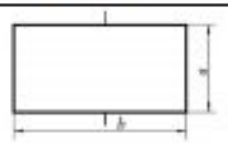
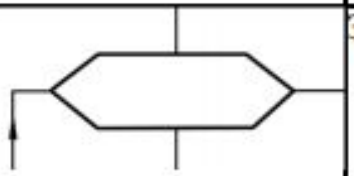
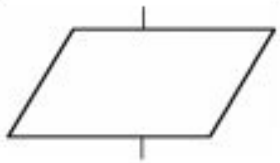
Властивості алгоритму:

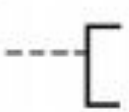
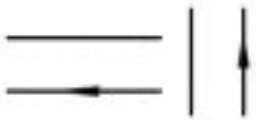
1. Дискретність алгоритму.
2. Масовість алгоритму.
3. Детермінованість алгоритму.
4. Елементарність кроків алгоритму.
5. Результативність алгоритму.

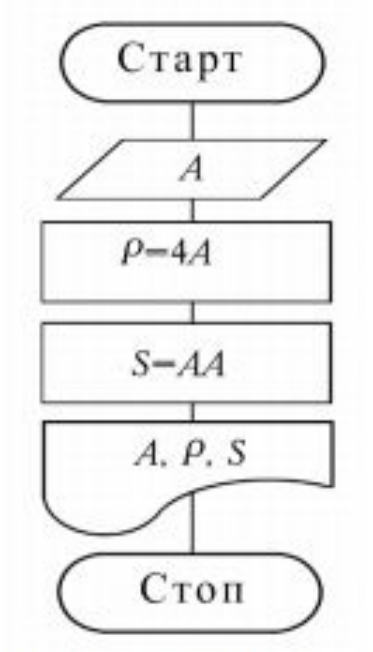
Способи представлення алгоритмів:

1. Спосіб представлення алгоритму у вербальній (словесній) формі
2. Спосіб задавання алгоритму в графічній формі
3. Спосіб задавання алгоритму у вигляді псевдокоду
4. Спосіб задавання алгоритму у вигляді комп’ ютерної програми

Основні елементи блок-схем

Назва	Елемент	Коментар
Процес		Обчислювальна дія або послідовність обчислювальних дій
Розв'язування		Перевірка умови
Модифікація		Заголовок циклу
Визначений процес		Звертання до процедури
Документ		Вивід даних, друк даних
Введення/Виведення		Ввід/Вивід даних

Назва	Елемент	Коментар
З'єднувач		Розрив лінії потоку
Початок, Кінець		Початок, кінець, пуск, зупинка, вхід і вихід у допоміжних алгоритмах
Коментар		Використовують для розміщення написів
Горизонтальні й вертикальні потоки		Лінії зв'язків між блоками, напрямки потоків
Злиття		Злиття ліній потоків
Міжсторінковый з'єднувач		Перехід на інший лист



. Приклад лінійного алгоритму

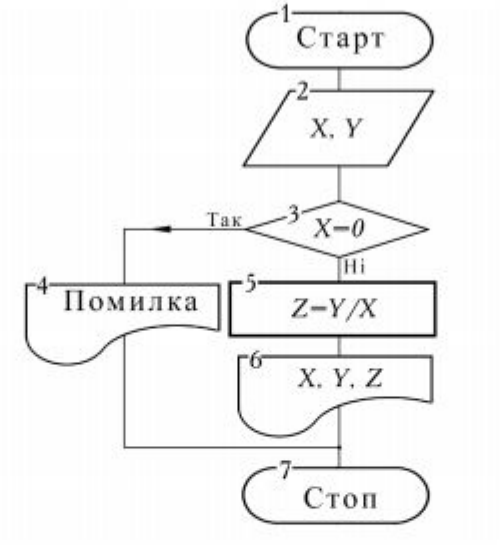
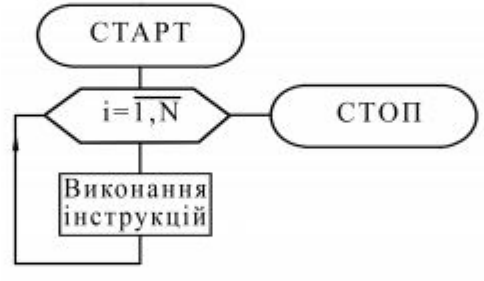


Рис. 1.5. Алгоритм, що розгалужується



.6. Алгоритм з детермінованим циклом

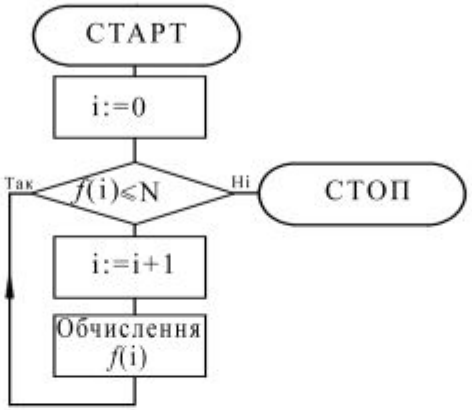


Рис. 1.7. Алгоритм із ітераційним циклом

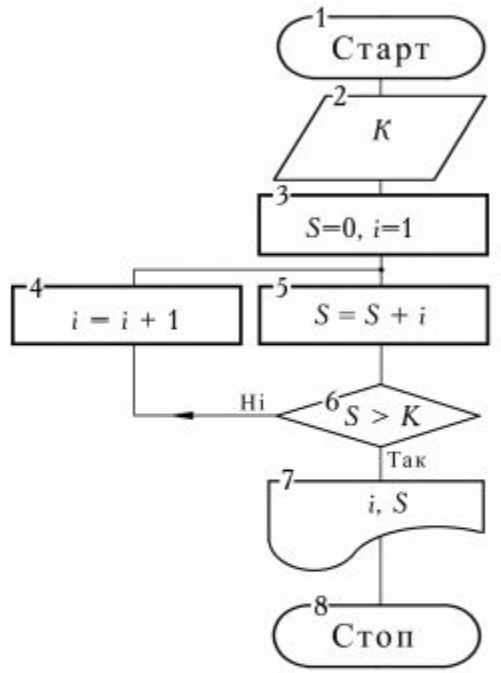


Рис. 1.8. Приклад циклічного алгоритму

Складність алгоритму:
1. Теоретична
2. Практична
3. Часова
4. Емнісна

На складність алгоритму впливають:
1. Швидкодія ПК та його ресурси
2. Математична модель та метод реалізації задачі
3. Мова програмування
4. Досвід програміста

Математичні операції з теоретичною часовою складністю

$$O(1{,}5N) = O(N).$$

$$O(17N \cdot N) = O(17N)\,O(N) = O(N)\,O(N) = O(N \cdot N) = O(N^2).$$

$$O(N^5 + N^2) = O(N^5).$$

Види функції складності

$O(1)$.

$O(N)$.

$O(N^2)$, $O(N^3)$, $O(N^a)$ —

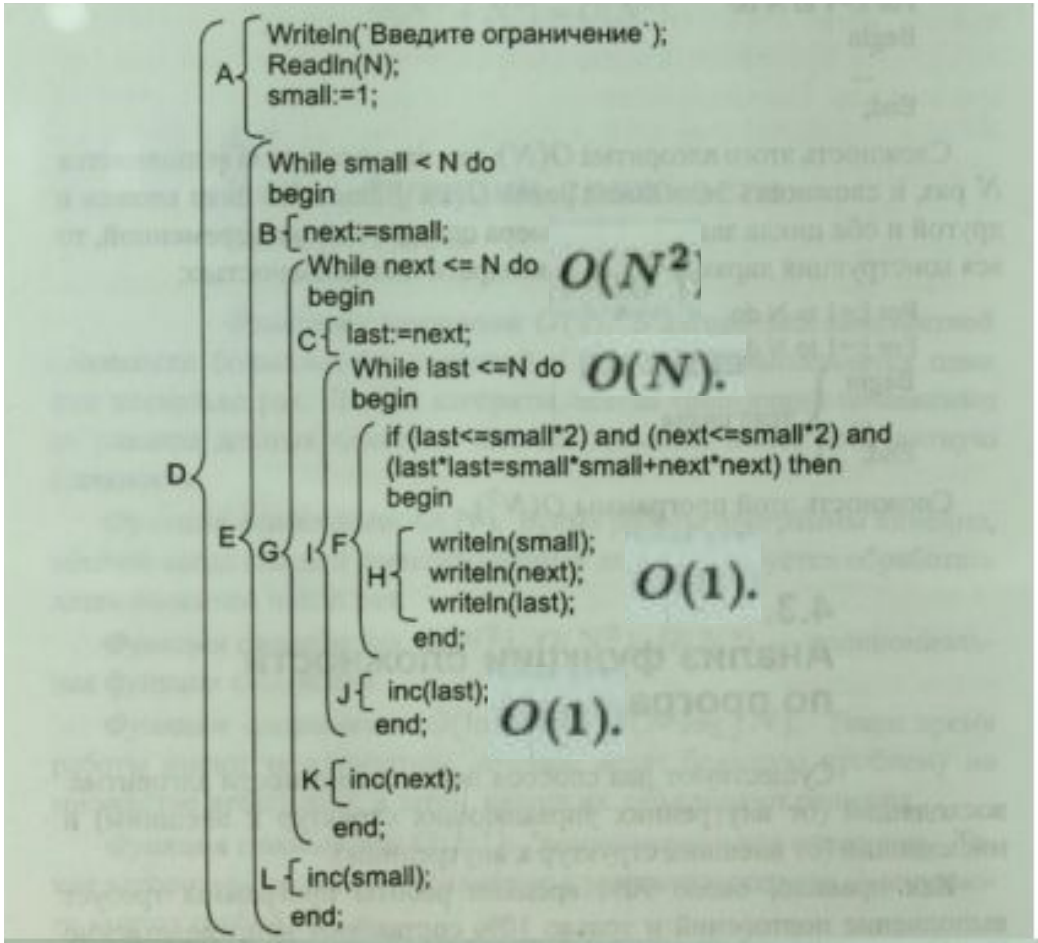
$O(\log_2 N)$, $O(N \log_2 N)$.

$O(N)$.

```
For i:=1 to N do
Begin
  ...
End;
```

```
For i:=1 to N do
For j:=1 to N do
Begin
  ...
End;
```
} тело цикла

Сложность этой программы $O(N^2)$.



$$O(H)=O(1)+O(1)+O(1)=O(1);$$

$$O(I)=O(N)\cdot(O(F)+O(J))=O(N)\cdot O(\text{доминанты условия})=O(N);$$

$$O(G)=O(N)\cdot(O(C)+O(I)+O(K))=O(N)\cdot(O(1)+O(N)+O(1))=O(N^2);$$

$$O(E)=O(N)\cdot(O(B)+O(G)+O(L))=O(N)\cdot O(N^2)=O(N^3);$$

$$O(D)=O(A)+O(E)=O(1)+O(N^3)=O(N^3).$$

Сложность данного алгоритма $O(N^3)$.

Складність оцінюють:

1. За визначеними формулами

Действительное время (Длина массива) =
$= \text{Длина массива}^2 + 5 \cdot \text{Длина массива} + 100.$

Грубое значение определяется вспомогательной функцией:

Оценка времени (Длина массива) $= 1{,}1 \cdot \text{Длина массива}^2$.

2. На основі результатів експерименту з варіаціного ряду

N	Количество сравнений
6	54

$$an, \quad an\log_2 n, \quad an^2, \quad an^3, \quad ae^n, \quad an!$$

$$1 \le a \le 2;$$

$$\lim_{n\to\infty} \frac{O_э(n)}{O_т(n)} = \text{const} \neq 0, \qquad \lim_{n\to\infty} \frac{1{,}5n^2}{X} = \text{const} \neq 0,$$

$$\lim_{n\to\infty} \frac{1{,}5n^2}{n^2} = \text{const} \neq 0.$$

Отсюда теоретическая функция сложности — $O(n^2)$.

3. На основі відношення теоретичної та практичної функції складності

. Полином $f(n) = 2n^5 + 6n^4 + 6n^2 + 18$ имеет $O(n^5)$,

$$\lim_{n\to\infty} \frac{2n^5 + 6n^4 + 6n^2 + 18}{n^5} = 2.$$

числе и в теории алгоритмов, функцию $f(n)$ определяют как $O(g(n))$ и говорят, что она порядка $g(n)$ для больших n, если

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \text{const} \neq 0,$$

где $f(n)$ и $g(n)$ — экспериментальная и теоретическая функции сложности. При этом если $f(n) = O(g(n))$, то предел отношения равен константе и тогда функция $f(n)$ имеет порядок $O(g(n))$ («О большое»).

4. Якщо цикл переглядає не повний список елементів, то будується модель переглядання елементів циклу

Первая итерация цикла имеет дело со всем списком. Каждая последующая итерация делит пополам размер массива. Так, размерами списка для алгоритма являются

$$n, \quad \frac{n}{2^1}, \quad \frac{n}{2^2}, \quad \frac{n}{2^3}, \quad \frac{n}{2^4}, \quad \ldots, \quad \frac{n}{2^m}.$$

В конце концов будет такое целое m, что $\frac{n}{2^m} < 2$ или $n < 2^{m+1}$. Так как m — это первое целое, для которого $\frac{n}{2^m} < 2$, то должно быть верно $\frac{n}{2^{m-1}} \geq 2$ или $2^m \leq n$. Из этого следует, что $2^m \leq n < 2^{m+1}$. Возьмем логарифм каждой части неравенства и получим $m \leq \log_2 n < m + 1$. Отсюда $O(\log_2 n)$.

5. За трудомісткістю

- Трудоемкость следования есть сумма трудоемкостей блоков, следующих друг за другом:

$$F = F_1 + F_2 + F_3 + \ldots + F_n.$$

- Трудоемкость ветвления определяется вероятностью перехода к каждой из инструкций исходя из условия, проверка условия также имеет трудоемкость:

$$F_{if} = F_1 + F_{then} \cdot P + F_{else} \cdot (1 - P),$$

где P и $(1 - P)$ — это вероятности перехода по условиям then и else ответственно.

- Трудоемкость цикла зависит от его вида, для цикла от 1 до n с параметрами будет справедливой формула:

$$F_{for} = 1 + 3 \cdot n + n \cdot F.$$

6. За кількістю процесорних операцій

Нехай $N_1,...,N_{k-1}$ – середнє число звернень до операторів $V_1,...,V_{k-1}$ за один прогін алгоритму, k_i – число операцій, які складають i-оператор. У такому випадку характеристики трудомісткості можуть бути обчислені таким чином.

Визначаємо середнє число процесорних операцій, що виконуються при одному прогоні алгоритму:

$$\theta \le \sum_{V_{ai}} n_i k_i, \qquad (1)$$

Після цього визначаємо середнє число звернень до файлу F_h:

$$N_h = \sum_{V_i} n_i (h = 1,...,H) \qquad (2)$$

та середню кількість інформації, що передається при одному зверненні до файла F_h:

$$\theta_h = \frac{1}{N_h} \sum_{V_i}^{k-1} n_i l_i (h = 1,...,H), \qquad (3)$$

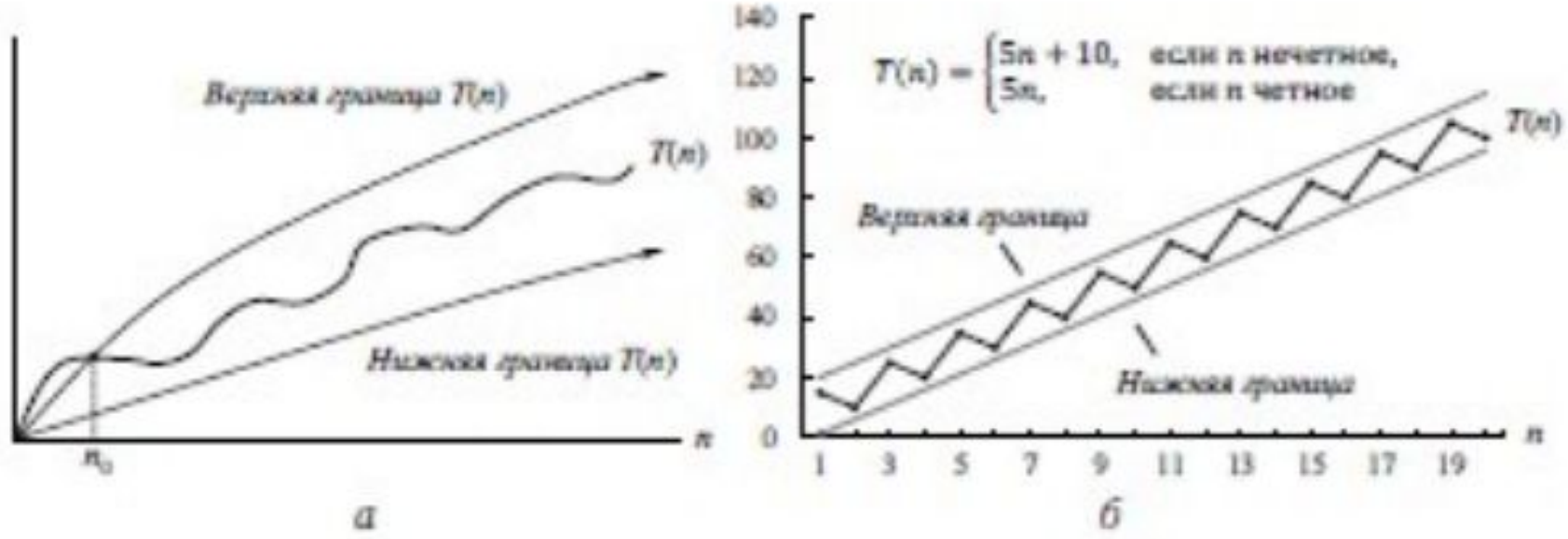


Рис. 1.3. Пример возможных верхней и нижней границ времени $T(n)$ выполнения алгоритмов.

1.5.1. O-обозначение

O-обозначение используют, если необходимо указать *асимптотическую верхнюю границу* (asymptotic upper bound) для функции $f(n)$, числа операций алгоритма. Говорят, что функция $f(n)$ принадлежит множеству функций $O(g(n))$, что записывается как $f(n) \in O(g(n))$, если существуют положительная константа $c > 0$ и $n_0 \in \{0, 1, 2, \ldots\}$, такие что для любых $n \geq n_0$

$$0 \leq f(n) \leq cg(n).$$

Формально множество $O(g(n))$ определяется следующим образом:

$$O(g(n)) = \left\{ f(n): \begin{array}{l} \exists c > 0, n_0 \in \{0, 1, 2, \ldots\}, \text{такие, что} \\ 0 \leq f(n) \leq cg(n), \quad \forall n \geq n_0. \end{array} \right\} \qquad (1.2)$$

1.5.2. Ω-обозначение

Ω-обозначение используется для записи *асимптотической нижней границы* (asymptotic lower bound) для функции $f(n)$. Говорят, функция $f(n)$ принадлежит множеству $\Omega(g(n))$ (записывается как $f(n) \in \Omega(g(n))$), если существуют константа $c > 0$ и $n_0 \in \{0, 1, 2, \ldots\}$, такие что для любых $n \geq n_0$

$$0 \leq cg(n) \leq f(n).$$

Читается как «f от n есть омега большое от g от n». Формально множество $\Omega(g(n))$ определяется следующим образом

$$\Omega(g(n)) = \left\{ f(n): \begin{array}{l} \exists c > 0, n_0 \in \{0, 1, 2, \ldots\}, \text{такие, что} \\ 0 \leq cg(n) \leq f(n), \quad \forall n \geq n_0. \end{array} \right\} \tag{1.3}$$

1.5.3. Θ-обозначение

Θ-обозначение позволяет записать *асимптотически точную оценку* (asymptotic tight bound) для функции $f(n)$. Функция $f(n)$ принадлежит множеству функций $\Theta(g(n))$, записывается как $f(n) \in \Theta(g(n))$, если существуют положительные константы $c_1 > 0, c_2 > 0$ и $n_0 \in \{0, 1, 2, \ldots\}$, такие что для любых $n \geq n_0$

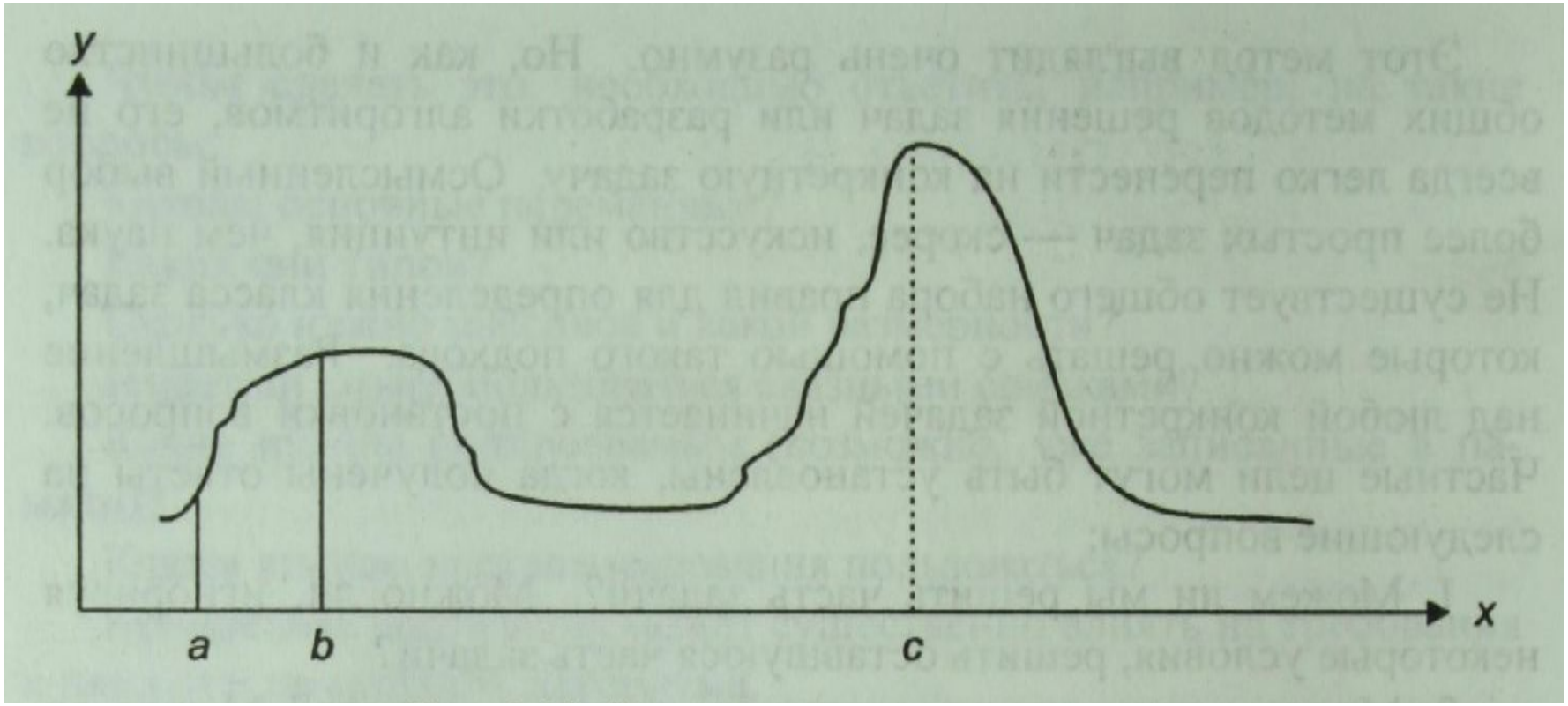
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n).$$

Читается как «f от n есть тета большое от g от n». Множество $\Theta(g(n))$ определяется следующим образом

$$\Theta(g(n)) = \left\{ f(n): \begin{array}{l} \exists c_1 > 0, c_2 > 0, n_0 \in \{0, 1, 2, \ldots\}, \text{такие, что} \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \quad \forall n \geq n_0. \end{array} \right\} \qquad (1.4)$$

Метод часткових цілей пов'язаний зі зведенням важкої (громіздкої) задачі до послідовності простіших задач. У цьому разі розв'язок початкової задачі отримують з розв'язків простіших задач.

Метод підйому починається з прийняття початкового припущення або обчислення початкового розв'язку задачі. Потім відбувається якнайшвидший рух "вверх" від початкового розв'язку в напрямі до ліпших розв'язків. Коли алгоритм досягає такої точки, з якої більше неможливо рухатися вверх, алгоритм зупиняється.


y
x
a
b
c

Метод відпрацювання назад починають з цілі чи розв'язку і рухаються назад за напрямом до початкового формулювання задачі. Далі, якщо ці дії оборотні, рухаються знову від формулювання задачі до розв'язку. Цей метод широко застосовують під час розв'язування різноманітних головоломок.

Рекурсія. Процедуру, яка прямо чи опосередковано звертається до себе, називають рекурсивною. Застосування рекурсії часто дає змогу побудувати зрозуміліші та стислі алгоритми, ніж це було б зроблено без неї. Рекурсія сама по собі не приводить до ефективнішого алгоритму. Однак у поєднанні з іншими методами, наприклад "поділяй і володарюй", дає алгоритми, одночасно ефективні й елегантні.

У методі “поділяй і володарюй” на кожному рівні рекурсії виділяють три етапи [8].

1. Поділ задачі на декілька підзадач.

2. Рекурсивне розв’язування цих підзадач. Коли обсяг підзадачі достатньо малий, виділені підзадачі розв’язують безпосередньо.

3. Комбінування розв’язку вихідної задачі з розв’язків допоміжних задач.

Розглянемо загальний стандартний алгоритм вигляду ПВ.

4.2. Евристичні алгоритми

Евристичні алгоритми визначають як алгоритми з такими властивостями [3]: алгоритм знаходить добрі, хоча не обов'язково оптимальні розв'язки; ці розв'язки можна швидше і простіше реалізувати, ніж відомий точний алгоритм, який гарантує оптимальний розв'язок.

4.3. Метод гілок і меж

Метод гілок і меж орієнтований головно на розв'язування задач оптимізації. Він досліджує деревоподібну модель простору розв'язків задачі та дає змогу серед елементів множини можливих розв'язків знайти найліпший (найоптимальніший). Головна властивість, яку повинна мати ця множина, – це можливість розділяти її на підмножини, що не перетинаються, для яких можна виконати деяку оцінку "оптимальності" можливого розв'язку. "Оптимальність" такого розв'язку означає, що не існує ліпшого розв'язку в межах вибраної підмножини (насправді такої оптимальності на цій підмножині може й не бути).

У разі практичного застосування цього методу всі можливі варіанти розв'язків задачі розбивають на класи (блоки), виконують оцінку знизу (у випадку мінімізації) для всіх розв'язків з одного класу, і якщо вона більша від раніше отриманої, то відкидають усі варіанти з цього класу.

4.4. Динамічне програмування

Р. Беллман увів поняття динамічного програмування для характеристики алгоритмів, які діють залежно від зміни обставин. Методи оптимізації з елементами динамічного програмування були відомі здавна, однак саме Р. Беллман дав строге математичне обґрунтування цієї області.

Динамічне програмування, як звичайно, застосовують до задач оптимізації, у яких для того, щоб отримати оптимальний розв'язок, необхідно зробити певну множину виборів. Після того, як вибір зроблено, часто виникають допоміжні підзадачі, такі ж за виглядом, як і основа.

4.5. Жадібні алгоритми

Для розв'язування багатьох задач оптимізації застосовують так звані жадібні алгоритми. Ці алгоритми діють, використовуючи в кожен момент лише частину вхідних даних, і намагаються зробити найліпший вибір на підставі доступної інформації. Тобто відбувається локально оптимальний вибір у надії, що він приведе до оптимального розв'язку глобальної задачі.

Класифікація алгоритмів сортування

Внутрішнє сортування – це алгоритм сортування, що у процесі впорядкування даних використовує тільки оперативну пам'ять (ОЗП) комп'ютера.

Зовнішнє сортування – це алгоритм сортування, що при проведенні впорядкування даних використовує зовнішню пам'ять, як правило, жорсткі диски.

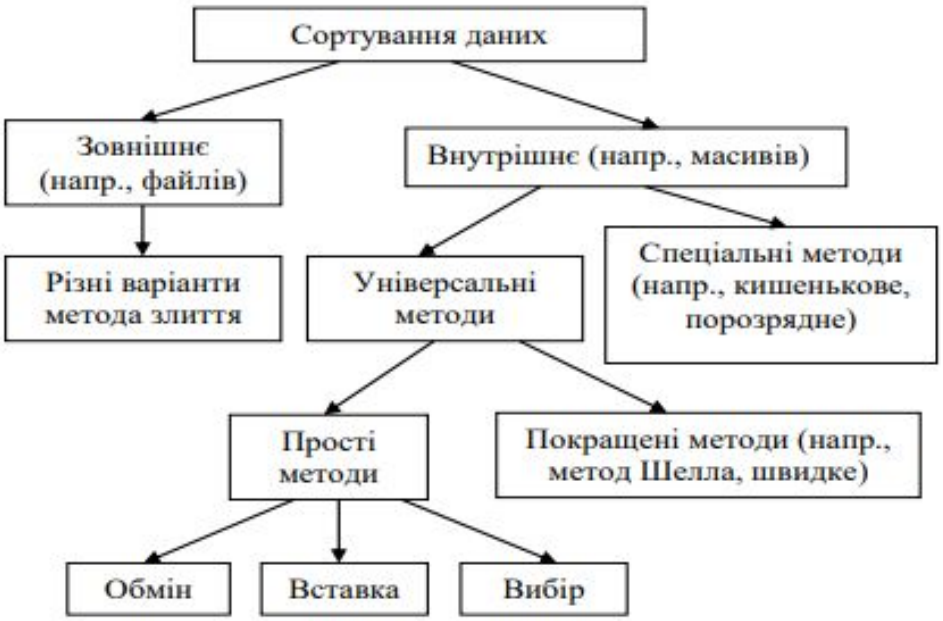


Рисунок 3.1 – Загальна класифікація методів сортування

Класифікація за часом виконання:

За час $O(n^2)$:

- сортування вибором;
- сортування вставкою;
- сортування обміном.

За час $O(n \log n)$:

- пірамідальне сортування;
- швидке сортування;
- сортування злиттям.

За час $O(n)$ з використанням додаткової інформації про елементи :

- сортування підрахунком;
- сортування за розрядами;
- сортування комірками.

За час $O(n \log^2 n)$:

- сортування злиттям модифіковане;
- сортування Шелла.

Сортування вибором Один з найпростіших методів сортування працює в такий спосіб:
1) знаходимо найменший елемент у масиві;
2) міняємо його місцями з елементом, що знаходиться на першому місці;
3) повторюємо процес із другої позиції у файлі й знайдений найменший елемент обмінюємо із другим елементом і так далі поки весь масив не буде відсортований.

Цей метод називається сортуванням вибором оскільки він працює циклічно вибираючи найменший з елементів, що залишилися.

Сортування вставкою Сортування вставкою – це метод який майже настільки ж простий, що й сортування вибором, але набагато більш гнучкий.

Суть алгоритму: беремо один елемент і вставляємо 35 його в потрібне місце серед тих, що ми вже обробили (тим самим залишаючи їх відсортованими).

Алгоритм:
1) зліва направо проходимо масив, порівнюючи сусідні елементи, поки не знайдемо елемент, що розташований не в порядку сортування;
2) обмінюємо цей елемент з елементами розташованими ліворуч від нього, поки він не займе потрібну позицію;
3) повторюємо перші дві дії, поки масив не буде відсортовано.

3	8	9	10	6	7	11	j=5, i=4	6<10
3	8	9	6	10	7	11	i=3	6<9
3	8	6	9	10	7	11	i=2	6<8
3	6	8	9	10	7	11	i=1	6>3
3	6	8	9	10	7	11	j=6, i=5	7<10
3	6	8	9	7	10	11	i=4	7<9
3	6	8	7	9	10	11	i=3	7<8
3	6	7	8	9	10	11	i=2	7>6

Бульбашкове сортування (сортування простими обмінами)

Алгоритм працює таким чином – у масиві порівнюються два сусідні елементи. Якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями. Прохід по списку продовжується до тих пір, доки дані не будуть відсортованими.

Перший прохід:

(**5** **1** 4 2 8) ⟶ (**1** **5** 4 2 8)
(1 **5** **4** 2 8) ⟶ (1 **4** **5** 2 8)
(1 4 **5** **2** 8) ⟶ (1 4 **2** **5** 8)
(1 4 2 **5** **8**) ⟶ (1 4 2 **5** **8**)

Другий прохід:

(**1** **4** 2 5 8) ⟶ (**1** **4** 2 5 8)
(1 **4** **2** 5 8) ⟶ (1 **2** **4** 5 8)
(1 2 **4** **5** 8) ⟶ (1 2 **4** **5** 8)
(1 2 4 **5** **8**) ⟶ (1 2 4 **5** **8**)

Третій прохід:

(**1** **2** 4 5 8) ⟶ (**1** **2** 4 5 8)
(1 **2** **4** 5 8) ⟶ (1 **2** **4** 5 8)
(1 2 **4** **5** 8) ⟶ (1 2 **4** **5** 8)
(1 2 4 **5** **8**) ⟶ (1 2 4 **5** **8**)

Покращення алгоритму бульбашкового сортування

Кролики і черепахи. Позиція елементів, що підлягають сортуванню відіграє велику роль у питанні продуктивності даного алгоритму. Великі елементи на початку списку не викликають проблеми, оскільки вони досить швидко переміщуються на свої місця. Однак, малі елементи у кінці списку переміщуються на його початок дуже повільно. Це призвело до того, що ці типи елементів було названо кроликами і черепахами, відповідно.

Сортування змішуванням – один із різновидів алгоритму сортування бульбашкою. Відрізняється від сортування бульбашкою тим, що сортування відбувається в обох напрямках, міняючи напрямок при кожному проході. Даний алгоритм лише трохи складніший за сортування бульбашкою, однак, вирішує так звану проблему "черепах".

Сортування гребінцем – спрощений алгоритм сортування, розроблений Влодзімежом Добосєвічем Сортування гребінцем є поліпшенням алгоритму сортування бульбашкою, і конкурує у швидкодії з алгоритмом швидкого сортування. Основна його ідея полягає в тому, щоб усунути так званих "черепах" (малі значення) ближче до кінця списку, оскільки у сортуванні бульбашкою вони сильно уповільнюють процес сортування. ("Кролики" або великі значення на початку списку у сортуванні бульбашкою не викликають проблеми).

Алгоритм сортування Шелла Сортування Шелла – це алгоритм сортування, що є узагальненням сортування вставкою.
Алгоритм базується на двох тезах:
 Сортування включенням ефективне для майже впорядкованих масивів.
Сортування включенням неефективне, тому що переміщує елемент тільки на одну позицію за раз. Тому сортування Шелла виконує декілька впорядкувань включенням, кожен раз порівнюючи і переставляючи елементи, що знаходяться на різній відстані один від одного.

Швидке сортування Хоара – удосконалений метод сортування, що базується на обміні. К.Хоар запропонував алгоритм QuickSort сортування масивів, що дає на практиці відмінні результати і дуже просто програмується. Це сортування називають швидким, тому що на практиці воно виявляється найшвидшим методом сортування з тих, що оперують порівняннями. Основна стратегія прискорення алгоритмів сортування – обмін між якомога більш віддаленими елементами вихідного файлу.

Спеціалізовані алгоритми внутрішнього сортування

Сортування підрахунком – алгоритм впорядкування, що застосовується при малій кількості різних елементів (ключів) у масиві даних. Час його роботи лінійно залежить, як від загальної кількості елементів у масиві, так і від кількості різних елементів.
Ідея алгоритму полягає в наступному: спочатку підрахувати скільки разів кожен елемент (ключ) зустрічається в вихідному масиві. Спираючись на ці дані можна одразу вирахувати, на якому місці має стояти кожен елемент, а потім за один прохід поставити всі елементи на свої місця. Псевдокод алгоритму Для простоти будемо вважати, що всі елементи (ключі) є натуральними числами, що лежать в діапазоні 1..K.

Сортування за розрядами (англ. Radix sort) – швидкий стійкий алгоритм впорядкування даних. Застосовується для впорядкування елементів, що є ланцюжками над будь-яким скінченним алфавітом (напр. рядки, або цілі числа).
В якості допоміжного використовує будь-який інший стійкий алгоритм сортування. Алгоритм застосовувався для впорядкування перфокарт.

Ідея полягає в тому, щоб спочатку впорядкувати всі елементи за молодшим розрядом, потім стабільно впорядкувати за другим розрядом, потім за третім і так далі аж до найстаршого. Оскільки, припускається, що кожен розряд приймає значення з невеликого діапазону, то кожен цикл впорядкування можна виконувати швидко і з малими затратами пам'яті.

Алгоритми зовнішнього сортування

Хоча диски називають пам'яттю прямого доступу, час звертання до даних набагато більшого порядку, ніж час читання/запису в основній пам'яті, через: а) затримки, пов'язані з очікуванням моменту, коли потрібний елемент циліндра пройде під головкою; б) переміщення головок, коли потрібні дані не з поточного циліндра. Прохід – це проходження файлу в одному напрямку зі зчитуванням даних в пам’ять й, можливо, їхнім поверненням у файл.

Під злиттям розуміється об'єднання двох (або більше) упорядкованих послідовностей в одну впорядковану послідовність за допомогою циклічного вибору елементів, доступних у цей момент. Злиття – набагато більш проста операція, ніж сортування. Ми розглянемо 2 алгоритми злиття: 1. Пряме злиття. Алгоритм Боуза-Нельсона 2. Природне (Неймановське) злиття.

Пряме злиття. Алгоритм Боуза-Нельсона

1. Послідовність a розбивається на дві половини b і c.
2. Послідовності b і c зливаються за допомогою об'єднання окремих елементів в упорядковані пари.
3. Отриманій послідовності привласнюється ім'я a, після чого повторюються кроки 1 і 2; при цьому впорядковані пари зливаються в упорядковані четвірки.
4. Попередні кроки повторюються, при цьому четвірки зливаються у вісімки і так далі, поки не буде впорядкована вся послідовність, тому що довжини послідовностей щоразу подвоюються.

Початкова послідовність:

A = 44 55 12 42 94 18 06 67

1 b = 44 55 12 42
c = 94 18 06 67
a = 44 94' 18 55' 06 12' 42 67'

2 b = 44 94' 18 55'
c = 06 12' 42 67'
a = 06 12 44 94' 18 42 55 67'

3 b = 06 12 44 94'
c = 18 42 55 67'
a = 06 12 18 42 44 55 67 94

Природне (Нейманівське) злиття Поєднуються впорядковані частини, що спонтанно 62 виникли у вихідному масиві; вони можуть бути також наслідком попередньої обробки даних. Розраховувати на однаковий розмір частин, що зливаються, не доводиться. Записи, що йдуть у порядку не зменшення ключів, зчіплюються у підсписок. Мінімальний підсписок один запис.

Сортування методом поглинання

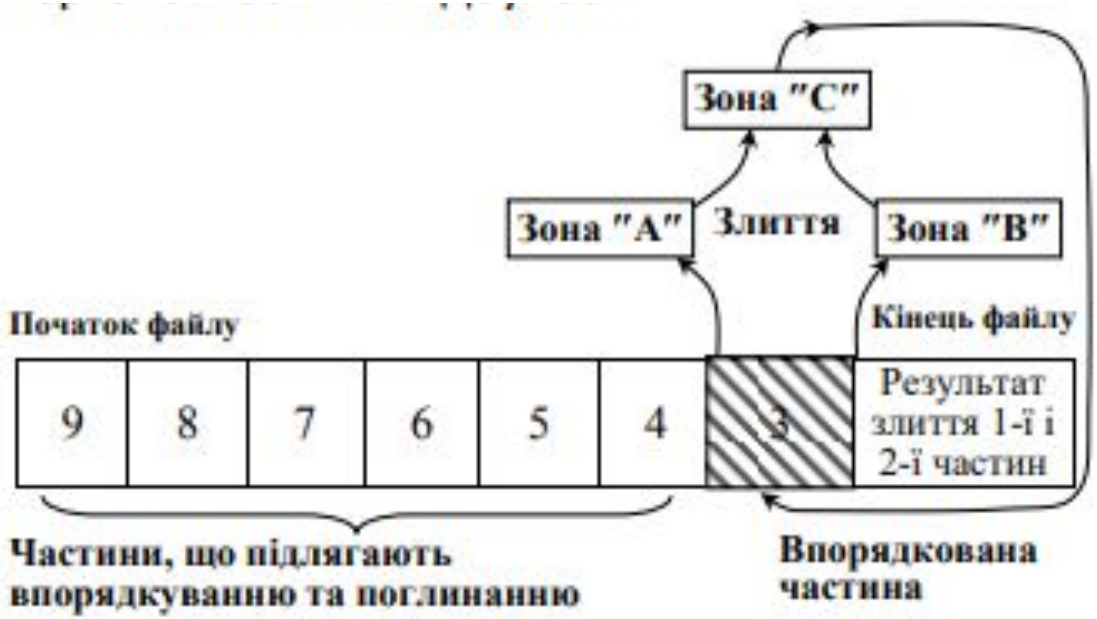


Рисунок 7.5 – Схема сортування поглинанням

Човникове балансове злиття

Рисунок 7.6 – Злиття частин розміром *R*

Рисунок 7.7 – Злиття частин розміром 2R

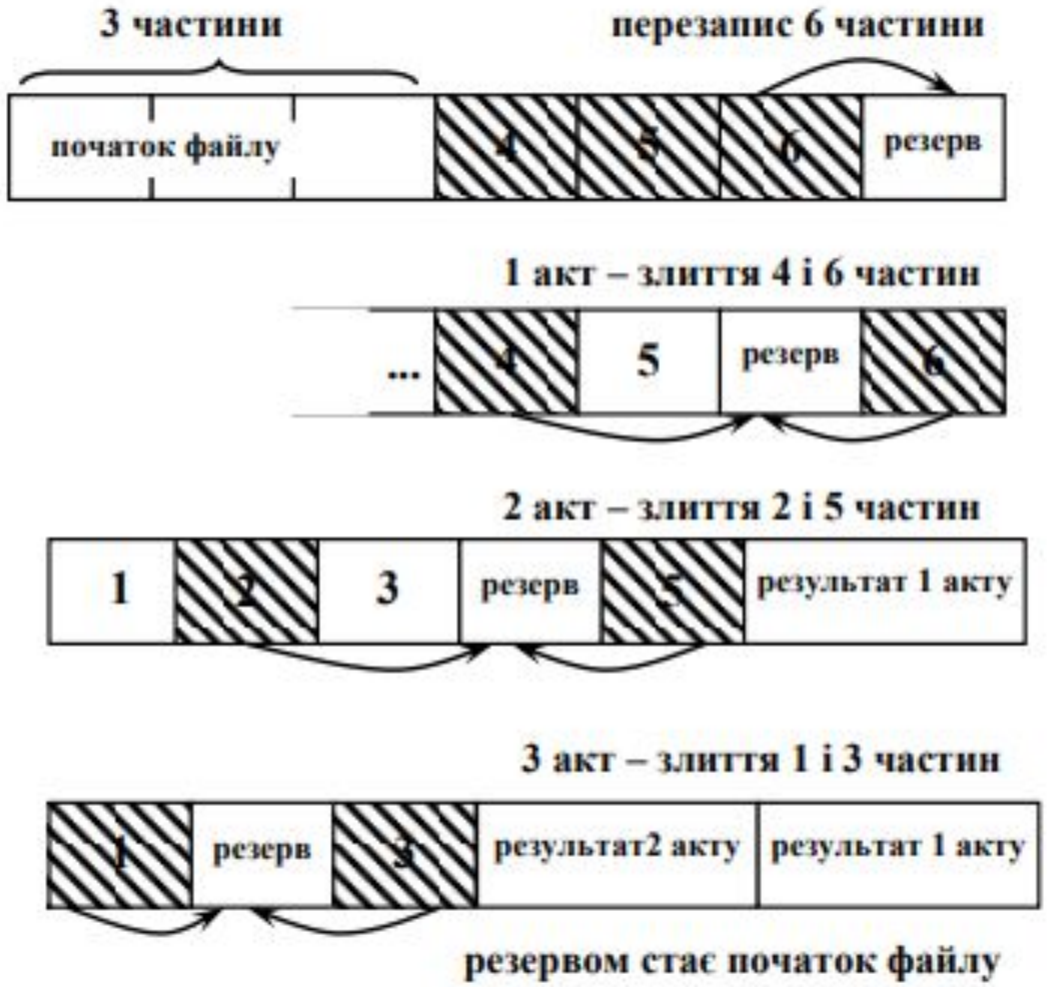


Рисунок 7.8 – Етап злиття (з підетапами)

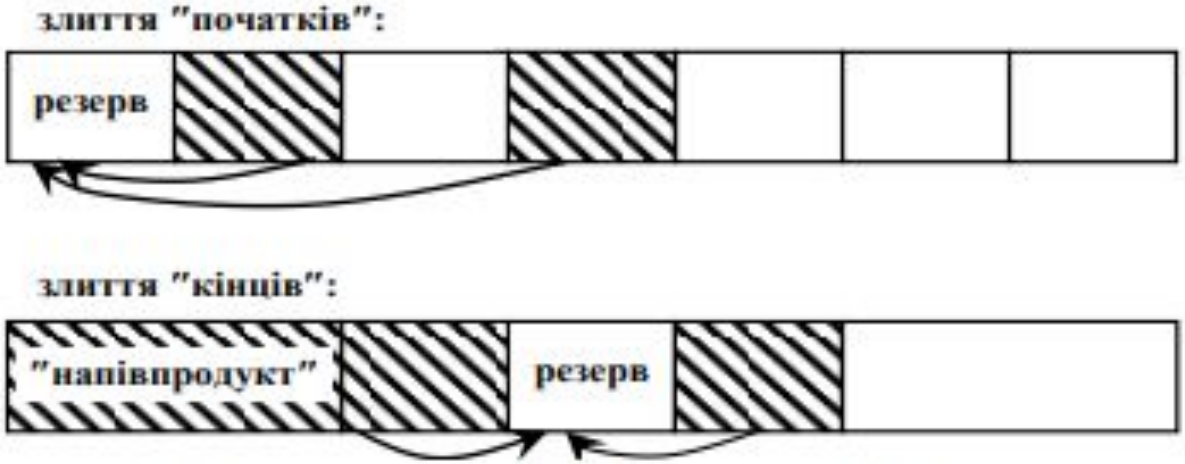


Рисунок 7.9 – Етап злиття "по половинах" (1-ша частина)

Рисунок 7.9 – Етап злиття "по половинах" (2-га частина)

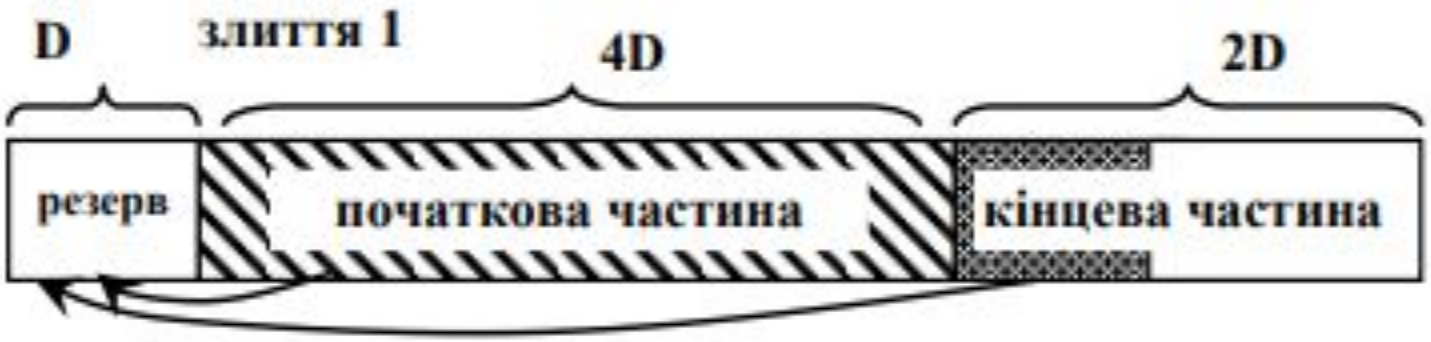


Рисунок 7.10 – Заключний етап злиття

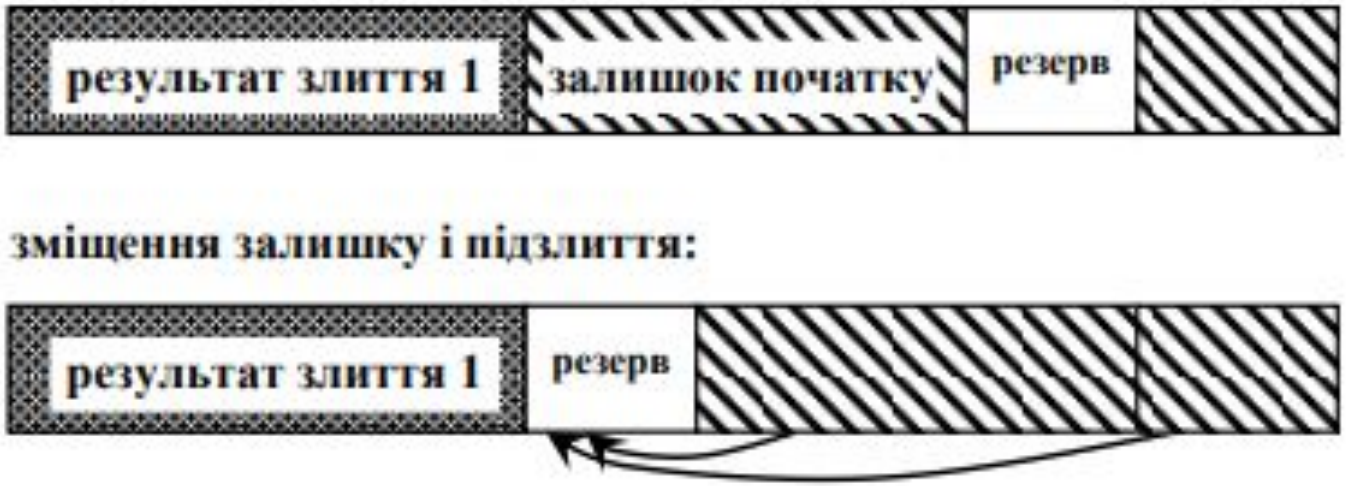


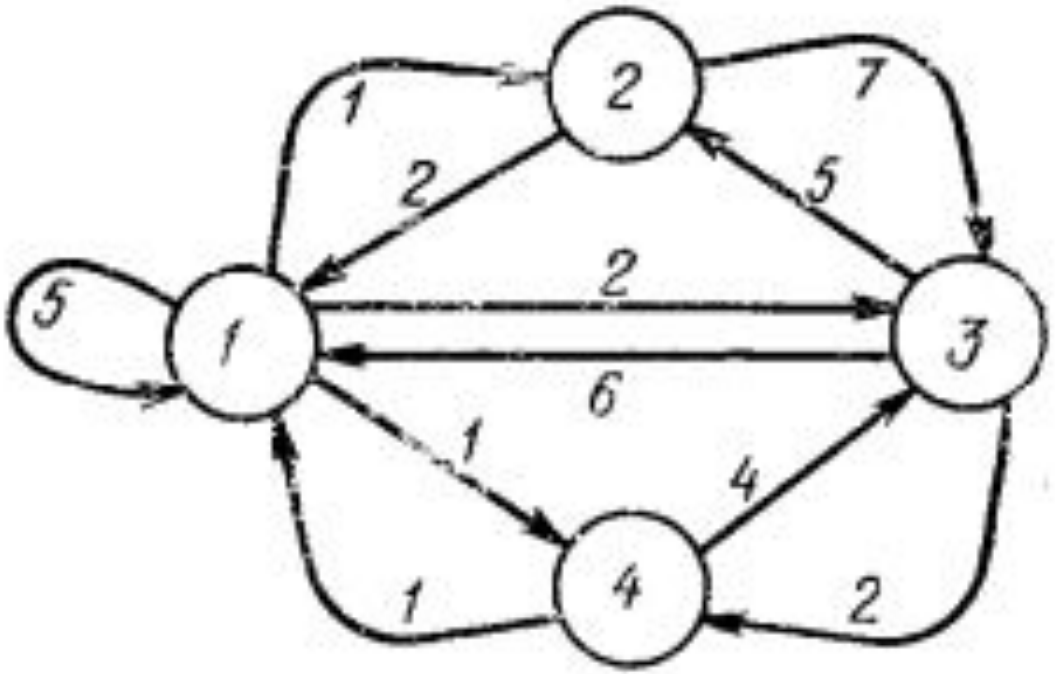
Рисунок 7.11 – Підзлиття

Алгоритм Флойда

Шаг 1. Перенумеровать вершины исходного графа целыми числами от 1 до N. Определить матрицу D^0, задав величину каждого ее элемента (i, j) равной длине кратчайшей дуги, соединяющей вершину i с вершиной j. Если в исходном графе указанные вершины не соединяются дугами, положить $d_{ij}^0 = \infty$. Кроме того, для всех i положить $d_{ij}^0 = 0$.

Шаг 2. Для целого m, последовательно принимающего значения 1, 2, ..., N, определить по величинам элементов матрицы D^{m-1} величины элементов матрицы D^m, используя рекурсивное соотношение (3), т. е. соотношение $d_{ij}^m = \min\{d_{im}^{m-1} + d_{mj}^{m-1}, d_{ij}^{m-1}\}$. При определении величины каждого элемента матрицы D^m фиксировать соответствующий кратчайший путь.

По окончании данной процедуры величина элемента (i, j) матрицы D^N определяет длину кратчайшего пути, ведущего из вершины i в вершину j.



	1	2	3	4
1	0	1	2	1
2	2	0	7	н
3	6	5	0	2
4	1	н	4	0

$$a^1_{ij} = \min \left\{ a^0_{i1} + a^0_{1j},\ a^0_{1j} \right\}$$

1. Занести обчислення значення з прикладу до матр. D1- матриця відстаней

2. Занести шляхи до матриці шляхів.

	1	2	3	4
1	(1,1)			
2				
3	(2,1) (1,3)			
4				

3. Обрахувати матрицю D2 і матр. шляхів

D3 ij

Перенести значення з обрахунків у матрицю D1 та використати її для обрахунків матриці D2

d2ij=min (d1 i2 + d1 2j, d1 ij)

	1	2	3	4
1	0	1	2	1
2	2	0	7	н
3	6	5	0	2
4	1	н	4	0

$$d^1_{ij} = \min \{ d^0_{i1} + d^0_{1j},\ d^0_{1j} \}$$

d2 24=min (0 d1 22 + 3 d1 24, 3 d1 24) = 3 (2,4)

$d^1_{11} = d^0_{11} = 0$

$d^1_{12} = d^0_{12} = 1$ (1,2)

$d^1_{13} = d^0_{13} = 2$ (1,3)

$d^1_{14} = d^0_{14} = 1$ (1,4)

$d^1_{21} = d^0_{21} = 2$ (2,1)

$d_{22} = 0$

$d^1_{23} = \min \{ d^0_{21} + d^0_{13},\ d^0_{23} \} = \min \{2 + 2, 7\} = 4$ (2,1), (1,3)

$d^1_{24} = \min \{ d^0_{21} + d^0_{14},\ d^0_{24} \} = \min \{2 + 1, \infty\} = 3$ (2,1), (1,4)

$d^1_{31} = d^0_{31} = 6$ (3,1)

$d^1_{32} = \min \{ d^0_{31} + d^0_{12},\ d^0_{32} \} = \min \{6 + 1, 5\} = 5$ (3,2)

$d^1_{33} = 0$

$d^1_{34} = \min \{ d^0_{31} + d^0_{14},\ d^0_{34} \} = \min \{6 + 1, 2\} = 2$ (3,4)

$d^1_{41} = d^0_{41} = 1$ (4,1)

$d^1_{42} = \min \{ d^0_{41} + d^0_{12},\ d^0_{42} \} = \min \{1 + 1, \infty\} = 2$ (4,1), (1,2)

$d^1_{43} = \min \{ d^0_{41} + d^0_{13},\ d^0_{43} \} = \min \{1 + 2, 4\} = 3$ (4,1), (1,3)

$d^1_{44} = 0$

$$D^2 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{bmatrix}$$

Кратчайшие пути для элементов матрицы D^2:

$$\begin{bmatrix} & (1,2) & (1,3) & (1,4) \\ (2,1) & & (2,1),\ (1,3) & (2,1),\ (1,4) \\ (3,1) & (3,2) & & (3,4) \\ (4,1) & (4,1),\ (1,2) & (4,1),\ (1,3) & \end{bmatrix}$$

Матрица D^3:

$$D^3 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{bmatrix}$$

Кратчайшие пути для элементов матрицы D^3:

$$\begin{bmatrix} & (1,2) & (1,3) & (1,4) \\ (2,1) & & (2,1)\ (1,3) & (2,1),\ (1,4) \\ (3,1) & (3,2) & & (3,4) \\ (4,1) & (4,1),\ (1,2) & (4,1),\ (1,3) & \end{bmatrix}$$

Матрица D^4:

$$D^4 = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 3 & 4 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{bmatrix}$$

Кратчайшие пути для элементов матрицы D^4

$$\begin{bmatrix} & (1,2) & (1,3) & (1,4) \\ (2,1) & & (2,1),\ (1,3) & (2,1),\ (\\ (3,4),\ (4,1) & (3,4),\ (4,1),\ (1,2) & & (3,4) \\ (4,1) & (4,1),\ (1,2) & (4,1),\ (1,3) & \end{bmatrix}$$

Підприємство имеет возможность приобрести не более 19 трехтонных автомашин и не более 17 пятитонных. Отпускная цена трехтонного грузовика - 4000 руб., пятитонного - 5000 руб. Колхоз может выделить для приобретения автомашин 141 тысяч рублей. **Сколько нужно приобрести автомашин**, чтобы их суммарная грузоподъемность была максимальной? Задачу решить графическими и аналитическими методами.

1. Задачу не переписуємо, але
2. Виписуємо у зошит дані задачі. Потрібно закупити : 19-3т
   17 – 5т
   3т-4000
   5т-5000
   всього 141000
3. Будуємо **математичну модель задачі. Через невідомі X1 та X2** позначаємо кількість трьохтонних та п’ятитонних автомобілів

3.1 Будуємо обмежання Кількість 0<= X1<=19, 0<= X2<=17

Вартість X1*4000 + X2*5000 <=141000 >

$f = 3x1 + 5\ x2 \rightarrow max$ (1)
обмежання

1. $4000x1 + 5000\ x2 \leq 141000$, (*) ------- будуємо лінію, що відповідає цій нерівності
2. $0 \leq x1 \leq 19$, ------- будуємо лінію, що відповідає цій нерівності

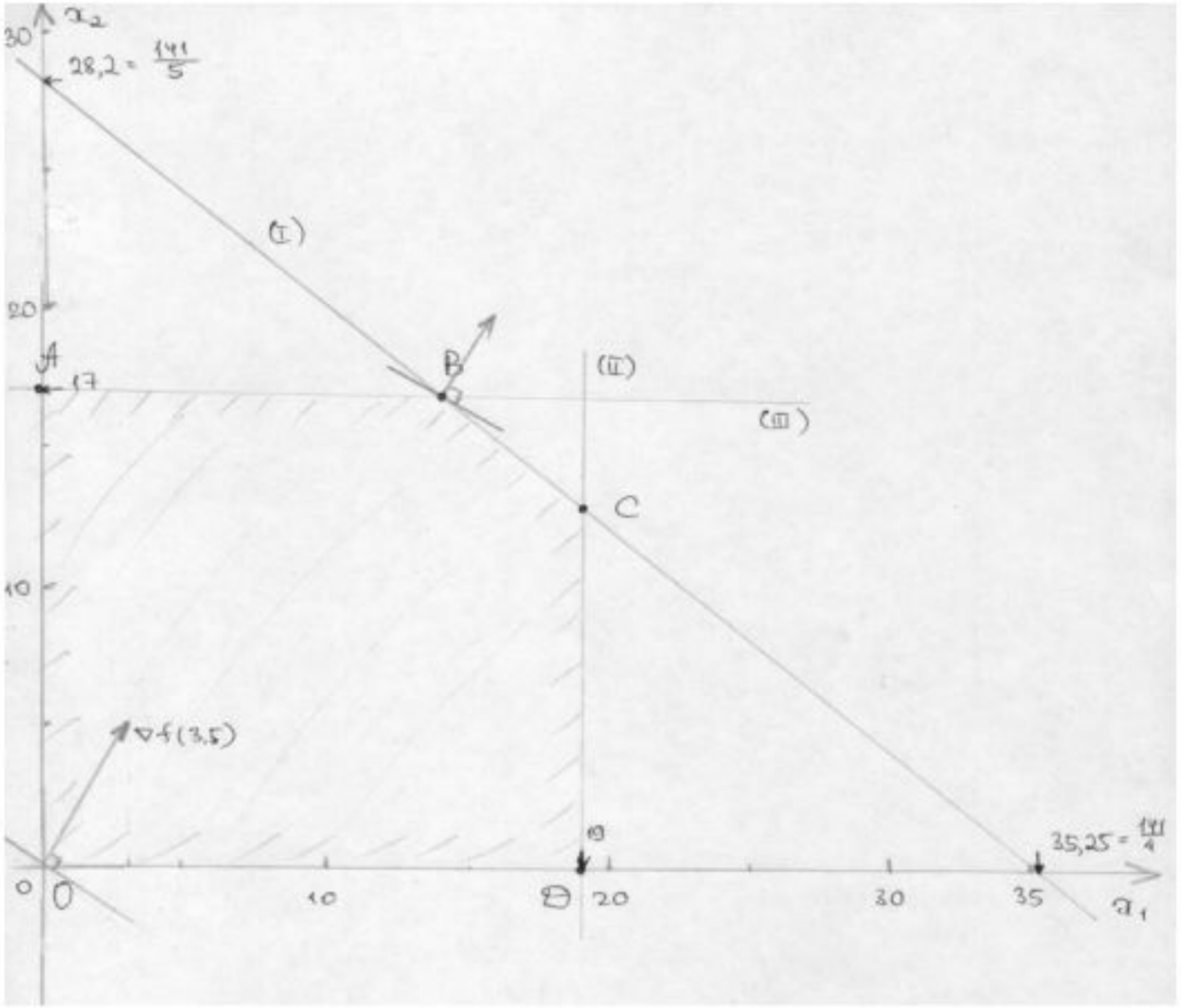
3. $0 \leq x2 \leq 17$. ------- будуємо лінію, що відповідає цій нерівності

Всі обмеження складають разом багатокутник розв'язків
Площина OX1X2

1. $4000x1 + 5000\ x2 \leq 141000$
   X1=0, тоді X2= 141000/5000 рахуємо Точка A (0, 141000/5000)
   X2=0, тоді X1= 141000/4000 рахуємо Точка B (141000/4000, 0)

Таким отримаємо координати двох точок прямої L1: A, B

Прийнято розв'язувати задачі лінійного програмування відповідними методами. Наприклад графічно, симплекс-метод.


x_2
30
$28{,}2 = \frac{141}{5}$
20
10
0
(I)
(II)
(III)
A
B
C
17
$\nabla f(3{,}5)$
0
10
20
30
35
$35{,}25 = \frac{141}{4}$
x_1

$$z = c_1 x_1 + c_2 x_2 \to \max,$$

$$\begin{cases} a_{11} x_1 + a_{12} x_2 \le b_1, \\ a_{21} x_1 + a_{22} x_2 \le b_2, \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ a_{m1} x_1 + a_{m2} x_2 \le b_m, \end{cases}$$

$$x_j \ge 0, \; j = 1,2.$$

$$z = 3x_1 + 2x_2 + x_3 = 3x_1 + 2x_2 + 4 - x_1 = 2x_1 + 2x_2 + 4 \to \max .$$

Так как $x_3 \geq 0$, то $4 - x_1 \geq 0$, что равносильно неравенству $x_1 \leq 4$. Следовательно, система ограничений примет следующий симметричный вид:

$$\begin{cases} x_1 + 2x_2 \leq 12, \\ x_1 \leq 4, \\ 2x_1 + 2x_2 \leq 14, \end{cases}$$
$$x_1 \geq 0,\ x_2 \geq 0.$$

Полученную двумерную задачу решим обычным графическим способом.

1. Так как $x_1 \geq 0$, $x_2 \geq 0$, то область допустимых решений будет находиться в первой координатной четверти. На плоскости Ox_1x_2 (рис. 1.1) построим прямые, порождаемые уравнениями:

$$\begin{cases} x_1 + 2x_2 = 12 \Leftrightarrow \dfrac{x_1}{12} + \dfrac{x_2}{6} = 1 & - \text{ прямая } l_1, \\ x_1 = 4 \Leftrightarrow \dfrac{x_1}{4} = 1 & - \text{ прямая } l_2, \\ 2x_1 + 2x_2 = 14 \Leftrightarrow \dfrac{x_1}{7} + \dfrac{x_2}{7} = 1 & - \text{ прямая } l_3. \end{cases}$$

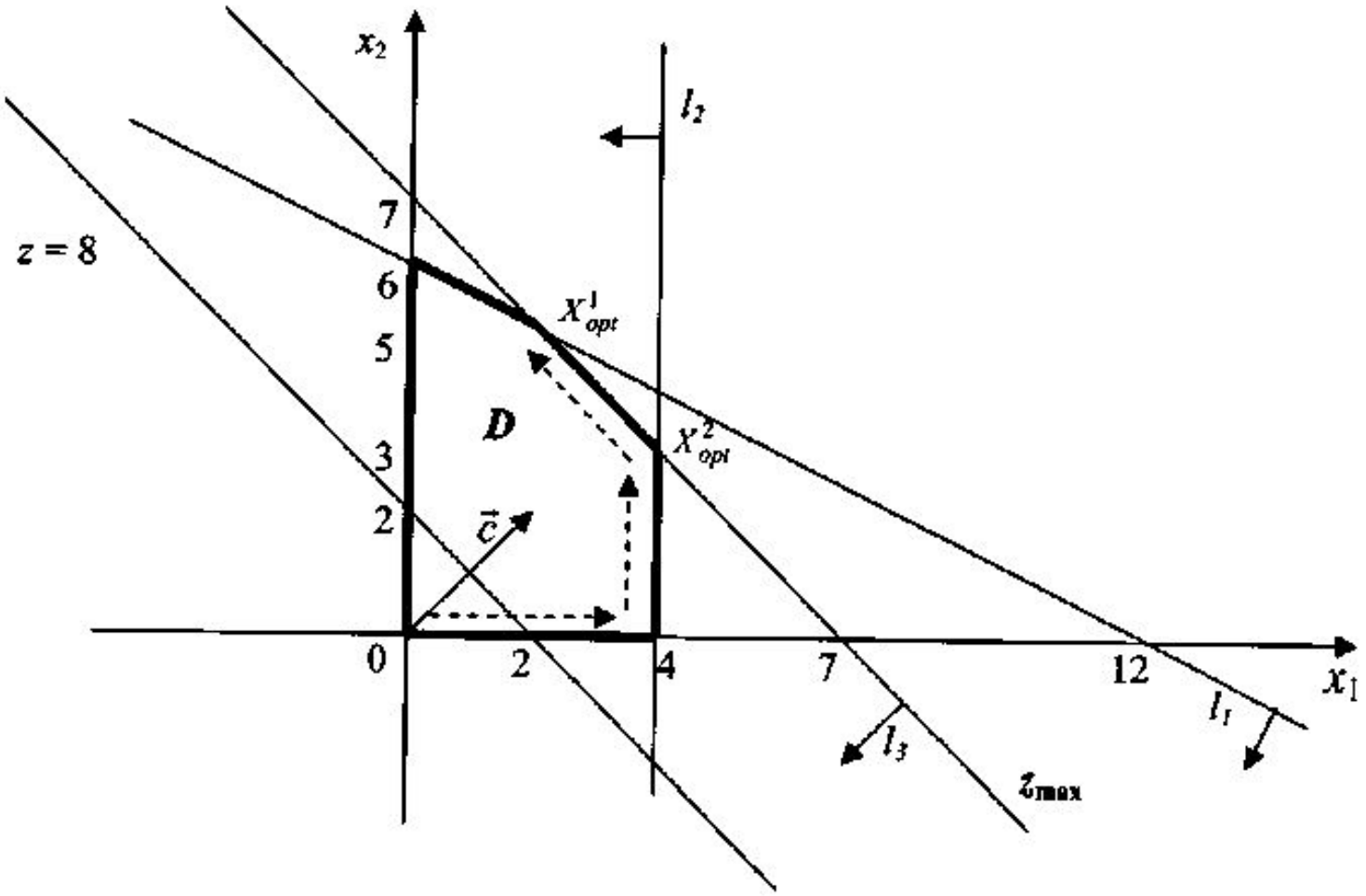


Рис. 1.1

1.4.2. *Геометрическая интерпретация целевой функции*

1. Уравнение $z = c_1 x_1 + c_2 x_2$ при фиксированном значении $z = z_0$ определяет на плоскости Ox_1x_2 прямую $z_0 = c_1 x_1 + c_2 x_2$. При изменении z получают семейство параллельных прямых, называемых *линиями уровня.*

2. Вектор коэффициентов целевой функции $\vec{c} = (c_1;\ \ c_2)$ называется *градиентом функции.* Он перпендикулярен линиям уровня.

3. Градиент функции $\vec{c} = (c_1;\ \ c_2)$ показывает направление наибольшего возрастания целевой функции.

4. Антиградиент $-\vec{c} = (-c_1;\ \ -c_2)$ показывает направление наибольшего убывания целевой функции.

1.4.3. *Графическое решение задач линейного программирования*

Суть графического метода решения задач ЛП основывается на следующих утверждениях:

1) совокупность опорных планов задачи ЛП совпадает с системой вершин многогранника решений;

2) целевая функция достигает оптимального значения в вершине многогранника решений.

Для практического решения задачи (1.13) – (1.15) необходимо:

1) построить с учетом системы ограничений область допустимых решений D (многогранник планов);

2) построить вектор градиента $\vec{c}$;

3) построить перпендикулярно к нему в области допустимых решений одну из прямых семейства $z = \text{const}$;

4) искомая точка экстремума X_{opt} найдется параллельным перемещением вспомогательной прямой $z = \text{const}$ в направлении вектора $\vec{c}$ (если ищется $z_{\max}$) и в направлении вектора $-\vec{c}$ (если ищется $z_{\min}$);

5) координаты точки X_{opt} можно определить, решив совместно уравнения прямых, пересекающихся в этой точке, или по чертежу.

$$z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \to \max \quad \text{(целевая функция)}, \qquad (1.1)$$

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n = b_1 \\ a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n = b_m \end{cases} \quad \text{(система ограничений)}, \qquad (1.2)$$

$$x_j \geq 0, \quad j = \overline{1, n} \quad \text{(ограничения на переменные)}. \qquad (1.3)$$

Здесь $A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$ – матрица коэффициентов системы ограничений;

$C = (c_1, c_2, \ldots, c_n)$ – матрица-строка коэффициентов целевой функции;

$B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$ – матрица-столбец свободных членов;

$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ – матрица-столбец неизвестных.

Тогда каноническую форму записи задачи ЛП (1.1)–(1.3) можно представить в следующем матричном виде, эквивалентном первоначальному:

$$Z = CX \to \max, \qquad (1.4)$$

$$AX = B, \qquad (1.5)$$

1.5.1. *Этапы решения задачи ЛП симплекс-методом*

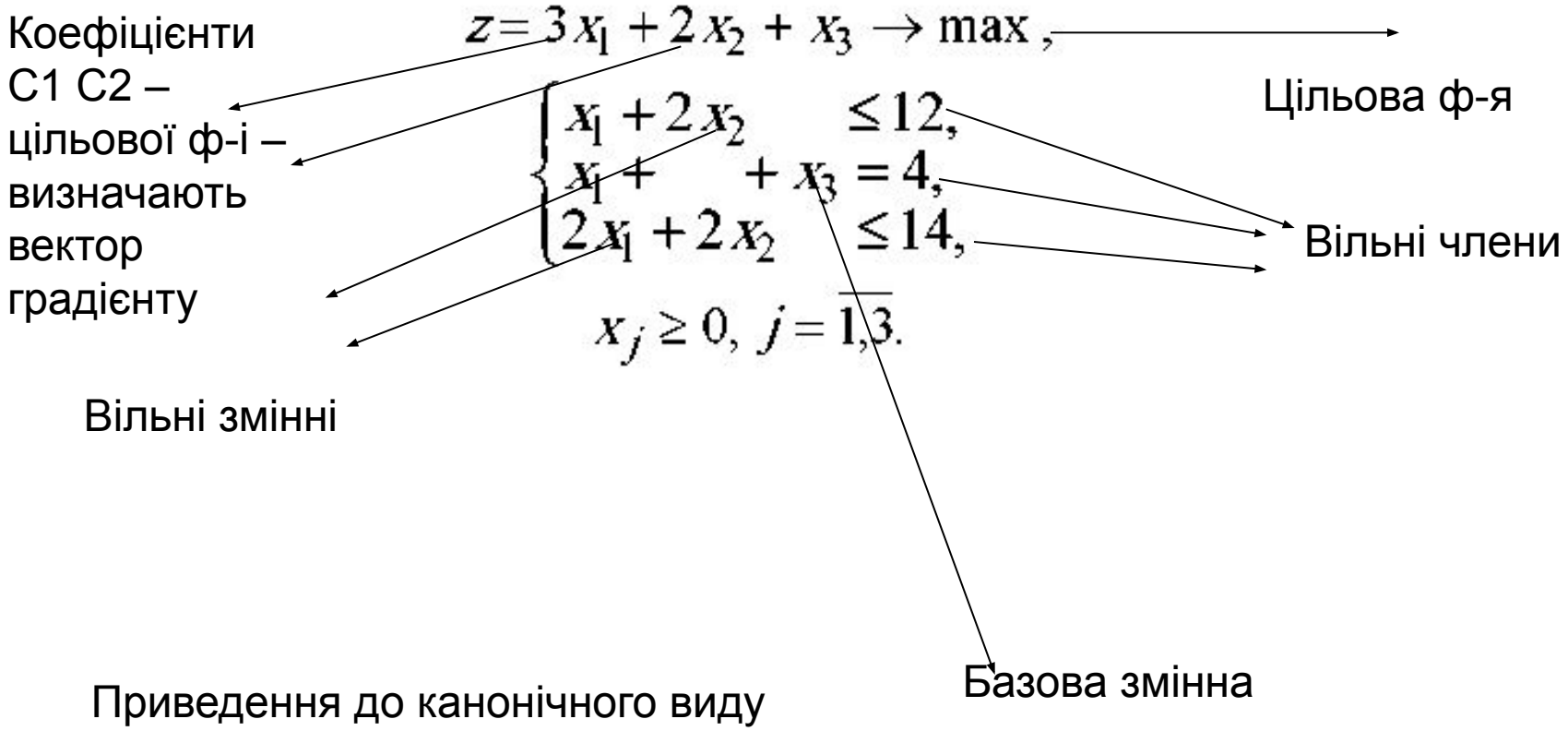
Решение задачи ЛП складывается из нескольких этапов:

1. Задача должна быть приведена к каноническому виду, притом все элементы столбца свободных членов должны быть неотрицательными.

2. Найден начальный опорный план задачи.

3. Целевая функция выражена через свободные переменные и максимизирована.

4. По симплексному методу находится оптимальный план задачи.



Приведення до канонічного виду

1. Всі нерівності замінюються на рівності шляхом додавання базових змінних: якщо менше і дорівнюються , то + x4, якщо більше, то віднімається +x5

Базова – це змынна яка входить тільки 1 раз тільки в одну нерівнсть з коефіцієнтом 1

2. Цільова ф-я має прямувати до max, якщо до min, то беремо –z ---- max

1. Приведемо задачу лінійного програмування до канонічного виду

$$z = 3x_1 + 2x_2 + x_3 \to \max,$$

$$\begin{cases} x_1 + 2x_2 + x_4 = 12, \\ x_1 + x_3 = 4, \\ 2x_1 + 2x_2 + x_5 = 14, \end{cases}$$

$$x_j \geq 0,\ j = \overline{1,5};$$

2. Вільні змінні прирівнюємо до 0, і визначаємо базові змінні

X1, X2 =0

X4=12 x3=4

X5=14

значення
ії для
начень х

ої

ову ф-ю

2) найден начальный опорный план задачи (см. пример 3):

$$X_0 = (0; 0; 4; 12; 14),\ z(X_0) = 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 4 = 4;$$

3) целевая функция выражена через свободные переменные и максимизирована (см. пример 4):

$$z = 2x_1 + 2x_2 + 4 \to \max.$$

Занесем коэффициенты целевой функции и системы ограничений в симплексную таблицу следующим образом:

- 1-е ограничение $x_1 + 2x_2 + \underline{x_4} = 12$ – в 1-ю строку:

а) в базисный столбец «Б» – базисную переменную x_4;

б) в столбец значений (базисной переменной) «З» – значение свободного члена, равное 12;

в) в столбцы коэффициентов « x_j» – коэффициенты при x_j, равные 1, 2, 0, 1, 0 соответственно;

- 2-е ограничение – во 2-ю строку (аналогично);
- 3-е ограничение – в 3-ю строку (аналогично);
- целевую функцию – в z-строку:

а) в столбец значений (целевой функции) «З» – константу со своим знаком, т.е. 4;

б) в столбцы коэффициентов « x_j» – коэффициенты при x_j с противоположными знаками, равные –2, –2, 0, 0, 0 соответственно.

Составим исходную симплекс-таблицу 1.

5. Заносимо задачу лінійного програмування до таблиці

2 етап розвя'зку задач симплекс методом

Базисні
змінні x4
x3,, x5

Симплекс-таблиця 1

Б	З	x_1 ↓	x_2	x_3	x_4	x_5	
x_4	12	1	2	0	1	0	12/1
←x_3	4	1	0	1	0	0	4/1
x_5	14	2	2	0	0	1	14/2
z	4	–2	–2	0	0	0	**4/-2**

мож

6. Визначаємо ведучий стовпець

Вибираємо max "–" у z стрічці

8. Визначаємо **ведучий елемент**

На перетині ведучої стрічки і ведучого стовпця

9. Ведучий елемент роблять рівним одиниці шляхо
ділення ведучої стрічки на ведучий елемент

7. Визначаємо **симплекс-відношення** – відношення ствпця значень до відповідних **додатніх** елементів ведучого стовпця.
Визначаэмо ведучу стрічку

Min 12/1 4/1 14/2

10. Позначаємо x3 як змінну, що виходить із базису

11. Всі значення ведучого стовпця окрім ведучого елементу роблять рівними нулю та застосовуємо метод ЖОРДАНА-ГАУССА

3 етап розвя'зку задач симплекс методом

Симплекс-таблиця 1

Б	З	$x_1\downarrow$	x_2	x_3	x_4	x_5	
x_4	12	1	2	0	1	0	12/1
$\leftarrow x_3$	4	1	0	1	0	0	4/1
x_5	14	2	2	0	0	1	14/2
z	4	−2	−2	0	0	0	4/-2

11. Всі значення ведучого стовпця окрім ведучого елементу роблять рівними нулю та застосовуємо метод **ЖОРДАНА-ГАУССА**

Для цього ведучу стрічку уявно множать на значення, яке б при додаванні із поточним значенням у ведучому стовпці давало 0.

Потім додаємо відповідні значення у поточній стрічці і ведучій. Результат записуємо у поточну стрічку

$$\min\{12/1;\ 4/1;\ 14/2\} = 4.$$

Проведем одну итерацию метода замещения (базисных элементов) Жордана–Гаусса. Столбцы «x_4» и «x_5» останутся базисными и в симплекс-таблице 2, а столбец «x_1» следует сделать «единичным». Новые данные в симплекс-таблицу 2 заносим по следующему алгоритму:

1. Ведущий элемент делают равным 1. Для этого ведущую строку делят на ведущий элемент. В нашем случае ведущий элемент равен 1. Значит, ведущая строка останется прежней. Перепишем ее в симплекс-таблицу 2 и назовем строкой, полученной из ведущей.

2. Остальные элементы ведущего столбца делают нулевыми.

- Чтобы в 1-й строке вместо 1 получить 0, необходимо каждый элемент сроки, полученной из ведущей, умножить на (–1) и прибавить почленно к 1-й строке. (Проще говоря, строку, полученную из ведущей, умножить на (–1) и прибавить к 1-й строке.)
- Чтобы в 3-й строке вместо 2 получить 0, необходимо строку, полученную из ведущей, умножить на (–2) и прибавить к 3-й строке.

• Чтобы в z-строке вместо (–2) получить 0, необходимо строку, полученную из ведущей, умножить на 2 и прибавить к z-строке.

Симплекс-таблица 2

Б	З	x_1	$x_2\downarrow$	x_3	x_4	x_5	
x_4	8 2	0	0 2	–1 1	1	-1 0	8/2
x_1	4	1	0 0	1	0	0	
$\leftarrow x_5$	6 3 -6	0	1 2 -2	–2 -1 2	0	-1 1 1/2	6/2=3 - min
z	12 18	0	0 –2	2 0	0	1 0	Не обчислюэться

Таблицы пересчитывают до тех пор, пока в z-строке все элементы (не считая значения) станут неотрицательными.

Симплекс-таблица 3

Б	З	x_1	x_2	$x_3\downarrow$	x_4	x_5
$\leftarrow x_4$	2	0	0	1	1	–1
x_1	4	1	0	1	0	0
x_2	3	0	1	–1	0	0,5
z	18	0	0	0	0	1

Так как в z-строке симплекс-таблицы 3 все элементы больше или равны нулю, то найден оптимальный план:

$$X^1_{opt} = (4; 3; 0; 2; 0),\ z_{\max} = z(X^1_{opt}) = 3 \cdot 4 + 2 \cdot 3 + 1 \cdot 0 = 18.$$

Он не единственный, так как существует нулевой элемент z-строки, соответствующий свободной переменной x_3. (Решение единственное, если нули в z-строке соответствуют только базисным переменным.)

Чтобы найти второй оптимальный план, столбец «x_3» принимают за ведущий и находят минимальное симплексное отношение: $\min\{2/1;\ 4/1\} = 2$. Тогда 1-я строка станет ведущей. Пересчитывают

симплекс-таблицу 3 методом замещения Жордана–Гаусса с ведущим элементом $a_{13} = 1$ и заносим в симплекс-таблицу 4.

Симплекс-таблица 4

Б	З	x_1	x_2	x_3	x_4	x_5
x_3	2	0	0	1	1	–1
x_1	2	1	0	0	–1	1
x_2	5	0	1	0	1	–0,5
z	18	0	0	0	0	1

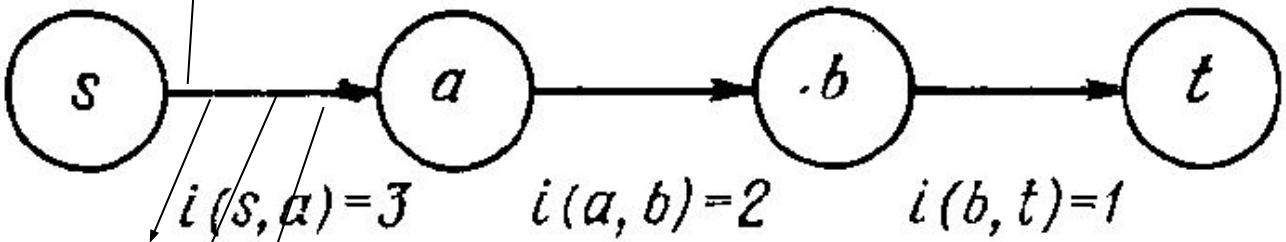
Из последней таблицы $X^2_{opt} = (2;\ 5;\ 2;\ 0;\ 0)$, а значение целевой функции $z_{\max} = 18$.

Общее решение записывается как выпуклая линейная комбинация решений X^1_{opt} и X^2_{opt}, т.е. $X_{opt} = \lambda_1 X^1_{opt} + \lambda_2 X^2_{opt}$, где $\lambda_1 + \lambda_2 = 1,\ \lambda_1 > 0,\ \lambda_2 > 0.$

уменьшен поток в дуге (x, y). Очевидно, $i(x, y) = c(x, y) - f(x, y)$ и $r(x, y) = f(x, y)$.

1. Пошук максимального потоку по збільшуючих дугах

C – максимальна пропускна здатність – макс. кільк (припустимо c=4 c(s,a)=4)



f – кількість одиниць потоку, що проходить по ній у теперішній час

(припустимо f=1 f(s,a)=**1**)

i – кількість одиниць потоку, на яку потік може бути збільшений (i=c-f=4-1=3) → Збільшуюча дуга

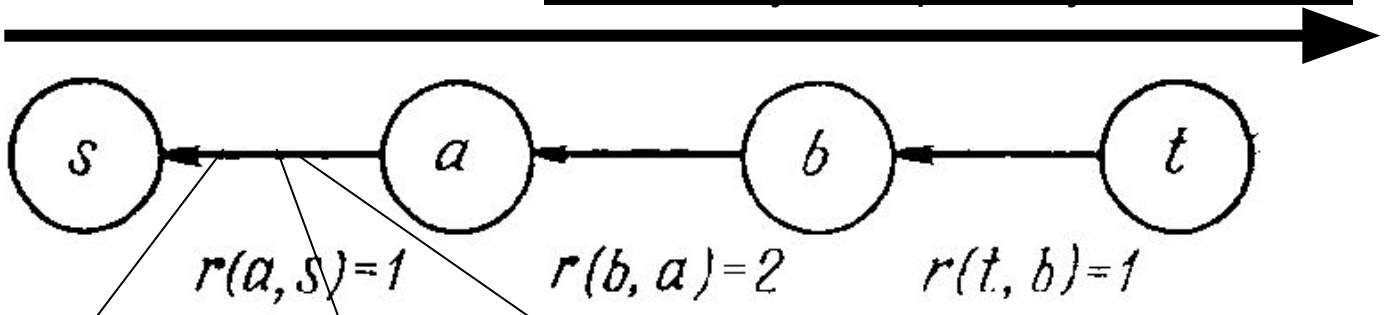
i(s,a)=3

r – кількість одиниць потоку, на яку потік може бути зменшений (r=f=1) r(s,a)=1 → Зменшуюча дуга

$$\min\{i(s, a), i(a, b), i(b, t)\} = \min\{3, 2, 1\} = 1.$$

2. Пошук максимального потоку по зменшуючих дугах

Max потоку, використовуючи тільки r

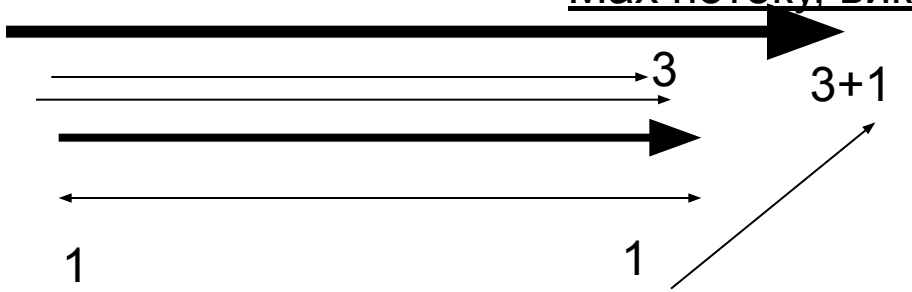


Зменшуюча дуга має зворотнІй напрямок (a,s) **r (a,s)=1**

r= f **(s,a)=1**

Збільшуюча має прямий напрямок **i(s,a)=3**

Max потоку, використовуючи тільки r

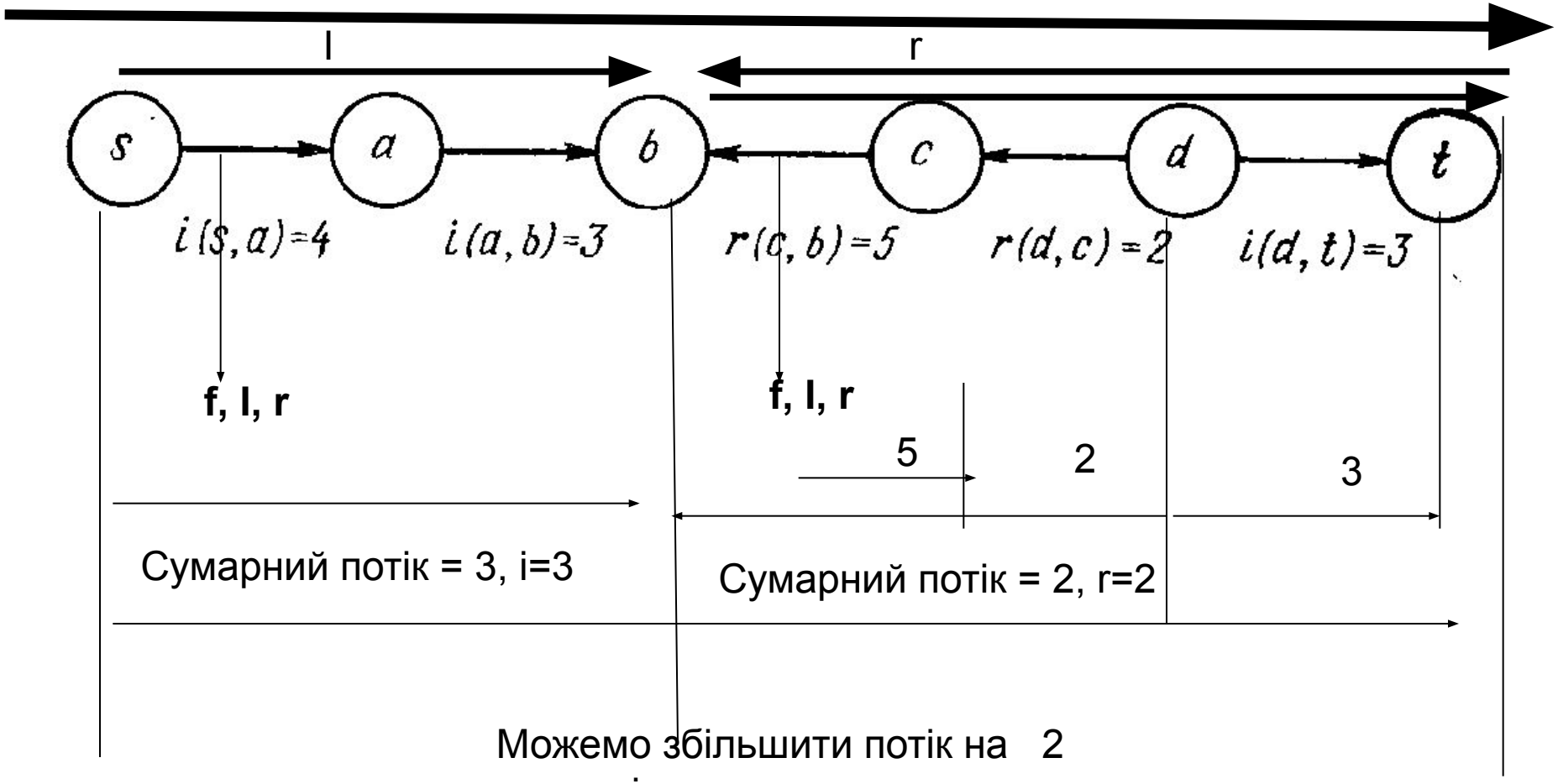


3+1

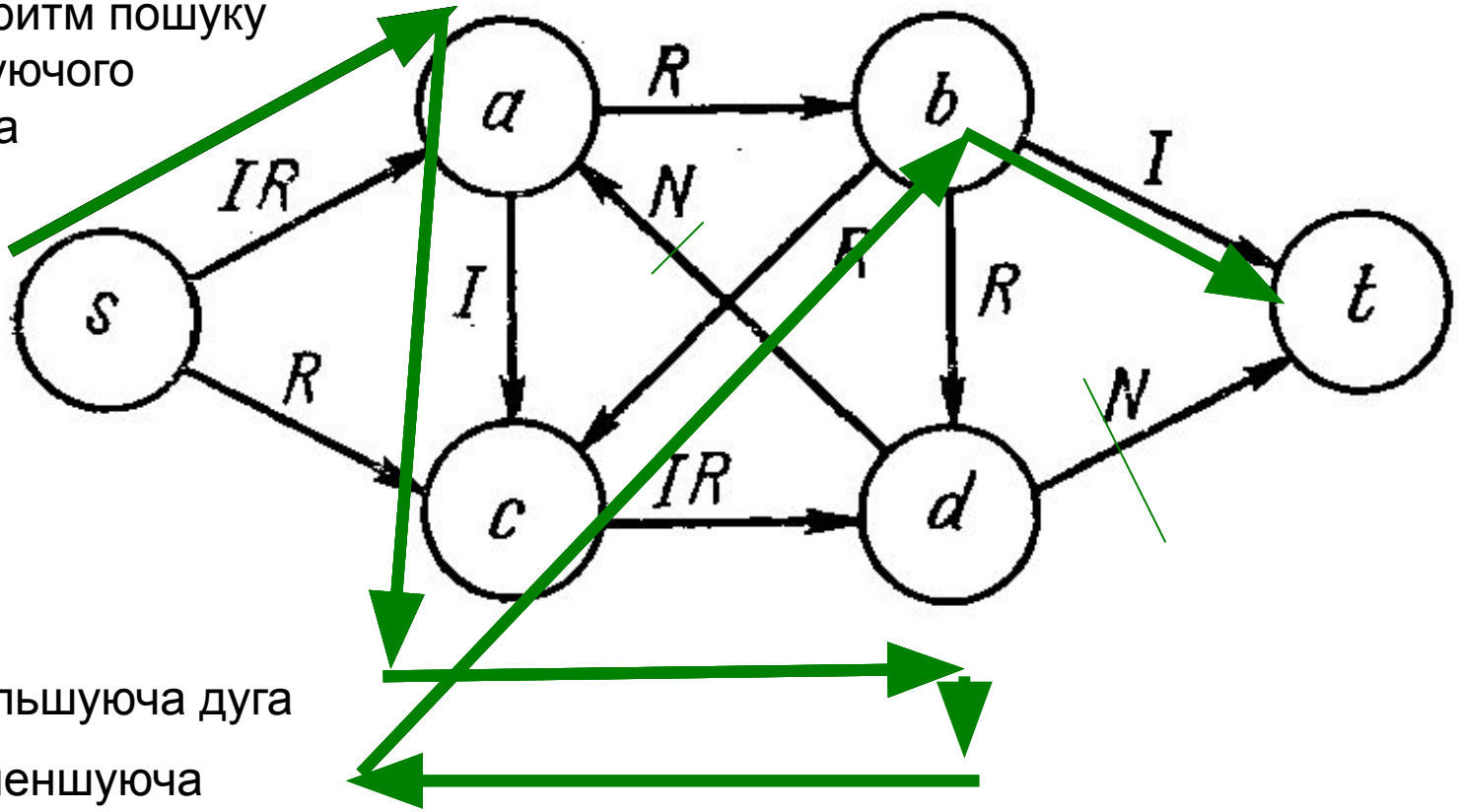
$$\min\{r(t, b),\ r(b, a),\ r(a, s)\} = \min\{1, 2, 1\} = 1.$$

3. Пошук максимального потоку по змішаних дугах

Max потоку, використовуючи r та i



4. Алгоритм пошуку збільшуючого ланцюга

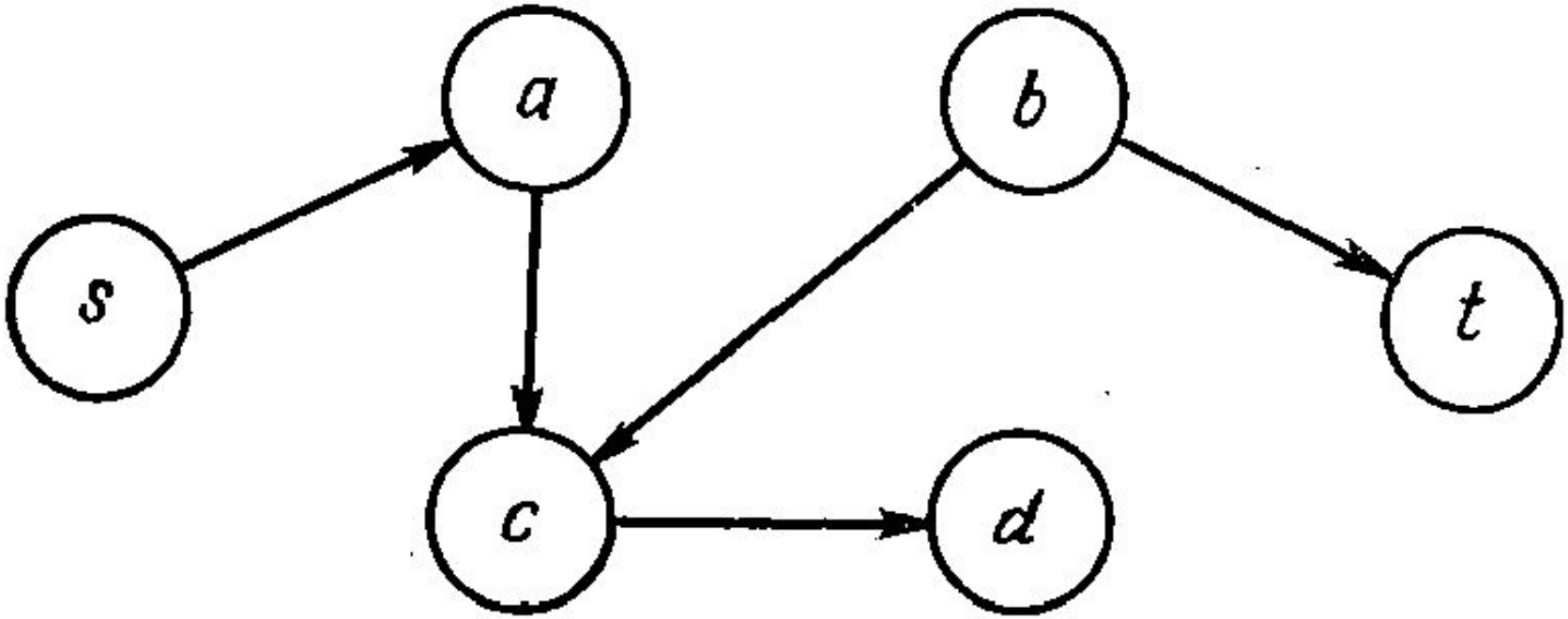


I- збільшуюча дуга

R- зменшуюча

I, R – дуга для збільшення потоку і для зменшення потоку

N - не збільшуюча і не зменшуюча дуга

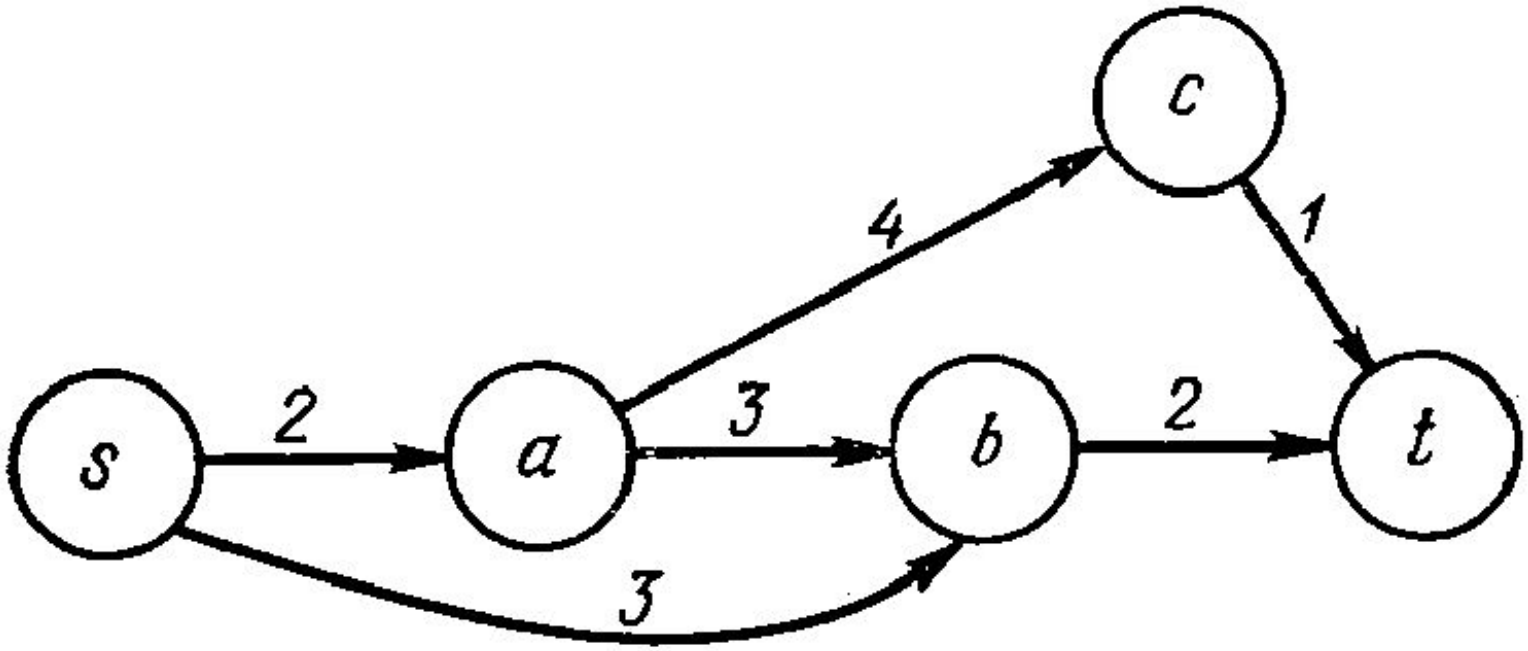

a
b
s
t
c
d

Алгоритм поиска максимального потока

Шаг 1. Выбрать любой начальный поток из вершины s (источник) в вершину t (сток), т. е. любой набор величин $f(x, y)$, удовлетворяющий соотношениям (1)—(4). Если ни один из начальных потоков из s в t неизвестен, задать начальный поток, положив для всех дуг (x, y) $f(x, y) = 0$.

Шаг 2. Для всех дуг (x, y) проделать следующее. Если $f(x, y) < c(x, y)$, то положить $i(x, y) = c(x, y) - f(x, y)$ и считать дугу (x, y) принадлежащей множеству I. Если $f(x, y) > 0$, то положить $r(x, y) = f(x, y)$ и считать дугу (x, y) принадлежащей множеству R.

Шаг 3. На множествах I и R, сформированных на предыдущем шаге, применить к исходному графу алгоритм поиска увеличивающей цепи. Если при этом увеличивающую поток цепь найти не удается, закончить процедуру алгоритма: текущий поток является максимальным. В противном случае осуществить максимально возможное увеличение потока вдоль найденной увеличивающей цепи. Затем возвратиться к шагу 2.


c
4
1
s
2
a
3
b
2
t
3