Ін. М. Грод, С. В. Мартинюк, О. М. Мартинюк

АНАЛІЗ ЕФЕКТИВНОСТІ ДЕЯКИХ АЛГОРИТМІВ

ТЕОРІЯ І ПРАКТИКА

Навчальний посібник

Тернопіль — 2017

УДК 004.942
Г 86

Рецензенти:

В. З. Чорний — завідувач кафедри математики та методики її навчання Тернопільського національного педагогічного університету імені Володимира Гнатюка, доцент, кандидат фізико-математичних наук

В. О. Єрьоменко — доцент кафедри економіко-математичних методів Тернопільського національного економічного університету, кандидат фізико-математичних наук

Грод Ін. М.

Г 86 Аналіз ефективності деяких алгоритмів. Теорія і практика : навчальний посібник / Ін. М. Грод, С. В. Мартинюк, О. М. Мартинюк. — Тернопіль : ТНПУ, 2017. — 64 с.

У посібнику розглянуто ефективність використання деяких класів алгоритмів. Значну увагу приділено питанням побудови й оптимізації математичних моделей.

Для студентів фізико-математичних факультетів.

УДК 004. 942

Поняття алгоритму

У всіх сферах діяльності, зокрема у сфері обробки інформації, людина стикається з різними способами або методиками рішення різноманітних завдань. Вони визначають порядок виконання дій для отримання бажаного результату — ми можемо трактувати це як початкове або інтуїтивне означення алгоритму. Таким чином, алгоритм можна визначити нестрого як послідовність команд для однозначного вирішення завдання. Додаткові вимоги про час виконання алгоритму для будь-яких вхідних даних приводять до наступного неформального визначення алгоритму:

Алгоритм — це заданий на деякій мові опис, який задає скінченну послідовність здійсненних і точно визначених елементарних операцій для вирішення завдання, загального для класу можливих вихідних даних.

Нехай DZ — область (множина) вихідних даних завдання Z, а R — множина можливих результатів. Тоді ми можемо говорити, що алгоритм здійснює відображення DZ в R. Оскільки таке відображення може бути не повним, то в теорії алгоритмів вводять такі поняття.

Алгоритм називають *частковим* алгоритмом, якщо ми отримуємо результат тільки для деяких d з DZ, і *повним* алгоритмом, якщо алгоритм отримує правильний результат для всіх $d \in DZ$.

Незважаючи на зусилля учених, сьогодні відсутнє одне строге визначення поняття «алгоритму». З різноманітних варіантів словесного визначення алгоритму найбільш вдалі, на думку авторів, належать А. М. Колмогорову і А. А. Маркову.

Означення 1 (Колмогоров). *Алгоритм* — це система обчислень, які виконують за строго визначеними правилами, яка після скінченної кількості кроків приводить до вирішення поставленого завдання.

Означення 2 (Марков). *Алгоритм* — це точна вказівка, яка визначає обчислювальний процес, який іде від варійованих вихідних даних до шуканого результату.

Зазначимо, що різні визначення алгоритму, в явній або неявній формі, постулюють такий ряд загальних вимог:

— алгоритм повинен містити скінченну кількість елементарно здійсненних завдань;

— алгоритм повинен привести до очікуваного результату;

— алгоритм повинен бути єдиним для всіх допустимих вихідних даних, тобто задовольняти вимогу універсальності;

— алгоритм повинен призводити до правильного відносно поставленого завдання рішенням, тобто задовольняти вимогу правильності.

Хоча загальне формальне визначення алгоритму дати неможливо, можна формально визначити той або інший ***клас алгоритмів***. Наприклад, визначаючи мову програмування, ми визначаємо клас алгоритмів, які можна представити у вигляді програм на цій мові.

Аналізуючи алгоритм, можна отримати уявлення про те, скільки часу займе вирішення даної задачі за допомогою даного алгоритму. Наприклад, ми можемо оцінити, скільки порівнянь потрібно алгоритму сортування для упорядкуванні списку з N величин за зростанням, або підрахувати, скільки арифметичних операцій потрібно для множення двох матриць розміром $N \times N$. Одну й ту саму задачу можна вирішити за допомогою різних алгоритмів. Аналіз алгоритмів дає нам інструмент для вибору алгоритму.

Усі алгоритми поділяються на такі, яким достатньо обмеженої пам’яті, і ті, яким потрібно додатковий простір. Нерідко програмістам доводилося вибирати повільніший алгоритм через те, що він обходився наявною пам’яттю і не вимагав зовнішніх пристроїв.

При розгляді питання програмного забезпечення, пропонованого на ринку сьогодні, стає зрозуміло, що подібний аналіз пам’яті проведений не був. Обсяг пам’яті, необхідний навіть для простих програм, обчислюється мегабайтами. Розробники програм, схоже, не відчувають потреби в економії місця, вважаючи, що якщо у користувача недостатньо пам’яті, то він зможе докупити відсутню для виконання програми пам’ять або новий жорсткий

диск для її зберігання. У результаті комп'ютери стають непридатними задовго до того, як вони справді застарівають.

Алгоритм як обчислювальна функція

Довільний алгоритм можна звести до обчислення значення деякої числової функції. І, навпаки, якщо для функції f існує алгоритм, який призводить до стандартного запису значення функції $m = f(n)$, то функцію f називають *алгоритмічно обчислювальною* або просто *обчислювальною*.

Обчислювальна функція — це числова функція $f(x_1, x_2, x_3, .., x_n)$, значення якої можна обчислювати за допомогою деякого алгоритму на підставі відомих значень аргументу.

Складність алгоритмів

Використовуючи алгоритмічні моделі, ми не замислювалися над обмеженням ресурсів:

— чи може певна програма (алгоритм) розміститися в пам'яті ПК;

— чи дасть вона результат за визначений час.

Дослідженням цих питань і займається розділ теорії алгоритмів — аналіз складності алгоритмів.

Складність алгоритмів — кількісна характеристика, яка визначає час, що необхідний для виконання алгоритму (часова складність), і об'єм пам'яті, необхідний для його розміщення (ємнісна складність).

Часова характеристика (фізичний час виконання) складності алгоритму — це величина $\tau \cdot t$, де t — кількість дій алгоритму (елементарних команд), а τ — середній час виконання однієї операції (команди).

Кількість команд t визначається описом алгоритму в певній алгоритмічній моделі та не залежить від фізичної реалізації цієї моделі. Середній час τ — величина фізична, яка залежить від швидкості обробки сигналів в елементах ПК. Ось чому об'єктивною математичною характеристикою складності алгоритму в певній моделі є кількість команд t.

Ємнісну характеристику складності алгоритмів визначають кількістю комірок пам'яті, що використовують у процесі його обчислення. Ця величина

не може перевищувати кількість дій *t*, що множиться на певну константу (кількість комірок, які використовують при виконанні однієї команди).

Слід зазначити, що часова складність алгоритму не є постійною величиною і залежить від розмірності задачі.

Складність алгоритму — функція, значення якої залежить від розмірності *n* даних задачі.

Виконуючи аналіз алгоритмів, використовують різні способи їх подання: словесний опис, блок-схеми чи запис однією з мов програмування.

Псевдокод – штучна неформальна мова, яка використовується для подання алгоритмів.

Пояснимо сказане на прикладі аналізу підпрограми обчислення факторіалу *n!*.

```
function factorial (n:integer): integer;
begin
(1) if n≤1 then
(2) factorial :=1
else
(3) factorial :=n* factorial (n-1)
    end;
```

Природною мірою кількості вхідних даних для функції *factorial* є значення *n*. Позначимо через $T(n)$ часову складність алгоритму. Часова складність рядків (1) і (2) має значення 1, а рядка (3) — $(1 + 1) + T(n–1)$. Таким чином, для деяких констант *c* і *d* маємо:

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

Припустивши, що $n > 2$ і, підставивши у співвідношення (1) $n – 1$ замість n, отримаємо: $T(n) = 2c + T(n–2)$. Аналогічно, якщо $n > 3$, отримаємо: $T(n) = 3c + T(n– 3)$.

Продовжуючи цей процес, у загальному випадку для деякого i, $n > 1$, маємо: $T(n) = ic + T(n–i)$. Поклавши в останньому виразі $i = n – 1$, отримуємо:

$$T(n)=c(n-1)+ T(1)= c(n-1)+d.$$

Таким чином, загальний метод рішення рекурентних співвідношень, полягає у послідовному розкритті виразу $T(k)$ у правій частині рівняння до тих пір, поки не вийде формула, в якій у правій частині відсутня $T(n)$.

Оскільки для різних конкретних даних однакової розмірності n алгоритм може витрачати різну кількість команд, функція складності $T(n)$ визначається різними способами. Розглянемо деякі з них на прикладі *алгоритмів впорядкування*.

Під *впорядкуванням* розуміємо процес перестановки об'єктів деякої множини у певному порядку [1, 74]. Методи впорядкування є ілюстрацією ідеї аналізу складності алгоритмів.

Існує кілька основних алгоритмів впорядкування (Д. Кнут наводить 25 основних алгоритмів [2, 85] та велику кількість їх модифікацій.

Упорядкування обміном

Алгоритм впорядкування обміном базується на принципі порівняння пари сусідніх елементів до тих пір, доки не будуть впорядковані всі елементи.

Щоб описати основну ідею цього методу, який іноді називають методом «бульбашки», уявимо, що елементи зберігаються в послідовності (масиві), розташованому вертикально. Елементи, що мають малі значення, є «легкішими» і «спливають» нагору подібно бульбашцікам. При першому проході уздовж масиву (перегляд починається знизу), береться перший елемент послідовності і його значення по черзі порівнюється зі значеннями наступних елементів. Якщо зустрічається елемент з «важчим» значенням, то ці елементи міняються місцями.

Проілюструємо роботу алгоритму впорядкування обміном наступним прикладом, що записаний на псевдокоді.

```
(1) for i←n-1 step -1 until 1 do
(2)   for j←1 step 1 until I do
(3)       if a[j] > a[j+1] then
      begin
(4)             swap(a[j], a[j+1])
      end
```

Звернемо увагу на те, що підпрограму-процедуру *swap* у рядку (4) використовують у багатьох алгоритмах впорядкування для перестановки елементів місцями. Код цієї процедури наведено нижче:

```
procedure swap (x,y)
  begin
     temp ← x
     x ← y
     y ←temp
  end
```

Для визначення часової складності алгоритму виконаємо такі дії. Біля кожного рядка алгоритму зазначимо його вартість (число операцій) і кількість разів, за яку виконується цей рядок. Зауважимо, що рядки усередині циклу виконуються на один раз менше, ніж перевірка, оскільки остання перевірка виводить з циклу.

Визначення часової складності алгоритму впорядкування обміном

№	Команда	Вартість	Кількість виконань
(1)	*for* i←n-1 *step* -1 *until* 1 *do*	c_1	n
(2)	*for* j←1 *step* 1 *until* i *do*	c_2	n-1
(3)	*if* a[j] > a[j+1] *then*	c_3	$\sum_{j=1}^{i} k_j$
(4)	swap (a[j], a[j+1])	c_4	$\sum_{j=1}^{i} t_j$

Рядок вартістю *c*, що повторений *m* разів, дає внесок *cm* у загальну кількість операцій. Склавши внески всіх рядків, одержимо вираз, що позначає часову складність алгоритму впорядкування обміном:

$$T(n) = c_1 n + c_2(n-1) + c_3 \sum_{j=2}^{i} k_j + c_4 \sum_{j=2}^{i} t_j$$

Обчислимо суму кількості виконань для рядку (3):

$$\sum_{j=1}^{i} k_j = \frac{1+n-1}{2}(n-1) = \frac{n(n-1)}{2}$$

Таким чином, часова складність алгоритму $T(n)$ дорівнює:

$$T(n) = c_1 n + c_2(n-1) + c_3(\frac{n(n-1)}{2}) + c_4 \sum_{j=2}^{i} t_j =$$

$$= c_1 n + c_2(n-1) + c_3(\frac{n^2 - n)}{2}) + c_4 \sum_{j=2}^{i} t_j =$$

$$= c_1 n + c_2(n-1) + c_3(\frac{1}{2}(n^2 - n)) + c_4 \sum_{j=2}^{i} t_j =$$

$$= c_1 n + c_2 n - c_2 + \frac{c_3 n^2}{2} - \frac{c_3 n}{2} + c_4 \sum_{j=2}^{i} t_j =$$

$$= (\frac{c_3}{2})n^2 + (c_1 + c_2 - \frac{c_3}{2})n - c_2 + c_4 \sum_{j=2}^{i} t_j$$

Як бачимо, функція $T(n)$ — квадратична, тобто має вигляд $T(n) = an^2 + bn + c$, де константи a, b і c визначаються значеннями c_1, …, c_4.

Упорядкування вибором

Ідея методу впорядкування вибором полягає у тому, що на i-му кроці вибирається найменший елемент серед елементів послідовності a[i]..a[n], який міняється місцями з елементом a[i].

У залежності від результату пошуку елемент a[i] або залишається в i-ій позиції або вони міняються місцями і процес повторюється.

Визначення часової складності алгоритму впорядкування вибором

№	Команда	Вартість	Кількість виконань
(1)	*for i←1 step 1 until n-1 do*	c_1	n
	begin {for}		
(2)	*min←a[i]*	c_2	n-1
(3)	*index←i*	c_3	n-1
(4)	*for j←i+1 step 1 until n do*	c_4	$\sum_{j=i+1}^{n} t_j$
	begin {for}		
(5)	*if a[j] < min then*	c_5	$\sum_{j=i+1}^{n} (t_j - 1)$
	begin {if}		
(6)	*min←a[j]*	c_6	$\sum_{j=i+1}^{n} k_j$
(7)	*index←j*	c_7	$\sum_{j=i+1}^{n} k_j$
	end {if}		
(8)	*swap(a[i], a[index])*	c_8	n-1
	end {for}		
	end {for}		

Склавши внески всіх рядків, одержимо вираз, що визначає часову складність алгоритму впорядкування вибором:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=i+1}^{n} t_j + c_5 \sum_{j=i+1}^{n} (t_j - 1) + c_6 \sum_{j=i+1}^{n} k_j + c_7 \sum_{j=i+1}^{n} k_j + c_8(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n^2+n}{2} - 1\right) + c_5\left(\frac{n^2-n}{2}\right) +$$

$$+ c_6 \sum_{j=i+1}^{n} k_j + c_7 \sum_{j=i+1}^{n} k_j + c_8(n-1)$$

Часова складність алгоритму

$$T_{\min}(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n^2+n}{2} - 1\right) + c_5\left(\frac{n^2-n}{2}\right) + c_8(n-1)$$

Упорядкування вставками

Ідея методу полягає у тому, що розглядають дії алгоритму на його *i*-му кроці. При цьому послідовність поділяють на дві частини: впорядковану *a*[1], ..., *a*[i] і неупорядковану *a*[i+1], ..., *a*[*n*]. На кожному наступному (*i*+1)-му кроці алгоритму береться *a*[*i*+1] елемент і вставляється у потрібне місце в упорядковану частину послідовності.

Пошук потрібного місця для чергового елемента вхідної послідовності здійснюється шляхом послідовних порівнянь з елементом, що знаходиться перед ним. Залежно від результату порівняння елемент або залишається на поточному місці (вставка закінчена), або вони міняються місцями і процес повторюється.

Визначення часової складності алгоритму впорядкування вставками

№	Команда	Вартість	Кількість виконань
(1)	***for*** *i←2* ***step*** 1 ***until*** *n* ***do***	c_1	n
	begin{for}		
(2)	*key←A[i]*	c_2	n-1
(3)	*j←i-1*	c_3	n-1
(4)	***while*** (*j*>0) ***and*** (*a[j]*>*key*) ***do***	c_4	$\sum_{i=2}^{n} t_i$
	begin{while}		
(5)	*a[j+1]←a[j]*	c_5	$\sum_{j=2}^{n} (t_j - 1)$
(6)	*j←j-1*	c_6	$\sum_{i=2}^{n} (t_i - 1)$
	end{while}		
(7)	*a[j+1]←key*	c_7	n-1
	end{for}		

Склавши внески всіх рядків, одержимо вираз, що визначає часову складність алгоритму впорядкування вставками:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i + c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7(n-1)$$

Підводячи підсумок, порівняємо ефективність розглянутих нами алгоритмів. Для цього скористаємося наведеними Н. Віртом аналітичними формулами [1, 106].

Аналітичні формулами для визначення часової складності алгоритмів

	C/M	$T_{min}(n)$	$T_{avg}(n)$	$T_{max}(n)$
Впорядкування обміном	C M	$(n^2-n)/2$ 0	$(n^2-n)/2$ $(n^2-n)*0.75$	$(n^2-n)/2$ $(n^2-n)*1.5$
Впорядкування Вибором	C M	$(n^2-n)/2$ $3(n-1)$	$(n^2-n)/2$ $n(\ln n+0.57)$	$(n^2-n)/2$ $n^2/4+3(n-1)$
Впорядкування Вставками	C M	$n-1$ $2(n-1)$	$(n^2+n-2)/4$ $(n^2+-9n-10)/4$	$(n^2-n)/2 -1$ $(n^2+3n-4)/2$

Як бачимо, для кожного із розглянутих нами алгоритмів функція визначення часової складності в середньому і найгіршому випадках є квадратичною. Існують більш ефективні алгоритму, які іноді називають «логарифмічними» за тієї причини, що функція визначення часової складності в середньому і найгіршому випадках є $T(n) = an\lg n + b + c$. Різницю між «квадратичними» і «логарифмічними» алгоритмами наочно демонструє приклад, наведений у роботі [3, 29–30].

Упорядкування злиттям

Розглянемо принцип роботи так званого *алгоритму впорядкування злиттям*.

У алгоритмі впорядкування злиттям послідовність спочатку поділяють на дві половини меншої розмірності. Потім відбувається впорядкування кожної з половин окремо. Після цього дві упорядковані послідовності половинного розміру зливаються в одну. Рекурсивне розбиття задачі на менші відбувається доти, поки розмірність послідовності не дорівнюватиме 1 (будь-яку послідовність розмірністю в 1 елемент можна вважати упорядкованим).

1.1. Основні поняття теорії графів

Чимало задач прикладного характеру пов’язують з упорядкованою множиною точок, під якою розуміють об’єкти, агрегати й інші дискретні елементи, які складаються із взаємозв’язаних елементів, з транспортними проблемами тощо.

Такі задачі зручно відображувати схемою, яка складається з точок і відрізків, які з’єднують будь-яку пару точок, якщо такий зв’язок має місце.

Схему, що має таку структуру, називають графом, а задачі з такими графами зараховують до задач теорії графів. За допомогою цієї теорії розв’язують задачі, які пов’язані з побудовою схем, плануванням послідовності виконання робіт, функціонуванням підсистем, балансові процеси тощо.

Найбільше використання теорія графів має при розв’язуванні інформаційних задач, задач з проблемами мінімізації кількості перетинів (так звана задача про мінімальну кількість аварій), задача про найкоротшу відстань, за-

дача про максимальний потік та інші задачі, які пов'язані з побудовою сітки зв'язків (газопроводи, мережі електропередач тощо).

Теорія графів дозволяє за допомогою єдиного підходу формулювати задачі, які начебто далекі одна від одної у своїй постановці. Усі задачі в термінах теорії графів можна розділити на два умовні класи:

— задачі, які пов'язані з підрахунком кількості об'єктів заданої властивості;

— задачі, що розв'язуються за алгоритмами на графах.

Позначимо деяку множину точок як $I = \{i\}$, де точку з цієї множини $i \in I$ назвемо вершиною. Ці вершини мають деякі зв'язки між собою, які утворюють множину зв'язків $U = \{u\}$ та відображують наявність або відсутність зв'язків між парами елементів $(i; j) \in U$, тобто $u \in U$, де $u = (i; j)$.

Якщо неважливо значення, в якій послідовності беруть пари вершин — $(i; j)$ або — $(j; i)$, то зв'язок такої пари вершин називають *ребром* (тобто без орієнтації).

Якщо порядок зв'язку вершин істотно змінює зміст, тобто коли пари $(i; j)$ та $(j; i)$ вважають різними, такий зв'язок називають *дугою* (випадок з орієнтацією).

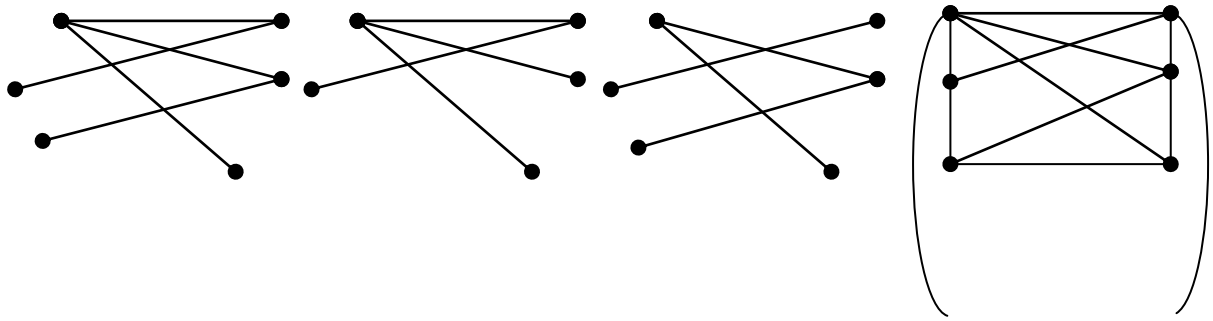
Граф позначають множиною $G = (I; U)$ і називають *неорієнтованим*, якщо зв'язки відображаються ребрами, та *орієнтованим* (орграфом), якщо зв'язками є дуги.

Якщо $\bar{I} \in I$ та $\bar{U} \in U$, то граф $(\bar{I}; \bar{U})$ називають *підграфом* графа $G = (I; U)$. У випадку коли $\bar{I}$ збігається з I, то граф $G' = (I; \bar{U})$ називають *остовним* підграфом графа $G = (I; U)$.

Якщо для будь-яких двох вершин є зв'язок, то такий граф називають *повним*.

Ребро $(i; j)$ інцидентне з вершинами i та j, та, в свою, чергу ці вершини інцидентні ребру $(i; j)$; в орграфі прийнято твердження, що дуга $(i; j)$ виходить з вершини i та заходить у вершину j.

Приклади графів наведено на рис. 1.



Граф $G = (I, U)$ **Підграф $(\bar{I}, \bar{U})$** **Остовний граф $G' = (I, \bar{U})$** **Повний граф**

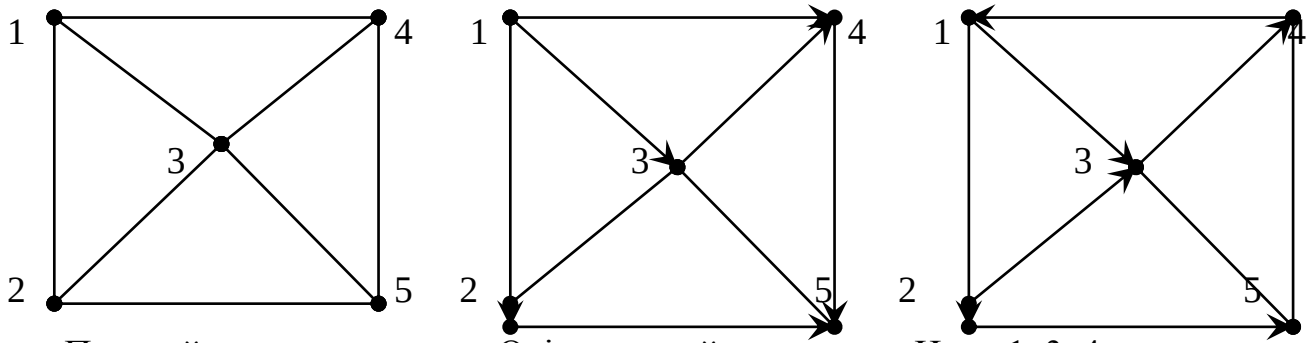
Рис. 1

Перехід від однієї вершини графа до другої виконують за допомогою *шляху*. У неорієнтованому графі шлях називають *ланцюгом*, який складається з послідовного набору суміжних ребер.

Шлях в орграфі називають *орієнтованим*. Якщо перша та остання вершини шляху збігаються, то такий шлях називають *циклом.*

Якщо для будь-якої пари є принаймні один шлях, то граф називають *зв'язним.*

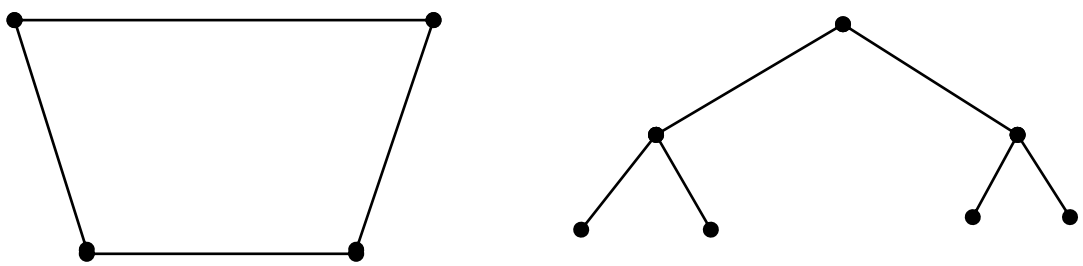
Особливим класом графів є *дерево* — це зв'язний граф, у якому відсутні цикли (рис. 2).



Простий шлях 1, 3, 5

Орієнтований шлях 1, 3, 4, 5

Цикл 1, 3, 4



Зв'язний граф

Граф-дерево

Рис. 2

Вершини та їх зв’язки можна зображувати на площині по-різному, виходячи із зручного зображення процесу. Внаслідок цього один і той самий граф можна зображувати по-різному, наприклад як на рис. 3.

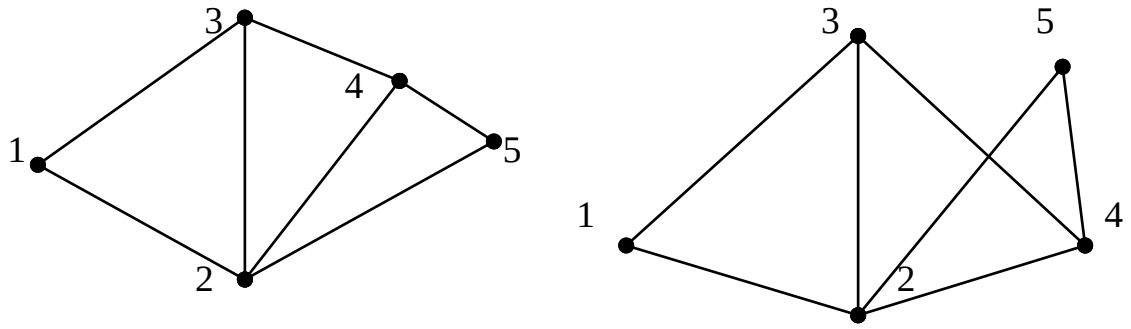


Рис. 3

Такі графи називають *ізоморфними*.

1.2. Засоби представлення графів. Зважені графи та мережі

Якщо граф зображують за допомогою схем у вигляді точок і зв’язків між ними, то таку форму графів відображають діаграмою графа. Але з великою кількістю елементів графа геометричне його зображення втрачає наочність. Тому в таких графах доцільно використовувати матричну форму.

Якщо наявність зв’язку між парою точок відображати через «1» а відсутність — через «0», то у такому випадку граф можна задавати матрицею суміжності (рис. 4).

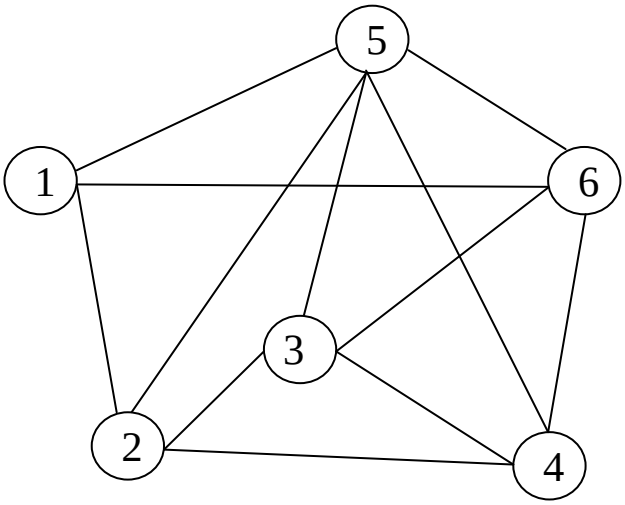


Рис. 4

	1	**2**	**3**	**4**	**5**	**6**
1	0	1	0	0	1	1
2	1	0	0	1	1	0
3	0	0	0	1	1	1
4	0	1	1	0	1	0
5	1	1	1	1	0	1
6	1	0	1	0	1	0

У деяких випадках зручно зображувати зв'язки елементів у вигляді *зваженого графа*. Це граф, у якого кожному ребру $(i; j)$ відповідає деяке число s_{ij}, яке називають *вагою* цього ребра. Іноді вагу зображують деякою функцією. Таку матриця суміжності у зважених графах називають *матрицею ваги*.

Якщо кожному елементу графа — вершині чи дузі — задані деякі параметри чи характеристики, то такий граф є *мережею*. Найчастіше в задачах теорії графів у випадку сіткової моделі дуги відображують функції, які показують їх пропускні спроможності s_{ij}, особливо це стосується задач транспортного типу. Якщо деяка величина $\propto_i$є параметром i-ї вершини, то цю величину називають *інтенсивністю* вершини. Якщо $\propto_i > 0$, то i-у вершину називають *джерелом*, якщо $\propto_i < 0$ — стоком, якщо $\alpha = 0$, то така вершина є *нейтральною*.

1.3. Потоки на мережах. Поняття розрізу

Якщо у мережі задано пропускні спроможності усіх дуг $(i; j)$, то *потоком* називають функцію, що відображує $(i; j)$ дугу числом x_{ij}, що відповідає властивостям

$$0 \leq x_{ij} \leq s_{ij}, \tag{1}$$

$$\sum_i x_{ik} - \sum_j x_{kj} = 0, \forall\, k \neq s, t; k \in I, \tag{2}$$

$$\sum_j x_{ik} - \sum_i x_{it} = \nu, i, j \in I. \tag{3}$$

Умова (1) означає, що потік x_{ij} не повинен перебільшувати величину s_{ij} для $(i; j)$ дуги; умова (2) означає, що для будь-якої проміжної вершини мережі обсягу продукту (ресурсу), який входить до цієї вершини, має дорівнювати обсягу продукту, який виходить із цієї вершини; умова (3) означає, що загальна кількість продукту ν, яка виходить з початкової вершини s мережі, дорівнює загальній кількості продукту, яка входить у кінцеву вершину t. Ця кількість є величиною потоку мережі і її називають *потужністю* потоку.

Визначимо поняття розрізу мережі.

Розглянемо мережу з одним джерелом та одним стоком t. У такій мережі множину її вершин *I* можна розбити на дві підмножини так:

$$S \cup S' = I,$$

$$S \cap S' = \emptyset,$$

$$s \in S, t \in S'.$$

Таке розбиття мережі відображується так званим *розрізом*, який розділяє джерело від стоку за допомогою множини (S, S'). У такий розріз входять усі дуги, що виходять з вершин $i \in S'$ і входять до вершин $j \in S'$.

Пропускна спроможність розрізу дорівнює сумі пропускних спроможностей дуг, які складають цей розріз, і її називають *величиною розрізу*. Запис його такий:

$$s(S, S') = \sum_{i \in S, j \in S'} S_{ij} = \sum_{i,j \in (S,S')} S_{ij}.$$

Для будь-якого розрізу $x(S, S')$ загальна величина потоку дорівнює

$$v = \sum_{i,j \in (S,S')} x_{ij}$$

і не повинна перебільшувати пропускну спроможність, тобто величини розрізу

$$v = x(S, S') \leq s(S, S'),$$

тому що

$$x_{ij} \leq S_{ij}.$$

Приклад наведено на рис. 5.

До означеного розрізу з S ={1, 2} та S'= {3, 4, 5} входить множина дуг $(S, S') = \{(1,3), (2,3), (2,4), (2,5)\}$, при цьому пропускна спроможність дорівнює $(S, S') = S_{12} + S_{23} + S_{24} + S_{25} = 2 + 2 + 2 + 5 = 11$.

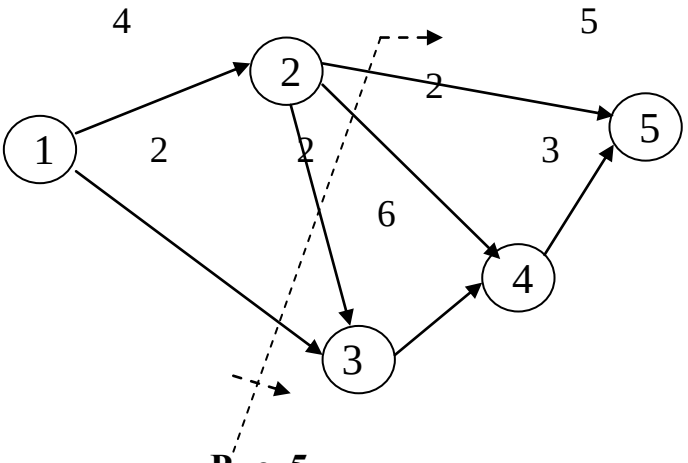


Рис. 5

1.4. Задача про максимальний потік

1.4.1. Загальна постановка

Основна задача на мережах — знаходження допустимого максимального потоку з початкової вершини до кінцевої, якщо задано пропускні спроможності кожної дуги мережі та топологія мережі.

Математична модель задачі така:

— цільова функція:

$$F = \nu \to max,$$

— обмеження:

$$\sum_i x_{ik} - \sum_j x_{kj} = 0, \forall\, k \neq s, t; k \in I, 0 \leq x_{ij} \leq S_{ij}.$$

Для будь-якої мережі можна знайти кілька розрізів з різними значеннями величини потоку ν

$$\nu = \sum_{i,j \in (S,S')} x_{ij}.$$

Згідно з умовою $x_{ij} \leq S_{ij}$, маємо

$$\sum_{i,j \in (S,S')} x_{ij} \leq \sum_{i,j \in (S,S')} S_{ij}$$

Цілком очевидно, що зростання величин можливо до того моменту, коли з усіх допустимих розрізів мережі буде знайдений розріз з максимальною величиною потоку

$$min\ s(S,S') = \sum_{i,j \in (S,S')} x_{ij}.$$

Ця величина відображає величину v і є «вузьким» місцем мережі. Внаслідок цього максимальний потік мережі дорівнює пропускній спроможності розрізу з мінімальною його величиною.

1.4.2. Алгоритм Форда — Фалкерсона

Щоб знайти максимальний потік, необхідно задати неорієнтований граф з прямою і зворотною пропускною спроможністю по кожному ребру. Якщо задано орграф, то зворотна пропускна спроможність ребра $S_{ij} = 0$.

Послідовність виконання дій така.

1. Будується початкова матриця графа $S = \|S_{ij}\|$.

2. Вибір розрізу нульового потоку: джерело s з одного боку $s \in S$ та вся решта вершин графа — з другого боку, тобто розподіл множини вершин *I* на дві підмножини: $S \cup S' = I$.

3. Для кожного ребра розрізу будується один з допустимих *L*–шляхів з s вершини до вершини*t* і знаходиться мінімальний потік кожного допустимого шляху як $\Delta_L = \min\{S_{ij}\}$, де$S_{ij}$ — пропускна спроможність *ij*-го ребра *L*-го шляху. Шлях через розріз доцільно вибрати найпростішим.

4. Будується матриця нульового потоку *X*. Для кожного *L*-шляху всі x_{ij} дорівнюють значенню Δ_L, а елементи, що симетричні їм, дорівнюють $-\Delta_L$; решта елементів матриці дорівнюють нулю. Якщо ребро зустрічається кілька разів у різних шляхах з величинами, то всі величини складаються для цього ребра, але ця сумарна величина не повинна перебільшувати s_{ij}.

5. Побудова матриці *S* – *X* (матриці резервів): якщо $s_{ij} = 0$, то (*ij*)-ребро насичене. Таке ребро має резерв і є можливість збільшити потік через це ребро.

6. Побудова списку ребер з резервами (насичені ребра) згідно з матрицею *X* від вершини s до вершини *t*. Структура списку:

Частина I	Частина II
S:	$L_1, L_2, \ldots$
L_1	$i_1^{(1)}, i_2^{(1)}, \ldots$
…	…
L_n	…t

У частину I входять вершини, від яких є зв'язки до вершин частини II; в частину II входить множина вершин, до яких є зв'язок від вершини частини I. Подалі в частину I вибираються вершини з попередніх рядків частини II.

Якщо список скласти неможливо до кінцевої вершини *t*, то це вказує на шлях з насиченими ребрами і такий шлях далі не розглядається. Якщо всі шляхи, що розглядаються, не досягають кінцевої вершини*t* у списку, то знайдено максимальний потік. У цьому разі треба знайти його величину, тобто перейти до п. 10.

7. Згідно з здобутим списком, складається ланцюг ненасичених ребер, які ведуть від *s* до *t*. Будувати такий ланцюг необхідно з кінця списку, тобто з ребра (i_n, *t*), який стоїть у кінці списку, потім (i_{n-1}, i_n) та далі до ребра (*s*, i_1): (*s*, i_1),… (i_{n-1}, i_n), (i_n, *t*).

8. Шлях з ненасичених ребер має можливість збільшити потік на додаткову величину $\Delta = \min\{(S_{ij} - x_{ij})\}$, де s_{ij} — пропускна спроможність (ij)-го ребра, $s_{ij} \in S$; x_{ij} — розрахований потік (ij)-го ребра, $x_{ij} \in X$.

9. Побудова нової матриці потоку X на основі попередньої матриці X: усі елементи ребер ненасиченого шляху (за п. 7) збільшуються на величину Δ, симетричні елементи при цьому змінюються на вказану величину. Для нової матриці потоку X треба також зробити аналіз, починаючи з п. 5.

10. Якщо неможливо скласти список до кінцевої вершини *t*, то це означає, що досягнуто насичення ребер шляхів, які ведуть від *s* до *t*, тобто знайдено максимальний потік, величина якого дорівнює сумі x_{ij} рядка *s* з останньої матриці *X*: $v = \sum_k x_{sk}$.

11. Побудова орграфа згідно з дугами (ij), для яких у матриці *X*. Блок-схема алгоритму наведена на рис. 6.

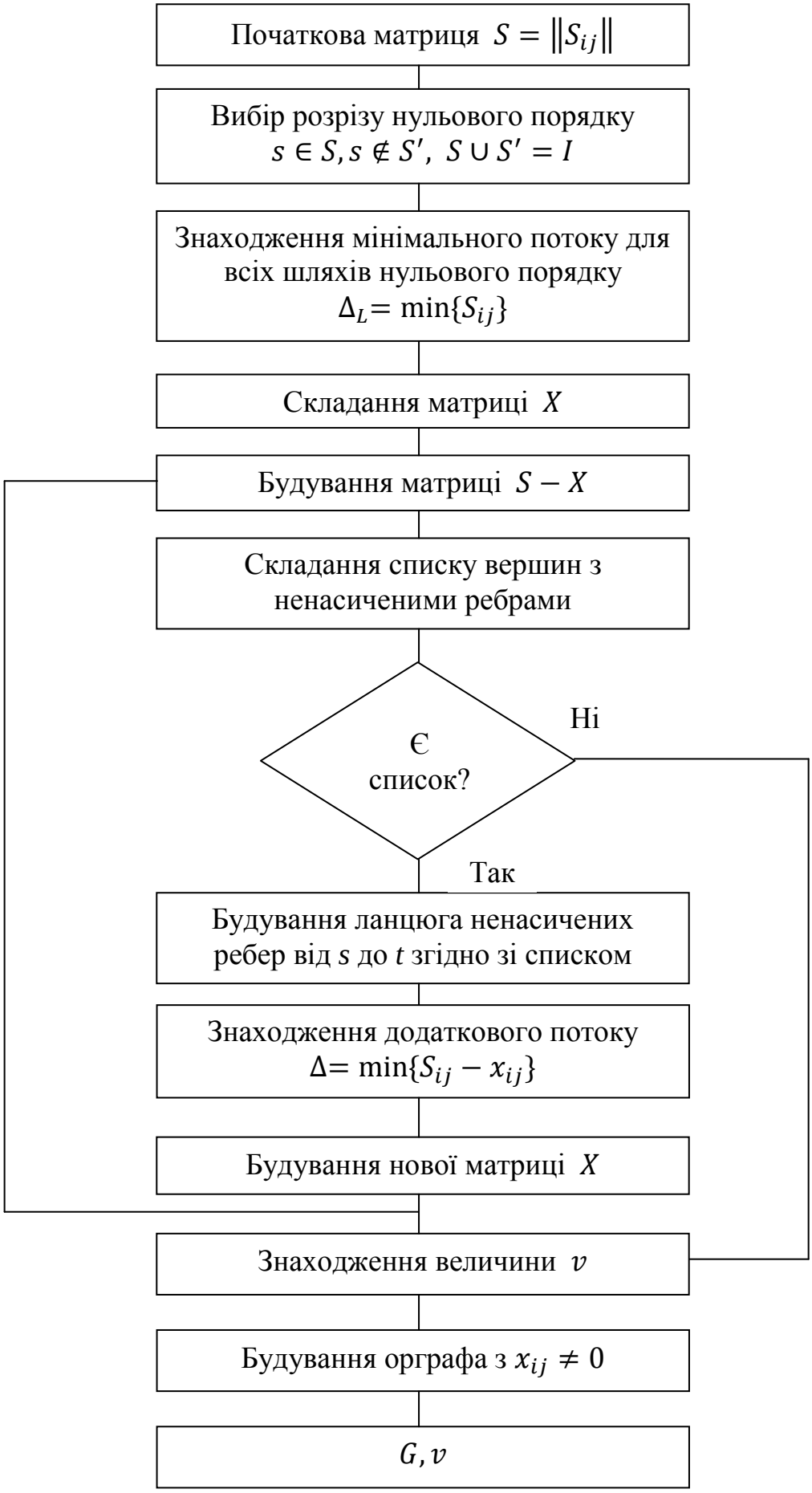

Початкова матриця $S = \|S_{ij}\|$
Вибір розрізу нульового порядку
$s \in S, s \notin S',\ S \cup S' = I$
Знаходження мінімального потоку для всіх шляхів нульового порядку
$\Delta_L = \min\{S_{ij}\}$
Складання матриці X
Будування матриці $S - X$
Складання списку вершин з ненасиченими ребрами
Є список?
Ні
Так
Будування ланцюга ненасичених ребер від *s* до *t* згідно зі списком
Знаходження додаткового потоку
$\Delta = \min\{S_{ij} - x_{ij}\}$
Будування нової матриці X
Знаходження величини v
Будування орграфа з $x_{ij} \neq 0$
G, v

Рис. 6

Приклад

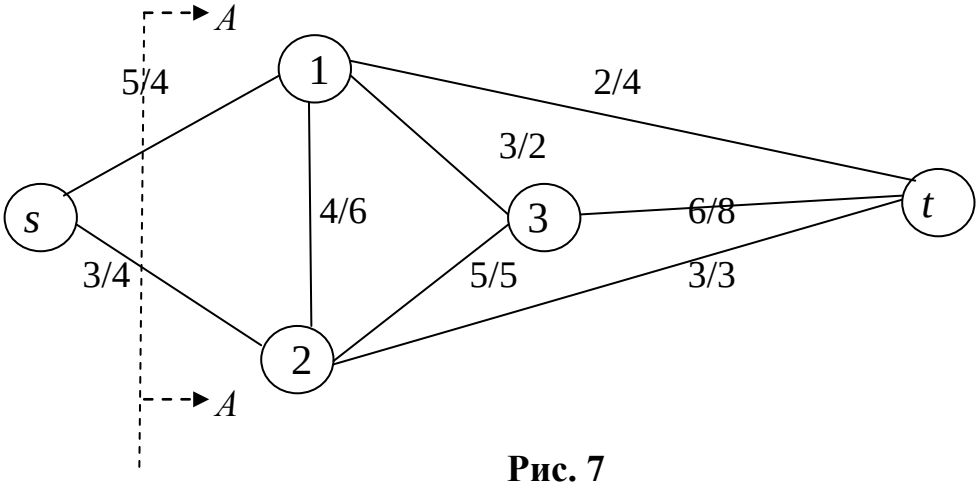


Рис. 7

Знайти максимальний потік та побудувати орграф розв'язку.

Будується вихідна матриця *s*:

	s	1	2	3	*t*
s	X	5	3	0	0
1	4	X	4	3	2
2	4	6	X	5	3
3	0	2	5	X	6
t	0	4	3	8	X

I: (*S*, 1), (1, *t*) та $\Delta_1 = \min\{5, 2\} = 2$;

II: (*S*, 2), (2, *t*) та $\Delta_2 = \min\{3, 3\} = 3$;

Матриця *X*

	s	1	2	3	*t*
s	X	2	3	0	0
1	-2	X	0	0	2
2	-3	0	X	0	3
3	0	0	0	X	0
t	0	-2	-3	0	X

Матриця *S* – *X*

	s	1	2	3	*t*
s	X	3	0	0	0
1	6	X	4	3	0
2	7	6	X	5	0
3	0	2	5	X	6
t	0	0	6	8	X

Далі список скласти неможливо. Тому остання матриця вказує на оптимальний розв’язок задачі.

Величина максимального потоку дорівнює:

$$v = 5 + 3 = 8.$$

Будується орграф за матрицею *X* (рис. 8):

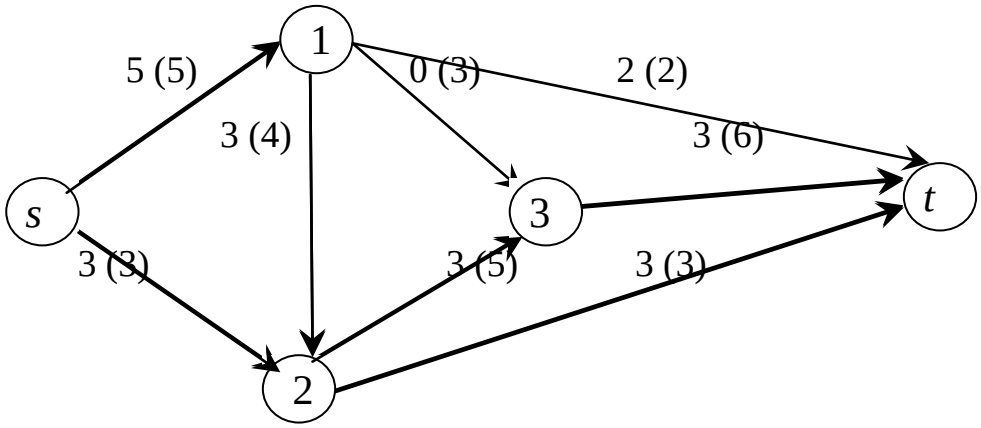


Рис. 8

У дужках наведена початкова пропускна спроможність s_{ij}.

1.5. Задача про потік мінімальної вартості

Граф є одним зі способів графічного представлення інформації про склад та структуру деякої системи. Розглянемо додаток лінійного програмування для розв’язку так званих *сіткових задач*. Нехай нам задана деяка транспортна мережа (трубопровід, залізна дорога, телефонна мережа та ін.), по якій ми хочемо переміщувати деякі однорідні одиниці (нафту, автомобілі, повідомлення та ін.) з однієї точки мережі (пункт відправлення) в іншу (пункт призначення), крім них мережа включає в себе множину проміжних вузлових пунктів, з’єднаних між собою з першими двома. Вузлові пункти при цьому можна інтерпретувати, як роздоріжжя транспортних шляхів. Домовимось позначити пункт відправки цифрою 0, пункт призначення — літерою *m*, а вузлові пункти — їх порядковими номерами, тоді загальна кількість вузлів може бути від 0 до *m*. Домовимось позначати шлях, який зв’язує вузлові пункти (наприклад, 2 і 3), записом (2; 3) або, в загальному випадку, (*i*; *j*). Звернемо увагу, що зв’язки, зображені в такій мережі, є направленими, тобто потік вантажу вздовж кожного з цих шляхів відбувається в напрямку, вказа-

ному стрілкою. Причому їх пропускна здатність обмежена (трубопровід може пропускати тільки конкретну кількість нафти за годину, лінія зв'язку обслуговує конкретну кількість викликів в день і т. д.). Позначимо максимальну пропускну здатність кожного з шляхів через f_{ij}, наприклад, на рис. 9, максимальний потік f_{23}, який пропускається шляхом (2; 3), дорівнює 1.

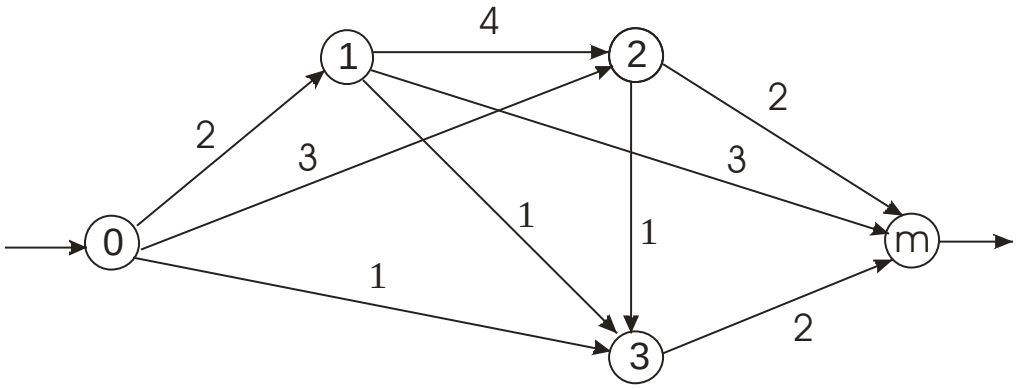


Рис. 9

При транспортуванні потік деякого вантажу прямує з пункту 0 по різних шляхах через вузлові пункти в пункт призначення до тих пір, поки весь цей вантаж не попаде в пункт m. Це означає, що ми накладаємо на мережу умову збереження потоку в проміжних вузлах: все, що потрапляє в такий вузол, повинно повністю залишити його.

Залежно від поставлених умов оптимізації виділяється декілька класів сіткових задач:

- задача про потік мінімальної вартості;
- задача про найкоротший шлях;
- задача про максимальний потік.

Задача (сіткова модель) про потік мінімальної вартості

Нехай задана деяка мережа, яка відображає процес доставки однорідного вантажу з пункту відправки 0 в пункт призначення m. З кожним шляхом $(i; j)$ зв'язана вартість C_{ij} транспортування одиниці вантажу з пункту i в пункт j. Усього з пункту відправки 0 в пункт призначення m повинна бути доставлена певна кількість F одиниць вантажу таким чином, щоб загальна вартість транспортування була мінімальною. Припустимо, що у вузлах виконується умова збереження потоку і що потік X_{ij} вздовж шляху $(i; j)$ — кількість ван-

тажу, доставленого з вузлового пункту i в пункт j, невід'ємний і обмежений, тобто $0 \le X_{ij} \le f_{ij}$.

Нехай загальний потік $F = 5$, якщо $m = 4$. Вартість транспортування C_{ij} і верхню границю f_{ij} для кожного шляху записуватимемо у вигляді пари чисел $[f_{ij}, C_{ij}]$, наприклад на рис. 10 для шляху (2; 3) позначення [1, 2] відповідає $f_{23} = 1$ і $C_{23} = 2$.

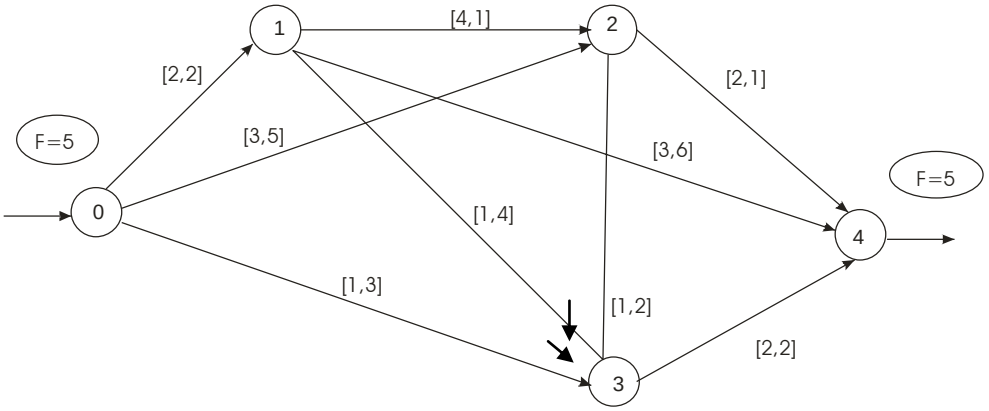


Рис. 10

Вихідні дані, представлені на рис. 10, можна зобразити у табличній формі:

Таблиця 1

Верхня границя потоку вантажу		Вартість C_{ij}		Фактичний потік вантажу вздовж шляху (i, j)
F_{01}	2	C_{01}	2	X_{01}
F_{02}	3	C_{02}	5	X_{02}
F_{03}	1	C_{03}	3	X_{03}
F_{12}	4	C_{12}	1	X_{12}
F_{13}	1	C_{13}	4	X_{13}
F_{14}	3	C_{14}	6	X_{14}
f_{23}	1	C_{23}	2	X_{23}
f_{24}	2	C_{24}	1	X_{24}
f_{34}	2	C_{34}	2	X_{34}

Суть цієї задачі полягає у мінімізації сумарної вартості перевезення вантажу, тобто дана цільова функція повинна досягнути мінімуму.

Побудуємо математичну модель для заданої мережі.

Цільова функція:

$$2X_{01} + 5X_{02} + 3X_{03} + X_{12} + 4X_{13} + 6X_{14} + 2X_{23} + X_{24} + 2X_{34}.$$

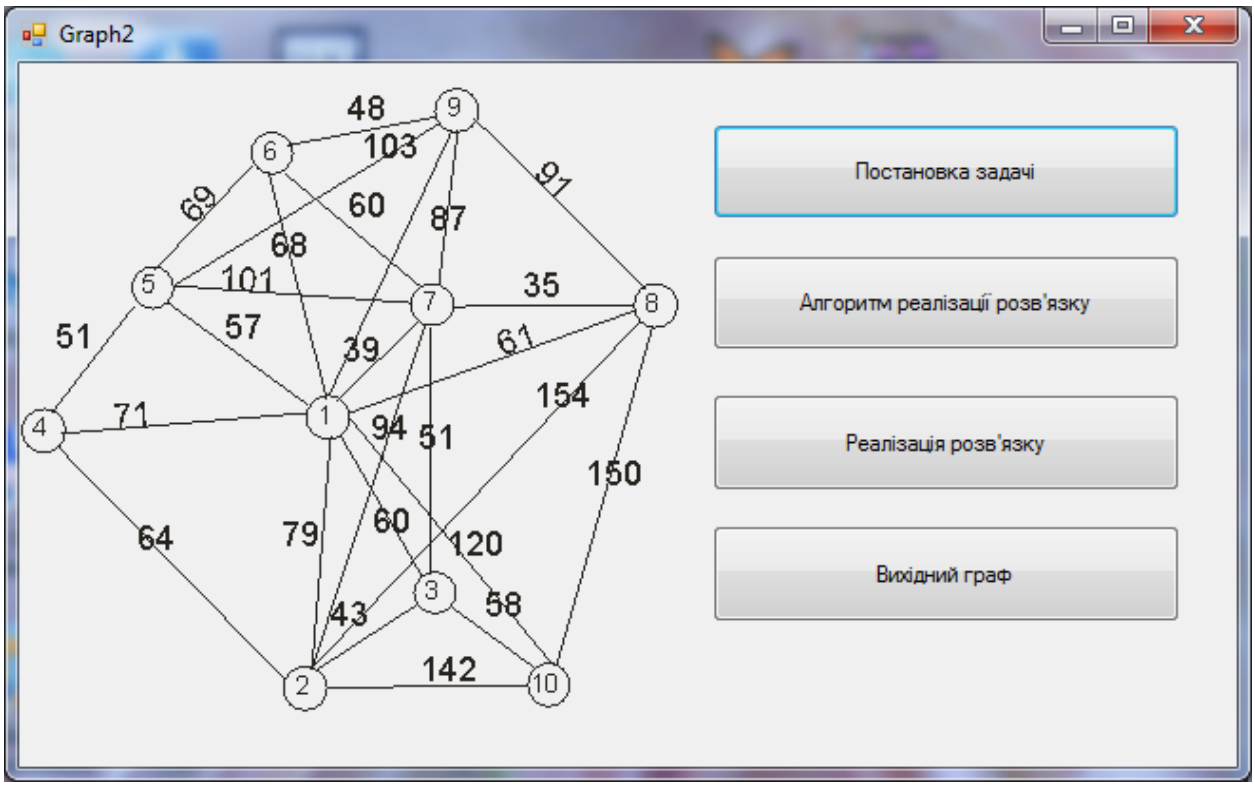
Граничні умови випливають з накладених обмежень:

Таблиця 2

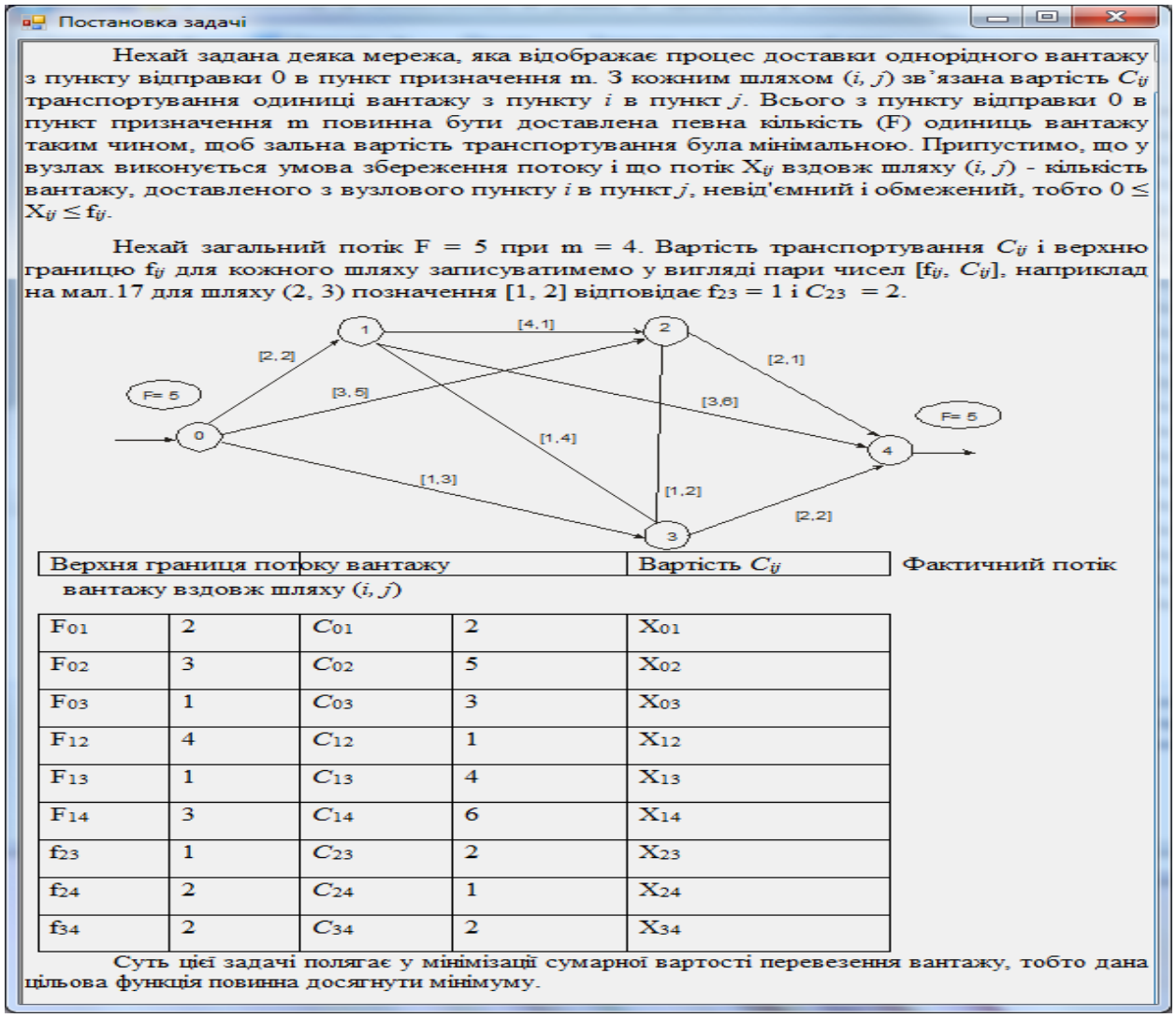
X_{01}	$+X_{02}$	$+X_{03}$						$=5$
X_{01}			$+X_{12}$	$+X_{13}$	$+X_{14}$			$=0$
	$-X_{02}$		$-X_{12}$			$+X_{23}$	$+X_{24}$	$=0$
		$-X_{03}$		$-X_{13}$		$-X_{23}$	$+X_{34}$	$=0$
					$-X_{14}$	$-X_{24}$	$-X_{34}$	$=-5$
X_{01}								≤ 2
	X_{02}							≤ 3
		X_{03}						≤ 1
			X_{12}					≤ 4
				X_{13}				≤ 1
					X_{14}			≤ 3
						X_{23}		≤ 1
							X_{24}	≤ 2
							X_{34}	≤ 2

Продемонструємо розв’язок цієї задачі за допомогою системи програмування C++ (для часткової реалізації розв’язку і побудови графа) і складової офісного пакету MSExcel (яка частково може використовуватися для моделювання розв’язків задач лінійного програмування).

1) Головна форма, з якої стартують всі підрозділи розв’язку, має вигляд:



2) по черзі відкриваємо постановку задачі



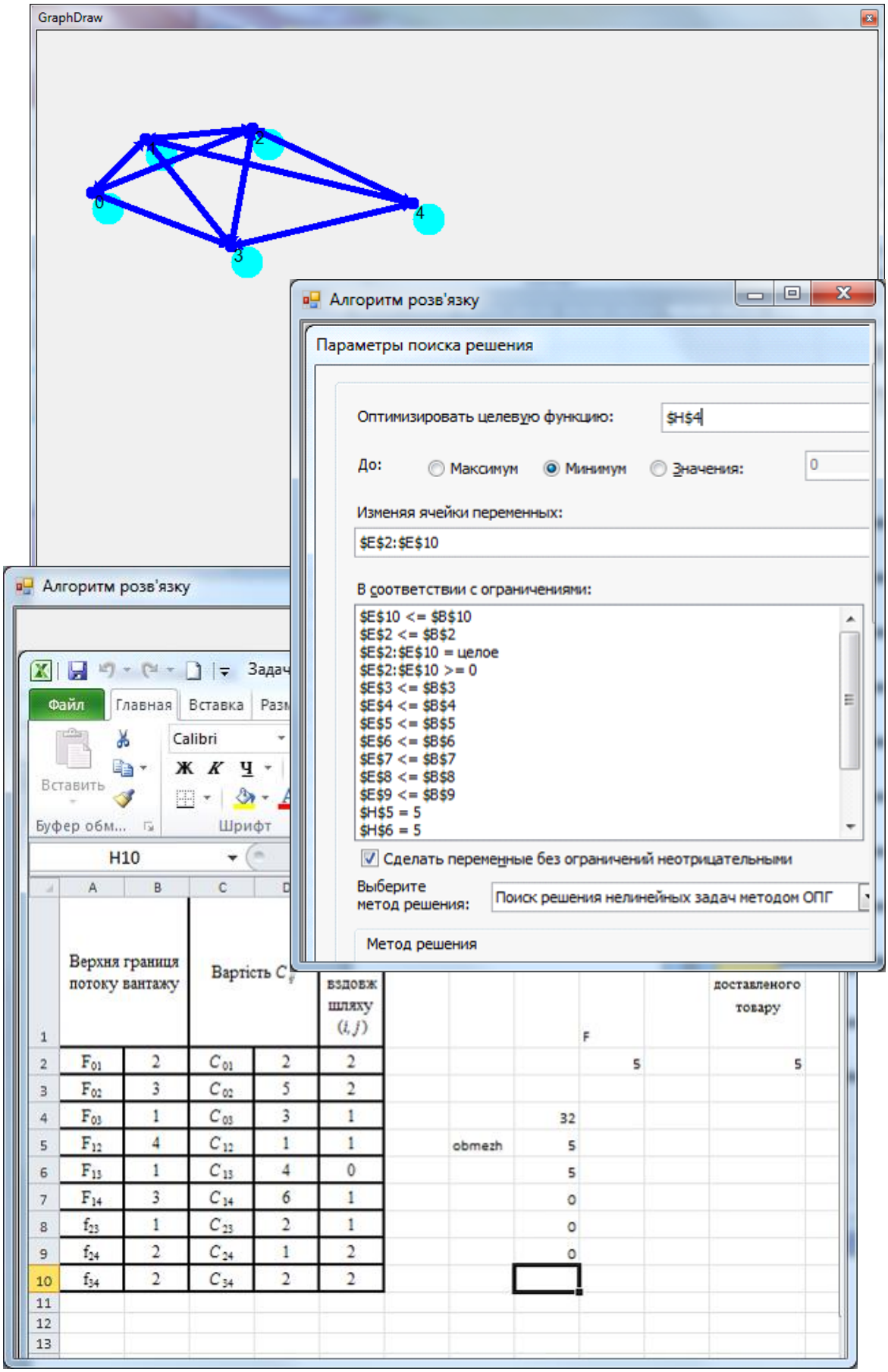
Нехай задана деяка мережа, яка відображає процес доставки однорідного вантажу з пункту відправки 0 в пункт призначення m. З кожним шляхом (i, j) зв'язана вартість C_{ij} транспортування одиниці вантажу з пункту i в пункт j. Всього з пункту відправки 0 в пункт призначення m повинна бути доставлена певна кількість (F) одиниць вантажу таким чином, щоб зальна вартість транспортування була мінімальною. Припустимо, що у вузлах виконується умова збереження потоку і що потік X_{ij} вздовж шляху (i, j) - кількість вантажу, доставленого з вузлового пункту i в пункт j, невід'ємний і обмежений, тобто $0 \le X_{ij} \le f_{ij}$.

Нехай загальний потік F = 5 при m = 4. Вартість транспортування C_{ij} і верхню границю f_{ij} для кожного шляху записуватимемо у вигляді пари чисел $[f_{ij}, C_{ij}]$, наприклад на мал.17 для шляху (2, 3) позначення [1, 2] відповідає $f_{23} = 1$ і $C_{23} = 2$.

Верхня границя потоку вантажу		Вартість C_{ij}		Фактичний потік вантажу вздовж шляху (i, j)
F_{01}	2	C_{01}	2	X_{01}
F_{02}	3	C_{02}	5	X_{02}
F_{03}	1	C_{03}	3	X_{03}
F_{12}	4	C_{12}	1	X_{12}
F_{13}	1	C_{13}	4	X_{13}
F_{14}	3	C_{14}	6	X_{14}
f_{23}	1	C_{23}	2	X_{23}
f_{24}	2	C_{24}	1	X_{24}
f_{34}	2	C_{34}	2	X_{34}

Суть цієї задачі полягає у мінімізації сумарної вартості перевезення вантажу, тобто дана цільова функція повинна досягнути мінімуму.

3) представляємо граф маршрутів і реалізацію розв’язку:



1.6. Задача про найкоротшу відстань

1.6.1. Загальна постановка

Задача про найкоротшу відстань є типовою транспортною задачею у сітьовій постановці. Вона об'єднує не тільки клас транспортних задач, до неї можна звести низку прикладних виробничих задач:

— про заміну обладнання;

— знаходження найнадійнішого маршруту;

— про мінімізацію аварійних ситуацій;

— проектування тощо.

Щоб розв'язати задачу у сітьовій постановці, треба знайти топологію мережі та величини пропускних спроможностей s_{ij}.

У термінах задачі лінійного програмування математична модель цієї задачі має такий вигляд:

— цільова функція:

$$F = \sum_{i,j \in U} \sum s_{ij}\, x_{ij} \rightarrow min,$$

— обмеження:

$$\sum_{(1j)} x_{1j} = 1\,, (\text{початковий пункт}),$$

$$\sum_{(in)} x_{in} = 1\,, (\text{кінцевий пункт}),$$

$$\sum_{(1k)} x_{1k} = \sum_{(kj)} x_{kj} \text{для } \forall\ \mathrm{k} \neq 1 \text{ та } n.$$

Таку модель можна розв'язувати симплекс-методом, але для сітьових задач розроблено простіші методи розв'язування, тому недоцільно для таких задач використовувати симплекс-метод.

1.6.2. Алгоритм розв'язування

Обчислювальна процедура розв'язування задачі розглядається для мережі без циклів і з однією початковою *s* та кінцевою *t* вершиною.

Заздалегідь усі вершини I графа G мають бути упорядковані, тобто пронумеровані згідно з напрямком дуг зліва направо. Кожна дуга повинна мати оцінку $\propto_{ij}$ — відстань від *i*-ї до *j*-ї вершини.

Алгоритм знаходження найкоротшої відстані від *s* до *t* такий.

1. Розбити множини вершин I на дві підмножини $S \cup S' = I$. Первісно у множинуS входить тільки одна вершина: $i = 1$, тобто джерело *s*.
2. Вибір вершини $j \in S'$ з$\propto_{ij} \neq \infty$ і знаходження оцінки

$$U_j = \min_i\{u_i + \propto_{ij}\}.$$

3. Закріплення за *j*-ю вершиною оцінки U_j.
4. Переведення *j*-ї вершини з підмножини S' у підмножинуS та привласнення їй наступного номера *i*.
5. Перехід до наступної вершини $j \in S'$.
6. Перевірка, чи всі вершини переглянуто, тобто, чи дійшов процес розв'язування до кінцевої вершини *t*. Якщо так, то перехід до п. 7, якщо ні, то перехід до п. 2.
7. Знаходження найкоротшої відстані $L = U_t$ та будування найкоротшого шляху $\{(ij)\}$.

Блок-схема алгоритму наведена на рис. 11.

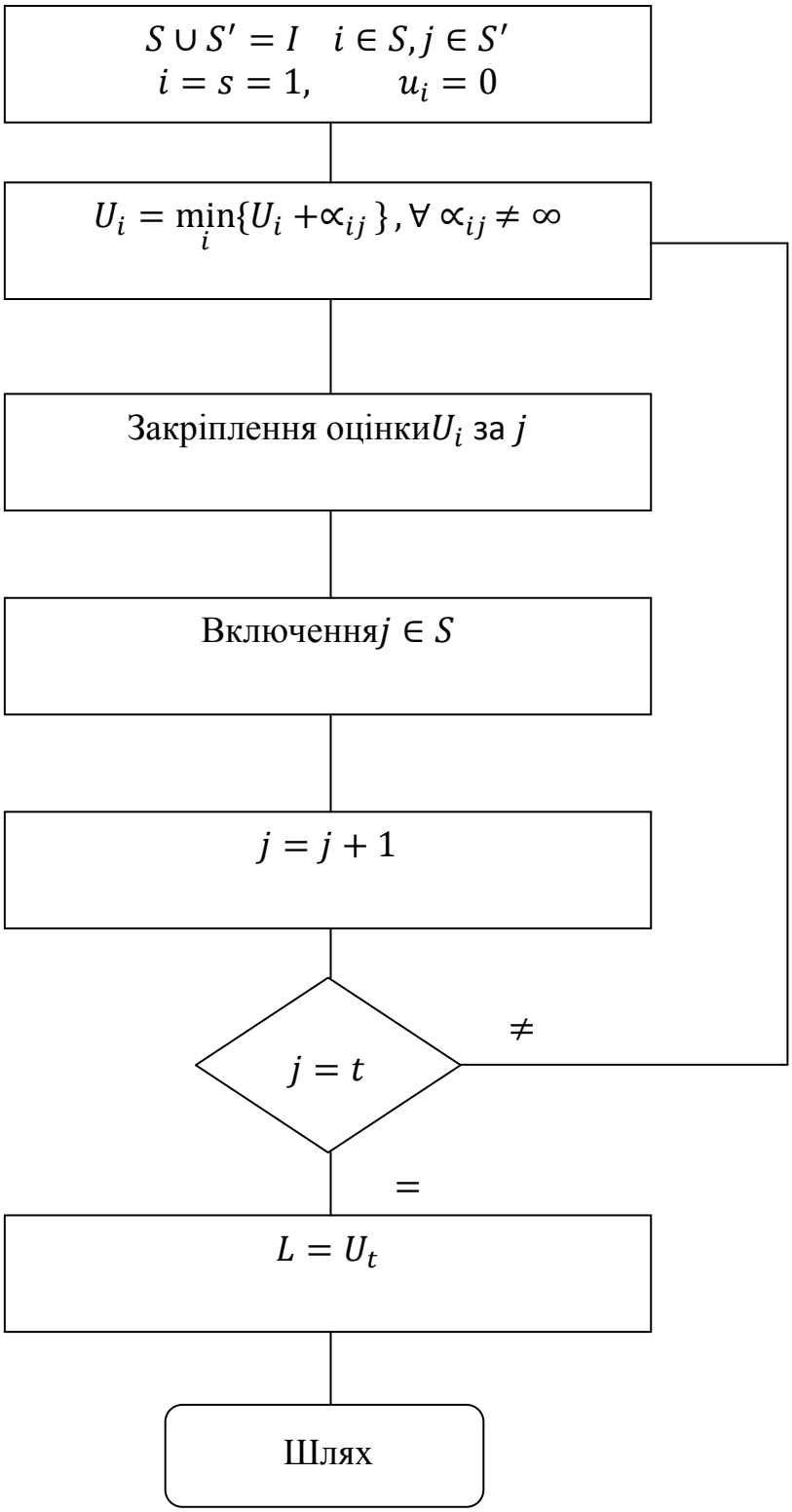


Рис. 11

8. Приклад

Знайти найкоротшу відстань для мережі, зображеної на рис. 12. Кожна дуга має оцінку $\propto_{ij}$.

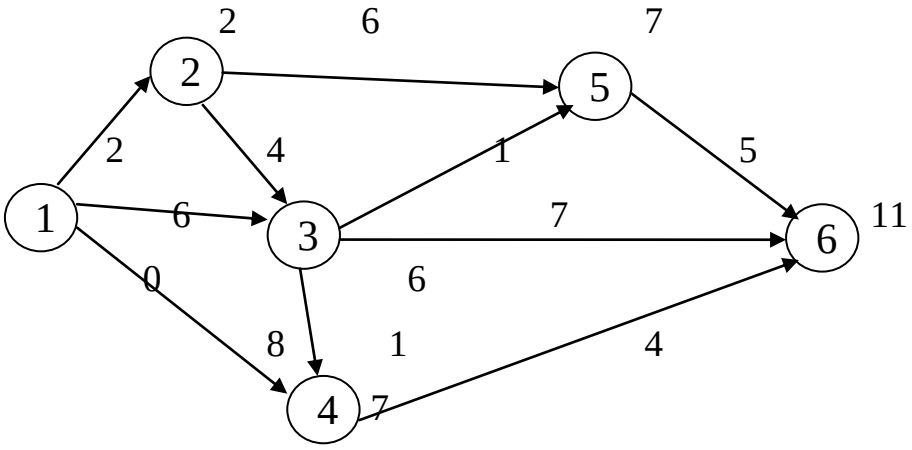


Рис. 12

Приймаємо для початкової вершини $i = 1$. Тоді $u_1 = 0$.

$$u_2 = u_1 + \propto_{12} = 0 + 2 = 2,$$

$$\begin{cases} u_3 = u_1 + \propto_{13} = 0 + 6 = (6), \\ u_3 = u_2 + \propto_{23} = 2 + 4 = 6, \end{cases}$$

$$\begin{cases} u_4 = u_1 + \propto_{14} = 0 + 8 = 8, \\ u_4 = u_3 + \propto_{34} = 6 + 1 = (7), \end{cases}$$

$$\begin{cases} u_5 = u_2 + \propto_{25} = 2 + 6 = 8, \\ u_5 = u_3 + \propto_{35} = 6 + 1 = 7, \end{cases}$$

$$\begin{cases} u_6 = u_5 + \propto_{56} = 7 + 5 = 12, \\ u_6 = u_3 + \propto_{36} = 6 + 7 = 13, \\ u_6 = u_4 + \propto_{46} = 7 + 4 = (11). \end{cases}$$

Відстань шляху $L = 11$.

Шлях (1, 3, 4, 6).

1.7Представлення транспортної задачі у вигляді графа

Транспортну задачу можна зобразити графом, на якому кожна дуга має значення c_{ij} , та кожна вершина – значенняa_i зі знаком «+» або зі знаком «-». Транспортна задача у сітьовій постановці відображається такою математичною моделлю:

$$F = \sum_{i,j \in U} C_{ij} x_{ij} \to min, \sum_{i,j \in u_V^+} x_{ij} - \sum_{i,j \in u_V^-} x_{ij} = b_v,$$

де b_v — потужність постачальника або попит споживача;u_V^+ — множина дуг, які входять до вершини; u_V^- — множина дуг, які виходять з вершини v;u - множина дуг графа G = (I; U).

Напрямок переміщування вантажу вказується дугою. До графу також входять вершини, які не відображують постачальника чи споживача; такі вершини є проміжними з нульовими запасами чи потребами (так звані перевалочні бази). Для знаходження оптимального варіанта згідно з графом транспортної задачі будується який-небудь перший розв’язок цієї задачі з урахуванням ребер неорієнтованого графа та значень a_i, b_j і c_{ij}. Таким чином, між пунктами, де йде переміщення вантажу, ребро графа має значення x_{ij}. Знаходження оцінки кожного ребра ведеться як

$$s_{ij} = C_{ij} - (u - v), \text{де } u > v.$$

Якщо для усіх ребер графа $s_{ij} \geq 0$, то план розв’язку є оптимальним.

Розміщення значеньu та за вершинами здійснюється так. Вибирається для однієї з вершин постачальника довільне значенняu. Потім, згідно з напрямком дуг і величинc_{ij} знаходяться значення для інших вершин, як, де знаки «+» або «-» відповідають обходу за напрямком чи протилежно напрямку дуг відповідно.

Якщо план довільний($\exists s_{ij} < 0$), то знаходяться значенняs_{ij} для ребер графа (без напрямку), а для ребер з$\max\{|-s_{ij}|\}$ будується цикл з дугою відj до i. Потім вибирають усі протилежні дуги і з них знаходиться величина$\Delta =$ на$\Delta = min\{x_{ij}\}$, яка і буде відповідати значенню перевезення вантажу від j до i. Це ребро доповнюється стрілкою-дугою. Дуга з мінімальним значенням постачання ліквідується, залишається тільки ребро між цими вершинами. ВеличинаΔ додається до значень усіх дуг цього напрямку та віднімається від дуг протилежного напрямку.

Приклад: Нехай є транспортна задача у сітьовій постановці (рис. 13).

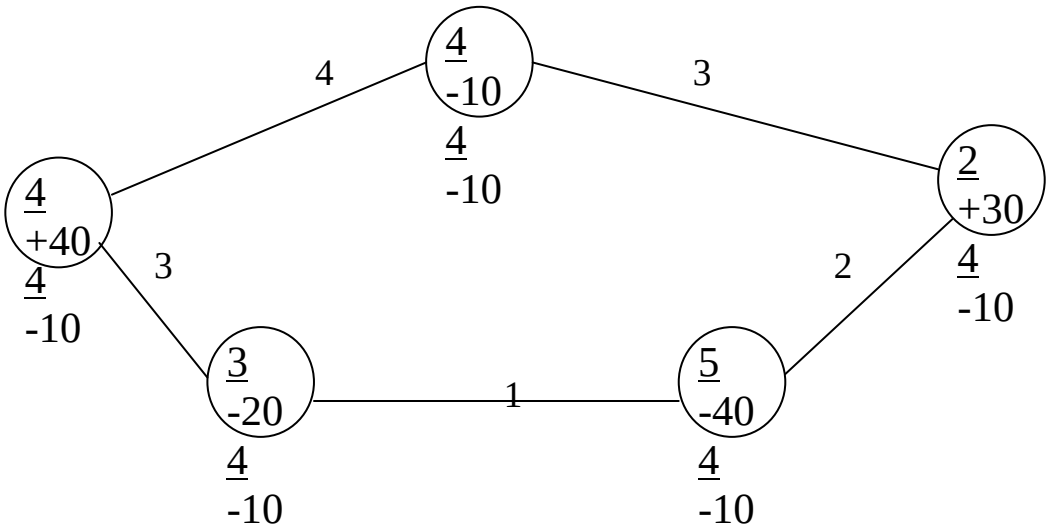


Рис. 13

Перший базисний розв’язок має вигляд (рис. 14).

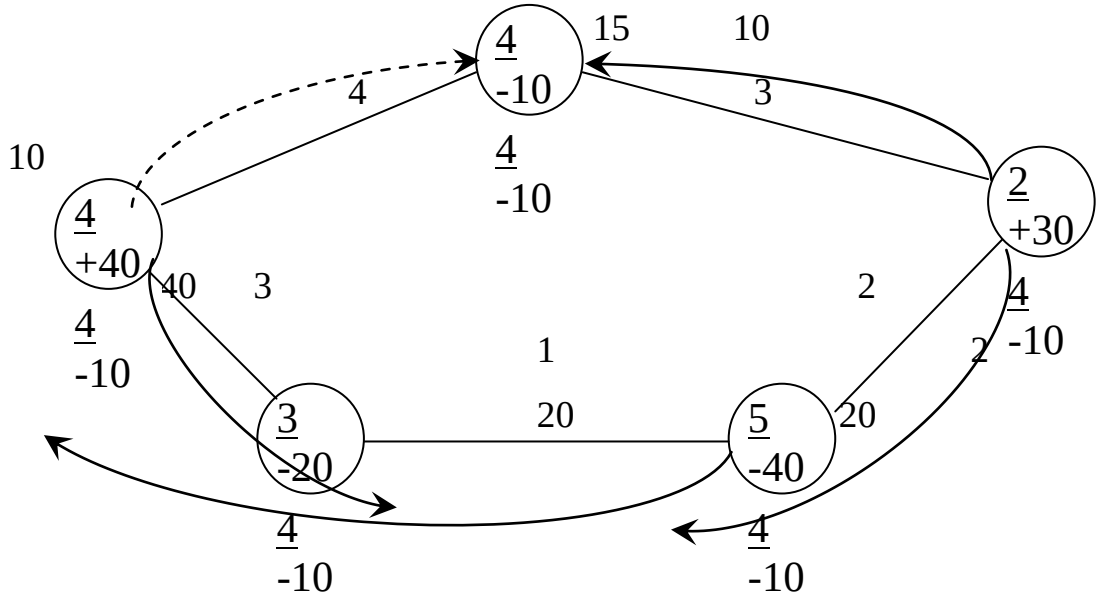


Рис. 14

Перевірка:

$S_{14} = 4 - (15 - 10) = -1 < 0$, тобто не є оптимальним.

Для ребра (14) будується дуга (14), потім для неї знаходиться цикл (1, 4, 2, 5, 3) та обчислюється$\Delta = min\{10{,}20\} = 10$ з дуг протилежного напрямку.

Дузі (14) надається значення Δ: $x_{14} = 10$: дуга (12) ліквідується (як з мінімальним значенням x_{ij}) та залишається тільки ребро між вершинами 2 та 4.

Потім знаходиться решта згідно з напрямком дуг:

$$x_{13} = 40 - 10 = 30,$$

$x_{35} = 20 - 10 = 10,$

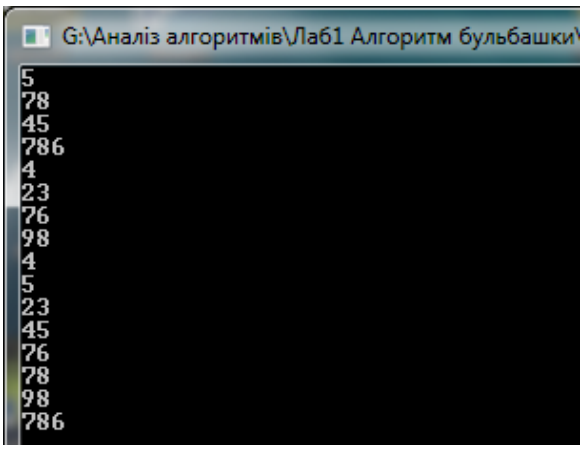
$x_{25} = 20 + 10 = 30,$

$x_{24} = 10 - 10 = 0.$

Лабораторна робота № 1

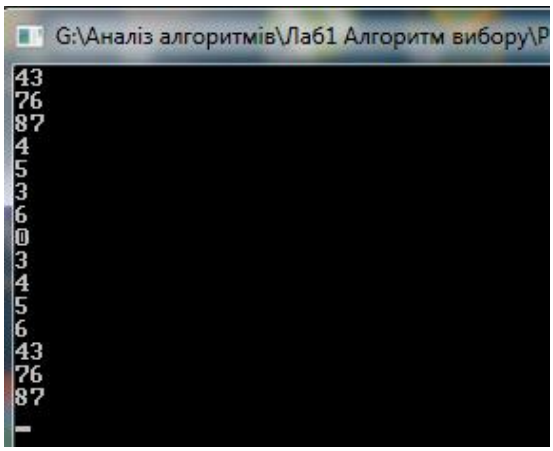
Впорядкування послідовності чисел.

Впорядкування (алгоритм бульбашки)




```
program Project2;
{$APPTYPE CONSOLE}
Uses SysUtils;
Const n=8 ;
Var   b,k,i:integer;
  A:array[1..n]of integer ;
begin
 { TODO -oUser -cConsole Main : Insert code here }
        for i:=1 to n do
                readln(a[i]);
                for k:=1 to n-1 do
                        for i:=1 to n-k do
                                if a[i] >a[i+1] then
                                        begin
                                                b:=a[i];
                                                a[i]:=a[i+1];
                                                a[i+1]:=b ;
                                        end;
        for i:=1 to n do
                writeln (a[i]) ;
        readln;
end.
```

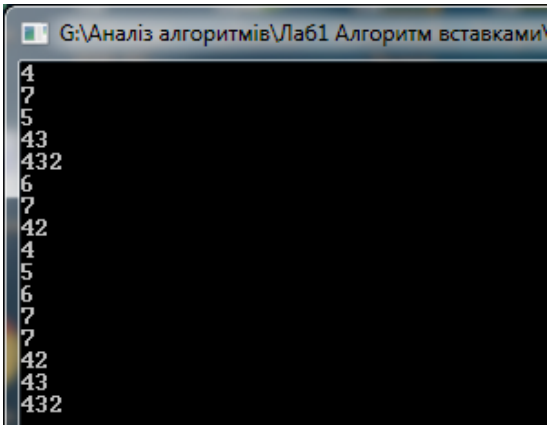
Впорядкування (алгоритм вибору)




```
program Project1;
{$APPTYPE CONSOLE}
Uses  SysUtils;
Const     n=8 ;
Var  b,k,j,i,min,index:integer;
        A:array[1..n] of integer ;
begin
 { TODO -oUser -cConsole Main : Insert code here }
        for k:=1 to n do
                readln (a[k]) ;
```

```
    for i:=0 to n-1 do
        begin
            min:=a[i];
            index:=i;
            for j:=i+1 to n do
                if a[j]<min then
                    begin
                        min:=a[j];
                        index:=j;
                    end;
                b:=a[i];
                a[i]:=a[index];
                a[index]:=b;
        end;
    for k:=1 to n do
        writeln(A[k]);
    readln;
end.
```

Впорядкування (алгоритм вставками)

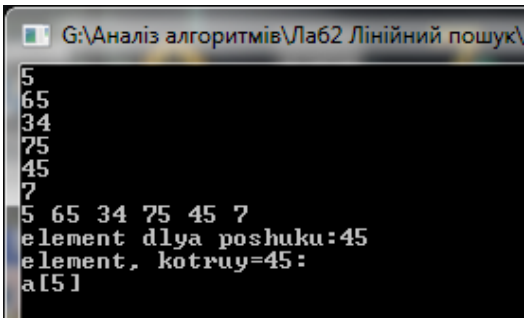



```
program Project1;
{$APPTYPE CONSOLE}
Uses  SysUtils;
const n=8;
var k,i,j,key: integer;
a: array [1..n] of integer;
begin
    for k:= 1 to n do readln(a[k]);
    for i:=2 to n do
        begin
            key:=a[i];
            j:=i-1;
            while (j>=0) and (a[j]>key) do
                begin
                    a[j+1]:= a[j];
                    j:=j-1;
                end;
            a[j+1]:=key;
```

```
            end;
        for i:=1 to n do
            writeln (a[i]);
        readln;
end.
```

Лабораторна робота № 2

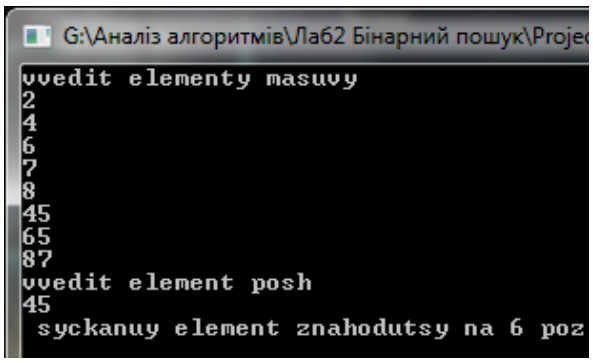
Пошук номера елемента послідовності.



Лінійний пошук

```
program Project2;
{$APPTYPE CONSOLE}
Uses  SysUtils;
const n=6;
    var     i:byte;     k,m:integer;
    a:array[1..n] of integer;
begin
  { TODO -oUser -cConsole Main : Insert code here }
        for i:=1 to n do
            readln(a[i]);
        for i:=1 to n do
            write(a[i],' ');
        writeln  ;
        write('element dlya poshuku:');
        readln(k);
        writeln('element, kotruy=',k,':');
        m:=0;
        for i:=1 to n do
            if a[i]=k then
                begin
                    writeln('a[',i,']');
                    m:=m+1;
                end;
        if m=0 then
            write('takuh elementiv nemae');
        readln
end.
```

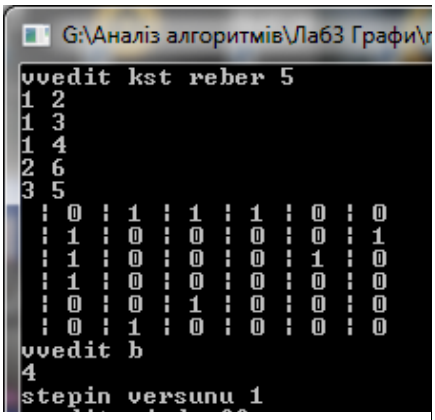
Бінарний пошук (для впорядкованого масиву)

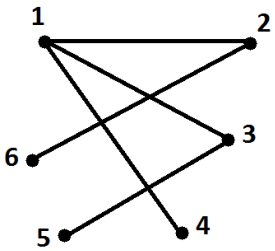



```
program Project2;
{$APPTYPE CONSOLE}
Uses  SysUtils;
const n=8;
var a:array [1..n] of integer;
b,p,i,j,k,c,x,y,min,index:integer;
s:boolean ;
begin
        writeln (‘vvedit elementy masuvy’);
        for i:=1 to n do
                readln (a[i]);
        writeln (‘vvedit element posh’);
        readln(p);
        s:=true;
        x:=1;
        y:=n;
        while (x<=y) and s do
                begin
                        c:=(x+y) div 2;
                        if a[c]=p then s:=false else
                        if a[c]<p then x:=c+1 else
                        if a[c]>p then y:=c-1
                end;
        if a[c]=p then writeln (‘ syckanuy element znahodutsy na ‘ , c ,’ poz’)
                else writeln (‘syckanogo elementa nemay y poslidovnosci’);
        readln;
end.
```

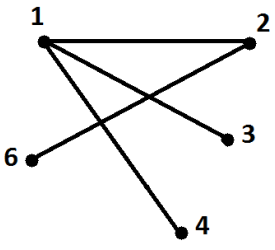
Лабораторна робота № 3

Побудова графа в пам'яті комп'ютера. Обчислення степеня вибраної вершини графа. Визначення того, чи є граф підграфом даного графа.





Граф



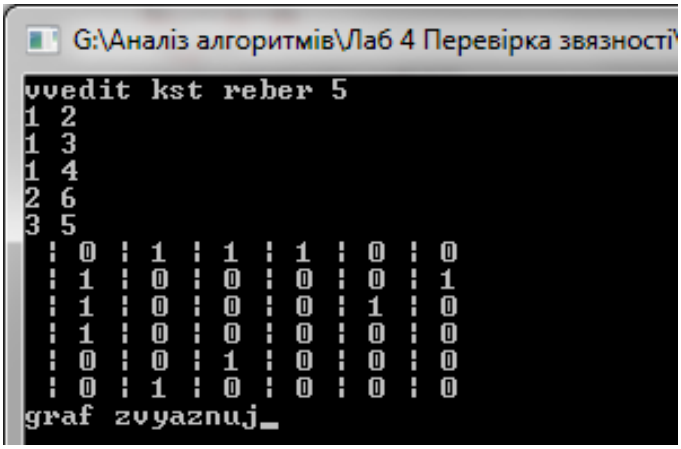
Підграф

```
program n_2;
{$APPTYPE CONSOLE}
Uses  SysUtils;
const n=6;
type graf= array[1..n,1..n] of byte;
var g,h:graf; u,v:byte; k,b,p,q,y,o,x,i,j:byte;
procedure make_G (var g:graf);
var   i,j,k,m:integer;
begin
        for i:=1 to n do
                for j:=1 to n do
                        g[i,j]:=0 ;
                        write('vvedit   kst
                reber ');
                        readln(m);
                        for i:=1 to m do
                                begin
                                        readln(u,v);
                                        g[u,v]:=1;
                                        g[v,u]:=1;
                                end;
end;
procedure stepin_g (g:graf;b:byte;var k:byte);
var j:integer;
begin
        for j:=1 to n do
                if g[b,j]=1 then k:=k+1;
end;
begin
        make_G(g);
        for i:=1 to n do
                begin
                        for j:=1 to n do
                                write(' | ',g[i,j])    ;
```

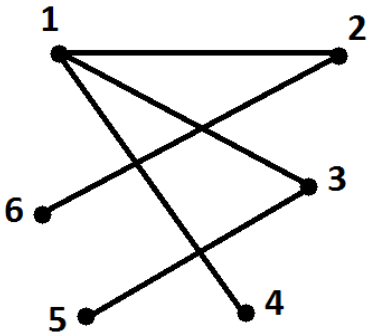
```
                writeln;
            end;
        writeln(‘vvedit b’);
        readln(b);
        stepin_g (g,b,k);
        writeln(‘stepin versunu ‘,k);
        writeln(‘vvedit pigdraff’);
begin
        for i:=1 to n do
            for j:=1 to n do
                h[i,j]:=0 ;
        write(‘vvedit kst reber grafa h ‘);
        readln(q);
        for i:=1 to q do
            begin
                readln(u,v);
                h[u,v]:=1;
                h[v,u]:=1;
            end;
        p:=0;
        o:=0;
        for i:=1 to n do
            for j:=1 to n do
                if (g[i,j]=1) and (h[i,j]=1)thenp:=p+1;
        if p<>(2*q) then writeln (‘H ne pidgraff A’) else writeln (‘H pidgraff a’);
        readln;
end;
end.
```

Лабораторна робота № 4

Перевірка зв’язності графа



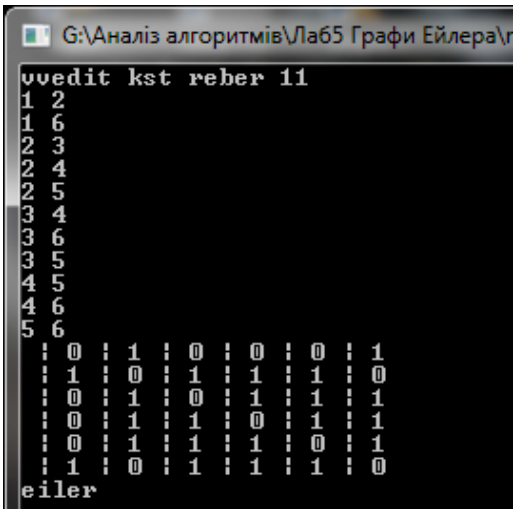

```
program n_2;
{$APPTYPE CONSOLE}
Uses  SysUtils;
const n=6;
type
link=^zap;
zap=record
        inf:byte;
        next:link;
end;
graf= array[1..n,1..n] of byte;
mas=array[1..n] of integer ;
var g,h:graf; u,v:byte; k,b,p,q,y,o,x,i,j:byte; a, m:mas; l:boolean;
procedure make_G (var g:graf);
var i,j,k,m:integer;
begin
        for i:=1 to n do
                for j:=1 to n do
                        g[i,j]:=0 ;
                write(‘vvedit kst reber ‘);
        readln(m);
        for i:=1 to m do
                begin
                        readln(u,v);
                        g[u,v]:=1;
                        g[v,u]:=1;
                end;
end;
procedure poshuk (g:graf; v0:integer; var a:mas);
var i:integer;l,r: link;
begin
        for i:=1 to n do
                a[i]:=-1;
        a[v0]:=0;
        new(l);
        l^.inf:=v0;
        l^.next:=nil;
```



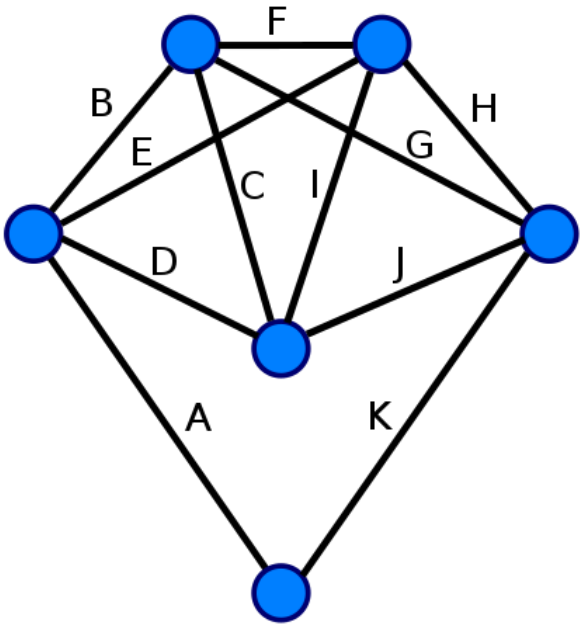
Граф

```
        r:=l;
        while r<>nil do
                begin
                        for i:=1 to n do
                                if (g[r^.inf,i]=1)and (a[i]=-1) then
                                        a[i]:=a[r^.inf]+1;
                        r:=r^.next;
                end;
end;
procedure zv_G(g:graf; var l:boolean);
var m:mas;i:integer;
begin
        poshuk(g,1,a);
        l:=true;
        for i:=1 to n do
                if a[i]=1 then
                        begin
                                l:=false;
                                exit;
                        end;
end;
begin
        make_G(g);
        for i:=1 to n do
                begin
                        for j:=1 to n do
                                write(‘ | ‘,g[i,j]) ;
                        writeln;
                end;
begin
        zv_G(G,l);
        if m[i]=1 then
        write (‘graf ne zvyaznuj’)
                else write (‘graf zvyaznuj’);
        readln;
end;
end.
```

Лабораторна робота №5



Перевірити, чи є заданий граф ейлеровим.




```
program n_2;
{$APPTYPE CONSOLE}
Uses  SysUtils;
const n=6;
type graf=array[1..n,1..n] of byte;
var   g,h:graf;   u,v:byte;   k,b,p,q,y,o,x,i,j:byte;
p1:boolean;
procedure make_G (var g:graf);
var   i,j,k,m:integer;
begin
        for i:=1 to n do
                for j:=1 to n do
                        g[i,j]:=0 ;
        write('vvedit kst reber ');
        readln(m);
        for i:=1 to m do
                begin
                        readln(u,v);
                        g[u,v]:=1;
                        g[v,u]:=1;
                end;
end;
procedure stepin_g (g:graf;b:byte;var k:byte);
var j:integer;
begin
        for j:=1 to n do
                if g[b,j]=1 then k:=k+1;
end;
procedure chu_pohogi(g:graf);
var l,i,j:integer;
begin
        for i:=1 to n do
                begin
                        l:=0;
                        for j:=1 to n do
```

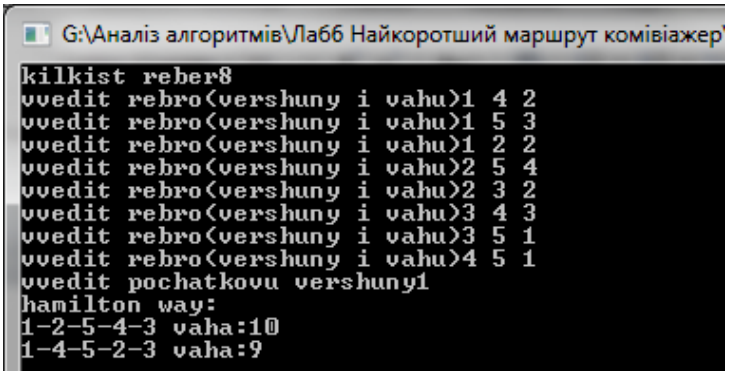
```
                    if g[i,j]=1 thenl:=l+1;
                if l mod 2 =0 thenp1:=true
                    else
                        begin
                            p1:=false;
                            break;
                        end;
            end;
end;
begin
    make_G(g);
    for i:=1 to n do
        begin
            for j:=1 to n do
                write(‘ | ‘,g[i,j])    ;
            writeln;
        end;
    chu_pohogi(g);
    if p1=true then  writeln(‘eiler’)
        else  writeln(‘ne eiler’);
readln;
end.
```

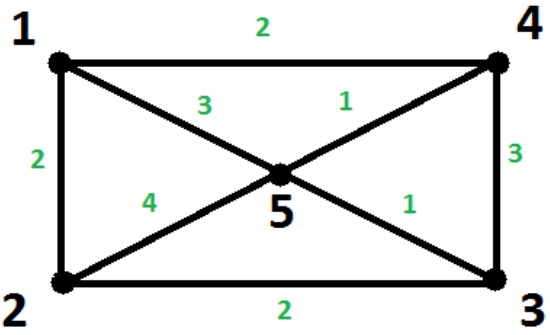
Лабораторна робота № 6

Знаходження найкоротшого маршруту комівояжера.

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const n=7;
type
    Graf=array[1..n,1..n] of byte;
    Strin=string[2*n+1];
    Stek=^Stek1;
     Stek1=record
                strit:Strin;
                waga,b:byte;
                next:Stek
                end;
var stt,tt,s,stk,nnn:Strin; graph:Graf;  r,l,min1:Stek;  k,j,i,a:byte;  t,c:boolean;

procedure Make_Gr(var graph:Graf);
 var k_reb,w,u,v:byte;
    begin
        for i:=1 to n do
            for j:=1 to n do
                graph[i,j]:=0;
        write('kilkist                reber');
readln(k_reb);
        for i:=1 to k_reb do
            begin
             write ('vvedit rebro(vershuny i vahu)'); readln(u,v,w);
             graph[u,v]:=w; graph[v,u]:=w;
            end;
    end;
procedure Add (stt:Strin; wt, bt:byte; var top:Stek);
var p:Stek;
    begin
        new(p);
        p^.strit:=stt; p^.waga:=wt; p^.b:=bt;  p^.next:=top; top:=p;
```





```
   end;
procedure Min_El (top:Stek;var min:Stek);
 var p:Stek;
   begin
        min:=top;  p:=top^.next;
        while p<>nil do
         begin
          if p^.waga<min^.waga then  min:=p;
          p:=p^.next
         end;
   end;
procedure Del_El (q:Stek; var p:Stek);
var q1,q2:Stek;
   begin
       if p=q then
          begin  p:=q^.next; exit;  end;
       q1:=p; q2:=p^.next;
       while q2<>q do
          begin  q1:=q2; q2:=q2^.next  end;
       q1^.next:=q2^.next
    end;
procedure Perev( var p:stek);
 var q,q1:stek;k:byte;
   begin
        q:=p;
        while q<>nil do
             begin
              if q^.b=a then
               begin
                 k:=length(q^.strit);
                 if k<(2*n+1) then Del_El(q,p);
                 if k=(2*n+1) then
                   begin
                     q1:=p;
                     while q1<>nil do
                         begin
                          if q1^.waga>q^.waga then  Del_El(q1,p);
```

```
                    q1:=q1^.next
                   end;
               end;
            end;
           q:=q^.next
           end
   end;
procedure street (graph:Graf);
 begin
   write (‘vvedit pochatkovu vershuny’); readln(a);
   str(a,tt);
   r:=nil; l:=nil;
   Add(tt,0,a,r);
   while r<>nil do
       begin
        Min_El(r,min1); stt:=min1^.strit;
        t:=true;
        for i:=1 to n do
           begin
            str(i,s);
            c:=true;
            for j:=2 to length(stt) do
               if stt[j]=s then c:=false;
               if c and (graph[min1^.b,i]<>0) then
                  begin
                   stk:=stt+’-’+s;
                   Add(stk,min1^.waga+graph[min1^.b,i],i,r);
                   t:=false;
                  end;
            end;
        if t then Add(min1^.strit,min1^.waga,min1^.b,l);
        Del_El(min1,r);
        Perev(r);
       end;
   writeln(‘hamilton way: ‘);
   while l<>nil do
    begin
```

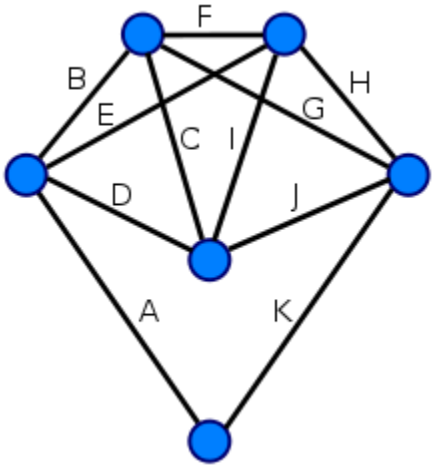
```
        writeln(l^.strit,' vaha:',l^.waga);
        l:=l^.next
       end;
end;
begin
    Make_Gr(graph);
    street(graph);
    readln
end.
```

Лабораторна робота № 7

Знаходження остова мінімальної ваги.

Алгоритм Краскала

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const roz=6;
type mat=array[1..roz,1..roz]  of real;
var a,b:mat; n:integer;
procedure make (var a:mat; var n:integer);
var r,i,j,u,v:integer; w:real;
begin
        write('vvedit kilkist vershyn');
        read(n);
        for i:=1 to n do
                for j:=1 to n do
                        a[i,j]:=0;
        writeln ('vvedit kilkist reber');
        read(r);
        for i:=1 to r do
```



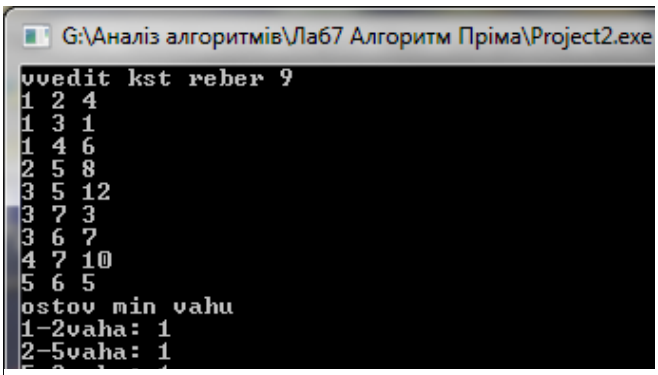
```
            begin
                writeln (‘vvedit rebro ‘);
                readln (u,v,w);
                a[u,v]:=w;
                a[v,u]:=w;
            end;
end;
procedure form (n:integer; a:mat; var b:mat);
var i,j,u,v:integer; p:real;  m:set of 1..roz;
begin
    for i:=1 to n do
            for j:=1 to n do b[i,j]:=0;
                i:=1;m:=[];
            while i<=n do
                begin
                    j:=1;
                    while j<=n do
                        begin
                            if a[i,j]<>0 then
                                begin
                                    p:=a[i,j];u:=i;v:=j;
                                    i:=n+1;j:=n+1;
                                end;
                            j:=j+1;
                        end;
                    i:=i+1;
                end;
    for i:=1 to n do
            for j:=1 to n do
                if (a[i,j]<p) and (a[i,j]<>0) then
```

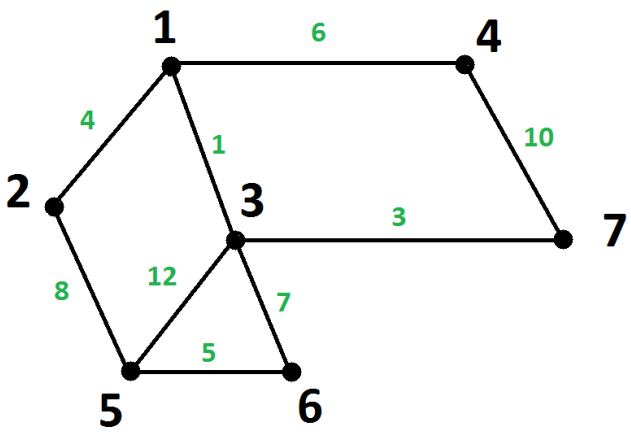
```
                begin
                        p:=a[i,j];
                        u:=i;
                        v:=j;
                end;
a[u,v]:=0;a[v,u]:=0;
b[u,v]:=p;b[v,u]:=p;
m:=m+[u]+[v];
writeln('rebro',u,'-',v,'maje vahu',p:3);
while true do
        begin
                i:=1;j:=2;p:=0;
                while i<=n do
                        begin
                                j:=1;
                                while j<=n do
                                        begin
                                                if (a[i,j]<>0) and (i in m) and (not(j in
                                                m)) then
                                                        begin
                                                                p:=a[i,j];
                                                                u:=i;v:=j;
                                                                i:=n+1;j:=n+1;
                                                        end;
                                                j:=j+1;
                                        end;
                                i:=i+1;
                        end;
                if p=0 then exit;
                for i:=1 to n do
```

```
                    for j:=1 to n do
                        if (a[i,j]<p) and (i in m) and (not(j in m)) and
                        (a[i,j]<>0) then
                            begin
                                p:=a[i,j];
                                u:=i;v:=j;
                            end;
                    a[u,v]:=0;a[v,u]:=0;
                    b[u,v]:=p;b[v,u]:=p;
                    m:=m+[u]+[v];
                    writeln('rebro',u,'-',v,'maje vahu',p:3);
                end;
            end;
begin
    make(a,n);
    writeln('ostov min vahy');
    form(n,a,b);
    readln
end.
```

Алгоритм Пріма

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const n=7;
type
   Graf=array [1..n,1..n] of byte;
   link=^zap;
   zap=record
   inf:integer;
```





```
    next:link;
    end;
    mas=array [1..n]of integer;
var  g,t:Graf;
i,j,z:integer;
procedure make_G (var g:graf);
var   i,j,k,m,u,v:integer;
begin
        for i:=1 to n do
                for j:=1 to n do
                        g[i,j]:=0 ;
        write(‘vvedit kst reber ‘);
        readln(m);
        for i:=1 to m do
                begin
                        readln(u,v);
                        g[u,v]:=1;
                        g[v,u]:=1;
                end;
end;
procedure app(u:integer; var left, right:link);
var p:link;
begin
        new(p);
        p^.inf:=u;
        p^.next:=nil;
        left^.next:=p;
        left:=p;
end;
procedure poshuk (g:graf; u:integer; var a:mas);
```

```
var p,left,right:link;
begin
        new(p);
        p^.inf:=u;
        p^.next:=nil;
        left:=p;
        right:=p;
        a[u]:=0;
        while right<>nil do
                begin
                        for i:=1 to n do
                                if (g[right^.inf,i]=1)and(a[i]=-1) then
                                        begin
                                                a[i]:=a[right^.inf]+1;
                                                app(i,left,right);
                                        end;
                        right:=right^.next;
                end;
end;
procedure min_el1(g:graf;i:integer;var j:integer);
var k,min:integer;
begin
        min:=99;
        for k:=1 to n do
                if (g[i,k]<>0)and (g[i,k]<min) then
                        begin
                                min:=g[i,k];
                                j:=k;
                        end;
end;
```

```
procedure min_el(g:graf; var k,l:integer);
var i,j,min: integer;
begin
        min:=g[1,2];
        k:=1;
        l:=2;
        for i:=1 to n do
                for j:=1 to n do
                        if (g[i,j]<>0)and(g[i,j]<min) then
                                begin
                                        min:=g[i,j];
                                        k:=i;
                                        l:=j;
                                end;
end;
function perev (var g:graf; k,l:integer):boolean;
var a:mas;
begin
        for i:=1 to n do
                a[i]:=-1;
        poshuk(g,k,a);
        if a[l]=-1 then perev:=true else perev:=false;
end;
procedure alg_prima (g:graf; var t:graf);
var k,l,v1,v:integer;
begin
        for i:=1 to n do
                for j:=1 to n do
                        t[i,j]:=0;
        min_el(g,k,l);
```

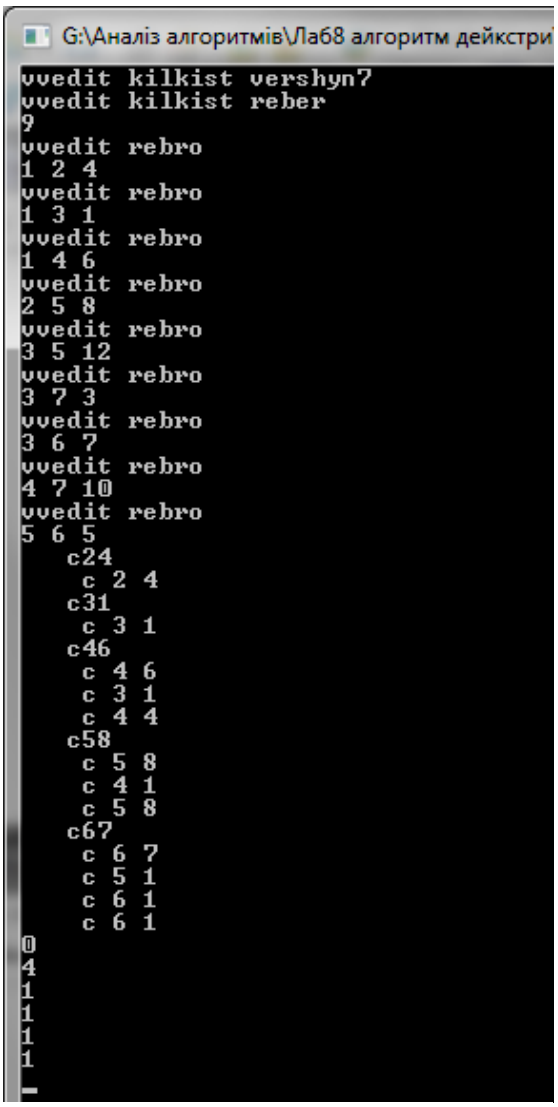
```
t[k,l]:=1;
t[l,k]:=1;
writeln(k,'-',l,'vaha: ',g[k,l]);
z:=1;
g[k,l]:=0;
g[l,k]:=0;
while z<n do
        begin
                min_el1(g,k,v);
                min_el1(g,l,v1);
                if (((perev(t,k,v))and(perev(t,l,v1)))and(g[k,v]>g[l,v1]))
                        or ((perev(t,k,v))and (not(perev(t,k,v1))))then
                                begin
                                        t[k,v]:=1;
                                        t[v,k]:=1;
                                        writeln(k,'-',v,'vaha: ',g[k,v]);
                                        g[k,v]:=0;
                                        g[v,k]:=0;
                                        z:= z+1;
                                        k:=v;
                                end
                                        else
                                                if((perev(t,k,v))and       (perev(t,l,v1)))
                                        then
                                                        begin
                                                        t[l,v1]:=1;
                                                        t[v1,l]:=1;
                                                        writeln(l,'-',v1,'vaha: ',g[l,v1]);
                                                        g[l,v1]:=0;
g[v1,l]:=0;
```

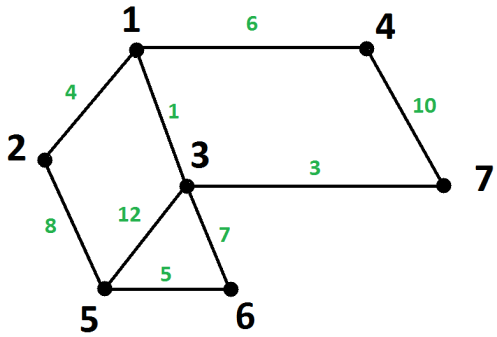
```
        z:=z+1;
        l:=v1;
        end
        else
                begin
                        g[l,v1]:=0;
                        g[v1,l]:=0;
                        z:=z+1;
                end ;
        end;
end;
begin
        make_g(g);
        writeln(‘ostov min vahu’);
        alg_prima(g,t);
        readln;
        readln;
end.
```

Лабораторна робота № 8

Алгоритм Дейкстри

```
program Project3;
{$APPTYPE CONSOLE}
Uses  SysUtils;
const roz=6;
type  mat=array[1..roz,1..roz] of integer;
        nag=record
                sh:integer;
                ver:integer;
        end;
        ty=array[1..roz]of nag;
var a:mat; n,i:integer; c: ty;
procedure DX3(var c:ty; a:mat);
var i,j:integer;
begin
        n:=roz;
        for i:=1 to roz do
                begin
                        c[i].ver:=i;
                        c[i].sh:=0;
                end;
        for i:=1 to n do
                for j:=1+i  to n do
                        begin
                                if (c[j].sh=0)and(a[i,j]<>0) then
                                        begin
                                                c[j].sh:=a[i,j];
                                                writeln('  ','c',j,c[j].sh)
                                        end;
                                if c[j].sh<>0 then
                                        begin
```

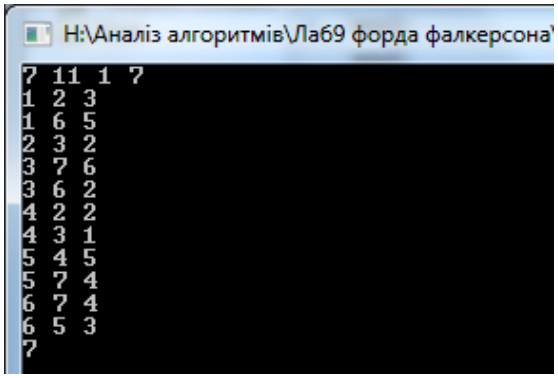




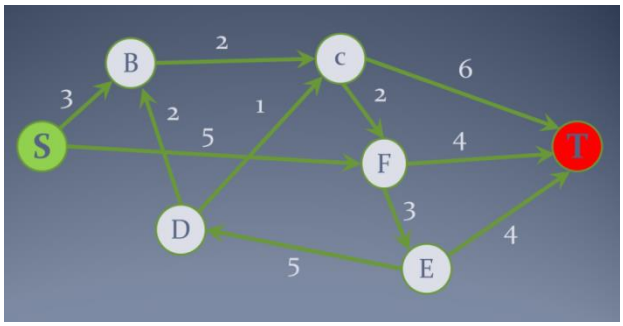
```
                                        if (a[i,j]+c[i].sh <c[j].sh) then
                                            c[j].sh:=a[i,j]+c[i].sh;
                                        writeln('    ','c ',j,' ',c[j].sh)
                                    end;
                                end;
end;
procedure make(var a:mat; var n:integer);
var r,i,j,u,v:integer; w:integer;
begin
    write('vvedit kilkist vershyn');
    read(n);
    for i:=1 to n do
        for j:=1 to n do
            a[i,j]:=0;
    writeln('vvedit kilkist reber');
    read(r);
    for i:=1 to r do
        begin
            writeln('vvedit rebro ');
            readln(u,v,w);
            a[u,v]:=w;
            a[v,u]:=w;
        end;
end;
begin
    n:=roz;
    make(a,n);
    DX3(c,a);
    for i:=1 to n do
    writeln(c[i].sh);
    readln
end.
```

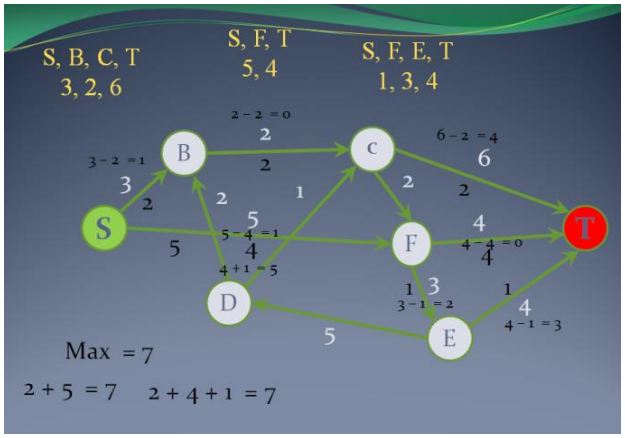
Лабораторна робота № 9

Пошук максимально можливого потоку.



Алгоритм Форда —Фалкерсона






```
program Project1;
{$APPTYPE CONSOLE}
uses  SysUtils;
constmaxn = 1000;
infinity = maxlongint;
        varn,m,vin,vout:longint;i,u,v,w,
head,tail,ans:longint;
ne,p,flow:array[1..maxn] of longint;
e,c,f:array[1..maxn,1..maxn] of longint;
q:array[0..maxn] of longint;
begin
        read(n,m,vin,vout);
        for i:=1 to m do
                begin
                        read(u,v,w);
                        if c[v,u]=0 then
                                begin
                                        inc(ne[u]); e[u,ne[u]]:=v;
                                        inc(ne[v]); e[v,ne[v]]:=u;
                                end;
                        c[u,v]:=w;
                end;
        repeat
                p[vout]:=-1;
                fillchar(flow,sizeof(flow),0);
                flow[vin]:=infinity;
```

```
        head:=0; tail:=1; Q[0]:=vin;
        while head<tail do
                begin
                        u:=Q[head]; inc(head);
                        for i:=1 to ne[u] do
                                begin
                                        v:=e[u,i];
                                        if (c[u,v]-f[u,v]>0) and (flow[v]=0) then
                                                begin
                                                        Q[tail]:=v; inc(tail);
                                                        p[v]:=u;
                                                        if    c[u,v]-f[u,v]<flow[u]    then
                                                                flow[v]:=c[u,v]-f[u,v]
                                                                        else
                                                                flow[v]:=flow[u];
                                                        if v=vout then break;
                                                end;
                                        end;
                                end;
                        if p[vout]=-1 then break;
                        u:=vout;
                        while u<>vin do
                                begin
                                        f[p[u],u]:=f[p[u],u]+flow[vout];
                                        u:=p[u];
                                end;
                        ans:=ans+flow[vout];
until false;
```

```
        write(ans);readln; readln;readln;
end.
```

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вирт. Н. Алгоритмы + структуры данных = программы. — М.: Мир, 1985. — 406 с.
2. Кнут Д. Искусство программирования для ЭВМ. — В 3-х т. Т. 3. Сортировка и поиск. — М. : Мир, 1977.
3. Кормен Т. и др. Алгоритмы : построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. — М. : МЦНМО, 2001. — 960 с.

Навчальне видання

Грод *Інна Миколаївна*

Мартинюк *Сергій Володимирович*

Мартинюк *Олеся Миронівна*

АНАЛІЗ ЕФЕКТИВНОСТІ ДЕЯКИХ АЛГОРИТМІВ

ТЕОРІЯ І ПРАКТИКА

Навчальний посібник

Виготовлено згідно із СОУ 22.2-02477019-07:2012

Формат 60×84/16. 3,73 ум. др. арк., 3,47 обл.-вид. арк.
Тираж 300. Замовлення № 17-26.