**І. В. Брисіна, В. О. Макарічев**

# АЛГОРИТМИ І АНАЛІЗ СКЛАДНОСТІ

2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

**І. В. Брисіна, В. О. Макарічев**

# АЛГОРИТМИ І АНАЛІЗ СКЛАДНОСТІ

Навчальний посібник

Харків «ХАІ» 2020

УДК 510.51:004(075.8)
Б15

Рецензенти: д-р техн. наук, проф. П. Ф. Горбачов,
д-р фіз.-мат. наук, проф. О. В. Макарічев

**Брисіна, І. В.**

Б15 Алгоритми і аналіз складності [Електронний ресурс] : навч. посіб. / І. В. Брисіна, В. О. Макарічев. – Харків: Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2020. – 41 с.

Наведено теоретичні відомості та методи розв’язання задач за такими напрямами: метод математичної індукції; коректність алгоритмів; аналіз складності алгоритмів; метод декомпозиції; динамічне програмування і жадні алгоритми.

Для студентів комп’ютерних спеціальностей і студентів, що вивчають системні науки.

Бібліогр.: 11 назв

**УДК 510.51:004(075.8)**

# 1. МЕТОД МАТЕМАТИЧНОЇ ІНДУКЦІЇ

## Теоретичний матеріал

Метод математичної індукції – це спосіб доведення тверджень, що залежать від деякого параметра. В основі цього методу лежить *принцип математичної індукції* : твердження P(n) є справедливим для будь-якого натурального n, якщо:

1) це твердження є справедливим при n = 1 або при деякому іншому початковому значенні параметра (база індукції);
2) зі справедливості твердження для будь-якого $n = k$ випливає його справедливість для n = k + 1 (індуктивний перехід).

Якщо друга частина доведення спирається не тільки на справедливість твердження для n = k, але й для n = k – 1, n = k – 2,…, n = k – i, то твердження у першій частині потрібно перевірити для n = 1, n = 2,…, n = i.

Якщо потрібно довести твердження для довільного цілого n, що перевищує деяке ціле число m, то у першій частині доведення твердження перевіряють для n = m + 1.

У подальшому метод математичної індукції може бути використано для доведення коректності алгоритмів та оцінювання їх складності.

## Приклади розв'язання задач

**Приклад 1.1.** Довести, що $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

*Розв'язання.* Доведемо справедливість цієї формули при n = 1. Дійсно, підстановкою $n = 1$ отримуємо $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$. Припустимо, що коректність формули при $n = k$, тобто $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$. Тоді $\sum_{i=1}^{k+1} i = \sum_{i=1}^{k} i + k + 1 = \frac{k(k+1)}{2} + k + 1 = \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+1)(k+2)}{2}$, що доводить твердження при n = k + 1. Таким чином, твердження $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ є справедливим для будь-якого натурального n.

**Приклад 1.2.** Довести, що $2^n \le n!$ при $n \ge 4$.

*Розв'язання.* Доведемо цю нерівність при $n = 4$. Дійсно, $2^4 = 16 \le 24 = 4!$. Припустимо, що $2^k \le k!$, де $k \ge 4$. Тоді можна записати $2^{k+1} = 2 \cdot 2^k \le 2k! \le (k+1)k! = (k+1)!$. Отже, для всіх $n \ge 4$ виконується нерівність $2^n \le n!$.

**Приклад 1.3.** Довести, що число $a_n = n^3 + 2n$ є кратним числу $3$.

*Розв'язання.* Встановимо справедливість цього вислову при n = 1. Якщо n = 1, то $a_1 = 3$ кратне 3. Припустимо, що $a_k$ кратне 3. Доведемо, що $a_{k+1}$ також кратне числу 3. Дійсно,

$$a_{k+1} = (k+1)^3 + 2(k+1) = k^3 + 3k^2 + 3k + 1 + 2k + 2 = a_k + 3\ k^2 + k + 1\ .$$

Таким чином, $a_{k+1}$ являє собою суму чисел, кожне з яких є кратним 3. Отже, $a_{k+1}$ також кратне 3. Твердження доведено.

**Приклад 1.4.** Довести, що для чисел Фібоначчі справедлива рівність $\sum_{i=1}^{n} F_i = F_{n+2} - 1,\ n \ge 1.$

*Розв'язання.* Нагадаємо, що числа Фібоначчі $\{F_n\}$ визначають так:

$$F_0 = 0,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2} \text{ при } n \ge 2.$$

Якщо $n = 1$, то $\sum_{i=1}^{n} F_i = F_1 = 1 = 2 - 1 = F_3 - 1$. Отже, формула коректна при $n = 1$.

Припустимо, що $\sum_{i=1}^{k} F_i = F_{k+2} - 1$. Доведемо, що $\sum_{i=1}^{k+1} F_i = F_{k+3} - 1$.

Дійсно, з цього припущення та означення чисел Фібоначчі випливає, що

$$\sum_{i=1}^{k+1} F_i = \sum_{i=1}^{k} F_i + F_{k+1} = F_{k+2} - 1 + F_{k+1} = \ F_{k+1} + F_{k+2}\ - 1 = F_{k+3} - 1.$$

Отже, твердження доведено.

**Приклад 1.5.** Знайти розв'язок рекурентного співвідношення $T(1) = 1$ та $T(n) = 3T\ n-1\ + 2$ при $n \ge 2$.

*Розв'язання.* Відповідно до умови

$$T(n) = 3T(n-1) + 2 = 3\ \ 3T(n-2) + 2\ \ + 2 = 3^2 T(n-2) + 2(1+3) =$$

$$= 3^2\ \ 3T(n-3) + 2\ \ + 2(1+3) = 3^3 T(n-3) + 2\ \ 1+3+3^2\ \ = \ldots$$

Ці міркування приводять до природної гіпотези

$$T(n) = 3^k T(n-k) + 3^k - 1 \text{ при } 1 \le k \le n-1.$$

Перевіримо цю рівність за допомогою методу математичної індукції. Зауважимо, що у даному випадку параметром індукції є k. При k = 1 справедливість формули випливає з умови задачі. Припустимо, що $T(n) = 3^m T(n-m) + 3^m - 1$. Доведемо, що

$$T(n) = 3^{m+1} T(n-m-1) + 3^{m+1} - 1.$$

Дійсно, з індуктивного припущення та формули $T(j) = 3T(j-1) + 2$ випливає, що

$$T(n) = 3^m T(n-m) + 3^m - 1 = 3^m\ \ 3T(n-m-1) + 2\ \ + 3^m - 1 =$$

$$= 3^{m+1} T(n-m-1) + 2 \cdot 3^m + 3^m - 1 = 3^{m+1} T(n-m-1) + 3^{m+1} - 1.$$

Отже, $T(n) = 3^k T(n-k) + 3^k - 1$ при $1 \le k \le n-1$. Підставимо $k = n-1$. Отримуємо

$$T(n) = 3^{n-1} T(1) + 3^{n-1} - 1 = 3^{n-1} + 3^{n-1} - 1 = 2 \cdot 3^{n-1} - 1.$$

Таким чином, $T(n) = 2 \cdot 3^{n-1} - 1$ для будь-якого $n \in \mathbb{N}$.

**Приклад 1.6.** Знайти розв'язок рекурентного співвідношення

$$T(n) = 2T\left(\frac{n}{2}\right) + 6n - 1 \text{ при } n = 2^m \ge 2 \text{ та } T(1) = 1.$$

*Розв'язання.* З умови випливає, що

$$T(n) = 2T\left(\frac{n}{2}\right) + 6n - 1 = 2\left(2T\left(\frac{n}{4}\right) + \frac{6n}{2} - 1\right) + 6n - 1 =$$

$$= 2^2 T\left(\frac{n}{4}\right) + 12n - (1+2) =$$

$$= 2^2\left(2T\left(\frac{n}{8}\right) + \frac{6n}{4} - 1\right) + 12n - (1+2) =$$

$$= 2^3 T\left(\frac{n}{8}\right) + 18n - \ 1+2+2^2\ \ = 2^3\left(2T\left(\frac{n}{16}\right) + \frac{6n}{8} - 1\right) +$$

$$+18n - \ 1+2+2^2 \ = 2^4 T\left(\frac{n}{16}\right) + 24n - \ 1+2+2^2+2^3 \ ,$$

звідки видно, що $T(n) = 2^k T\left(\frac{n}{2^k}\right) + 6kn - 2^k + 1$ при $k = 1,\ldots,\log_2 n$.

Доведемо цю рівність, використовуючи метод математичної індукції по k. При k = 1 справедливість цієї формули випливає з умови задачі.

Припустимо, що $T(n) = 2^j T\left(\frac{n}{2^j}\right) + 6jn - 2^j + 1$. Тоді

$$T(n) = 2^j T\left(\frac{n}{2^j}\right) + 6jn - 2^j + 1 = 2^j\left(2T\left(\frac{n}{2^{j+1}}\right) + \frac{6n}{2^j} - 1\right) + 6jn - 2^j + 1 =$$

$$= 2^{j+1} T\left(\frac{n}{2^{j+1}}\right) + 6n - 2^j + 6jn - 2^j + 1 = 2^{j+1} T\left(\frac{n}{2^{j+1}}\right) + 6(j+1)n - 2^{j+1} + 1.$$

Отже, $T(n) = 2^k T\left(\frac{n}{2^k}\right) + 6kn - 2^k + 1$ для всіх $k = 1,\ldots,\log_2 n$. Якщо підставити $k = \log_2 n$, то отримаємо

$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + 6n\cdot\log_2 n - 2^{\log_2 n} + 1 =$$

$$= nT(1) + 6n\cdot\log_2 n - n + 1 = 6n\cdot\log_2 n + 1.$$

Таким чином, $T(n) = 6n\cdot\log_2 n + 1$ при $n = 2^m \geq 2$.

### Задачі для самостійного розв'язання

1.1. Встановити справедливість таких рівностей:

1) $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$;

2) $\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$;

3) $\sum_{i=0}^{n} (2i+1)^2 = \frac{(n+1)(2n+1)(2n+3)}{3}$;

4) $\sum_{i=1}^{n} i(i+1) = \frac{n(n+1)(n+2)}{3}$;

5) $\sum_{i=1}^{n} i(i+1)(i+2) = \frac{n(n+1)(n+2)(n+3)}{4}$;

6) $\sum_{i=1}^{n} i \cdot i! = (n+1)! - 1$;

7) $\sum_{i=1}^{n} i2^i = (n-1)2^{n+1} + 2$;

8) $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$;

9) $\sum_{i=1}^{n} a^i i = \frac{na^{n+2} - (n+1)a^{n+1} + a}{(a-1)^2}$ при $a \neq 1$;

10) $\sum_{i=1}^{n} i^2 2^i = n^2 2^{n+1} - n2^{n+2} + 3 \cdot 2^{n+1} - 6$.

1.2. Довести такі нерівності:

1) $(1+x)^n \geq 1 + nx$ при $n \geq 1$ и $x > -1$;

2) $3^n < n!$ при $n \geq 7$;

3) $n^2 < 2^n$ при $n \geq 5$;

4) $\sum_{i=1}^{n} i^k \leq \frac{n^k(n+1)}{2}$ при $k \geq 1$;

5) $\sum_{i=1}^{n} \frac{1}{i^2} < 2 - \frac{1}{n}$.

1.3. Довести такі твердження:

1) $n^5 - n$ кратне числу 5;

2) $5^{n+1} + 2 \cdot 3^n + 1$ кратне числу 8;

3) $8^{n+2} + 9^{2n+1}$ кратне числу 73;

4) $11^{n+2} + 12^{2n+1}$ кратне числу 133.

1.4. Довести, що для чисел Фібоначчі виконуються рівності:

1) $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$;

2) $\sum_{i=1}^{n} F_i^2 = F_n F_{n+1}$;

3) $F_{n+1}F_{n+2} = F_n F_{n+3} + (-1)^n$;

4) $F_{n-1}F_{n+1} = F_n^2 + (-1)^n$.

1.5. Довести з використанням методу математичної індукції такі рівності:

1) $\sum_{i=0}^{n} C_n^i = 2^n$;

2) $C_n^i \le n^i$;

3) $(a+b)^n = \sum_{i=0}^{n} C_n^i a^i b^{n-i}$;

4) $\sum_{i=0}^{n} i C_n^i = n2^{n-1}$;

5) $\sum_{i=0}^{n} 2^{-i} C_n^i = \left(\frac{3}{2}\right)^n$.

1.6. Розв'язати рекурентні співвідношення:

1) $T(n) = 3T(n-1) - 15$ при $n \ge 2$ та $T(1) = 8$;

2) $T(n) = T(n-1) + n - 1$ при $n \ge 2$ та $T(1) = 2$;

3) $T(n) = T(n-1) + 2n - 3$ при $n \ge 2$ та $T(1) = 3$;

4) $T(n) = 2T(n-1) + n - 1$ при $n \ge 2$ та $T(1) = 1$;

5) $T(n) = 2T(n-1) + 3n + 1$ при $n \ge 2$ та $T(1) = 5$;

6) $T(n) = 2T\left(\frac{n}{2}\right) + 3n + 2$ при $n = 2^m \ge 2$ та $T(1) = 4$;

7) $T(n) = 6T\left(\frac{n}{6}\right) + 2n + 3$ при $n = 6^m \ge 6$ та $T(1) = 1$;

8) $T(n) = 6T\left(\frac{n}{6}\right) + 3n - 1$ при $n = 6^m \ge 6$ та $T(1) = 3$;

9) $T(n) = 4T\left(\frac{n}{3}\right) + 2n - 1$ при $n = 3^m \ge 3$ та $T(1) = 3$;

10) $T(n) = 4T\left(\frac{n}{3}\right) + 3n - 5$ при $n = 3^m \geq 3$ та $T(1) = 2$;

11) $T(n) = 3T\left(\frac{n}{2}\right) + n^2 - n$ при $n = 2^m \geq 2$ та $T(1) = 1$;

12) $T(n) = 3T\left(\frac{n}{2}\right) + n^2 - 2n + 1$ при $n = 2^m \geq 2$ та $T(1) = 4$;

13) $T(n) = 3T\left(\frac{n}{2}\right) + n - 2$ при $n = 2^m \geq 2$ та $T(1) = 1$;

14) $T(n) = 3T\left(\frac{n}{2}\right) + 5n - 7$ при $n = 2^m \geq 2$ та $T(1) = 1$;

15) $T(n) = 4T\left(\frac{n}{3}\right) + n^2$ при $n = 3^m \geq 3$ та $T(1) = 1$;

16) $T(n) = 4T\left(\frac{n}{3}\right) + n^2 - 7n + 5$ при $n = 3^m \geq 3$ та $T(1) = 1$;

17) $T(n) = T\left(\frac{n}{4}\right) + \sqrt{n} + 1$ при $n = 4^m \geq 4$ та $T(1) = 1$;

18) $T(n) = T(n-2) + 3n + 4$ при $n \geq 3$ та $T(1) = 1,\ T(2) = 6$;

19) $T(n) = T(n-2) + n$ при $n > 1$ та $T(0) = c,\ T(1) = d$;

20) $T(n) = T(n-3) + 5n$ при $n \geq 4$ та $T(1) = T(2) = 1,\ T(3) = 13$;

21) $T(n) = 2T(n-1) + n^2 - 2n + 1$ при $n \geq 2$ та $T(1) = 1$;

22) $T(n) = nT(n-1) + n$ при $n \geq 2$ та $T(1) = 1$;

23) $T(n) = \sum_{i=1}^{n-1} T(i) + 1$ при $n \geq 2$ та $T(1) = 1$;

24) $T(n) = \sum_{i=1}^{n-1} T(i) + 7$ при $n \geq 2$ та $T(1) = 1$;

25) $T(n) = \sum_{i=1}^{n-1} T(i) + n^2$ при $n \geq 2$ та $T(1) = 1$;

26) $T(n) = 2\sum_{i=1}^{n-1} T(i) + 1$ при $n \geq 2$ та $T(1) = 1$;

27) $T(n) = \sum_{i=1}^{n-1} T(n-i) + 1$ при $n \geq 2$ та $T(1) = 1$;

28) $T(n) = \sum_{i=1}^{n-1} (T(i) + T(n-i)) + 1$ при $n \geq 2$ та $T(1) = 1$.

## 2. КОРЕКТНІСТЬ АЛГОРИТМІВ

### Теоретичний матеріал

*Алгоритмом* називають скінченний набір чітких, недвозначних інструкцій, дотримуючись яких можна за набором вхідних даних отримати на виході деякий результат. Необхідно зазначити, що на теперішній час алгоритми – це технологічний продукт, від якого суттєво залежать швидкість та ефективність обчислювальних схем. Алгоритм називається коректним, якщо для кожного набору вхідних даних результат його виконання є коректним.

Коректність ітераційних алгоритмів можна довести з використанням інваріантів циклу. *Інваріантом циклу* називається деяке твердження або вислів відносно змінних, що використовуються у циклі, а також залишається істинним перед початком виконання циклу та є таким після кожного проходу тіла циклу. Щоб довести коректність роботи ітераційного алгоритму, спочатку виділяють інваріант циклу та доводять його справедливість, після цього за допомогою інваріанта циклу доводять, що алгоритм закінчиться й на виході буде отримано коректний результат.

Коректність рекурсивного алгоритму зазвичай доводять на основі методу математичної індукції. При цьому базою індукції є база рекурсії, а індуктивний перехід здійснюється з використанням припущення про коректність рекурсивних викликів. Також потребує перевірки відсутність нескінченної рекурсії.

### Приклади розв'язання задач

**Приклад 2.1.** Довести коректність алгоритму обчислення факторіала числа:

```
function factorial(n)
 begin
   result:=1
   i:=2
   while i<=n do
```

```
    result:=result*i
    i:=i+1
  return result
end
```

*Розв'язання.* Нехай $result_j$, $i_j$ – значення змінних result, і після j-го проходу тіла циклу. Інваріант циклу: $P(j) = result_j = (j+1)!, i_j = j+2$ , де j – кількість проходів тіла циклу. Доведемо, що P(j) = true для будь-якого j. Дійсно, перед виконанням циклу змінні result, і відповідно дорівнюють 1 і 2. Отже, вислів $P(0) = result_0 = 1!, i_0 = 2$ є справедливим. Припустимо, що $P(k) = true$, тобто $result_k = (k+1)!$ та $i_k = k+2$. Тоді відповідно до алгоритму

$$result_{k+1} = result_k i_k = (k+1)!(k+2) = (k+2)!,$$
$$i_{k+1} = i_k + 1 = k+2+1 = k+3,$$

звідки випливає, що P(k+1) = true. Вихід із циклу відбувається при $i_j = n+1$. Оскільки $i_j = j+2$, то $j = n-1$. Іншими словами, виконання циклу завершиться після (n-1)-го проходу, після чого отримуємо значення $result_j = (j+1)! = (n-1+1)! = n!$, яке є результатом виконання функції factorial(n). Отже, наведений алгоритм обчислення факторіала числа не приводить до нескінченного циклу і є коректним.

**Приклад 2.2.** Довести, що наведений нижче алгоритм обчислення суми елементів масиву є коректним:

```
functiom sum(A[1..n])
 begin
   result:=0
   i:=1
   while i<=n do
    result:=result+A[i]
    i:=i+1
   return result
 end
```

*Розв'язання. Я*к і у попередньому прикладі, позначимо через $result_j$, $i_j$ значення змінних result, і після j-го проходу тіла циклу. Інваріантом циклу є твердження $P(j) = \left\{ result_j = \sum_{m=1}^{j} A[m],\ i_j = j+1 \right\}$, де j – кількість проходів тіла циклу. Покажемо істинність цього твердження для будь-якого j. Перед

виконанням циклу змінні result, і відповідно дорівнюють 0 та 1. Отже, $P(0) = \text{true}$. Припустимо, що $P(k) = \text{true}$. Іншими словами,

$$\text{result}_k = \sum_{m=1}^{k} A[m] \text{ та } i_k = k + 1.$$

Тоді

$$\text{result}_{k+1} = \text{result}_k + A[k+1] = \sum_{m=1}^{k} A[m] + A[k+1] = \sum_{m=1}^{k+1} A[m],$$

$$i_{k+1} = i_k + 1 = k + 1 + 1 = k + 2.$$

Отже, P(k+1) = true. Вихід з циклу відбувається при $i_j = n + 1$. Оскільки $i_k = k + 1$, то $j = n$. Тому цикл завершиться після n-го проходу. Після цього отримуємо

$$\text{result}_j = \sum_{m=1}^{j} A[m] = \sum_{m=1}^{n} A[m],$$

що, у свою чергу, являє собою суму елементів масиву. Таким чином, наведений алгоритм обчислення суми елементів масиву є коректним.

**Приклад 2.3.** Довести коректність алгоритму пошуку максимального елемента масиву:

```
function max(A[1..n])
 begin
   result=A[1]
   i:=2
   while i<=n do
     if result<A[i]
     then
      result:=A[i]
     i:=i+1
   return result
 end
```

*Розв'язання.* Нехай $\text{result}_j$, $i_j$ – це значення змінних result, і після j-го проходу тіла циклу. Інваріант циклу у цьому випадку має такий вигляд:

$$P(j) = \left\{ \text{result}_j = \max_{m=1,\ldots,j+1} A[m],\ i_j = j + 2 \right\},$$

де j – кількість проходів тіла циклу. Доведемо, що P(j) = true для кожного j.

Дійсно, перед виконанням циклу змінні result, і відповідно дорівнюють A[1] та 2. Отже, вислів

$$P(0) = \ \text{result}_0 = \max_{m=1} A[m],\ i_0 = 0 + 2$$

є справедливим. Припустимо, що P(k) = true, тобто

$$\text{result}_k = \max_{m=1,\ldots,k+1} A[m] \text{ та } i_k = k + 2.$$

Тоді відповідно до алгоритму

$$\text{result}_{k+1} = \max\ \text{result}_k, A[i]\ =$$

$$= \max\ \max_{m=1,\ldots,k+1} A[m], A[k+2]\ = \max_{m=1,\ldots,k+2} A[m],$$

$$i_{k+1} = k + 3.$$

Отже, $P(k+1) = \text{true}$. Вихід із циклу відбувається при $i_j = n + 1$. Тому $j = n - 1$, тобто тіло циклу виконується n–1 разів. Функція max(A[1...n]) повертає значення $\text{result}_{n-1}$, яке дорівнює $\max_{m=1,\ldots,n} A[m]$.

Отже, коректність алгоритму встановлено.

**Приклад 2.4.** Довести коректність алгоритму обчислення чисел Фібоначчі:

```
function fibonacci(n)
 begin
   if n<=1
  then
   result:=n
   else
    result:=fibonacci(n-1)+fibonacci(n-2)
   return result
end
```

*Розв'язання.* База індукції. Якщо n = 0 або n = 1, то функція fibonacci(n) повертає відповідно значення 0 або 1, які дорівнюють числам Фібоначчі $F_0$ та $F_1$. Припустимо, що алгоритм є коректним при n = k та n = k+1, тобто fibonacci(k) $= F_k$ й fibonacci(k+1) $= F_{k+1}$. Функція fibonacci(k+2) повертає значення

fibonacci(k)+ fibonacci(k+1) = $F_k + F_{k+1}$,

яке за означенням чисел Фібоначчі дорівнює $F_{k+2}$. Отже, наведений рекурсивний алгоритм є коректним для кожного n.

**Приклад 2.5.** Довести коректність алгоритму пошуку мінімального елемента масиву:

```
function min_array(n)
 begin
    if n=1
  then
   result:=A[1]
    else
     result:=min(min_array(n-1),A[n])
    return result
 end
```

*Розв'язання.* База індукції. При n = 1 функція min_array(n) повертає значення $A[1]$ або, що те ж саме, $\min\limits_{m=1} A[m]$. Припустимо, що алгоритм є коректним при $n = k$, тобто функція min_array(k) повертає значення $\min\limits_{m=1,\ldots,k} A[m]$. Тоді за побудовою функція min_array(k+1) повертає

$$\min\ \min\_array(k), A[k+1] =$$

$$= \min \min\limits_{m=1,\ldots,k} A[m], A[k+1] = \min\limits_{m=1,\ldots,k+1} A[m].$$

Таким чином, алгоритм є коректним для кожного n.

## Задачі для самостійного розв'язання

2.1. Довести коректність алгоритму обчислення чисел Фібоначчі:

```
function fibonacci(n)
 begin
   if n<=1
   then
    current:=n
   else
    last:=0
    current:=1
    i:=2
     while i<=n do
       temp:=last+current
       last:=current
       current:=temp
       i:=i+1
  return current
```

```
 end
```

2.2. Довести коректність наведеного алгоритму пошуку мінімального елемента масиву:

```
function min(A[1..n])
 begin
  result:=A[1]
  i:=2
  while i<=n do
   if A[i]<result
   then
    result:=A[i]
   i:=i+1
  return result
 end
```

2.3. Довести коректність алгоритму сортування бульбашкою:

```
procedure bubblesort(A[1..n])
 begin
  i:=1
  while i<=n-1 do
   j:=1
   while j<=n-I do
    if A[j]>A[j+1]
    then
     swap(A[j],A[j+1])
    j:=j+1
   i:=i+1
 end
```

Тут swap(A[j],A[j+1]) – процедура, яка міняє місцями елементи A[j] та A[j+1].

2.4. Довести коректність алгоритму обчислення добутку матриць:

```
function product(A,B)
  begin
   for i:=1 to n do
    for j:=1 to n do
     C[i,j]:=0
     for k:=1 to n do
      C[i,j]:=C[i,j]+A[i,k]*B[k,j]
  return C
```

```
 end
```

2.5. Довести коректність алгоритму обчислення факторіала:

```
function factorial(n)
 begin
  if n<=1
  then
   result:=n
  else
   result:=n*factorial(n-1)
  return result
 end
```

2.6. Довести, що наведений нижче алгоритм обчислення суми елементів масиву є коректним:

```
function sum(n)
 begin
  if n=1
  then
   result:=A[1]
  else
   result:=A[n]+sum(n-1)
  return result
 end
```

2.7. Довести коректність алгоритму пошуку максимального елемента масиву:

```
function max_array(n)
 begin
  if n=1
  then
   result:=A[n]
  else
   result:=max(max_array(n-1),A[n])
 end
```

# 3. АНАЛІЗ АЛГОРИТМІВ

## Теоретичний матеріал

Аналіз складності алгоритму – це дослідження обсягу обчислювальних ресурсів (часу, пам'яті, апаратного забезпечення тощо), необхідних для його виконання. Надалі в основному буде досліджуватися саме часова складність алгоритмів. Під асимптотичною складністю алгоритму розуміють порядок зростання часу роботи алгоритму при збільшенні кількості вхідних даних. Для описання асимптотичної складності алгоритму використовують асимптотичну символіку.

Розглянемо додатні функції f(n) та g(n) натурального аргументу.

За означенням $f(n) = O(g(n))$, якщо існують такі додатні константи $c$ і $n_0$, що для будь-якого $n \geq n_0$ виконується нерівність $f(n) \leq cg(n)$. $O$-символ використовують для опису верхньої межі часу роботи алгоритму в найгіршому випадку.

Якщо існують додатні константи $c$ і $n_0$, такі, що для кожного $n \geq n_0$ справедлива нерівність $f(n) \geq cg(n)$, то $f(n) = \Omega(g(n))$. Таке позначення використовують для опису нижньої межі часу роботи алгоритму в найкращому випадку.

Якщо існують додатні константи $c_1, c_2$ і $n_0$, такі, що для будь-якого $n \geq n_0$ виконується нерівність $c_1 g(n) \leq f(n) \leq c_2 g(n)$, то $f(n) = \Theta(g(n))$. У цьому випадку функція g(n) також є асимптотично точною оцінкою функції f(n).

Час виконання алгоритму будемо позначати через T(n), де n – кількість вхідних даних. Функція T(n) являє собою суму проміжків часу, необхідних для виконання кожної виконуваної інструкції, що входить до складу алгоритму.

## Приклади розв'язання задач

**Приклад 3.1.** Довести, що $(n+1)^2 = O(n^2)$.

*Розв'язання.* Довести це твердження можна, наприклад, так: для будь-якого натурального $n \geq 2$ справедливо

$$(n+1)^2 = n^2 + 2n + 1 \leq n^2 + n^2 + n^2 = 3n^2.$$

Таким чином, існують додатні числа $c$, $n_0$, такі, що $(n+1)^2 \leq cn^2$ для будь-якого $n \geq n_0$. Отже, $(n+1)^2 = O(n^2)$.

**Приклад 3.2.** Довести, що $2^n = \Omega(n)$.

*Розв'язання.* З використанням методу математичної індукції можна довести, що для будь-якого $n \in \mathbb{N}$ справедлива нерівність $2^n \geq n$, звідки випливає справедливість рівності $2^n = \Omega(n)$.

**Приклад 3.3.** Довести, що $n^2 + 2^n = \Theta(2^n)$.

*Розв'язання.* За допомогою методу математичної індукції можна довести, що $2^n \geq n^2$ для будь-якого натурального $n \geq 4$. Тоді

$$2^n \leq 2^n + n^2 \leq 2^n + 2^n = 2 \cdot 2^n$$

при $n \geq 4$.

Отже, $n^2 + 2^n = \Theta(2^n)$.

Зауважимо, що в цьому випадку константи $c_1, c_2$ і $n_0$ з означення, що було наведено вище, відповідно дорівнюють 1, 2 та 4.

**Приклад 3.4.** Довести, що якщо $f_1(n) = O(g_1(n))$, $f_2(n) = O(g_2(n))$, то $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

*Розв'язання.* За умовою $f_1(n) = O(g_1(n))$. Отже, існують додатні числа $c_1$ та $n_1$, такі, що для будь-якого $n \geq n_1$ виконується нерівність $f_1(n) \leq c_1 g_1(n)$. Аналогічно можна знайти такі додатні числа $c_2$ та $n_2$, що для будь-якого $n \geq n_2$ виконується нерівність $f_2(n) \leq c_2 g_2(n)$. Нехай $c_0 = \max\{c_1, c_2\}$ та $n_0 = \max\{n_1, n_2\}$. Тоді для будь-якого $n \geq n_0$ можна записати

$$f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \leq c_0 \ g_1(n) + g_2(n) \ .$$

Таким чином, існують додатні числа $c_0$ і $n_0$, такі, що для будь-якого $n \geq n_0$ виконується нерівність

$$f_1(n) + f_2(n) \leq c_0 \ g_1(n) + g_2(n) \ .$$

Отже,

$$f_1(n) + f_2(n) = O(g_1(n) + g_2(n)).$$

Зазначимо, що у такому доведенні суттєвим є той факт, що функції $f_1(n), f_2(n), g_1(n)$ і $g_2(n)$ – додатні.

**Приклад 3.5.** Обчислити час роботи алгоритму з прикладу 2.1.

*Розв'язання.* Розглянемо алгоритм обчислення факторіала числа:

| | Час | Кількість разів |
|---|---|---|
| **function** factorial(n) | | |
| **begin** | | |
| 1**.** result:=1 | $c_1$ | 1 |
| 2. i:=2 | $c_2$ | 1 |
| 3. **while** i<=n **do** | $c_3$ | **n** |
| 4. result:=result*i | $c_4$ | **n-1** |
| 5. i:=i+1 | $c_5$ | **n-1** |
| 6. **return** result | $c_6$ | 1 |
| **end** | | |

Прокоментуємо виконання цього алгоритму. Витрати часу на операції у строках 1 і 2 з урахуванням повторень становлять відповідно $c_1$ і $c_2$. Перевірка умов у рядку 3 потребує часу $c_3$ та виконується n разів. Операції у строках 4 і 5 повторюються n – 1 разів і потребують відповідно $c_4$ і $c_5$ часу. Операція у рядку 6 виконується один раз і потребує $c_6$. Тоді час роботи даного алгоритму становить

$$T(n) = c_1 + c_2 + nc_3 + (n-1)c_4 + (n-1)c_5 + c_6 = an + b,$$

з чого випливає, що $T(n) = \Theta(n)$. Іншими словами, час роботи наведеного алгоритму обчислення факторіала є лінійним.

**Приклад 3.6.** Обчислити час роботи алгоритму з прикладу 2.5.

*Розв'язання.* Розглянемо алгоритм пошуку мінімального елемента масиву (нехай T(n) – час роботи цього алгоритму):

| | Час | Кількість повторень |
|---|---|---|
| **function** min_array(n) | | |
| **begin** | | |
| 1. **if** n=1 | $c_1$ | 1 |
| 2. **then** | | |
| 3. result:=A[1] | $c_2$ | 1 |
| 4. **else** | | |
| 5. result:=min(min_array(n-1),A[n]) | $T(n-1)+c_3$ | 1 |
| 6. **return** result | $c_4$ | 1 |
| **end** | | |

Операції у строках 1, 3 і 6 виконуються один раз і потребують $c_1, c_2$ і $c_4$ часу відповідно. Операція у рядку 5 передбачає пошук мінімального елемента в масиві розміром n – 1 та його порівняння з елементом A[n]. Тому час виконання становить $T(n-1)+c_3$. Отже,

$$T(n)=\begin{cases} c_1+c_2+c_4, & n=1, \\ c_1+T(n-1)+c_3+c_4. \end{cases}$$

За допомогою методу математичної індукції можна довести, що $T(n)=an+b$. Отже, $T(n)=\Theta(n)$.

## Задачі для самостійного розв'язання

3.1. Довести справедливість таких рівностей:

1) $3n^2-8n+9=O(n^2)$;
2) $\log_2 n=O(n)$;
3) $3n\log_2 n=O(n^2)$;
4) $n^2-3n=\Omega(n)$;
5) $n^3-2n^2+1000=\Theta(n^3)$;
6) $2n+100=O(2^n)$;
7) $cn+d=O(2^n)$ при $c,d>0$;
8) $cn^k+d=O(2^n)$ при $c,d,k>0$;
9) $2^n=O(n!)$;
10) $n^n=\Omega(n!)$.

3.2. Довести або спростувати співвідношення:

1) $2n^2+1=O(n^2)$;
2) $n\log_4 n=O(n\sqrt{n})$;
3) $\sqrt{n}=O(\log_2 n)$;
4) $\log_2 n=O(\sqrt[3]{n})$;
5) $n^3=O(n^2(1+n^2))$;
6) $n^2(3+\sqrt[10]{n})=O(n^2)$;
7) $n^2(1+\sqrt{n})=O(n^2\log_5 n)$;

8) $n^2 + n = O(n + n\sqrt{n} + \sqrt{n})$;

9) $\log_2 n + \sqrt[1000]{n} = O(\sqrt[1000]{n})$;

10) $\sqrt{n}\log_2 n = O(n)$.

3.3. Визначити, яка з рівностей $f(n) = O(g(n))$, $g(n) = O(n)$, $f(n) = \Omega(g(n))$, $g(n) = \Omega(f(n))$, $f(n) = \Theta(g(n))$ буде виконуватися для наведених функцій:

1) $f(n) = n^2 + 3n + 4,\ g(n) = 6n + 7$;

2) $f(n) = \sqrt{n},\ g(n) = \log_2(n+1)$;

3) $f(n) = n\sqrt{n},\ g(n) = n^2 - n$;

4) $f(n) = n + n\sqrt{n},\ g(n) = 4n\log_2(n^2 + 1)$;

5) $f(n) = n^2 - 10,\ g(n) = n\log_{10000}(n)$;

6) $f(n) = 2^n - n^3 + n^2,\ g(n) = n^{20} + n^{23}$.

3.4. Довести такі твердження:

1) якщо $f_1(n) = \Omega(g_1(n))$ та $f_2(n) = \Omega(g_2(n))$, то
$$f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n));$$

2) якщо $f_1(n) = O(g_1(n))$ і $f_2(n) = O(g_2(n))$, то
$$f_1(n) + f_2(n) = O\ \max\{g_1(n), g_2(n)\}\ ;$$

3) якщо $f_1(n) = \Omega(g_1(n))$ і $f_2(n) = \Omega(g_2(n))$, то
$$f_1(n) + f_2(n) = \Omega\ \min\{g_1(n), g_2(n)\}\ ;$$

4) якщо $f_1(n) = \Omega(g_1(n))$ і $f_2(n) = \Omega(g_2(n))$, то
$$f_1(n)f_2(n) = \Omega(g_1(n)g_2(n));$$

5) якщо $f_1(n) = O(g_1(n))$ і $f_2(n) = O(g_2(n))$, то
$$f_1(n)f_2(n) = O(g_1(n)g_2(n)).$$

3.5. Знайти час роботи алгоритмів з прикладів 2.2 – 2.4 і задач 2.1 – 2.7.

3.6. Знайти складність алгоритмів пошуку суми елементів матриці:

```
function sum1(A,m,n)
begin
  S:=0;
  for  i := 1 to m  do
     for  j := 1 to  n  do
        S := S + A[i][j];
  return  S;
end
```

```
function sum2(A,m,n)
begin
  S:=0;
  for  j := 1 to n  do
    for  i := 1 to  m  do
      S := S + A[i][j];
  return  S;
end
```

Реалізувати ці алгоритми мовою C++ та порівняти час їх виконання на прикладі деякої матриці великого розміру. Прокоментувати отримані результати.

3.7. Розглянемо задачу пошуку добутку двох квадратних матриць A і B, що складаються з n рядків і стовпців. Порівняти складність і час виконання алгоритмів пошуку їх добутку:

1)
```
for  i := 1  to  n  do
  for  j := 1  to  n  do
begin
   sum := 0;
   for  k := 1  to  n  do
     sum := sum + A[i][k]*B[k][j];
   C[i][j] := C[i][j] + sum;
end
```

2)
```
for  j := 1  to  n  do
  for  i := 1  to  n  do
    begin
      sum := 0;
      for  k := 1  to  n  do
        sum := sum + A[i][k]*B[k][j];
      C[i][j] := C[i][j] + sum;
    end
```

3)
```
for  j := 1  to  n  do
  for  k := 1  to  n  do
    begin
      r := B[k][j];
      for  i := 1  to  n  do
        C[i][j] := C[i][j] + A[i][k]*r;
    end
```

```
4) for  k := 1  to  n  do
     for  j := 1  to  n  do
        begin
          r := B[k][j];
          for  i := 1  to  n  do
             C[i][j] := C[i][j] + A[i][k]*r;
        end

5) for  k := 1  to  n  do
     for  i := 1  to  n  do
        begin
          r := A[i][k];
          for  j := 1  to  n  do
              C[i][j] := C[i][j] + r*B[k][j];
        end

6) for  i := 1  to  n  do
     for  k := 1  to  n  do
        begin
          r := A[i][k];
          for  j := 1  to  n  do
              C[i][j] := C[i][j] + r*B[k][j];
        end
```

Реалізувати ці алгоритми мовою C++ та порівняти час їх виконання на прикладі деяких матриць великого розміру. Як відрізняється асимптотична складність алгоритмів? Як відрізняється час виконання відповідних реалізацій цих алгоритмів? Чому?

## 4. МЕТОД ДЕКОМПОЗИЦІЇ

### Теоретичний матеріал

Метод декомпозиції (принцип «divide & conquer») – це спосіб розв'язання задач, що передбачає розбиття задачі на декілька більш простих задач, розв'язок яких знаходять рекурсивно, після чого комбінують з наміром отримати розв'язок вихідної задачі. Цей метод містить такі етапи:

1) розбиття вихідної задачі на декілька більш простих задач;

2) рекурсивне розв'язання кожної задачі;
3) побудова розв'язку вихідної задачі за допомогою отриманих розв'язків відповідних задач.

Важливим прикладом застосування цього методу є алгоритми сортування масиву, які розглядаються в інших курсах (ознайомитися з цим матеріалом можна, наприклад, у роботі [2]).

## Приклади розв'язання задач

**Приклад 4.1.** Довести коректність і знайти час роботи алгоритму обчислення суми елементів масиву:

```
function sum(A[1...n])
 begin
  if n=1
  then
   result:=A[1]
  else
   m:=[n/2]
   result:=sum(A[1...m])+sum(A[m+1...n])
  return result
 end
```

*Розв'язання.* Доведемо коректність наведеного алгоритму. Для цього будемо використовувати метод математичної індукції. Якщо n = 1, то функція sum(A[1...n]) повертає значення A[1], яке збігається із сумою всіх елементів масиву. Нехай алгоритм є коректним для будь-якого n = 2,3,…,k, де k > 1. Тоді функція sum(A[1…n]) повертає значення, що дорівнює sum(A[1…m]) + sum(A[m+1…n]), яке за припущенням становить

$$\sum_{i=1}^{m} A[i] + \sum_{i=m+1}^{n} A[i] = \sum_{i=1}^{n} A[i].$$

Тому наведений алгоритм є коректним. Знайдемо час його роботи.

Нехай T(n) – час роботи алгоритму. Тоді

$$T(n) = \begin{cases} c, & n = 1, \\ T(m) + T(n-m) + d, & n>1. \end{cases}$$

З використанням методу математичної індукції отримуємо $T(n) = an + b$. Таким чином, $T(n) = \Theta(n)$. Отже, час є лінійним.

**Приклад 4.2.** Задача про біноміальні коефіцієнти. Довести коректність і оцінити час роботи алгоритму обчислення коефіцієнта $C_n^k$:

```
function binom_coef(n,k)
 begin
  if k=0 or n=k
  then
   result:=1
  else
   result:=binom_coef(n-1,k-1)+binom_coef(n-1,k)
  return result
 end
```

*Розв'язання.* Коректність алгоритму є наслідком формул

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k, \quad C_n^0 = C_n^n = 1.$$

Позначимо час роботи цього алгоритму через T(n,k). Тоді T(n,k) = c, якщо n = k або k = 0, і T(n,k) = T(n – 1,k – 1) + T(n – 1,k) в усіх інших випадках.

Розглянемо випадок, коли k не перевищує n / 2. Якщо k > 1, то $T(n,k) \geq 2T(n-1,k-1)+d$. Із останньої нерівності випливає, що $T(n,k) \geq 2^k a + b$ (це неважко перевірити за допомогою методу математичної індукції). Отже, при фіксованому n виконується рівність $T(n,k) = \Omega\ 2^k$ . Аналогічну експоненціальну оцінку можна отримати у випадку, коли k > n / 2.

Таким чином, наведений алгоритм потребує як мінімум експоненціальних витрат часу.

## Задачі для самостійного розв'язання

4.1. Довести коректність і знайти час роботи алгоритму пошуку мінімального елемента масиву:

```
function min_array(A[1...n])
 begin
  if n=1
  then
   result:=A[1]
  else
   m:=[n/2]
   result:=min(min_array(A[1...m]),min_array(A[m+1...n]))
  return result
```

```
 end
```

4.2. Довести коректність і знайти час роботи алгоритму пошуку максимального елемента масиву:

```
function max_array(A[1...n])
 begin
  if n=1
  then
   result:=A[1]
  else
   m:=[n/2]
   result:=max(max_array(A[1...m]),max_array(A[m+1...n]))
  return result
 end
```

4.3. Задача про «рюкзак–виконуваність». Задані n чисел $s_1, s_2, \ldots, s_n$. Потрібно з'ясувати, чи можна подати деяке число S у вигляді суми $x_1 s_1 + \ldots + x_n s_n$, де $x_i \in \{0;1\}$ для будь-якого $i = 1, \ldots, n$. Довести коректність і оцінити час роботи алгоритму розв'язання цієї задачі:

```
function knapsack(n,S)
 begin
  result:=false
  if n=0
  then
   if S=0
   then
    result:=true
   else
    result:=false
  else
   if knapsack(n-1,S)
   then
    result:=true
   else
    if s[n]<=S
    then
     result:=knapsack(n-1,S-s[n])
  return result
 end
```

4.4. Розробити алгоритм обчислення добутку матриць $A_1, A_2, \ldots, A_n$ порядку $m \times m$ за допомогою методу декомпозиції.

# 5. ДИНАМІЧНЕ ПРОГРАМУВАННЯ

## Теоретичний матеріал

Динамічне програмування являє собою спосіб розв'язання задач шляхом комбінування рішень більш простих підзадач. Однак на відміну від методу декомпозиції розв'язання кожної підзадачі записується в таблицю, що, в свою чергу, дозволяє уникнути одних і тих же повторних обчислень.

Динамічне програмування знайшло своє застосування при розв'язанні оптимізаційних задач. Процес розроблення алгоритмів зазвичай складається з таких етапів:

1) опис структури розв'язку;
2) рекурсивне визначення значення, яке відповідає розв'язку вихідної задачі;
3) обчислення значення, що відповідає розв'язку задачі;
4) складання оптимального розв'язку.

## Приклади розв'язання задач

**Приклад 5.1.** Задача про біноміальні коефіцієнти. Знайти час виконання алгоритму обчислення коефіцієнта $C_n^k$:

```
function binom_coef(n,k)
 begin
   for i:=0 to n-k do T[i,0]:=1
   for i:=0 to k do T[i,i]:=1
   for j:=1 to k do
     for i:=j+1 to n-k+j do
       T[i,j]:=T[i-1,j-1]+T[i-1,j]
   return T[n,k]
 end
```

*Розв'язання.* Як видно, цей алгоритм являє собою модифікацію алгоритму з прикладу 4.2. Позначимо через T(n,k) час його роботи. Тоді

$$T(n,k) = \sum_{i=0}^{n-k} c_1 + \sum_{i=0}^{k} c_2 + \sum_{j=1}^{k} \sum_{i=j+1}^{n-k+j} c_3 + c_4 = ank + b = \Theta(nk).$$

У найгіршому варіанті час роботи становить $O\ n^2$ .

Отже, наведений динамічний алгоритм порівняно з алгоритмом із прикладу 4.2 має такі переваги:

1) поліноміальний час роботи замість експоненціального;

2) динамічний алгоритм, який дозволяє знайти одразу цілий набір біноміальних коефіцієнтів, а не тільки одне значення.

Зауважимо, що експоненціальний час роботи алгоритму, який реалізує метод декомпозиції для розв'язання задачі про біноміальні коефіцієнти, обумовлено тим, що при її розв'язанні виникає велика кількість підзадач, які перетинаються, тобто є однаковими. Динамічне програмування дозволяє усунути цей недолік.

**Приклад 5.2.** Задача про «рюкзак–виконуваність». Задано натуральні числа $s_1,\ldots,s_n$ і $S$. Чи можна подати $S$ у вигляді лінійної комбінації чисел $s_i$ з коефіцієнтами $x_i$, кожен з яких може дорівнювати 0 або 1?

*Розв'язання.* Розробимо функцію knapsack(S,n), яка повертає true або false залежно від того, чи можна подати S у вигляді потрібної лінійної комбінації. Ця функція повертає значення true в одному з випадків:

1) якщо $x_n = 0$ і knapsack(S,n-1);
2) якщо $x_n = 1$ і knapsack(S,n-1).

Отже, рекурсивный алгоритм, що базується на принципі «divide and conquer», буде таким:

```
function knapsack(S,n)
 begin
  if n=0
  then
   if S=0
   then
    return true
   else
    return false
  else
   if knapsack(S,n-1)
   then
    return true
   else
    return knapsack(S-s[n],n-1)
 end
```

За допомогою методу математичної індукції можна показати, що час роботи цього алгоритму становить $O\ 2^n$, тобто є експоненціальним. Причина цього полягає у наявності великої кількості підзадач, що перетинаються. Усунемо цей недолік за допомогою зберігання розв'язків цих підзадач у матриці t. Елементами цієї матриці є такі значення: t[i,j]

дорівнює true або false залежно від того, чи можна подати число j у вигляді лінійної комбінації $s_1,\ldots,s_i$ з коефіцієнтами $x_1,\ldots,x_i$, кожен із яких дорівнює 0 або 1. Елемент t[i,j] дорівнює true тоді й тільки тоді, коли $t\,i-1,j=\text{true}$ або $t\,i-1,j-s_i=\text{true}$. Тому динамічний алгоритм розв'язання цієї задачі буде таким:

```
function knapsack(n,S)
 begin
   t[0,0]=true
   for j:=1 to S do
     t[0,j]:=false
   for i:=1 to n do
     for j:=0 to S do
       t[i,j]:=t[i-1,j]
       if j-s[i]>=0
       then
         t[i,j]:=t[i,j] or t[i-1,j-s[i]]
   return t[n,S]
 end
```

Час роботи алгоритму становить $\Theta(nS)$.

**Приклад 5.3.** Задача про добуток набору матриць. Вартістю добутку двох матриць $A_{m\times n}$ і $B_{n\times k}$ будемо називати величину $mnk$. Задано набір матриць $A_1,A_2,\ldots,A_n$, де кожна матриця $A_i$ має розмір $r_{i-1}\times r_i$. Вартість добутку $A_1A_2\ldots A_n$ визначається порядком виконання відповідних операцій. Потрібно знайти мінімальну вартість добутку цих матриць.

*Розв'язання.* Нехай cost(i, j) – мінімальна вартість добутку $A_iA_{i+1}\ldots A_j$. Тоді мінімальна вартість добутку $A_i\ldots A_k\ A_{k+1}\ldots A_j$ дорівнює сумі мінімальних вартостей добутків $A_i\ldots A_k$, $A_{k+1}\ldots A_j$ та величини $r_{i-1}r_kr_j$. Отже,

$$\text{cost}(i,j)=\min_{k=i,\ldots,j-1}\ \text{cost}(i,k)+\text{cost}(k+1,j)+r_{i-1}r_kr_j\ .$$

Рекурсивный алгоритм розв'язання цієї задачі має такий вигляд:

```
function cost(i,j)
 begin
   if i=j
   then
     return 0
```

```
  else
    return min_{k=i,...,j-1}(cost(i,k)+cost(k+1,j)+r[i-1]*r[k]*r[j])
 end
```

Оцінимо час роботи алгоритму. Нехай T(n) – найгірший час виконання алгоритму, коли j – i + 1 = n. Тоді T(0) = c та при n > 0 виконується рівність

$$T(n) = \sum_{m=1}^{n-1} (T(m) + T(n-m)) + dn,$$

звідки випливає, що $T(n) = \Theta\ 3^n$ . Отже, час роботи запропонованого алгоритму є експоненціальним.

Модифікуємо наведений вище алгоритм.

Будемо зберігати значення cost(i, j) в m[i, j]. Тоді m[i, j] = 0, якщо i = j, і $m[i, j] = \min_{k=i,\ldots,j-1}(m[i, k] + m[k+1, j] + r_{i-1}r_k r_j)$ в усіх інших випадках.

Динамічний алгоритм розв'язання задачі:

```
function matrix(n)
 begin
  for i:=1 to n do
    m[i,i]:0
  for d:=1 to n-1 do
    for i:=1 to n-d do
      j:=i+d
      m[i,j]:=min_{k=i,...,j-1}(m[i,k]+m[k+1,j]+r[i-1]*r[k]*r[j])
  return m[1,n]
 end
```

Час роботи цього алгоритму становить $O\ n^3$ . Таким чином, час роботи динамічного алгоритму розв'язання задачі про добуток набору узгоджених матриць є поліноміальним.

**Приклад 5.4.** Задача про всі найкоротші шляхи у графі. Нагадаємо, що графом називається упорядкована пара G = (V,E), де $V = v_1,\ldots,v_n$ – множина вершин, E – множина неупорядкованих пар елементів множини V (іноді вважають, що E – це множина ребер). Якщо елементами множини E є упорядкованими парами вершин, то граф називають орієнтованим. Якщо кожному ребру орієнтованого графа поставлено у відповідність деяке число (вага або вартість), то такий граф називають зваженим. Шляхом у графі G = (V,E) називається набір ребер $v_{i_1}, v_{i_2}$ , $v_{i_2}, v_{i_3}$ ,..., $v_{i_k}, v_{i_{k+1}}$ , кожне з яких належить множині E. Довжина шляху – кількість ребер, з яких він складається, а вартість – сума вартостей відповідних ребер.

Задано зважений орієнтований граф G = (V,E) з додатною функцією вартостей c: $E \to R$. Для кожної пари вершин $u, v \in V$ потрібно знайти шлях із u в v, що має мінімальну вартість.

*Розв'язання.* Розглянемо набір матриць $A_k \ {}^{n}_{k=0}$ розміром $n \times n$ (n – кількість вершин графа). Елементом $A_k[i, j]$ матриці $A_k$ є вартість найкоротшого шляху із $v_i$ в $v_j$ з внутрішніми вершинами $v_1, \ldots, v_k$ .

Ідея розв'язання поставленої задачі полягає у послідовному знаходженні матриць $A_k \ {}^{n}_{k=1}$. Зазначимо, що елементами матриці $A_n$ будуть вартості всіх найкоротших шляхів у графі.

Розглянемо найкоротший шлях із $v_i$ в $v_j$ з внутрішніми вершинами $v_1, \ldots, v_k$ . Якщо цей шлях не містить вершину $v_k$, то $A_k[i, j] = A_{k-1}[i, j]$, у протилежному випадку цей шлях містить $v_k$ і $A_k[i, j] = A_{k-1}[i, k] + A_{k-1}[k, j]$. Таким чином,

$$A_k[i, j] = \min \ A_{k-1}[i, j], A_{k-1}[i, k] + A_{k-1}[k, j] \ .$$

Із цієї рівності випливає, що для заповнення матриці $A_k$ потрібні лише k-й рядок і k-й стовпець матриці $A_{k-1}$. Окрім того, в матрицях $A_k$ і $A_{k-1}$ рядки і стовпці з номером k збігаються. Тому при складанні алгоритму можна використовувати тільки одну матрицю.

Для кожних $i, j = 1, \ldots, n$ виконується таке:

1) $A_0[i, j] = c(v_i, v_j)$, якщо і відрізняється від j і ребро $(v_i, v_j)$ належить множині E;

2) $A_0[i, j]$ дорівнює нескінченності у випадку, коли і відрізняється від j і ребро $(v_i, v_j)$ не належить множині E;

3) $A_0[i, j] = 0$, якщо i = j.

Алгоритм розв'язання задачі про всі найкоротші шляхи у графі:

```
for i:=1 to n do
  for j:=1 to n do
    P[i,j]:=0
    if  v_i, v_j  ∈ E
    then
      A[i,j]:= c  v_i, v_j
    else
      A[i,j]:=∞
  A[i,i]:=0
```

```
for k:=1 to n do
  for i:=1 to n do
    for j:=1 to n do
      if A[i,k]+A[k,j]<A[i,j]
      then
        A[i,j]:=A[i,k]+A[k,j]
        P[i,j]:=k
```

Наведений алгоритм називається алгоритмом Флойда. Час його роботи становить $O(n^3)$. По завершенні алгоритму матриця A буде містити вартості всіх найкоротших шляхів. Елементом P[i, j] матриці P є номер вершини, яка належить найкоротшому шляху із $v_i$ у $v_j$. Щоб вивести на екран всі найкоротші шляхи, можна використати такий алгоритм:

```
for i:=1 to n do
 for j:=1 to n do
  if A[i, j]<∞
  then
    print(v_i)
    shortest(i, j)
    print(v_j)

procedure shortest(i, j)
 begin
  k:=P[i,j]
  if k>0
  then
    shortest(i,k)
    print(v_k)
    shortest(k,j)
 end
```

Тут процедура shortest(i, j) виводить на екран внутрішні вершини найкоротшого шляху з вершини $v_i$ у вершину $v_j$.

### Задачі для самостійного розв'язання

5.1. Нехай $X(n)$ – це кількість способів розстановки дужок у добутку матриць $A_1A_2,\ldots,A_n$. Довести, що $X(n) \geq 2^{n-2}$.

5.2. Довести, що величина $X(n)$ із попередньої задачі дорівнює $\frac{1}{n}C_{2(n-1)}^{n-1}$.

5.3. Розробити алгоритм обчислення чисел Фібоначчі з використанням динамічного програмування. Оцінити час роботи цього алгоритму.

5.4. Задано зважений орієнтований граф. Знайти кількість всіх шляхів між всіма парами його вершин.

5.5. Задано зважений орієнтований граф. Знайти кількість всіх найкоротших шляхів між всіма парами його вершин.

5.6. Задача про розмін монет. Розглянемо множину номіналів монет $A_n = \{a_1, a_2, ..., a_n\}$, де $a_i \in \mathbb{N}$ і $a_1 < a_2 < ... < a_n$. Задано деяку суму $C \in \mathbb{N}$. Вважаючи, що кількість монет кожного номіналу є необмеженою, знайти найменшу комбінацію монет, за допомогою якої можна подати C. Побудувати динамічний алгоритм розв'язання цієї задачі. Оцінити час його роботи.

5.7. Функцію S(n) визначено рівністю

$$S(n) = n\sum_{i=1}^{n}\left(1+\frac{1}{i}\right)^i,\ n \in \mathbb{N}.$$

Використовуючи динамічне програмування, знайти значення цієї функції для $n = 1,2,...,100$.

5.8. Модифікувати динамічний алгоритм із прикладу 5.3 так, щоб можна було визначити порядок оптимальної розстановки дужок.

5.9. Довести коректність процедури shortest(i, j) з прикладу 5.4.

5.10. Знайти час роботи алгоритму виведення найкоротших шляхів у прикладі 5.4.

## 6. ЖАДНІ АЛГОРИТМИ

### Теоретичний матеріал

Алгоритми розв'язання багатьох оптимізаційних задач найчастіше складаються з набору кроків, на кожному з яких виходячи з тих або інших міркувань потрібно зробити деякий вибір. Алгоритм називають жадним, якщо цей вибір завжди є локально оптимальним. Іншими словами, на кожному кроці приймають локально оптимальний розв'язок. Основним недоліком таких алгоритмів є те, що отримані з їх допомогою розв'язки не завжди є глобально оптимальними. Водночас жадні алгоритми поєднують простоту реалізації зі швидким часом роботи.

## Приклади розв'язання задач

**Приклад 6.1.** Неперервна задача про рюкзак. Задано додатні числа $S$, $s_1,\ldots,s_n$, $w_1,\ldots,w_n$. Розв'язати екстремальну задачу

$$\begin{cases} x_1w_1 + x_2w_2 + \ldots + x_nw_n \to \min; \\ x_1s_1 + x_2s_2 + \ldots + x_ns_n \le S; \\ x_i \in [0,1],\ i=1,\ldots,n. \end{cases}$$

*Розв'язання.* Позначимо $\rho_i = w_i / s_i$. Будемо вважати, що пари $w_i, s_i \ _{i=1}^{n}$ розташовані так, що виконується умова $\rho_1 \le \rho_2 \le \ldots \le \rho_n$.

Далі виконаємо такі дії:

```
s:=S
i:=1
while s[i]<=s do
  x[i]:=1
  s:=s-s[i]
  i:=i+1
x[i]:=s/s[i]
for j:=i+1 to n do
  x[j]:=0
```

Час роботи цього жадного алгоритму становить O(n). Доведемо його коректність.

Нехай $X = x_1,\ldots,x_n$ – розв'язок, що отримано за допомогою наведеного жадного алгоритму.

Якщо всі $x_i$ дорівнюють 1, то $X = x_1,\ldots,x_n$, і це є розв'язком екстремальної задачі.

У протилежному випадку позначимо через j найменший номер, такий, що $x_j \ne 1$. Тоді $x_i = 1$ для будь-якого i = 1,…,j – 1, $x_j \in 0,1$ й $x_i = 0$ для всіх i = j+1,…,n. Із цього отримуємо рівність $\sum_{i=1}^{j} x_i s_i = S$.

Нехай $Y = y_1,\ldots,y_n$ – розв'язок екстремальної задачі. Доведемо, що $\sum_{i=1}^{n} x_i w_i = \sum_{i=1}^{n} y_i w_i$. Будемо вважати, що $X \ne Y$. Нехай k – найменший номер, такий, що $x_k \ne y_k$. Покажемо, що $y_k < x_k$. Можливі три випадки.

1. k < j. Тоді $x_k = 1$. Оскільки $x_k \neq y_k$ й $y_k \leq 1$, то $y_k < 1 = x_k$.
2. k = j. За означенням номера k маємо $x_k \neq y_k$. Якщо $y_k > x_k$, то

$$S = \sum_{i=1}^{n} y_i s_i = \sum_{i=1}^{k-1} y_i s_i + y_k s_k + \sum_{i=k+1}^{n} y_i s_i =$$

$$= \sum_{i=1}^{k-1} x_i s_i + y_k s_k + \sum_{i=k+1}^{n} y_i s_i =$$

$$= \sum_{i=1}^{k} x_i s_i + (y_k - x_k) s_k + \sum_{i=k+1}^{n} y_i s_i =$$

$$= \sum_{i=1}^{j} x_i s_i + (y_k - x_k) s_k + \sum_{i=k+1}^{n} y_i s_i =$$

$$= S + (y_k - x_k) s_k + \sum_{i=k+1}^{n} y_i s_i > S,$$

що приводить до протиріччя. Отже, $y_k < x_k$.

3. k > j. У цьому випадку $x_k = 0$ і $y_k > 0$. Тоді

$$S = \sum_{i=1}^{n} y_i s_i = \sum_{i=1}^{j} y_i s_i + \sum_{i=j+1}^{n} y_i s_i = \sum_{i=1}^{j} x_i s_i + \sum_{i=j+1}^{n} y_i s_i = S + \sum_{i=j+1}^{n} y_i s_i > S,$$

звідки випливає, що цей випадок є неможливим.

Отже, $y_k < x_k$.

Нехай $Z = (z_1, \ldots, z_n)$ – набір коефіцієнтів, отриманих зі збільшенням $y_k$ до $x_k$ і зменшенням $y_{k+1}, \ldots, y_n$:

1) $\sum_{i=1}^{n} z_i s_i = S$;
2) $(z_k - y_k) s_k > 0$;
3) $\sum_{i=k+1}^{n} (z_i - y_i) s_i < 0$;
4) $(z_k - y_k) s_k + \sum_{i=k+1}^{n} (z_i - y_i) s_i = 0$.

Тоді

$$\sum_{i=1}^{n} z_i w_i = \sum_{i=1}^{k-1} z_i w_i + z_k w_k + \sum_{i=k+1}^{n} z_i w_i = \sum_{i=1}^{k-1} y_i w_i + z_k w_k + \sum_{i=k+1}^{n} z_i w_i =$$

$$= \sum_{i=1}^{n} y_i w_i + (z_k - y_k)\, s_k \frac{w_k}{s_k} + \sum_{i=k+1}^{n} (z_i - y_i)\, s_i \frac{w_i}{s_i} \le$$

$$\le \sum_{i=1}^{n} y_i w_i + (z_k - y_k)\, s_k \frac{w_k}{s_k} + \sum_{i=k+1}^{n} (z_i - y_i)\, s_i \frac{w_k}{s_k} =$$

$$= \sum_{i=1}^{n} y_i w_i + \frac{w_k}{s_k}\left( (z_k - y_k)\, s_k + \sum_{i=k+1}^{n} (z_i - y_i)\, s_i \right) = \sum_{i=1}^{n} y_i w_i .$$

Оскільки Y є розв'язком задачі, то $\sum_{i=1}^{n} z_i w_i = \sum_{i=1}^{n} y_i w_i$.

Отже, Z також є розв'язком екстремальної задачі.

Якщо скористатися наведеною вище процедурою перетворення, то за деяку скінченну кількість кроків буде отримано розв'язок задачі Z, який повністю збігається з X. Отже, X – розв'язок екстремальної задачі.

Таким чином, жадний алгоритм має достатньо просту реалізацію і лінійний час роботи. Водночас доведення його коректності не є тривіальним.

**Приклад 6.2.** Алгоритм Дейкстри. Задано зважений орієнтований граф $G = \{V,E\}$ без ребер від'ємної ваги. Знайти найкоротшу відстань від вершини $s \in V$ до всіх інших вершин графа.

*Розв'язання.* Позначимо через $\delta(v,w)$ вартість найкоротшого шляху із вершини v у вершину w. Нехай C[v,w] – вартість ребра (v,w). Якщо v = w, то C[v,w] = 0. Якщо $(v,w) \notin E$, то $C[v,w] = \infty$.

Жадна ідея розв'язання поставленої задачі полягає у послідовному заповненні множини S, яка є множиною таких вершин графа, мінімальна відстань до яких вже відома. На початку S = {s}. Послідовним заповненням цієї множини отримуємо S = V.

Шлях із вершини s будемо називати внутрішнім, якщо всі його внутрішні вершини належать S.

Нехай D – масив таких елементів:

1) якщо $w \notin S$, то D[w] дорівнює вартості найкоротшого внутрішнього шляху в w;
2) якщо $w \in S$, то D[w] є вартістю найкоротшого шляху із s у w.

Вважаємо, що вершина w лежить на межі, якщо $w \notin S$ й існує така вершина $u \in S$, що $(u,w) \in E$. Всі внутрішні шляхи завершуються у S або на межі.

Додавати до множини S будемо ту вершину $w \in V \setminus S$ графа, якій відповідає мінімальне значення D[w]. Ця вершина, зрозуміло, лежить на межі. Доведемо, що для неї виконується рівність $D[w] = \delta(s,w)$. Для цього розглянемо найкоротший шлях із s у w. Нехай x – перша вершина межі, що лежить на цьому шляху. Тоді можна записати

$$\delta(s,w) = \delta(s,x) + \delta(x,w) = D[x] + \delta(x,w) \geq D[w] + \delta(x,w) \geq D[w].$$

За означенням $\delta(s,w) \leq D[w]$. Отже, $D[w] = \delta(s,w)$.

Установимо справедливість вислову: якщо вершину u було додано до множини S перед вершиною v, то виконується нерівність $\delta(s,u) \leq \delta(s,v)$. Доведення проводимо за допомогою індукції. Параметр індукції – кількість вершин, які вже знаходяться в S у момент додавання вершини v. При |S|=1 це твердження, зрозуміло, є коректним. Припустимо його справедливість при |S| = k – 1. Нехай x – остання внутрішня вершина, що лежить на найкоротшому шляху в v. Розглянемо декілька випадків:

1. Вершину x додано до S перед u. Повернемося до моменту, коли було додано вершину u. Оскільки $x \in S$, то v лежить на межі, але оскільки вибір було зроблено на користь u, то $D[u] \leq D[v]$. Згідно з раніше встановленою властивістю $D[u] = \delta(s,u)$. За означенням x через те, що вершину x було додано до S перед u, маємо $\delta(s,u) = D[u] \leq D[v] = \delta(s,v)$.
2. Вершину x додано до S після u. Оскільки x додано до S перед v, то у відповідний момент часу $|S| < n$. Унаслідок цього припущення справедлива нерівність $\delta(s,u) \leq \delta(s,x)$. Отже,
$$\delta(s,u) \leq \delta(s,x) \leq \delta(s,v).$$

Тепер залишилося з'ясувати, як зміниться масив D після додавання вершин до множини S.

Нехай до множини S додано вершину w. Значення D[v] для вершин межі може змінитися, оскільки можливим є виникнення нового внутрішнього шляху, що містить w. Це може трапитися в одному з двох випадків:

1) w є останньою внутрішньою вершиною, що лежить на новому внутрішньому шляху;
2) w не є останньою внутрішньою вершиною, що лежить на новому внутрішньому шляху.

У першому випадку вартість нового внутрішнього шляху становить D[w]+C[w,v], у другому – унаслідок доведеного вище новий шлях не може бути коротшим від будь-якого іншого шляху із s в v.

Якщо вершина v належить S, то D[v] не зміниться, оскільки відповідний шлях вже є найкоротшим.

Якщо вершина v була зовні межі, то перед додаванням w існує рівність $D[y] = \infty$, а після – D[y]=D[w]+C[w,y].

Таким чином, після додавання вершини w до множини S отримуємо

$$D[v] := \min\ D[v], D[w] + C[w,v] \quad \text{для всіх } v \in V.$$

Тепер можна побудувати алгоритм розв'язання поставленої задачі:

```
S:={s}
for each u∈V do
  D[u]:=C[s,u]
for i:=1 to n-1 do
  choose w∈V\S with smallest D[w]
  S:=S∪{w}
  for each u∈V do
    D[v]:=min{D[u],D[w]+C[w,u]}
```

Цей алгоритм називають алгоритмом Дейкстри. Час його роботи становить $O\ n^2$ .

Як і у попередньому прикладі, наведений жадний алгоритм поєднує простоту реалізації і поліноміальний час роботи з достатньо громіздким й нетривіальним доведенням коректності.

Зауважимо, алгоритм Дейкстри допускає різні варіанти реалізації, що мають складність, яка може відрізнятися від наведеної вище.

### Задачі для самостійного розв'язання

6.1. Розв'язати задачу 5.6 за допомогою жадних алгоритмів.

6.2. Модифікувати алгоритм із прикладу 6.2 з урахуванням можливості знаходження найкоротших шляхів.

6.3. Оцінити складність алгоритмів із прикладів 6.1, 6.2 і 6.4. Чи є суттєвою властивість відсутності ребер від'ємної вартості у прикладі 6.2?

## БІБЛІОГРАФІЧНИЙ СПИСОК

1. Асанов, М. О. Дискретная математика: графы, матроиды, алгоритмы / М. О. Асанов, В. А. Баранский, В. В. Расин. – Ижевск : Регулярная и хаотическая динамика, 2001. – 288 с.

2. Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. – М. : Вильямс, 2005. – 1296 с.

3. Кнут, Д. Искусство программирования для ЭВМ. В 4 т. Т.1. Основные алгоритмы / Д. Кнут. – М. : Вильямс, 2006. – 720 с.

4. Кнут, Д. Искусство программирования для ЭВМ. В 4 т. Т. 2. Получисленные алгоритмы / Д. Кнут. – М. : Вильямс, 2007. – 832 с.

5. Кнут, Д. Искусство программирования для ЭВМ. В 4 т. Т. 3. Сортировка и поиск / Д. Кнут. – М. : Вильямс, 2007. – 824 с.

6. Новиков, Ф. А. Дискретная математика для программистов / Ф. А. Новиков. – СПб. : Питер, 2000. – 304 с.

7. Кормен, Т. Алгоритмы. Вводный курс / Т. Кормен. – М. : Вильямс, 2014. – 208 с.

8. Корте, Б. Комбинаторная оптимизация. Теория и алгоритмы / Б. Корте, Й. Фиген. – М. : МЦНМО, 2015. – 720 с.

9. Седжвик, Р. Фундаментальные алгоритмы на С++. В 4 ч. / Р. Седжвик. – СПб. : ДиаСофтЮП, 2002. – 688 с.

10. Стивенс, Р. Алгоритмы. Теория и практическое применение / Р. Стивенс. – СПб. : Питер, 2016. – 272 с.

11. Bryant, R.E. Computer systems: a programmer's perspective / R. E. Bryant, D. R. O'Hallaron. – Pearson, 2010. – 1080 p.

**ЗМІСТ**

Навчальне видання

**Брисіна Ірина Вікторівна**
**Макарічев Віктор Олександрович**

# АЛГОРИТМИ І АНАЛІЗ СКЛАДНОСТІ

Редактор В. І. Філатова

Зв. план, 2020
Підписано до видання 26.06.2020
Ум. друк. арк. 2,3. Обл.-вид. арк. 2,56. Електронний ресурс

---

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001