 x^1 + \ldots + c_m x^m$

Лінійна складність $O(n)$

Довільний алгоритм називається алгоритмом з **лінійною складністю** $O(n)$, якщо **кількість операцій** перебуває в лінійній залежності від обсягу вхідних даних.

Приклад 1. Пошук у несортованому списку

Алгоритм простого пошуку в несортованому списку є алгоритмом лінійної складності.

```
while (i<=n && a[i]!=k)i++;
```

Очевидно, що чим далі в списку перебуває конкретне значення ключа, тим більше операцій необхідно виконати для його пошуку.

Квадратична складність $O\left(n^2\right)$

Довільний алгоритм називається алгоритмом із **квадратичною складністю** $O\left(n^2\right)$, якщо **кількість операцій** перебуває у квадратичній залежності від обсягу вхідних даних.

Приклад 2. Сортування вибором.

1. Знаходимо найменший елемент і міняємо його місцями з першим.
2. Серед елементів, що залишилися, знаходимо найменший і міняємо його місцями із другим і т. д.

```
for (int i:=1;i<=n;i++){
for (int j:=i+1;j<=n;j++){
if (a[i]>a[j]){
x=a[i]; a[i]:=a[j]; a[j]:=x;}
}
}
```

Алгоритм має квадратичну складність: $O\left(n^2\right)$

Приклад 2. Бульбашкове сортування

1. Проходимо по масиву з кінця до початку.
2. На шляху переглядаємо пари сусідніх елементів.
3. Якщо елемент із більшим індексом менший за величиною, то міняємо місцями елементи в парі.
4. Після першого проходу на початку масиву виявиться найменший елемент.
5. Наступний прохід робиться до другого елемента.
6. Усього виконується *n* проходів.

```
for (int i=1; i<=n; i++) {
int j=n;
while (j>=i+1) {
if (a[i]>a[j]) { k=a[i]; a[i]=a[j];a[j]=k;}
j=j-1;
}}
```

Алгоритм має квадратичну складність: $O\left(n^2\right)$

Кубічна складність $O\left(n^3\right)$

Довільний алгоритм називається алгоритмом з **кубічною складністю** $O\left(n^3\right)$**, якщо кількість опреацій** перебуває в кубічній залежності від обсягу вхідних даних.

Приклад. Множення матриць.

Правило множення двох матриць.

1. Для обчислення добутку двох матриць i-ий рядок першої матриці поелементно множиться j-й стовпець другої матриці.

2. Потім обчислюється сума таких добутків і записується в клітинку (i, j) результуючої матриці.

Приклад множення матриць

$$A_{3\times4} \bullet B_{4\times2} = C_{3\times2}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{pmatrix}$$

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}; \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42}; \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41}; \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}; \\ c_{31} &= a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41}; \\ c_{32} &= a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42}; \end{aligned}$$

```
var
 a: array[1..3,1..4] of integer;
 b: array[1..4,1..2] of real;
 c: array[1..3,1..2] of real
 for i := 1 to 3 do
  for j := 1 to 2 do
  begin
   c[i,j] := 0;
   for k := 1 to 4 do
    c[i,j] := c[i,j] + a[i,k] * b[k,j];
  end;
```

Стандартний алгоритм множення матриці розміром $a \times b$ на матрицю розміром $b \times c$ виконує abc множень і $a\left(b-1\right)c$ додавань.

Обчислювальна складність алгоритму множення матриць

	Множення	Додавання
Стандартний алгоритм	n^3	$n^3 - n^2$
Алгоритм Виноградова	$\frac{n^3 + 2n^2}{2}$	$\frac{3n^3 + 4n^2 - 4n}{2}$
Алгоритм Штрассена	$n^{2.81}$	$6n^{2.81} - 6n^2$

Розглянуті алгоритми з асимптотичною обчислювальною складністю

$$O(n), \quad O\left(n^2\right), \quad O\left(n^3\right)$$

називаються алгоритмами з **поліноміальною складністю або алгоритмами класу** P.

1. Алгоритми множення матриць – складність $O\left(n^3\right)$.

2. Алгоритми сортування – складність $O\left(n^2\right)$.

3. Алгоритми лінійного пошуку – складність $O\left(n\right)$.

Основна риса алгоритмів поліноміальної складності – ВОНИ ДОЗВОЛЯЮТЬ ЗНАЙТИ РОЗВ'ЯЗОК ЗАДАЧІ ЗА ПРИЙНЯТНИЙ ПРОМІЖОК ЧАСУ.

Алгоритми недетермінованої поліноміальної складності (клас NP задач)

Клас **NP**-задач – це клас недетермінованої поліноміальної складності.

1. Складність усіх відомих детермінованих алгоритмів для розв'язування цих задач або експоненціальна, або факторіальна

2. Складність деяких з них дорівнює $O\left(2^n\right)$, де n – кількість вхідних даних.

Якщо для розв'язування такої задачі на вході з 10 елементів алгоритму було потрібно 1024 операцій, то на вході з 11 елементів число операцій складе вже 2048. Це значне зростання часу при невеликому збільшенні кількості вхідних даних.

Технологія розв'язування задач класу NP

Термін «**недетерміновані поліноміальні**», що характеризує задачі із класу **NP**, пояснюється наступним двокроковим підходом до їх розв'язування.

Крок 1. Створюють алгоритм, що генерує *можливий* розв'язок такої задачі.

Якщо така спроба виявляється успішною, то одержуємо оптимальний або близький до оптимального розв'язок. *Однак найчастіше розв'язок буває далеким від оптимального*.

Крок 2. Перевіряють, чи дійсно розв'язок, отриманий на першому кроці, є розв'язком вхідної (початкової) задачі.

Як визначається загальний час розв'язування задач класу NP

Кожний із цих кроків окремо вимагає поліноміального часу.

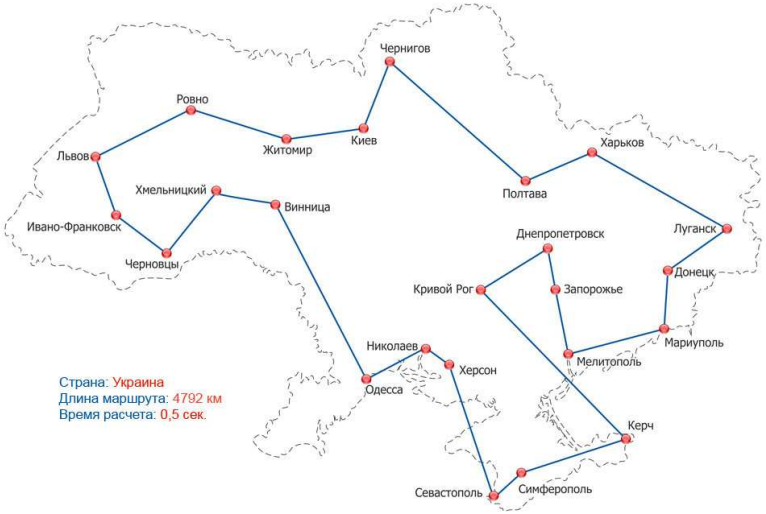
Проблема, однак, у тому, що ми не знаємо, скільки разів нам доведеться повторити обидва кроки, щоб одержати шуканий розв'язок.

Хоча обидва кроки й поліноміальні, число звертань до них може бути експоненціальним або факторіальним.

Типові NP задачі

Задача комівояжера

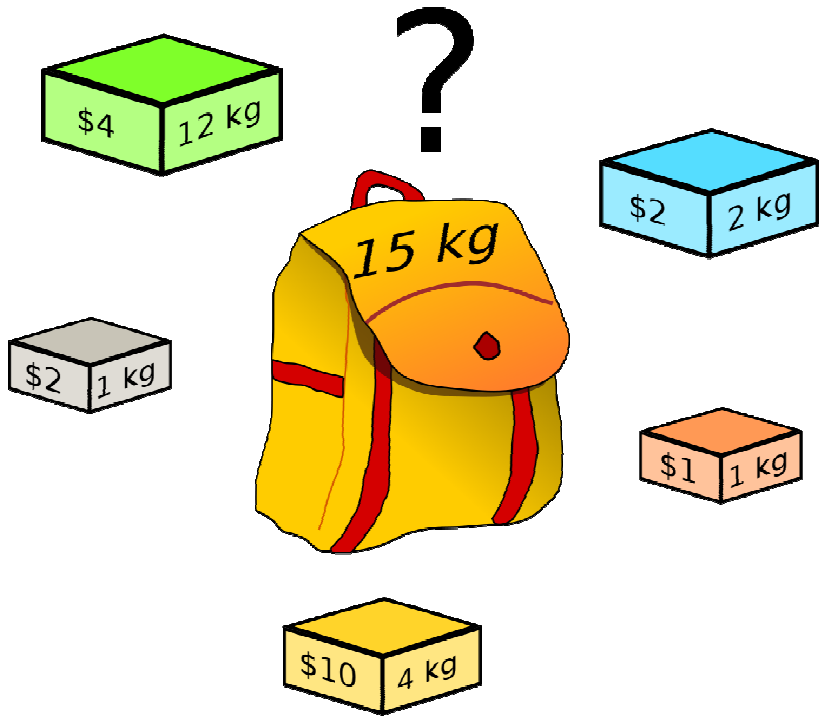
Необхідно відшукати найвигідніший **маршрут**, відвідуючи зазначені міста хоча б по одному разу з наступним поверненням у початкове місто.



Указують критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій і т. п.)
Задають відповідні матриці відстаней, вартості і т. п.
Можлива вимога, що маршрут повинен проходити через кожне місто тільки один раз.

Задача впакування рюкзака

Нехай необхідно наповнити рюкзак предметами.



Кожний предмет характеризується певною вартістю.

Кожний предмет має свою вагу.

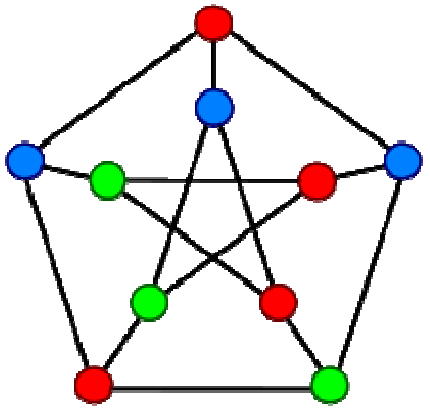
При наповненні рюкзака необхідно вибрати предмети таким чином, щоб сумарна їх вартість (або корисність) були максимальні,

але сумарна вага обраних предметів не повинна перевищити максимально припустиму.

Розфарбування графа

Граф $G(V,E)$ є набором вершин V,

і набором ребер E, що з'єднують вершини попарно.



Вершини графа можна розфарбувати в різні кольори, які зазвичай позначаються цілими числами. Нас цікавлять такі розфарбування, у яких кінці кожного ребра пофарбовані різними кольорами.

Очевидно, що в графі з N вершинами можна пофарбувати вершини в N різних кольорів, але чи можна обійтися меншою кількістю кольорів?

У задачі оптимізації нас цікавить мінімальне число кольорів, необхідних для розфарбування вершин графа.