МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

МЕТОДИЧНІ ВКАЗІВКИ І ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ ЗА КУРСОМ «ДИСКРЕТНА МАТЕМАТИКА»

(для студентів спеціальності "Комп’ютерна інженерія")

Розглянуто
на засіданні кафедри ВТ
протокол №__ від 2013 р.

Затверджено
на засіданні навчально-
видавничої ради НТУУ «КПІ»
протокол № від 2013 р.

- 2013-

УДК 681.3.06

Методичні вказівки до виконання лабораторних робіт з дисципліни «Комп’ютерна інженерія». /Укл.: Марковський О.Г, Новотарський М.А.: НТУУ «КПІ»- 2013, 46 с.

Методичні вказівки до виконання лабораторних робіт з дисципліни «Дискретна математика» містять завдання до лабораторних робіт. До кожної роботи наведено методичні рекомендації з теорії, яка використовується при виконанні лабораторних робіт, приведені деякі алгоритми вирішення завдань, вимоги до виконання роботи, індивідуальні завдання та контрольні питання.

Укладачі: Новотарський М.А.

Рецензент: Стиренко С.Г.

<u>**Лабораторна робота №1**</u>

<u>**Тема:**</u> «Множини: основні властивості та операції над ними, діаграми Венна».

<u>**Мета:**</u> вивчити основні аксіоми, закони і теореми теорії множин, навчитися застосовувати їх на практиці. Виконати наступні операції над множинами: доповнення множин; об’єднання, перетин, різниця, симетрична різниця

<u>**Завдання:**</u> написати програму для виконання даних операцій над множинами.

<u>**Теоретичні основи:**</u>

1.1. Властивості множин

Множина **-** є сукупність визначених об’єктів, різних між собою, об’єднаних за певною ознакою чи властивістю.

Множини позначаються **великими** латинськими буквами. Об’єкти, що складають множини, називаються елементами і позначаються **малими** буквами латинського алфавіту.

На практиці часто застосовують такі позначення:

N – множина натуральних чисел;

P – множина додатних цілих чисел;

Z – множина додатних і від’ємних цілих чисел, включаючи нуль;

Q – множина раціональних чисел;

R – множина дійсних чисел.

Якщо множина не містить жодного елемента, її називають порожньою і позначають Ø.

Скінченна множина – це така множина, кількість елементів якої може бути виражена скінченним числом, причому не важливо, чи можемо ми порахувати це число в даний момент.

Нескінченна множина – це така множина, що не є скінченна.

Способи завдання множин:

- ***перерахуванням,*** тобто списком всіх своїх елементів. Такий спосіб завдання прийнятний тільки при завданні кінцевих множин. Позначення списку – у фігурних дужках. Наприклад, множина, що є з перших п’яти простих чисел $A=\{2,3,5,7,11\}$. Множина спортсменів університетської хокейної команди: $B=\{Іванов, Петров, Сидоров, Бубликов, Сироєжкін, Волосюк\}$;

- ***процедурою***, що породжує і описує спосіб одержання елементів множини із уже отриманих елементів або з інших об’єктів. Наприклад, множина усіх цілих чисел, що є степенями двійки $M_{2^n}, n \in N$, де N - множина натуральних чисел, може бути представлена породжуючою процедурою, заданою двома правилами, названими рекурсивними: а) $1 \in M_{2^n}$; б) якщо $m \in M_{2^n}$, тоді $2m \in M_{2^n}$;

- ***описом характеристичних властивостей***, якими повинні володіти елементи множини. Так, множина A, що складається з таких елементів x, які мають властивість $P(x)$, позначимо в такий спосіб:

$$A = \{x | P(x)\}.$$

Так, розглянута вище множина всіх цілих чисел, що є степенями числа 2 може бути записана як $A = \{x | x = 2^n, n \in N\}$. До A ще треба додати 1.

Якщо елемент a належить множині A, то пишуть $a \in A$. Якщо a не є елементом множини A, то пишуть $a \notin A$. Наприклад, $5 \in \{1,3,5,7\}$, але $4 \notin \{1,3,5,7\}$. Якщо $A = \{x | \textit{студентки групи } MM_{21}\}$, то *Іванова* A, а *Петров* $\notin A$.

Підмножина. Множину A називають підмножиною (або *включенням*) множини B $(A \subseteq B)$, якщо кожен елемент множини A є елементом множини B, тобто, якщо $x \in A$, то $x \in B$. Якщо $A \subseteq B$ й $A \neq B$, то A називається строгою підмножиною й позначається $A \subset B$.

Рівність множин. Дві множини рівні $(A = B)$, якщо всі їхні елементи збігаються. Множини A і B рівні, якщо $A \subseteq B$ і $B \subseteq A$.

Потужність множини. Кількість елементів у скінченій множині A називається *потужністю* множини A і позначається $|A|$.

Універсальна множина U є множина, що володіє такою властивістю, що всі розглянуті множини є його підмножинами.

Варто розрізняти поняття належності елементів множини і включення! Так, наприклад, якщо множина $A = \{1, 3, 6, 13\}$, то $3 \in A$, $6 \in A$, але $\{3, 6\} \notin A$, у той час як $\{3, 6\} \subseteq A$.

Булеан. Множина всіх підмножин, що складаються з елементів множини A, називається *булеаном* $P(A)$.

Приклад булеану. Нехай $A = \{a, b, c, d\}$. Визначити булеан множини A. Яка потужність множини $P(A)$?

Розв'язок:

$$P(A) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}, \{a, b, c\},$$
$$\{a, b, d\}, \{a, c, d\}, \{b, c, d\}, \{a, b, c, d\}\}.$$

Потужність $|P(A)| = 16$.

1.2. Операції над множинами

Об'єднання. *Об'єднанням* множин A і B називається множина, що складається із всіх тих елементів, які належать хоча б однієї з множин A або B. Об'єднання множин A і B позначається $A \cup B$. Це визначення рівносильне наступному: $A \cup B = \{x | x \quad A$ *або* $x \quad B\}$.

Приклад. Нехай $A=\{2, 3, 5, 6, 7\}$, $B=\{1, 2, 3, 7, 9\}$. Знайти $A\cup B$.

Розв'язок: $A\cup B=\{1, 2, 3, 5, 6, 7, 9\}$.

Перетин. *Перетином* множин A і B називається множина, що складається із всіх тих елементів, які належать і множині A й множині B. Перетин множин A і B позначається $A\cap B$. Це визначення рівносильне наступному: $A\cap B=\{x | x \quad A\ \ i\ \ x \quad B\}$.

Приклад. Нехай $A=\{2, 3, 5, 6, 7\}$, $B=\{1, 2, 3, 7, 9\}$. Знайти $A\cap B$.

Розв'язок: $A\cap B=\{2, 3, 7\}$.

Доповнення. *Доповненням (*або *абсолютним доповненням)* множини A називається множина, що складається із всіх елементів універсальної множини, які не належать A. Доповнення множини A позначається $\overline{A}$. Це визначення рівносильне наступному: $\overline{A}=U-A=\{x | x\in U \ \ и \ \ x\notin A\}$.

Різниця. *Різницею* множин A й B *(*або *відносним доповненням)* називається множина, що складається із всіх елементів множини A, які не належать B Різницю множин A і B позначають $A-B$. Це визначення рівносильне наступному: $A-B=\{x | x\in A \ \ и \ \ x\notin B\}$.

Приклад. Нехай $A=\{2, 3, 5, 6, 7\}$, $B=\{1, 2, 3, 7, 9\}$. Знайти $A-B$.

Розв'язок: $A-B=\{5, 6\}$.

Симетрична різниця. *Симетричною різницею* множин A і B називається множина, що складається з об'єднання всіх елементів, що належать множині A і не містяться в B, і елементів, що належать множині B і не містяться в A. Симетрична різниця множин A і B позначається $A+B$. Це визначення рівносильне наступному: $A\Delta B=(A-B)\cup(B-A)$.

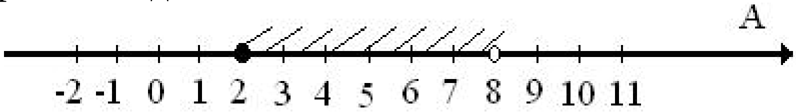
Види операцій. Операції, які виконують над однією множиною, називають *унарними*. Операції, які виконують над двома множинами, називають *бінарними*. Прикладом унарної операції є знаходження доповнення. Прикладами бінарних операцій є об'єднання, перетинання, різниця, симетрична різниця.

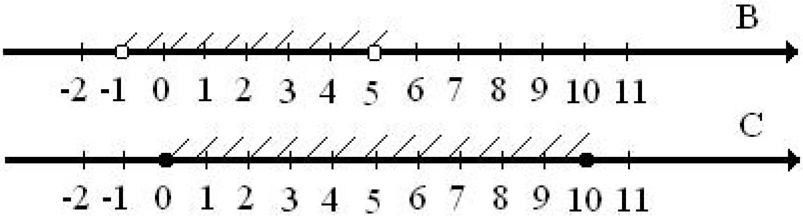
Приклад. Нехай $A=\{2, 3, 5, 6, 7\}$, $B=\{1, 2, 3, 7, 9\}$. Знайти $A+B$.

Розв'язок: $A\Delta B=\{1,5, 6, 9\}$.

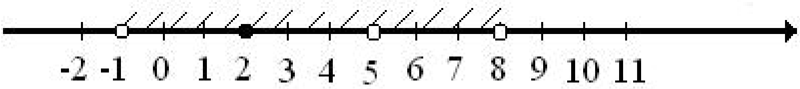
Приклад. Нехай $A=[2, 8)$, $B=(-1, 5)$; $C=[0, 10]$. Знайти $A\cup B$, $B\cap C$, $C-A$, $B\Delta C$, $\overline{B}$.

Розв'язок: Зобразимо задані множини на числовій осі

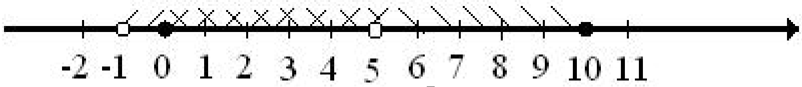




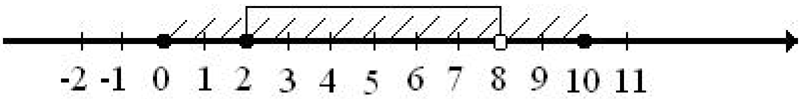
Тоді шукані множини будуть мати вигляд (рис. 1.1):



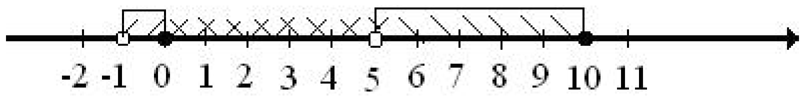
$$A \cup B = (-1, 8);$$



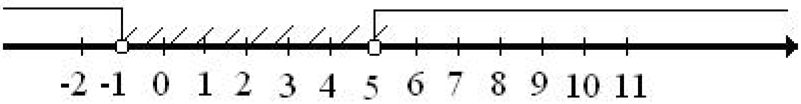
$$B \cap C = [0, 5);$$



$$C - A = [0, 2) \cup [8, 10];$$



$$B \,\Delta\, C = (-1, 0) \cup [5, 10];$$

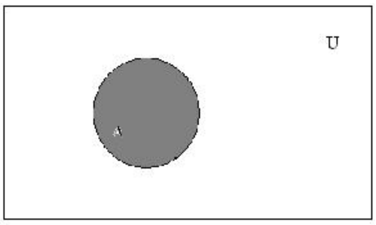


$$\overline{B} = (-\infty, -1] \cup [5, \infty).$$

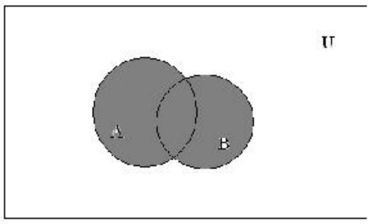
Рис. 1.1.

1.3. Діаграми Венна

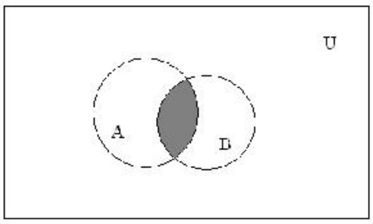
Для графічної ілюстрації відносин між множинами даної універсальної множини U використають діаграми Венна. Діаграма Венна – це зображення множини у вигляді геометричної множини, наприклад, кола. При цьому універсальну множину зображують у вигляді прямокутника. На рис. 1.2 зображені діаграми Венна для розглянутих операцій над множинами.

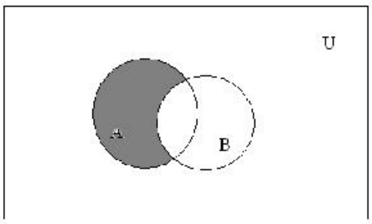


A

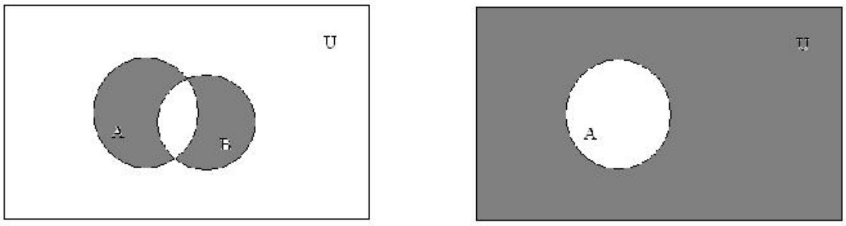


$A \cup B$





$A \cap B$ $\quad\quad\quad\quad$ $A - B$



$A \Delta B$ $\quad\quad\quad\quad$ $\overline{A}$

Рис. 1.2.

Для будь-яких підмножин A, B, C універсальної множини U справедливо наступне:

1.4. Тотожності алгебри можин

1. Комутативність об’єднання $X \cup Y = Y \cup X$	1. Комутативність перетину $X \cap Y = Y \cap X$
2. Асоціативність об’єднання $X \cup \left(Y \cup Z\right) = \left(X \cup Y\right) \cup Z$	2. Асоціативність перетину $X \cap \left(Y \cap Z\right) = \left(X \cap Y\right) \cap Z$
3. Дистрибутивність об’єднання відносно перетину $X \cup \left(Y \cap Z\right) = \left(X \cup Y\right) \cap \left(X \cup Z\right)$	3. Дистрибутивність перетину відносно об’єднання $X \cap \left(Y \cup Z\right) = \left(X \cap Y\right) \cup \left(X \cap Z\right)$
4. Закони дій з пустою та універсальною множинами $X \cup \varnothing = X$ $X \cup \overline{X} = U$ $X \cup U = U$	4. Закони дій з пустою та універсальною множинами $X \cap U = X$ $X \cap \overline{X} = \varnothing$ $X \cap \varnothing = \varnothing$
5.Закон ідемпотентності об’єднання Термін **ідемпотентність** означує властивість математичного об’єкта , яка проявляється в тому, що повторна дія над об’єктом *<u>не змін</u>ює* його $X \cup X = X$	5. Закон ідемпотентності перетину $X \cap X = X$
6. Закон де Моргана $\overline{X \cup Y} = \overline{X} \cap \overline{Y}$	6. Закон де Моргана $\overline{X \cap Y} = \overline{X} \cup \overline{Y}$
7. Закон поглинання $X \cup \left(X \cap Y\right) = X$	7. Закон поглинання $X \cap \left(X \bigcup Y\right) = X$
8. Закон склеювання $\left(X \cap Y\right) \cup \left(X \cap \overline{Y}\right) = X$	8. Закон склеювання $\left(X \cup Y\right) \cap \left(X \cup \overline{Y}\right) = X$
9. Закон Порецького	9. Закон Порецького

$X \cup \left(\bar{X} \cap Y \right) = X \cup Y$	$X \cap \left(\bar{X} \cup Y \right) = X \cap Y$
10. Закон подвійного доповнення $\bar{\bar{X}} = X$	

1.5. Вимоги до програмного забезпечення:

1. Лабораторна робота виконується в середовищі візуального програмування Lazarus.
2. В якості початкового коду використати проект LAB1_Project, попередньо скачавши його з сайту викладача.
3. Початкові данні вводяться за допомогою програмних засобів уже реалізованих в згаданому проекті.
4. Необхідно в рамках форми OperForm реалізувати алгоритм відповідно до варіанту лабораторної роботи.
5. На формі OperForm відобразити як початкові дані, так і результати роботи алгоритму.

Зміст звіту:

1. Титульний лист;
2. Тема завдання;
3. Завдання;
4. Теоретичні відомості по темі лабораторної роботи.
5. Блок-схема алгоритму;
6. Роздруківка тексту програми;
7. Роздруківка результатів виконання програми;
8. Аналіз результатів.

1.6. Контрольні питання.

1. Чим відрізняється множина від довільної сукупності елементів?

2. Назвіть способи задавання множин.

3. Які ви знаєте операції над множинами.

4. Чи може існувати множина, яка не містить жодного елемента.

5. Чи мають множини A і B однакові потужності, якщо A={1,9,25}, B={2300,25,1}?

6. Зобразіть діаграми Венна для вказаних викладачем операцій над двома множинами.

7. Запишіть вказане викладачем тотожне перетворення алгебри множин.

8. Дано дві множини $A=\{1,54,12,45,11,34\}$ і $B=\{2,11,12,13,45,54,34\}$ та результат операції над ними: $C=\{1,2,13\}$. Вкажіть цю операцію.

1.7. Блок-схеми алгоритмів виконання операцій над множинами

Доповнення множини

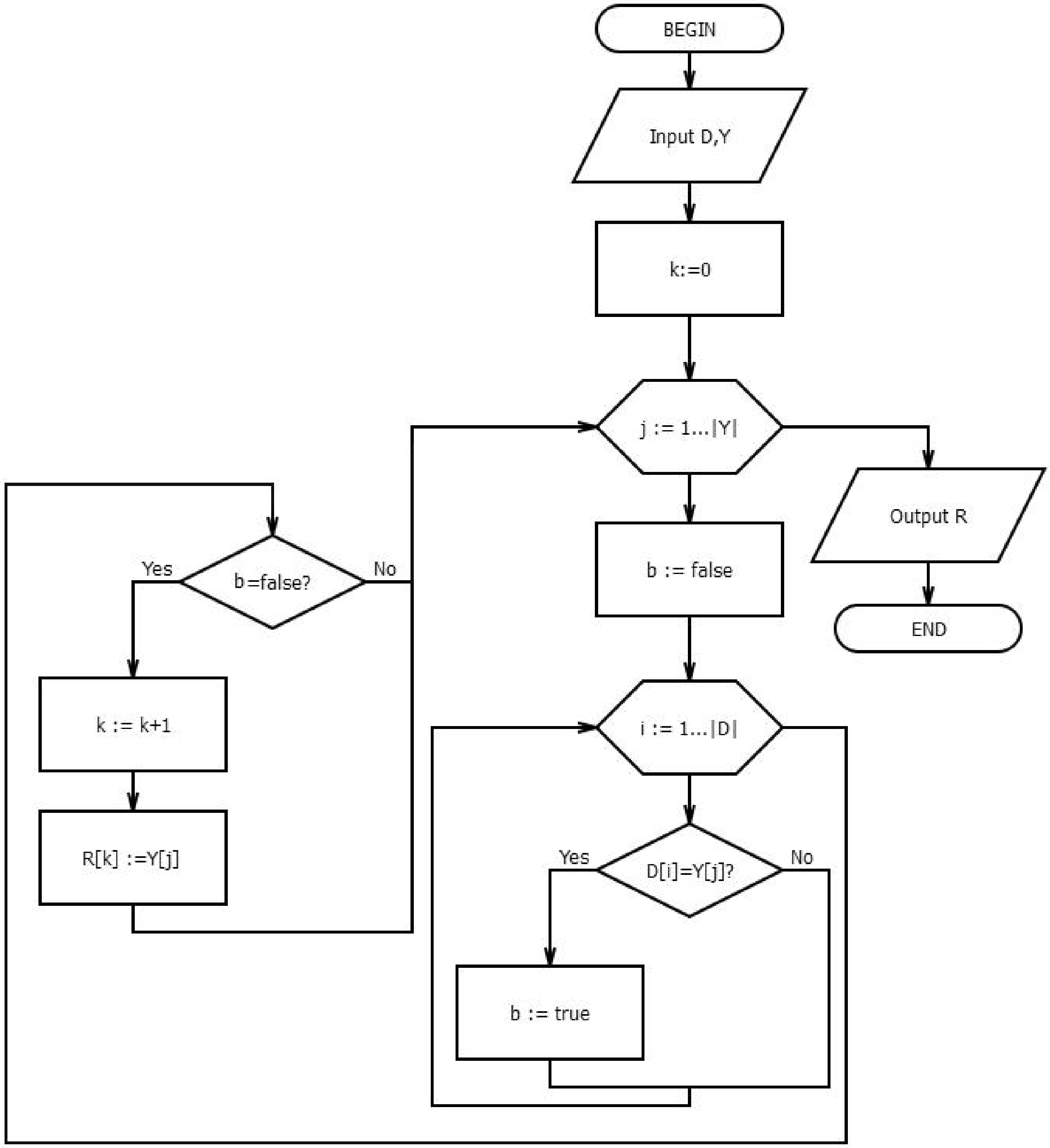


Рис.1.3 Блок-схема операції доповнення множини до універсальної

Об’єднання двох множин

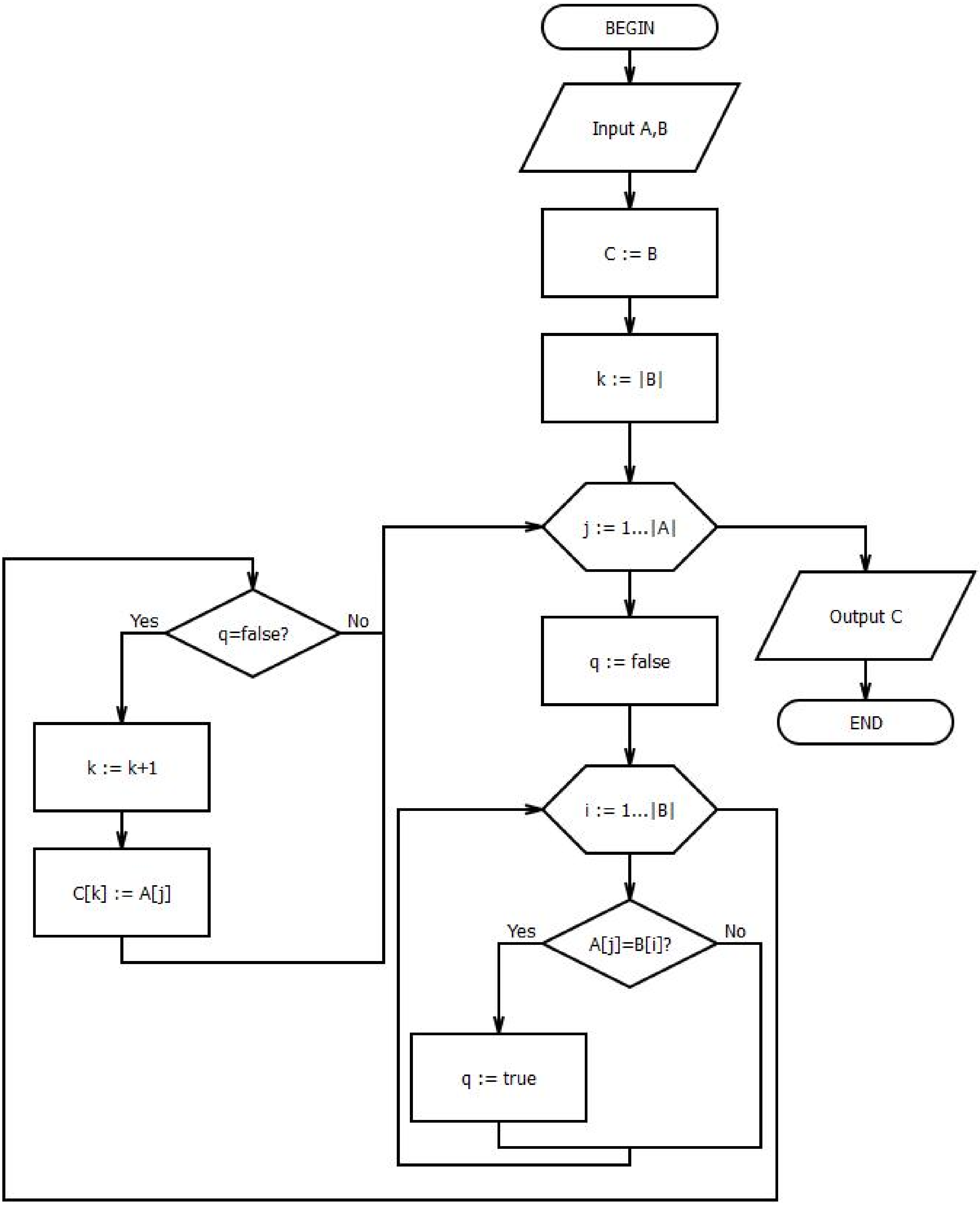


Рис.1.4. Блок-схема операції об’єднання двох множин

Перетин двох множин

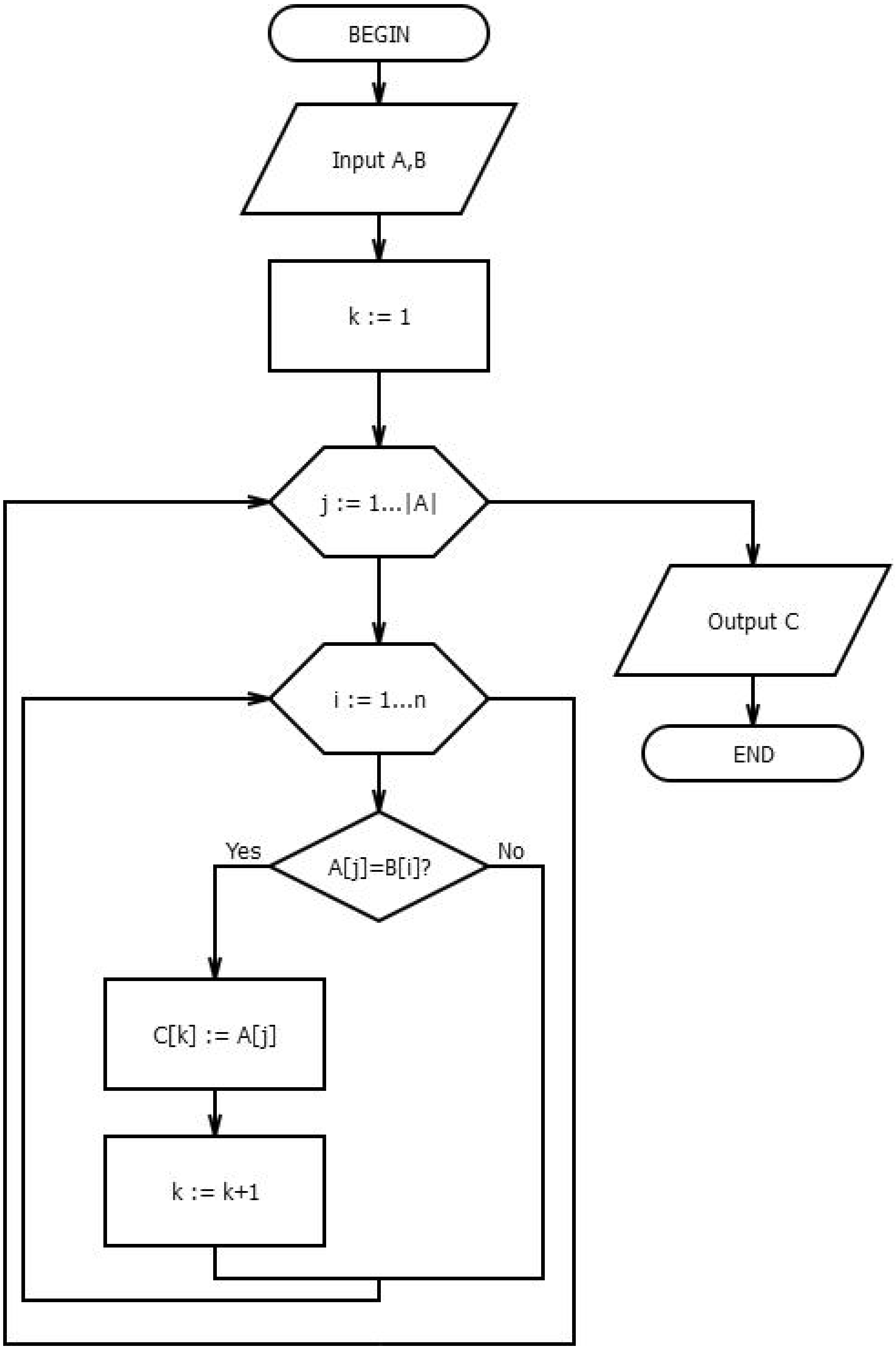


Рис. 1.5. Блок-схема операції перетину двох множин

Різниця двох множин

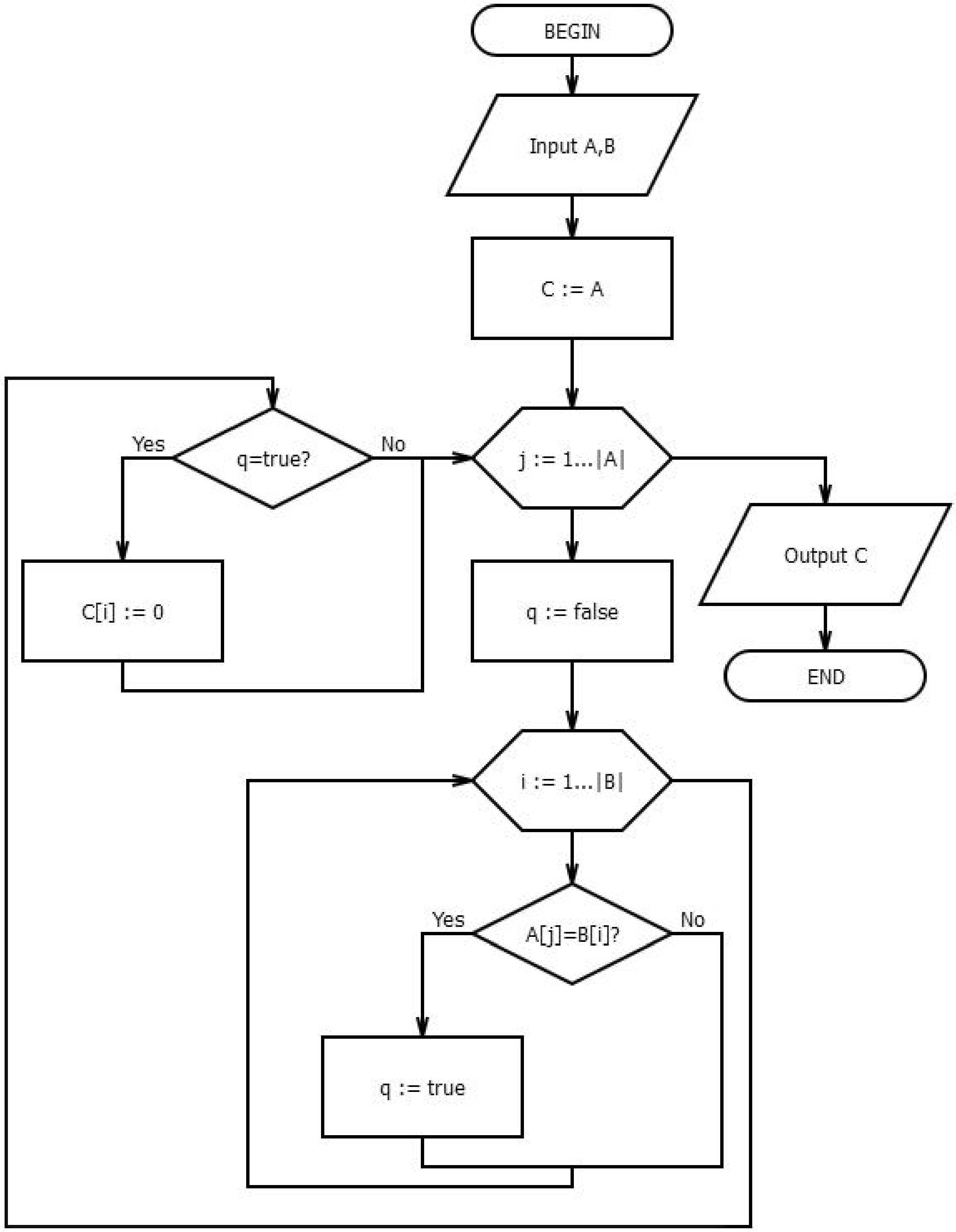


Рис. 1.6. Блок-схема операції різниці множин

Симетрична різниця двох множин

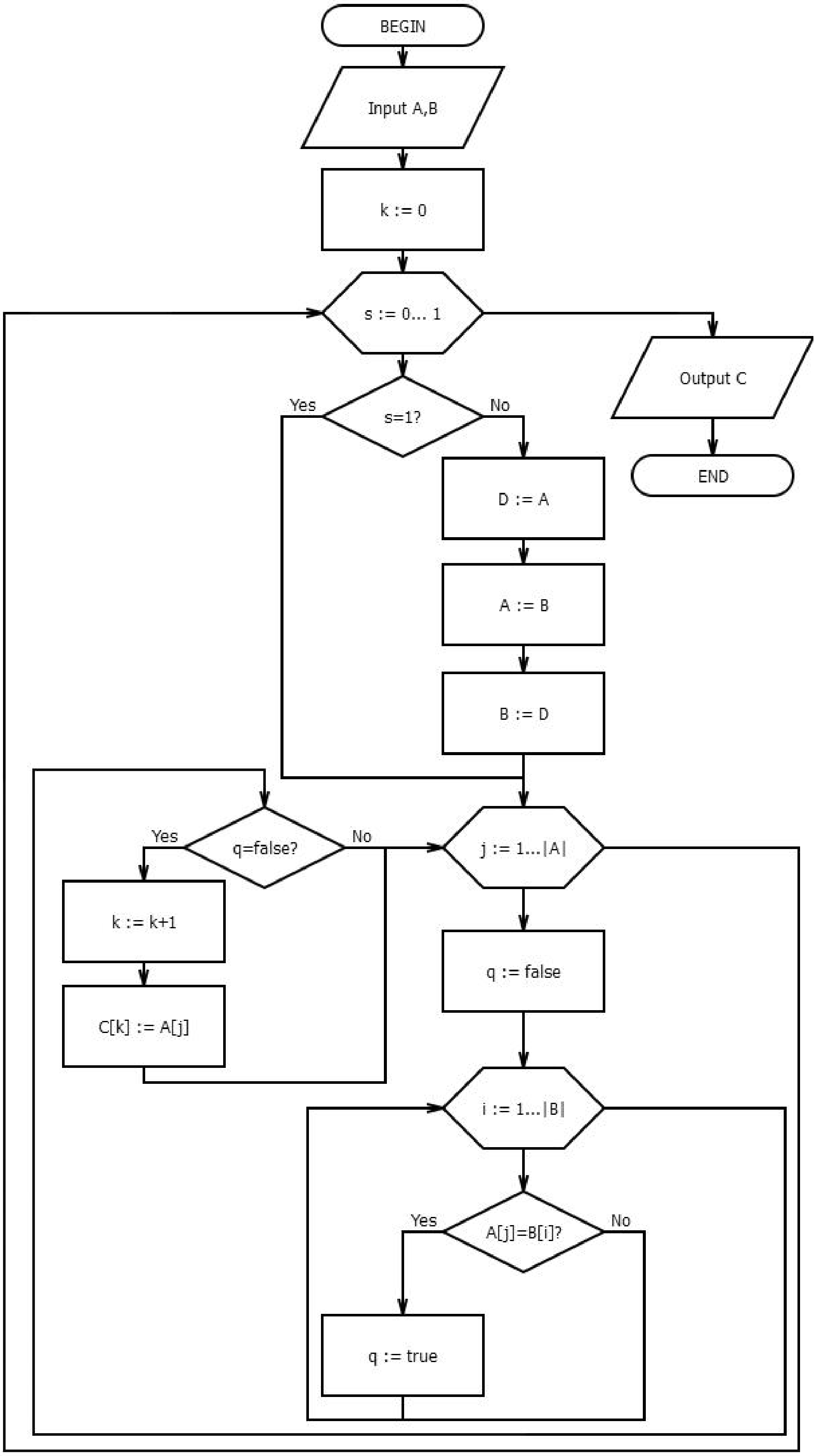


Рис. 1.7. Блок-схема операції симетричної різниці множин

1.8. Завдання лабораторної роботи

А) Використовуючи блок-схему алгоритму рис.1.3, або створену самостійно блок-схему ефективнішого алгоритму, доповнити множину D до відповідної для неї універсальної множини Y. В результаті виконання доповнення сформувати множину R. Вивести елементи множин Y та R.

Б) Використовуючи блок-схему алгоритму рис. 1.4 або створену самостійно блок-схему ефективнішого алгоритму, виконати об’єднання множин А і В. В результаті виконання об’єднання сформувати множину С. Вивести елементи множини С.

В) Використовуючи блок-схему алгоритму рис.1.5, або створену самостійно блок-схему ефективнішого алгоритму, виконати перетин множин А і В. В результаті виконання перетину сформувати множину С. Вивести елементи множини С.

Г) Використовуючи блок-схему алгоритму рис.1.6, або створену самостійно блок-схему ефективнішого алгоритму, знайти різницю множин А та В. В результаті виконання перетину сформувати множину С. Вивести елементи множини С.

Д) Використовуючи блок-схему алгоритму рис.1.7, або створену самостійно блок-схему ефективнішого алгоритму, знайти симетричну різницю множин А та В. В результаті виконання перетину сформувати множину С. Вивести елементи множини С.

Е) Побудувати діаграми Венна для кожної з операцій з виділенням результуючої підмножини.

Варіанти для виконання лабораторної роботи

Номер варіанту I визначається як результат операції I = NZK mod 10, де NZK – номер залікової книжки .

При виконанні завдань лабораторної роботи використати такий спосіб задавання універсальної множини:

Варіант 0. ASCI код

Варіант 1. Цілі числа 0...255

Варіант 2. Букви англійського алфавіту

Варіант 3. Букви українського алфавіту

Варіант 4. Букви російського алфавіту

Варіант 5. Цілі числа 0...1024

Варіант 6. Парні числа від 0…1024

Варіант 7. Непарні числа від 0…1024

Варіант 8. Числа від 0…1024, які діляться без остачі на 5

Варіант 9. Числа від 0…1024, які діляться без остачі на 3

<u>Лабораторна робота №2</u>

<u>Тема:</u> «Бінарні відношення та їх основні властивості, операції над відношеннями».

<u>Мета:</u> вивчити основні властивості бінарних відношень та оволодіти операціями над бінарними відношеннями.

<u>Завдання:</u> написати програму для виконання операцій над бінарними відношеннями.

<u>Теоретичні основи:</u>

2.1. Основні означення

Упорядкована пара предметів – це сукупність, що складається із двох предметів, розташованих у деякому певному порядку. При цьому впорядкована пара має наступні властивості:

а) для будь-яких двох предметів x і y існує об'єкт, який можна позначити як $\langle x, y\rangle$, названий упорядкованою парою;

б) якщо $\langle x, y\rangle$ і $\langle u, v\rangle$ – упорядковані пари, то $\langle x, y\rangle = \langle u, v\rangle$ тоді і тільки тоді, коли $x = u$, $y = v$.

При цьому x будемо називати першою координатою, а y – другою координатою впорядкованої пари $\langle x, y\rangle$.

Бінарним (або *двомісним*) відношенням R називається підмножина впорядкованих пар, тобто множина, кожен елемент якої є впорядкована пара.

Якщо R є деяким відношенням, це записують як $\langle x, y\rangle \in R$ або xRy.

Один з типів відношень − це множина всіх таких пар $\langle x, y\rangle$, що x є елементом деякої фіксованої множини X, а y − елементом деякої фіксованої множини Y. Таке відношення називається *прямим* або *декартовим добутком*.

Декартовим добутком $X \times Y$ множин X і Y є *множина* $\{\langle x, y\rangle | x \in X,\ y \in Y\}$.

При цьому множина X називається *областю визначення* відношення R, а Y – його *областю значень*:

$D(R) = \{x | \langle x, y\rangle \in R\}$; $E(R) = \{y | \langle x, y\rangle \in R\}$

Бінарним відношенням R називається підмножина пар $\langle x, y\rangle \in R$ прямого добутку $X \times Y$, тобто $R \subseteq X \times Y$.

У силу визначення бінарних відношень, як **спосіб їх задавання** можуть бути використані будь-які способи задавання множин. Відношення, визначені на скінченних множинах, звичайно задаються:

1. *Списком (перерахуванням)* упорядкованих пар, для яких це відношення виконується.

2. *Матрицею* – бінарному відношенню $R \subseteq X \times X$, де $X = \{x_1; x_2; \dots; x_n\}$ відповідає квадратна матриця порядку n, кожен елемент a_{ij} якої дорівнює 1, якщо між x_i й x_j є відношення R, і 0 у протилежному випадку, тобто:

$$a_{ij} = \begin{cases} 1, & \text{якщо } x_i R x_j, \\ 0, & \text{у протилежному випадку.} \end{cases}$$

Приклад. Нехай $A = \{1, 2, 3\}$, $B = \{2, 3, 4\}$. Знайти декартовий добуток множин $A \times B$ й $B \times A$. Записати $(A \times B) - (B \times A)$, $(A \times B) \cap (B \times A)$, $(A \times B) + (B \times A)$.

Рішення: $A \times B = \{\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 2,2\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 3,3\rangle, \langle 3,4\rangle\}$;

$B \times A = \{\langle 2,1\rangle, \langle 2,2\rangle, \langle 2,3\rangle, \langle 3,1\rangle, \langle 3,2\rangle, \langle 3,3\rangle, \langle 4,1\rangle, \langle 4,2\rangle, \langle 4,3\rangle\}$;

$(A \times B) - (B \times A) = \{\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 2,4\rangle, \langle 3,4\rangle\}$;

$(A \times B) \cap (B \times A) = \{\langle 2,2\rangle, \langle 2,3\rangle, \langle 3,2\rangle, \langle 3,3\rangle\}$;

$(A \times B) + (B \times A) = \{\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 2,4\rangle, \langle 3,4\rangle, \langle 2,1\rangle, \langle 3,1\rangle, \langle 4,1\rangle, \langle 4,2\rangle, \langle 4,3\rangle\}$.

2.2. Властивості бінарних відношень

1. Відношення R на $A \times A$ називається **рефлексивним**, якщо має місце $\langle a, a\rangle \in R$ для кожного $a \in A$. Головна діагональ матриці такого відношення містить тільки одиниці.

2. Відношення R на $A \times A$ називається **антирефлексивним**, якщо для жодного $a \in A$ не виконується $\langle a, a\rangle \in R$, тобто із $\langle a, b\rangle \in R$ потрвбгл, щоб $a \neq b$. Головна діагональ матриці такого відношення містить тільки нулі.

3. Відношення R на $A \times A$ називається **симетричним**, якщо для всіх $a, b \in R$ з умови $\langle a, b\rangle \in R$ потрібно, щоб $\langle b, a\rangle \in R$. Матриця симетричного відношення симетрична щодо головної діагоналі, тобто $c_{ij} = c_{ji}$ для всіх i і j.

4. Відношення R на $A \times A$ називається **антисиметричним**, якщо для всіх $a, b \in R$, з умов $\langle a, b\rangle \in R$ і $\langle b, a\rangle \in R$ потрібно, щоб $a = b$, тобто для жодних елементів a і b, що розрізняються $(a \neq b)$, не виконуються одночасно відношення $\langle a, b\rangle \in R$ і $\langle b, a\rangle \in R$. У матриці антисиметричного відношення відсутні одиниці, симетричні щодо головної діагоналі.

5. Відношення R на $A \times A$ називається **транзитивним**, якщо для будь-яких a, b, c з умов $\langle a, b\rangle \in R$ і $\langle b, c\rangle \in R$ випливає $\langle a, c\rangle \in R$. У матриці такого відношення повинна виконуватися наступна умова: якщо в *i*-тому рядку і в *j*-тому стовпці стоїть одиниця, тобто $c_{ij} = 1$, то всім одиницям в j-тому рядку і k-тому стовпці $(c_{jk} = 1)$ повинні відповідати одиниці в i-тому рядку і у тих же k-тих стовпцях, тобто $c_{ik} = 1$ (і, можливо, в інших стовпцях).

6. Бінарне відношення називається **еквівалентним**, якщо воно рефлексивне, симетричне і транзитивне.

2.3. Операції над відношеннями

Оскільки відношення на множині A задаються підмножинами $R \subseteq A \times B$, то для них визначні ті ж операції, що й над множинами, а саме:

1. **Об'єднання**: $R_1 \quad R_2 = \{\langle a,b\rangle | \langle a,b\rangle \quad R_1 \;\; або \;\; \langle a,b\rangle \quad R_2\}$.

2. **Перетин**: $R_1 \cap R_2 = \{\langle a,b\rangle | \langle a,b\rangle \quad R_1 \;\; і \;\; \langle a,b\rangle \quad R_2\}$.

3. **Різниця**: $R_1 - R_2 = \{\langle a,b\rangle | \langle a,b\rangle \in R_1 \;\; i \;\; \langle a,b\rangle \in R_2\}$.

4. **Доповнення**: $\overline{R} = U - R$, де $U = A \times B$.

Крім того, необхідно визначити інші операції над бінарними відношеннями.

5. **Обернене відношення** R^{-1}.

Якщо $\langle a,b\rangle \in R$ – відношення, то відношення R^{-1} називається **оберненим відношенням** до даного відношення R тоді й тільки тоді, коли $R^{-1} = \{\langle b,a\rangle | \langle a,b\rangle \in R\}$.

Наприклад, якщо R – "бути старіше", то R^{-1} – "бути молодше"; якщо R – "бути дружиною", то R^{-1} – "бути чоловіком".

Нехай $R = \{\langle a,b\rangle | b \;\; є \;\; родич \;\; a\}$ або $R = \{\langle x,y\rangle | x^2 + y^2 = 1\}$. У такому випадку $R = R^{-1}$.

Нехай $R \subseteq A \times B$ – відношення на $A \times B$, а $S \subseteq B \times C$ – відношення на $B \times C$. **Композицією** відношень R і S називається відношення $T \subseteq A \times C$, таке, що $T = \{\langle a,c\rangle | існує \;\; такий \;\; елемент \;\; b \;\; з \;\; B, \; що \;\; \langle a,b\rangle \quad R \;\; і \;\; \langle b,c\rangle \quad S\}$.
Ця множина позначається $T = S \circ R$.

Зокрема, якщо відношення R визначене на множині A $\left(R \subseteq A^2\right)$, то композиція визначається як

$$R \circ R = R^{(2)} = \{\langle a,c\rangle | \langle a,b\rangle, \langle b,c\rangle \in R\}.$$

Транзитивним замиканням R^{O} називається множина, що складається з таких і тільки таких пар елементів a,b з A, тобто $\langle a,b\rangle \in R^{\mathrm{O}}$, для яких в A існує ланцюжок з $n+2$ $(n \geq 0)$ елементів $A: a, c_1, c_2, ..., c_n, b$, між сусідніми елементами якої виконується відношення R: $\langle a,c_1\rangle \in R$, $\langle c_1,c_2\rangle \in R, \ldots, \langle c_n,b\rangle \in R$, тобто

$$R^{\mathrm{O}} = \{\langle a,b\rangle | \langle a,c_1\rangle, \langle c_1,c_2\rangle, ..., \langle c_n,b\rangle \in R\}.$$

Наприклад, для відношення R – "бути дочкою" композиція $R \circ R = R^{(2)}$ – "бути онукою", $R \circ R \circ R = R^{(3)}$ – "бути правнучкою" і т. ін. Тоді об'єднання всіх цих відносин є транзитивним замиканням R^{O} – "бути прямим нащадком".

Якщо відношення R транзитивне, то $R^{\mathrm{O}} = R$.

Транзитивне замикання R° на R є найменшим транзитивним відношенням на A, що містить R як підмножину.

Процедура обчислення транзитивного замикання R° для відношення $R \in A \times A$:

1) присвоїти $R_1 \leftarrow R$;

2) обчислити $R_1 \cup R_1^{(2)} = R_1 \cup R_1$, присвоїти $R_2 \leftarrow R_1^{(2)}$;

3) прирівняти R_1 і R_2. Якщо $R_1 = R_2$, то перейти до кроку 4, якщо $R_1 \neq R_2$, то присвоїти $R_1 \leftarrow R_2$ і перейти до кроку 2;

4) $R_1 = R_2 = R^{\circ}$.

8. **Рефлексивне замикання**:

Нехай тотожне відношення $E = \{\langle a, a \rangle | a \in A\}$. Тоді *рефлексивне замикання* визначається як $R^* = R^{\circ} \cup E$.

Якщо R транзитивне і рефлексивне, то $R^* = R$.

Рефлексивне замикання R^* на R є найменшим рефлексивним відношенням на A, що містить R як підмножину.

2.4. Завдання лабораторної роботи

А) Написати програму для знаходження $S \cup R$, $S \times \mathrm{R}$, R^{-1}, $S \times \mathrm{R}^{-1}$, де бінарні відношення задані на множині людей, пов’язаних родинними зв’язками (задані у варіантах до виконання лабораторної роботи).

Вимоги до програмного забезпечення:

1. Лабораторна робота виконується в середовищі візуального програмування Lazarus.
2. В якості початкового коду використати проект LAB2_Project, попередньо скачавши його з сайту викладача.
3. Початкові данні вводяться за допомогою програмних засобів уже реалізованих в згаданому проекті.
4. Необхідно в рамках форми OperForm реалізувати алгоритм відповідно до варіанту лабораторної роботи.
5. На формі OperForm відобразити як початкові дані, так і результати роботи алгоритму.

Зміст звіту:

1. Титульний лист.
2. Тема завдання.
3. Завдання.
4. Короткі теоретичні відомості.
4. Роздруківка результатів виконання програми.
5. Роздруківка тексту програми.

6. Аналіз результатів.

Контрольні питання

1. Дати визначення бінарного відношення.
2. Способи задавання відношень.
3. Властивості бінарних відношень.
4. Визначення композиції відношень.

Етапи виконання роботи .

Етап.1. Створити програму, яка коректно формує відношення у відповідності з варіантом завдання та виконує операції над цими відношеннями.

Етап.2.Ввести елементи множин A та B. Наприклад: $A=${Антоніна, Оксана, Галина, Ольга, Світлана, Петро, Тетяна, Іван, Катерина, Олег}, $B=${Борис, Василь, Максим, Ольга, Тетяна, Іван, Аркадій, Артем, Оксана, Петро}.

Етап 3. Задати програмно відношення S і R між елементами множин A і B. Наприклад використовуючи варіант:

aSb, якщо a сестра b. aRb, якщо a дружина b

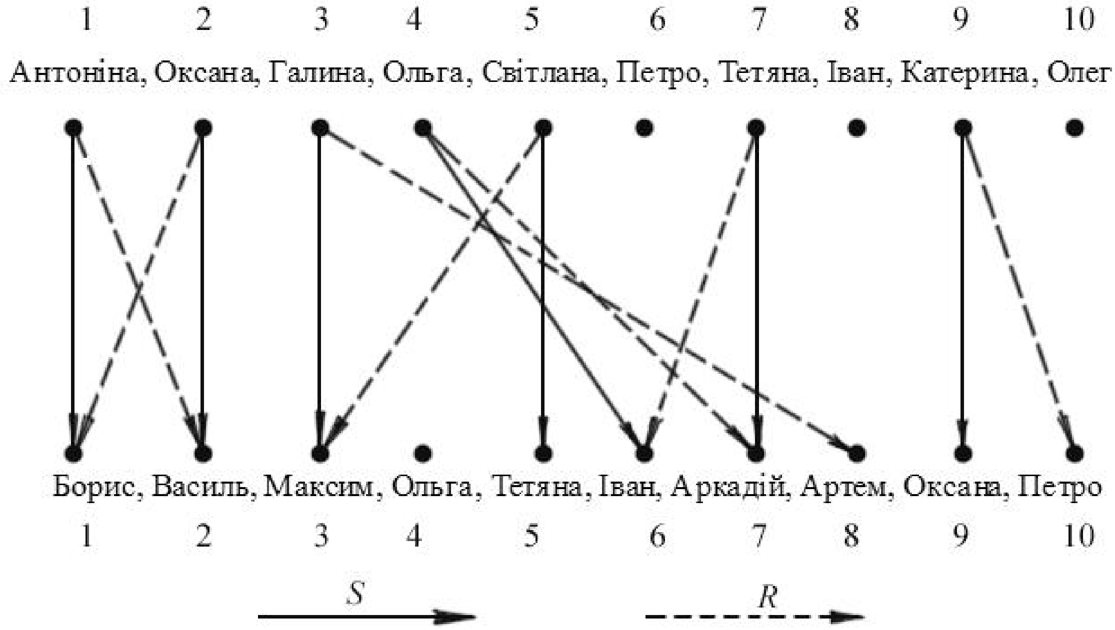


Рис.2.1. Приклад відношення S і R

Етап 5. Виконати програмно перевірку коректності задавання відношень. Наприклад:

$S=\{\langle 1,1\rangle,\langle 2,2\rangle,\langle 3,3\rangle,\langle 4,6\rangle,\langle 5,5\rangle,\langle 7,7\rangle,\langle 9,9\rangle\}$ - елементи відношення коректні,

$R=\{\langle 1,2\rangle,\langle 2,1\rangle,\langle 3,8\rangle,\langle 4,7\rangle,\langle 5,3\rangle,\langle 7,6\rangle,\langle 9,10\rangle\}$ - елементи відношення коректні.

Приклади некоректних елементів відношень:

$\langle 1,1\rangle \notin R$ -Антоніна не може бути дружиною Бориса, оскільки вона – його сестра.

$\langle 3,4\rangle \notin R$ - Ольга –особа жіночої статі, тому не може бути дружиною Галини.

$\langle 6,9\rangle \notin R$ - Петро –особа чоловічої статі, тому не може бути дружиною Оксани.

$\langle 4,4\rangle \notin S$ - Ольга не може бути сестрою сама собі. І т. д.

Етап 6. Роздрукувати та вивести на екран результати операцій $S \cup R$, $S \times R$, R^{-1}, $S \times R^{-1}$

Варіанти для виконання лабораторної роботи

Номер варіанту I визначається як результат операції I = NZK mod 20, де NZK – номер залікової книжки.

Варіанти до пункту А завдання

0. aSb, якщо a сестра b. aRb, якщо a дружина b.
1. aSb, якщо a мати b. aRb, якщо a внучка b.
2. aSb, якщо a дружина b. aRb, якщо a мати b.
3. aSb, якщо a сестра b. aRb, якщо a мати b.
4. aSb, якщо a свекруха b. aRb, якщо a батько b.
5. aSb, якщо a сестра b. aRb, якщо a чоловік b.
6. aSb, якщо a батько b. aRb, якщо a внук b.
7. aSb, якщо a дружина b. aRb, якщо a мати b.
8. aSb, якщо a сестра b. aRb, якщо a мати b.
9. aSb, якщо a свекруха b. aRb, якщо a батько b.
10. aSb, якщо a брат b. aRb, якщо a чоловік b.
11. aSb, якщо a чоловік b. aRb, якщо a батько b.
12. aSb, якщо a тесть b. aRb, якщо a батько b.
13. aSb, якщо a брат b. aRb, якщо a мати b.
14. aSb, якщо a син b. aRb, якщо a свекор b.
15. aSb, якщо a брат b. aRb, якщо a дружина b.
16. aSb, якщо a чоловік b. aRb, якщо a батько b.
17. aSb, якщо a тесть b. aRb, якщо a батько b.
18. aSb, якщо a брат b. aRb, якщо a мати b.
19. aSb, якщо a син b. aRb, якщо a свекор b.

Лабораторна робота №3

Тема: «Комбінаторика: перестановки, розміщення, сполучення»
Мета роботи: вивчення правил утворення комбінацій можин: *перестановок, розміщень, сполучень.*
Завдання: Вивчити алгоритми формування перестановок, сполучень та розбиття. Написати програми для виконання даних алгоритмів.

Теоретичні основи

3.1. Основні означення

Перестановки

Комбінації з n елементів, які відрізняються одна від одної тільки порядком елементів, називаються перестановками.

Перестановки позначаються символом P_n, де n — число елементів, що входять у кожну перестановку.

Приклад. Нехай множина *M* містить три букви *A, B, C*. Складемо всі можливі комбінації із цих букв*: ABC, ACB, BCA, CAB, CBA, BAC* (усього 6 комбінацій). Видно, що вони відрізняються одна від одної тільки порядком розташування букв.

Добуток всіх натуральних чисел від 1 до n включно називають n-факторіалом і пишуть: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$. Вважають, що $0! = 1$ і $n \in N$. Основна властивість факторіала: $(n+1)! = (n+1) \cdot n!$.

Отже, число перестановок обчислюємо за формулою: $P_n = n!$

Розміщення

Комбінації з n елементів по m елементів, які відрізняються одна від одної або самими елементами, або порядком елементів, називаються розміщеннями.

Розміщення позначаються символом A_n^m, де n – число всіх наявних елементів, m – число елементів у кожній комбінації. Число розміщень можна обчислити за формулою:

$$A_n^m = n(n-1)(n-2)\ldots(n-m+1), \qquad \text{де } 0 \le m \le n; \qquad m, n \in N.$$

Вважають, що $A_n^0 = 1$.

Приклад. Нехай множина *M* містить чотири букви *A, B, C, D*. Склавши всі комбінації тільки із двох букв, одержимо: *AB, AC, AD, BA, BP, BD, CA, CB, CD, DA, DB, DC.*

Формулу розміщення можна записати у факторіальній формі:

$$A_n^m = \frac{n!}{(n-m)!}.$$

Основні властивості розміщень:

1) $A_n^{m+1} = A_n^m \cdot (n-m)$;

2) $A_n^n = P_n = n!$.

Сполучення

Сполученнями називаються всі можливі комбінації з n елементів по m, які відрізняються одна від одної принаймні хоча б одним елементом

($m,\ n \in N$ *і* $n \geq m$).

У загальному випадку число сполучень із *n* елементів по *m* дорівнює числу розміщень з *n* елементів по *m*, діленому на число перестановок з *m* елементів: $C_n^m = \dfrac{A_n^m}{P_m}$. Використовуючи для кількості розміщень і перестановок факторіальні формули $A_n^m = \dfrac{m!}{(m-n)!}$ і $P_n = n!$, одержимо формулу кількості сполучень у вигляді: $C_n^m = \dfrac{n!}{(n-m)!\, m!}$.

Основні властивості сполучень: $C_n^{n-m} = \dfrac{P_n}{P_{n-m} \cdot P_m} = \dfrac{n!}{(n-m)!\, m!}$; $C_n^m = C_n^{n-m}$.

Приклад. Множина *M* утворена з чотирьох букв *A, B, C, D*. Скласти комбінації з двох букв, що відрізняються хоча б одним елементом.

Маємо *AB, AC, AD, BP, BD, CD.* Виходить, що кількість сполучень з чотирьох елементів по два дорівнює 6. Це коротко записується так: $C_4^2 = 6$.

Розміщення з повтореннями

Розміщення з *n* елементів по *k* відображають упорядковані комбінації різних елементів множини *M*, $|M|=n$. Часто доводиться утворювати упорядковані комбінації з повтореннями деяких елементів. Наприклад, з множини *M = {A, Б}* можна утворити вісім комбінацій з трьох елементів: *ААА, ААБ, АБА, БАА, БАБ, ББА, АББ, БББ.* Тут $n = 2$, $k = 3$. Такі упорядковані *k*-комбінації називають кортежами довжини *k.* Два кортежі (тобто дві загальні комбінації) вважаються однаковими, якщо вони мають однакову довжину і на місцях з однаковими номерами стоять однакові елементи.

Кортеж довжини k з n елементів називається розміщенням з повтореннями з n елементів по k .

Кількість кортежів обчислюється за формулою: $\overline{A_n^k} = n^k$.

Дійсно, після заповнення першого місця кортежу довжиною *k* одним з *n* елементів (що можливо зробити *n* варіантами) заповнити друге місце кортежу можна знову будь-яким елементом з усієї множини (повторюючи в одному з варіантів елемент, який знаходиться на першому місці), і так далі k разів. За правилом добутку одержимо, що $\overline{A_n^k} = \underbrace{n \cdot n \ldots n}_{k\ разів} = n^k$

3.2. Комбінаторні алгоритми

Алгоритми перестановок

Розглянемо методи генерування послідовностей всіх $n!$ перестановок множини, що складається з n елементів. Для цього задану множину представимо у вигляді елементів масиву $P[1], P[2], ..., P[n]$

Методи, які будуть нами розглядатися, базуються на застосуванні до масиву $P[i]$, $i = 1, 2, ..., n$ елементарної операції, що носить назву поелементної транспозиції. Суть операції полягає у обміні даними між елементами масива $P[i]$ і $P[j]$, $1 \le i, j \le n$ за такою схемою:

$$vrem := P[i],\ P[i] := P[j],\ P[j] := vrem\,,$$

де $vrem$ – деяка допоміжна змінна, що використовується для тимчасового збереження значення елемента масива $P[i]$.

Пояснимо також термін „лексикографічний порядок”. Назва терміну походить від його застосування у словниках. Всі слова у словниках розміщені у лексикографічному порядку.

Визничення лексикографічного порядку. Нехай існують перестановки у вигляді послідовностей $\{x_1, x_2, x_3, ..., x_n\}, \{y_1, y_2, y_3, ..., y_n\}, ...$ однієї і тієї ж множини X. Говорять, що перстановки з елементів множини X упорядковані у лексикографічному порядку , якщо

$$\{x_1, x_2, x_3, ..., x_n\} < \{y_1, y_2, y_3, ..., y_n\}$$

тоді і тільки тоді, коли для деякого k

$$x_k \le y_k \text{ і } x_i = y_i \text{ для всіх } i < k\,.$$

Визничення антилексикографічного порядку. Нехай існують перестановки у вигляді послідовностей $\{x_1, x_2, x_3, ..., x_n\}, \{y_1, y_2, y_3, ..., y_n\}, ...$ однієї і тієї ж множини X. Говорять, що перстановки з елементів множини X упорядковані у антилексикографічному порядку , якщо

$$\{x_1, x_2, x_3, ..., x_n\} <' \{y_1, y_2, y_3, ..., y_n\}$$

тоді і тільки тоді, коли для деякого k

$$x_k > y_k \text{ і } x_i = y_i \text{ для всіх } i < k\,.$$

Приклад. Перестановки множини $X = \{1, 2, 3\}$ в лесикографічному (а) та антилексикографічному (б) пордку

(а)	(б)
1 2 3	1 2 3
1 3 2	2 1 3
2 1 3	1 3 2
2 3 1	3 1 2
3 1 2	2 3 1
3 2 1	3 2 1

3.2.1. Алгоритм побудови перестановок у лексикографічному порядку

Алгоритм формування перестановок у лексикографічному порядку почнемо, наприклад, з початкової перестановки $(1,2,...,n-1,n)$. Подальший розвиток алгоритму полягає у переході від перестановки $(x_1,x_2,...,x_{n-1},x_n)$ до безпосередньо наступної за нею перестановки $(y_1,y_2,...,y_n)$ поки не одержимо найбільшу перестановку $(n,n-1,...,2,1)$. Розглянемо спосіб побудови перестановки $(y_1,y_2,...,y_{n-1},y_n)$.

(1, 2, 3) ***Цикл 1***

11. Розглядаємо з права наліво перестановку $x=(x_1,x_2,...,x_{n-1},x_n)$ в пошуках самої правої позиції i такої, що $x_i<x_{i+1}$. Якщо такої позиції не знайдено, то тоді $x_1>x_2>...>x_n$, тобто $x=(n,n-1,...,1)$. Дана перестановка є завершальною перестановкою нашого алгоритму.

Приклад. $x=(1,2,3)$. $x_1=1,\ x_2=2,\ x_3=3$.
$x_2<x_3$ оскільки 2 < 3. Отже сама права позиція, для якої виконується $x_i<x_{i+1}$ визначається індексом $i=2$

12. Якщо ж згадану позицію i знайдено, то $x_i<x_{i+1}>x_{i+2}>...>x_n$.

Приклад. $x=(1,2,3)$При $i=2$ одержуємо 2 < 3 > ∅ оскільки 3 є останнім елементом кортежу.

13. На наступному кроці алгоритму шукаємо першу позицію j при переході від позиції n до позиції i таку, що $x_i<x_j$. Тоді $i<j$.

Приклад. $x=(1,2,3)$. Шукана позиція $j=3$ оскільки $x_2<x_3$. Тоді 2 < 3.

14. Далі виконуємо операцію транспозиції над елементами x_i та x_j в результаті якої одержуємо перестановку $x'=(x'_1,x'_2,...,x'_n)$.

Приклад. $x=(1,2,3)$.Виконали транспозицію елементів x_2 і x_3. Одержали $x=(1,3,2)$

15. В цій перестановці частину елементів $x_{i+1},...,x_{n-1},x_n$ перевертаємо, тобто міняємо порядок їх слідування на протилежний. Одержана таким чином перестановка є перестановкою $y=(y_1,y_2,...,y_n)$. Саме ця перестановка є наступною в лексикографічному порядку слідування перестановок.

Приклад. $x=(1,3,2)$.В даному випадку необхідно перевернути терм, що складається з одного елемента (x_3), що нічого не змінює. Тому одержуємо $x=(1,3,2)$.

Переходимо до циклу 2 алгоритму.

(1, 3, 2) ***Цикл 2.***

21. Розглядаємо з права наліво перестановку $x=\left(x_1,x_2,...,x_{n-1},x_n\right)$ в пошуках самої правої позиції i такої, що $x_i<x_{i+1}$. Якщо такої позиції не знайдено, то тоді $x_1>x_2>...>x_n$, тобто $x=\left(n,n-1,...,1\right)$. Дана перестановка є завершальною перестановкою нашого алгоритму.

Приклад. $x=\left(1,3,2\right)$. $x_1=1,\ x_2=3,\ x_3=2$.

$x_1<x_2$ оскільки 1 < 3. Отже сама права позиція, для якої виконується $x_i<x_{i+1}$ визначається індексом $i=1$

22. Якщо ж згадану позицію i знайдено, то $x_i<x_{i+1}>x_{i+2}>...>x_n$.

Приклад. $x=\left(1,2,4,3\right)$ При $i=1$ одержуємо 1 < 3 >2.

23. На наступному кроці алгоритму шукаємо першу позицію j при переході від позиції n до позиції i таку, що $x_i<x_j$. Тоді $i<j$.

Приклад. $x=\left(1,3,2\right)$. Шукана позиція $j=3$ оскільки $x_1<x_3$. Тоді 1 < 2.

24. Далі виконуємо операцію транспозиції над елементами x_i та x_j в результаті якої одержуємо перестановку $x'=\left(x_1',x_2',...,x_n'\right)$.

Приклад. $x=\left(1,3,2\right)$.Виконали транспозицію елементів x_1 і x_3. Одержали $x=\left(2,3,1\right)$

25. В цій перестановці частину елементів $x_{i+1},...,x_{n-1},x_n$ перевертаємо, тобто міняємо порядок їх слідування на протилежний. Одержана таким чином перестановка є перестановкою $y=\left(y_1,y_2,...,y_n\right)$. Саме ця перестановка є наступною в лексикографічному порядку слідування перестановок.

Приклад. $x=\left(2,3,1\right)$.В даному випадку необхідно перевернути терм, що складається з двох елементів $\left(x_2,x_3\right)$. Одержуємо

$x=\left(2,1,3\right)$**.**

Переходимо до циклу 3 алгоритму.

(2, 1, 3) ***Цикл 3.***

31. Розглядаємо з права наліво перестановку $x=\left(x_1,x_2,...,x_{n-1},x_n\right)$ в пошуках самої правої позиції i такої, що $x_i<x_{i+1}$. Якщо такої позиції не знайдено, то тоді $x_1>x_2>...>x_n$, тобто $x=\left(n,n-1,...,1\right)$. Дана перестановка є завершальною перестановкою нашого алгоритму.

Приклад. $x=\left(2,1,3\right)$. $x_1=2,\ x_2=1,\ x_3=3$.

А) Спочатку перевіряємо $x_2<x_3 \Rightarrow$ 1 < 3. $i=2$. Справджується, оскільки $1<3$.

32. Якщо ж згадану позицію i знайдено, то $x_i<x_{i+1}>x_{i+2}>...>x_n$.

Приклад. $x=\left(2,1,3\right)$ При $i=2$ одержуємо 1 <3>$\varnothing$.

33. На наступному кроці алгоритму шукаємо першу позицію j при переході від позиції n до позиції i таку, що $x_i < x_j$. Тоді $i < j$.

Приклад. $x = (2,1,3)$. Шукана позиція $j = 3$ оскільки $x_2 < x_3$. Тоді 1 < 3.

34. Далі виконуємо операцію транспозиції над елементами x_i та x_j в результаті якої одержуємо перестановку $x' = (x_1', x_2', ..., x_n')$.

Приклад. $x = (2,1,3)$.Виконали транспозицію елементів x_2 і x_3. Одержали $x = (2,3,1)$

35. В цій перестановці частину елементів $x_{i+1}, ..., x_{n-1}, x_n$ перевертаємо, тобто міняємо порядок їх слідування на протилежний. Одержана таким чином перестановка є перестановкою $y = (y_1, y_2, ..., y_n)$. Саме ця перестановка є наступною в лексикографічному порядку слідування перестановок.

Приклад. $x = (2,3,1)$.В даному випадку необхідно перевернути терм, що складається з одного елемента (x_3). Одержуємо $x = (2,3,1)$**.**

Переходимо до циклу 4 алгоритму.

(2, 3, 1) *Цикл 4.*

41. Розглядаємо з права наліво перестановку $x = (x_1, x_2, ..., x_{n-1}, x_n)$ в пошуках самої правої позиції i такої, що $x_i < x_{i+1}$. Якщо такої позиції не знайдено, то тоді $x_1 > x_2 > ... > x_n$, тобто $x = (n, n-1, ..., 1)$. Дана перестановка є завершальною перестановкою нашого алгоритму.

Приклад. $x = (2,3,1)$. $x_1 = 2,\ x_2 = 3,\ x_3 = 1$.

А) Спочатку перевіряємо $x_2 < x_3 \Rightarrow$ 3 > 1. Не справджується, оскільки $3 > 1$.

Б) Далі перевіряємо $x_1 < x_2 \Rightarrow$ 2 < 3. Справджується, оскільки $2 < 3$.

Отже $i = 1$.

42. Якщо ж згадану позицію i знайдено, то $x_i < x_{i+1} > x_{i+2} > ... > x_n$.

Приклад. $x = (2,3,1)$При $i = 1$ одержуємо 2 <3>1 .

43. На наступному кроці алгоритму шукаємо першу позицію j при переході від позиції n до позиції i таку, що $x_i < x_j$. Тоді $i < j$.

Приклад. $x = (2,3,1)$. Шукана позиція $j = 2$ оскільки $x_1 < x_2$. Тоді 2 < 3.

44. Далі виконуємо операцію транспозиції над елементами x_i та x_j в результаті якої одержуємо перестановку $x' = (x_1', x_2', ..., x_n')$.

Приклад. $x = (2,3,1)$.Виконали транспозицію елементів x_2 і x_3. Одержали $x = (3,2,1)$

45. В цій перестановці частину елементів $x_{i+1},...,x_{n-1},x_n$ перевертаємо, тобто міняємо порядок їх слідування на протилежний. Одержана таким чином перестановка є перестановкою $y=(y_1,y_2,...,y_n)$. Саме ця перестановка є наступною в лексикографічному порядку слідування перестановок.

Приклад. $x=(3,2,1)$.В даному випадку необхідно перевернути терм, що складається з двох елементів (x_2,x_3). Одержуємо $x=(3,1,2)$.

Переходимо до циклу 5 алгоритму.

(3, 1, 2) ***Цикл 5.***

51. Розглядаємо з права наліво перестановку $x=(x_1,x_2,...,x_{n-1},x_n)$ в пошуках самої правої позиції i такої, що $x_i<x_{i+1}$. Якщо такої позиції не знайдено, то тоді $x_1>x_2>...>x_n$, тобто $x=(n,n-1,...,1)$. Дана перестановка є завершальною перестановкою нашого алгоритму.

Приклад. $x=(3,1,2)$. $x_1=3,\ x_2=1,\ x_3=2$.

А) Спочатку перевіряємо $x_2<x_3 \Rightarrow$ 1 < 2.Справджується, оскільки $1<2$.

Отже $i=2$.

52. Якщо ж згадану позицію i знайдено, то $x_i<x_{i+1}>x_{i+2}>...>x_n$.

Приклад. $x=(3,1,2)$При $i=2$ одержуємо 1 <2>∅ .

53. На наступному кроці алгоритму шукаємо першу позицію j при переході від позиції n до позиції i таку, що $x_i<x_j$. Тоді $i<j$.

Приклад. $x=(3,1,2)$. Шукана позиція $j=3$ оскільки $x_2<x_3$. Тоді 1 < 2.

54. Далі виконуємо операцію транспозиції над елементами x_i та x_j в результаті якої одержуємо перестановку $x'=(x_1',x_2',...,x_n')$.

Приклад. $x=(3,1,2)$.Виконали транспозицію елементів x_2 і x_3. Одержали $x=(3,2,1)$

55. В цій перестановці частину елементів $x_{i+1},...,x_{n-1},x_n$ перевертаємо, тобто міняємо порядок їх слідування на протилежний. Одержана таким чином перестановка є перестановкою $y=(y_1,y_2,...,y_n)$. Саме ця перестановка є наступною в лексикографічному порядку слідування перестановок.

Приклад. $x=(3,2,1)$.В даному випадку необхідно перевернути терм, що складається з одного елемента (x_3). Одержуємо $x=(3,2,1)$.

Переходимо до циклу 6 алгоритму.

(3, 2, 1) ***Цикл 6.***

61. Розглядаємо з права наліво перестановку $x=(x_1,x_2,...,x_{n-1},x_n)$ в пошуках самої правої позиції i такої, що $x_i<x_{i+1}$. Якщо такої позиції не знайдено, то тоді

$x_1 > x_2 > ... > x_n$, тобто $x = (n, n-1, ..., 1)$. Дана перестановка є завершальною перестановкою нашого алгоритму.

Приклад. $x = (3,2,1)$. $x_1 = 3,\ x_2 = 2,\ x_3 = 1$.

Не знайдено випадку $x_i < x_{i+1}$. Алгоритм завершено.

3.2.2. Блок-схема алгоритму побудови перестановок у лексикографічному порядку

Блок 1.

Ітерфейс вводу початкових даних:

$n-$кількість елементів масиву для виконання перестановок у лексикографічному порядку.

$s-$ кількість перестановок, які необхідно виконати.

P - початкові значення елементів масиву для виконання перестановок.

Блоки 2-3.

Проверка получения количества заказанных перестаовок s.

Блоки 4-7.

Відбувається попереднє встановлення змінних , пошук самої правої позиції i такої, що $x_i < x_{i+1}$ та закінчення алгоритму у випадку, коли i не знайдено.

Блоки 8-10.

Якщо позиція i знайдена, то відбувається пошук позиції j, починаючи від позиції n до позиції i такої, що $x_i < x_j$.

Блок 11.

Якщо знайдено i та j, то виконуємо операцію транспозиції над елементами x_i та x_j

Блоки 12-17.

Частину елементів $x_{i+1}, ..., x_{n-1}, x_n$ перевертаємо, тобто міняємо порядок їх слідування на протилежний. Роздруківка перестановки.

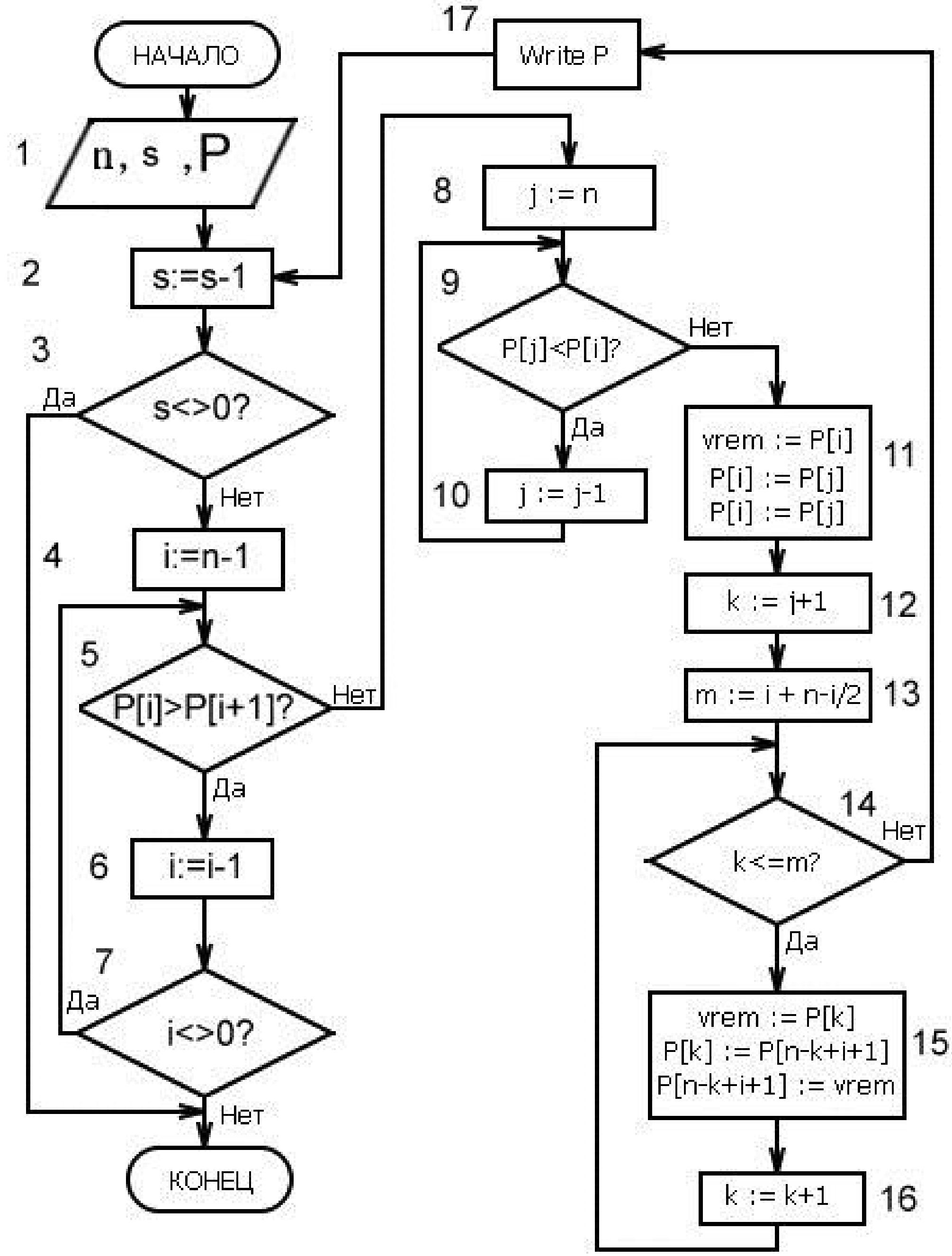


Рис.3.1. Блок-схема алгоритму перестановок в лексикографічному порядку

3.2.3. Алгоритм генерації двійкових векторів довжини n.

Алгоритм породжує всі двійкові вектори $b=\left(b_{n-1},b_{n-1},...,b_1,b_0\right)$ довжини n в лексикографічному порядку, починаючи з найменшого елемента.

Будемо використовувати масив b[n],b[n-1], ….,b[1],b[0], установивши b[n]:=0.

Приклад. $b=\left(0,0,0\right)$. $n=2$ $b\left[2\right]=0$, $b\left[1\right]=0$, $b\left[0\right]=0$

(0,0,0) ***Цикл 1.***

1.1. Переглядаючи справа наліво, знаходимо першу позицію b[i] таку, що b[i]=0.

Приклад. $b=(0,0,0)$, $i=0$, $b[0]:=0$

1.2. Записуємо b[i]:=1, а всі елементи b[j], j<i, що розміщені праворуч від b[i], задаємо рівними 0.

Приклад. $b[0]:=1$. Оскільки $i=0$, то розряди справа відсутні.

(0,0,1) ***Цикл 2.***

2.1. Переглядаючи справа наліво, знаходимо першу позицію b[i] таку, що b[i]=0.

Приклад. $b=(0,0,1)$, $i=1$, $b[1]=0$.

2.2. Записуємо b[i]:=1, а всі елементи b[j], j<i, що розміщені праворуч від b[i], задаємо рівними 0.

Приклад. $b[1]:=1$, $b[0]:=0$.

(0,1,0) ***Цикл 3.***

1.1. Переглядаючи справа наліво, знаходимо першу позицію b[i] таку, що b[i]=0.

Приклад. $b=(0,1,0)$, $i=0$, $b[0]:=0$

1.2. Записуємо b[i]:=1, а всі елементи b[j], j<i, що розміщені праворуч від b[i], задаємо рівними 0.

Приклад. $b[0]:=1$. Оскільки $i=0$, то розряди справа відсутні.

(0,1,1) ***Цикл 4.***

2.1. Переглядаючи справа наліво, знаходимо першу позицію b[i] таку, що b[i]=0.

Приклад. $b=(0,1,1)$, $i=1$, $b[2]=0$.

2.2. Записуємо b[i]:=1, а всі елементи b[j], j<i, що розміщені праворуч від b[i], задаємо рівними 0.

Для всіх породжуваних послідовностей елемент b[n] не змінюється, за винятком генерації останнього вектора (1,1,…,1), i=n. Рівність b[n]=1 є умовою зупинки алгоритму.

Приклад. $b[2]:=1$. Оскільки $n=2$, то елемент масиву $b[n]=1$ є ознакою закінчення алгоритму.

3.2.4. Блок-схема алгоритму генерації двійкових векторів довжини n.

Блок 1. Ввод початкових параметрів алгоритму.

$n-$ кількість елементів двійкового вектора.

$m-$ кількість двійкових векторів послідовності, які потрібно згенерувати за допомогою даного алгоритму.

$(b[n-1],b[n-2],...,b[0])-$ початковий вектор послідовності векторів.

Блок 2. $b[n]:=0$. Встановлення початкового стану для ознаки закінчення роботи алгоритму внаслідок досягнення максимального вектора послідовності.

Блок 3. Перевірка ознаки досягнення максимального вектора послідовності.

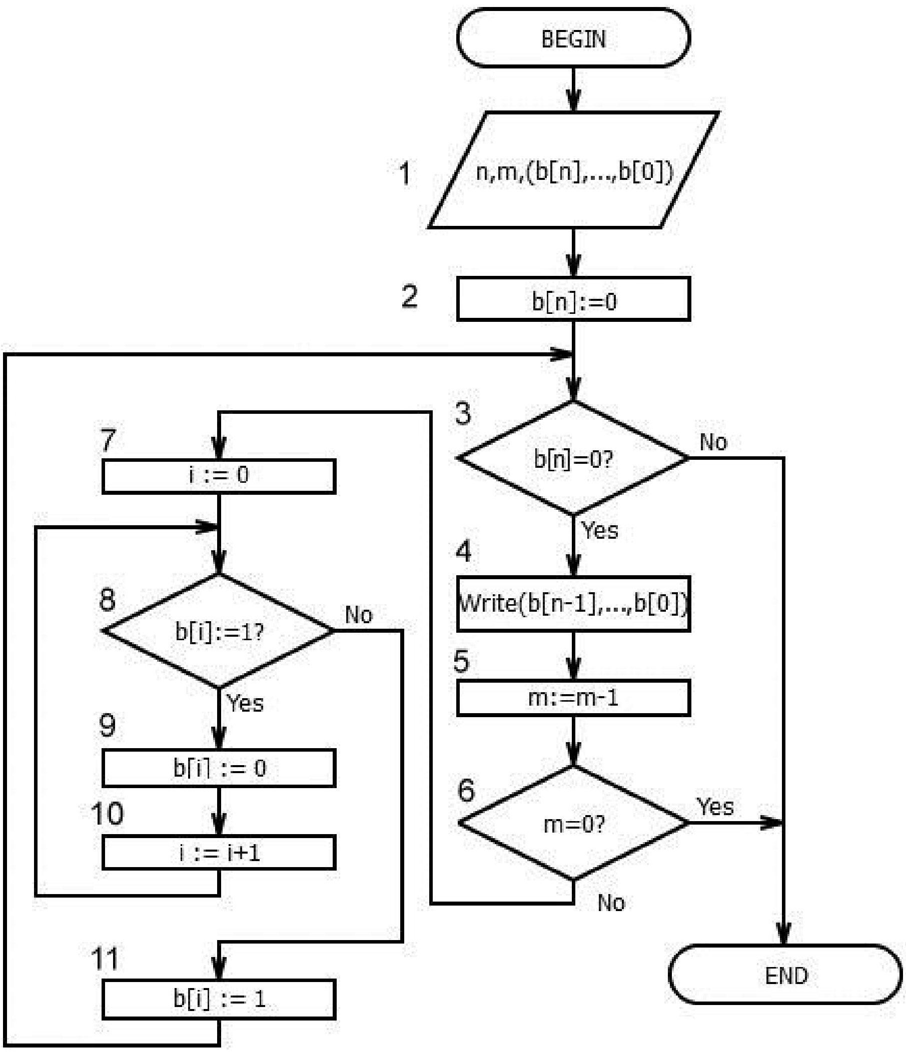


Рис.3.2. Блок-схема алгоритму генерації двійкових векторів довжини n.

Блок 4.Роздруківка чергового вектора послідовності.

Блок 5. Зміна значення лічильника кількості векторів, заданих для генерації.

Блок 6. Перевірка ознаки кількості згенерованих векторів.

Блок 7. Встановити в початкове значення індекс для пошуку.
Блоки 8,9,10. Пошук справа наліво першої нульової позиції з одночасним обнулінням переглянутих одиничних елементів.

Блок 11.Встановлення в 1 знайденої в результаті пошуку справа наліво нульової позиції.
Після виконання блоку 11 завжди відбуваються перевірки на досягнення максимального значення двійкового вектора та на досягнення заданої кількості згенерованих векторів.

3.2.5. Алгоритм генерації підмножин заданої множини.

Нехай дано множину $A = \{a_0, a_1, ..., a_i, ..., a_{n-1}\}$. Введемо фіктивний елемент $a_n \in A$.

Будемо далі розглядати алгоритм пошуку підмножин B^i множини $A = \{a_0, a_1, ..., a_i, ..., a_{n-1}, a_n\}$.

Кожна наступна підмножена B^i формується на основі попередньої підмножини B^{i-1}

Приклад. $A = \{a_0, a_1, a_2\}$ $n = 2$

Першою завжди вибираємо пусту підмножину $B^0 = \varnothing$.

$\varnothing$ ***Цикл 1.***

1.1. Переглядаємо елементи множини в порядку збільшення індекса i з метою пошуку відсутнього в поточній підмножині елемента.

Приклад. $a_0 \notin B^0$**.**

1.2. Для формування наступної підмножини включаємо в попередню підмножину знайдений відсутній елемент та виключаємо з попередньої підмножини всі елементи з індесом, що не перевищує індекс включеного елемента.

Приклад. $B^1 := B^0 \cup a_0$**.** Оскільки B^0 - це пуста множина, то виключати нічого.

$B^1 = \{a_0\}$ ***Цикл 2.***

2.1. Переглядаємо елементи множини в порядку збільшення індекса i з метою пошуку відсутнього в поточній підмножині елемента.

Приклад. $a_1 \notin B^1$**.**

2.2. Для формування наступної підмножини включаємо в попередню підмножину знайдений відсутній елемент та виключаємо з попередньої підмножини всі елементи з індесом, що не перевищує індекс включеного елемента.

Приклад. Включаємо a_1: $B^2 := B^1 \cup a_1$**.** Виключаємо $a_0 : B^2 := B^2 \setminus \{a_0\}$.

$B^2 = \{a_1\}$ ***Цикл 3.***

2.1. Переглядаємо елементи множини в порядку збільшення індекса i з метою пошуку відсутнього в поточній підмножині елемента.

Приклад. $a_0 \notin B^2$**.**

2.2. Для формування наступної підмножини включаємо в попередню підмножину знайдений відсутній елемент та виключаємо з попередньої підмножини всі елементи з індесом, що не перевищує індекс включеного елемента.

Приклад. Включаємо a_0: $B^3 := B^2 \cup a_0$**.** Оскільки індекс $i = 0$ є мінімальним, то в даному випадку немає виключень з підмножини B^3

$B^3 = \{a_0, a_1\}$ ***Цикл 4.***

2.1. Переглядаємо елементи множини в порядку збільшення індекса i з метою пошуку відсутнього в поточній підмножині елемента.

Приклад. $a_2 \notin B^3$**.**

2.2. Для формування наступної підмножини включаємо в попередню підмножину знайдений відсутній елемент та виключаємо з попередньої підмножини всі елементи з індесом, що не перевищує індекс включеного елемента.

Приклад. Оскільки елемент a_2 є фіктивним елементом, введеним в множину в якості ознаки закінчення роботи алгоритму, то вибір цього елемента означає, що всі підмножини B^i множини A згенеровано. Тому в даному випадку алгоритм закінчує свою роботу.

3.2.6. Блок-схема алгоритму генерації підмножин заданої множини .

Блок 1. Ввод початкових параметрів алгоритму.

$n -$ кількість елементів множини.

$m -$ кількість підмножин, які потрібно згенерувати.

початкова множина.

Блок 2. $B := \varnothing$. Встановлення початкового стану для ознаки закінчення роботи алгоритму після генерації послідовності всіх підмножин.

Блок 3. Перевірка ознаки досягнення максимальної кількості підмножин.

Блок 4. Роздруківка чергової підмножини.

Блок 5. Зміна значення лічильника кількості підмножин, заданих для генерації.

Блок 6. Перевірка ознаки кількості згенерованих підмножин.

Блок 7. Встановити в початкове значення індекс для пошуку.

Блоки 8,9,10. Пошук першого елемента у напрямку зростання індекса, який не входить в поточну підмножину з видаленням знайдених елементів.

Блок 11. Обєднання знайденого елемента з поточною підмножиною.

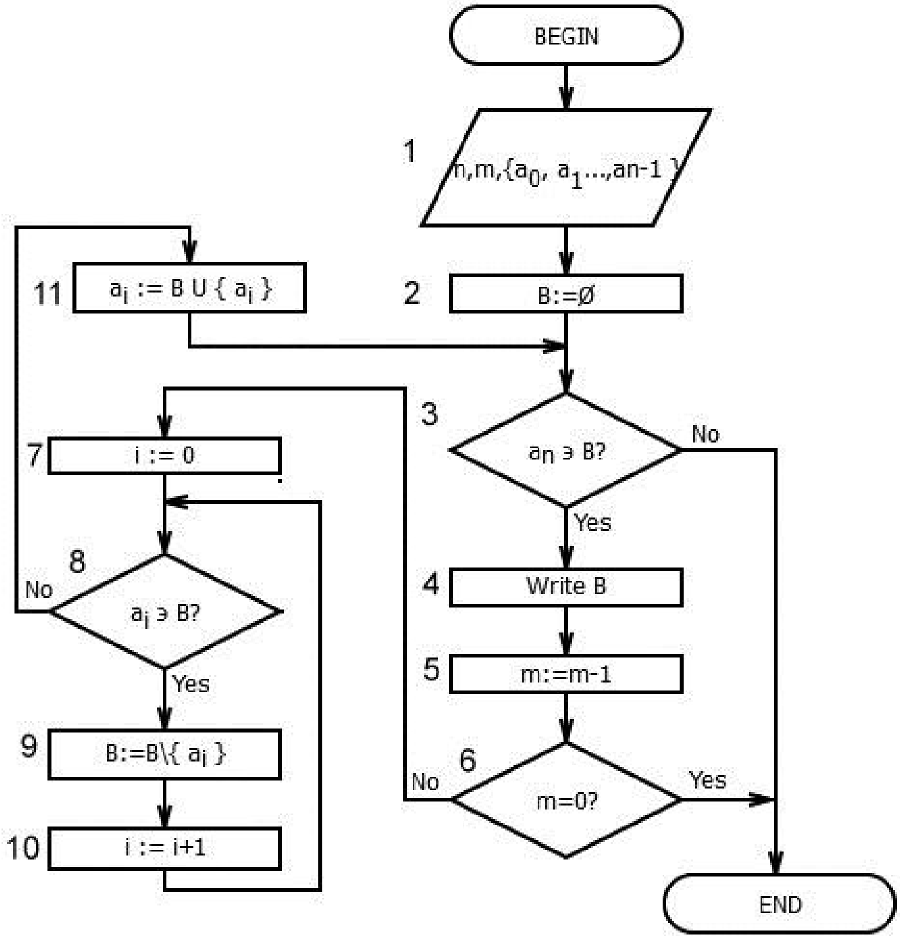


Рис.3.3. Блок-схема алгоритму генерації підмножин заданої множини.

3.2.7. Перший алгоритм генерації коду Грея.

Нехай $b_1 b_2 ... b_n$ – деяке двійкове число.

Код Грея числа одержують шляхом зсуву цього числа на один розряд вправо, відкидання самого правого (n -го розряду), порозрядного додавання по модулю два із цим же числом без зсуву:

$$\begin{array}{r} b_1 b_2 b_3 ... b_{n-1} b_n \\ \oplus b_1 b_2 b_3 ...... b_{n-1} \cancel{b_n} \\ \hline c_1 c_2 c_3 .... c_{n-1} c_n \end{array}$$

Таким чином, кожний результуючий розряд c_i,. одержують по формулі $c_i = b_i \oplus b_{i-1}$, вважаючись, що $b_0 = 0$.

Приклад. Розглянемо порядок генерації коду Грея для трьохрозрядных двійкових чисел:

i	Двоичн. число	Операция	Код Грея
0	000	$000 \oplus 00 = 000$	000
1	001	$001 \oplus 00 = 001$	001
2	010	$010 \oplus 01 = 011$	011
3	011	$011 \oplus 01 = 010$	010
4	100	$100 \oplus 10 = 110$	110
5	101	$101 \oplus 10 = 111$	111
6	110	$110 \oplus 11 = 101$	101
7	111	$111 \oplus 11 = 100$	100

Виберемо початковий код $(b_1 b_2 b_3) = 000$.

Алгоритм містить цикл, кожний прохід якого забезпечує перетворення двійкового коду в код Грея.

(000) ***Цикл 1.***

1.1. Виконуємо ссув вправо на один двійковий розряд, відкидаючи ссунутий правий разряд та приписуючи 0 зліва.

$(g_0\, g_1\, g_2) := 000\ \mathrm{shr}1$. Звідси одержуємо $(g_0\, g_1\, g_2) = 000$.

1.2. Виконуємо порозрядне додавання по модулю 2 двох чисел: $(b_1 b_2 b_3)$ та $(g_0\, g_1\, g_2)$: $(g_1 g_2 g_3) := (b_1 b_2 b_3)\,\mathrm{xor}\,(g_0 g_1 g_2)$ Звідси $(g_1 g_2 g_3) := 000$

(001) ***Цикл 2.***

2.1. Виконуємо ссув вправо на один двійковий розряд, відкидаючи ссунутий правий разряд та приписуючи 0 зліва.

$(g_0\, g_1\, g_2) := 001\,\mathrm{shr}1$. Звідси одержуємо $(g_0\, g_1\, g_2) = 000$.

2.2. Виконуємо порозрядне додавання по модулю 2 двох чисел: $(b_1 b_2 b_n)$ та $(g_0\, g_1\, g_2)$: $(g_1 g_2 g_3) := (b_1 b_2 b_3)\,\mathrm{xor}\,(g_0 g_1 g_2)$ Звідси $(g_1 g_2 g_3) := 001$

(010) ***Цикл 3.***

3.1. Виконуємо ссув вправо на один двійковий розряд, відкидаючи ссунутий правий разряд та приписуючи 0 зліва.

$(g_0\, g_1\, g_2) := 010\ \mathrm{shr}1$. Звідси одержуємо $(g_0\, g_1\, g_2) = 001$.

3.2. Виконуємо порозрядне додавання по модулю 2 двох чисел: $(b_1 b_2 b_n)$ та $(g_0\, g_1\, g_2)$: $(g_1 g_2 g_3) := (b_1 b_2 b_3)\,\mathrm{xor}\,(g_0 g_1 g_2)$ $011 := 010\,\mathrm{xor}\,001$

Звідси $(g_1 g_2 g_3) := 011$

(011) ***Цикл 4.***

3.1. Виконуємо ссув вправо на один двійковий розряд, відкидаючи ссунутий правий разряд та приписуючи 0 зліва.

$(g_0\, g_1\, g_2) := 011\,\mathrm{shr}1$. Звідси одержуємо $(g_0\, g_1\, g_2) = 001$.

3.2. Виконуємо порозрядне додавання по модулю 2 двох чисел: $(b_1 b_2 b_n)$ та $(g_0\, g_1\, g_2)$: $(g_1 g_2 g_3) := (b_1 b_2 b_3)\,\mathrm{xor}\,(g_0 g_1 g_2)$ $010 := 011\,\mathrm{xor}\,001$

Звідси $(g_1 g_2 g_3) := 010$

(100) ***Цикл 5.***

3.1. Виконуємо ссув вправо на один двійковий розряд, відкидаючи ссунутий правий разряд та приписуючи 0 зліва.

$(g_0\, g_1\, g_2) := 100\ \mathrm{shr}1$. Звідси одержуємо $(g_0\, g_1\, g_2) = 010$.

3.2. Виконуємо порозрядне додавання по модулю 2 двох чисел: $(b_1 b_2 b_n)$ та $(g_0\, g_1\, g_2)$: $(g_1 g_2 g_3) := (b_1 b_2 b_3)\,\mathrm{xor}\,(g_0 g_1 g_2)$ $110 := 100\,\mathrm{xor}\,010$

Звідси $(g_1 g_2 g_3) := 110$

Максильмано для трьохрозрядного числа виконуємо $2^3 = 8$ циклів.

3.2.8. Блок-схема першого алгоритму генерації коду Грея.
<u>Блок 1.</u> Ввод початкових параметрів алгоритму.
$n -$ кількість розрядів числа у двійковій системі числення.
$m -$ кількість чисел у коді Грея, які потрібно згенерувати.
$(b_1 b_2 \ldots b_n)$ початкове число у двійковій системі числення.

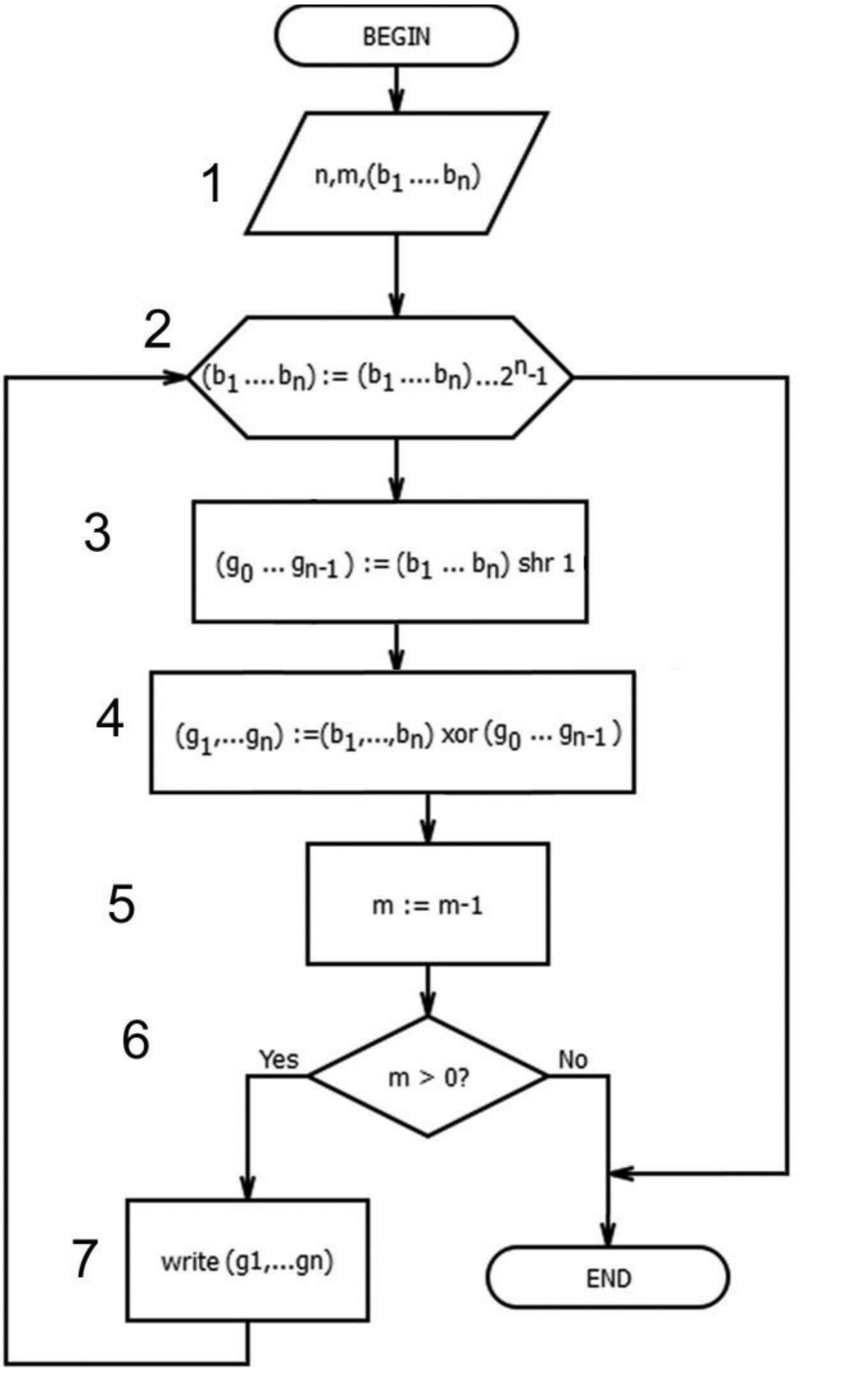


Рис.3.4. Блок-схема першого алгоритму генерації коду Грея.

Блок 2. Головний цикл генерації чисел у коді Грея починаючи з числа $(b_1 b_2 ... b_n)$ у двійковій системі числення та закінчуючи числом $2^n - 1$, яке є максимальним $n-$розрядним числом.
Блок 3. Операція зсуву вправо на один розряд числа $(b_1 ... b_n)$. Результатом зсуву є число $(g_0 g_1 ... g_{n-1})$.
Блок 4. Операція порозрядного додавання по модулю два числа $(b_1 ... b_n)$ та числа $(g_0 g_1 ... g_{n-1})$. Результатом цієї операції є число $(g_1 g_2 ... g_n)$, яке є числом у коді Грея, що відповідає числу $(b_1 b_2 ... b_n)$ у двійковій системі числення.
Блок 5. Зміна значення лічильника кількості чисел, заданих для генерації.
Блок 6. Перевірка ознаки кількості згенерованих чисел.
Блок 7. Роздруківка чергового числа у коді Грея.

3.2.9. Другий алгоритм генерації коду Грея

Другий алгоритм генерації коду Грея містить наступні кроки:
1. У якості базової використовуємо двохрозрядну послідовність: 00,01,11,10.
2. Будуємо трьохрозрядну послідовність:
   2а. Припишемо до 00,01,11,10 праворуч 0: 000,010,110,100.
   2б. Переставимо елементи 00,01,11,10 у зворотному порядку: 10,11,01,00.
   2в. До елементів 10,11,01,00 припишемо праворуч 1: 101,111,011,001.
   2г. Об’єднаємо послідовності з п.2а й п.2в:
   000, 010,110,100,101,111,011,001.
3. Для одержання четирьохрозрядної послідовності перейдемо до п.1 алгоритму, замінивши двохрозрядну послідовність послідовністю, що отримано в п. 2г. даного алгоритму.
4. Повторюючи дії $n-2$ рази, одержимо $n-$розрядний код Грея.

Реалізація даного алгоритму містить два вкладені цикли.
У зовнішньому циклі відбувається нарощування розрядів згенерованого коду Грея, а у внутрішньому циклі формується послідовність всіх чисел у коді Грея в рамках даної кількості розрядів.

3.2.10. Блок-схема другого алгоритму генерації коду Грея.

Блок 1. Ввод початкових параметрів алгоритму.
$n-$ кількість розрядів чисел у коді Грея, які потрібно згенерувати.
$A = (00, 01, 11, 10)$ - початкова послідовність чисел, яка відповідає двохрозрядним числам у коді Грея.
Блок 2. Початкова установка лічильника поточної кількості розрядів чисел, що генеруються алгоритмом у коді Грея.
Блок 3. Лічильник кількості розрядів. Алгоритм починає свою роботу з початкової кількості розрядів, яка дорівнює 2.
Блок 4. Цикл генерації $(i+1)$-розрядної послідовності чисел у коді Грея на основі $i-$ розрядної послідовності

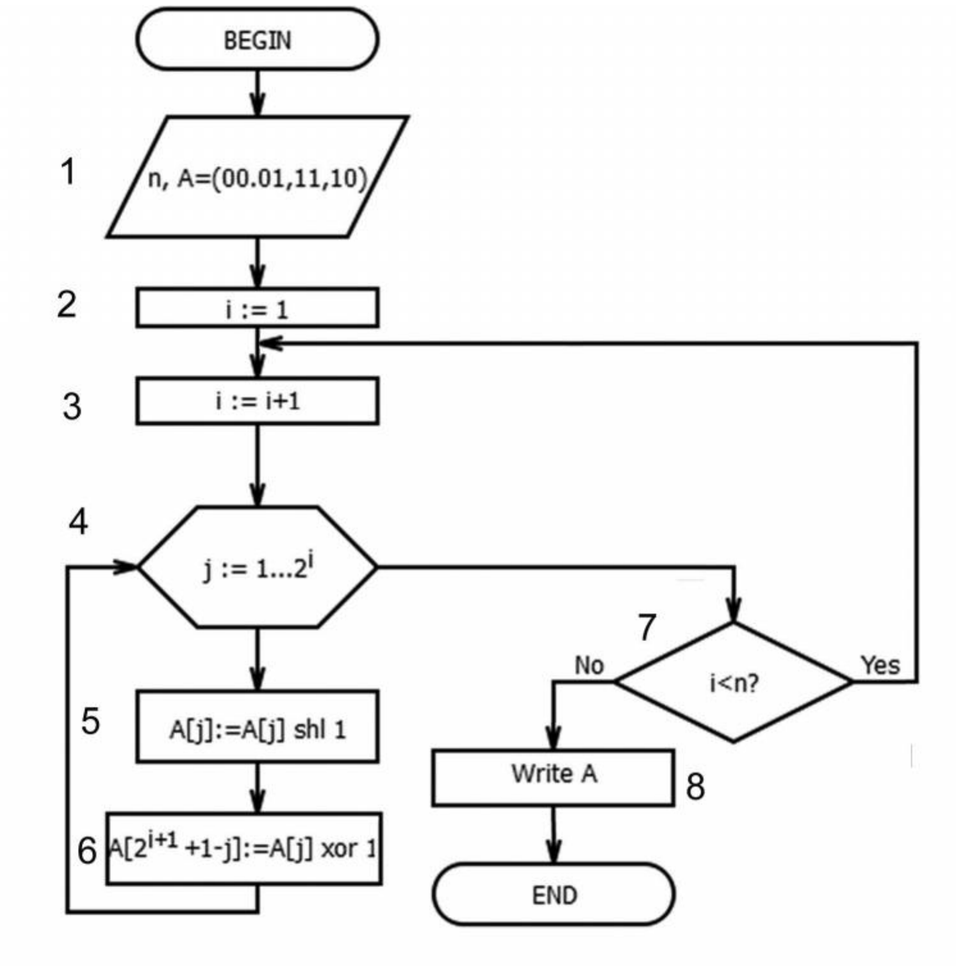


Рис.3.5. Блок-схема другого алгоритму генерації коду Грея

Блок 5. Зсув вліво на один розряд чисел початкової послідовності у коді Грея. Операція аналогічна множенню даного числа на 2.
Наприклад: $0100 * 0010 = 1000$ та $0100 \mathrm{shl} 1 = 1000$

Ця операція дозволяє сформувати першу половину $(i+1)-$розрядної послідовності чисел у коді Грея.

Блок 6. До числа, сформоравоного у попередньому блоці додаємо 1 в молодший розряд. Одержані числа записуємо, починаючи з кінця $(i+1)-$ розрядної послідовності. Таким чином формуємо другу половину $(i+1)-$розрядної послідовності чисел у коді Грея.

Блок 7. Перевірка ознаки досягнення розрядності n послідовності исел у коді Грея.

Блок 8. Роздруківка послідовності n-розрядних чисел у коді Грея.

3.2.11. Перший алгоритм генерації підмножин з умовою мінімальної відмінності елементів.

Завдання полягає у генеруванні всіх підмножин множини $A = \{a_1, a_2, a_3\}$ з умовою мінімальної відмінності сусідніх породжуваних підмножин.

Для розв’язання цієї задачі необхідно поставити у відповідність кожному елементу множини a_i той же по номеру розряд числа у коді Грея. Формування чергової підмножини множини A відбувається шляхом виключення з множини A тих елементів для яких відповідні розряди коду Грея дорівнюють 0.

Результати представимо у вигляді таблиці:

i	$b_1b_2b_3$	$g_1g_2g_3$	B_i
0	000	000	$\varnothing$
1	001	001	a_3
2	010	011	a_2, a_3
3	011	010	a_2
4	100	110	a_1, a_2
5	101	111	a_1, a_2, a_3
6	110	101	a_1, a_3
7	111	100	a_1

Для реалізації цього алгоритму використаємо перший спосіб генерації коду Грея.

3.2.12. Блок-схема першого алгоритму генерації підмножин з умовою мінімальної відмінності елементів.

Блок 1. Ввод початкових параметрів алгоритму.
n – кількість розрядів числа у двійковій системі числення.
m – кількість чисел у коді Грея, які потрібно згенерувати.
$(b_1 b_2 ... b_n)$ - початкове число у двійковій системі числення.
$\{a_1, a_2, ..., a_n\}$ - базова множина для формування підмножин з умовою мінімальної відмінності елементів.

Блок 2. Цикл генерації чисел у коді Грея починаючи з числа $(b_1b_2...b_n)$

Блок 3. Операція зсуву вправо на один розряд числа $(b_1...b_n)$. Результатом зсуву є число $(g_0g_1...g_{n-1})$.

Блок 4. Операція порозрядного додавання по модулю два числа $(b_1...b_n)$ та числа $(g_0g_1...g_{n-1})$. Результатом цієї операції є число $(g_1g_2...g_n)$, яке є числом у коді Грея, що відповідає числу $(b_1b_2...b_n)$ у двійковій системі числення.

Блок 5. Встановлення в початковий стан для формування чергової підмножини з використанням числа у коді Грея.

Блок 6. Цикл формування підмножини шляхом послідовного аналізу розрядів числа $(g_1g_2...g_n)$ к коді Грея

Блок 7. Перевірка значення розряду числа у коді Грея

Блок 8. Операція виконується, якщо значення поточного розряду симла у коді Грея дорівнює 1. У цьому випадку до підмножини додається відповідний

елемент a_i множини
$\{a_1, a_2, ..., a_n\}$.

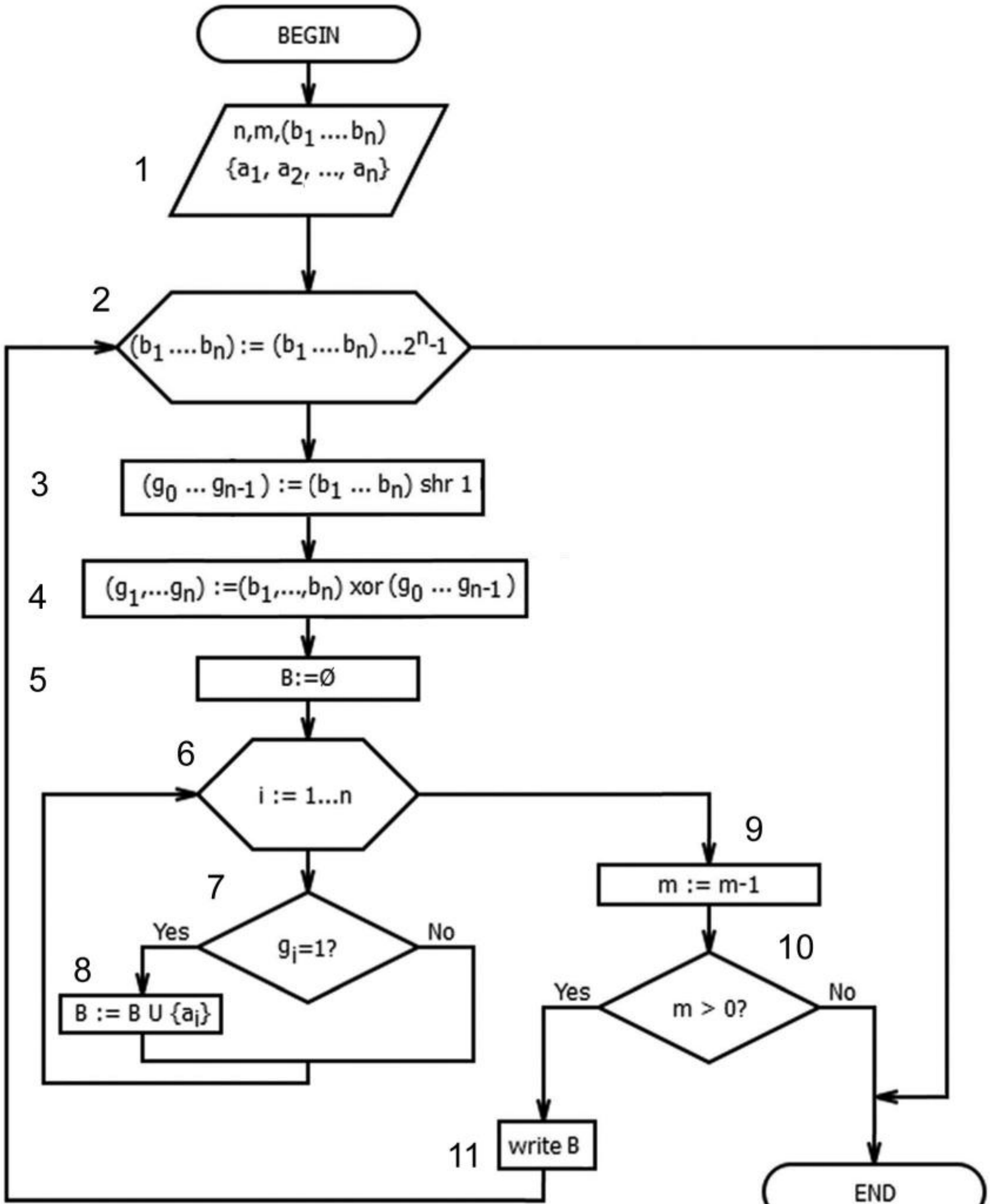


Рис.3.6. Блок-схема першого алгоритму генерації підмножин з умовою мінімальної відмінності елементів.

Блок 9. Зміна значення лічильника кількості згенерованих підмножин.

Блок 10. Перевірка стану лічильника кількості згенерованих підмножин.

Блок 11. Роздруківка чергової підмножини B .

3.2.13. Другий алгоритм генерації підмножин з умовою мінімальної відмінності елементів.

Як і в попередньому випадку, будемо генерувати підмножини множини $X = \{x_1, x_2, x_3\}$ з умовою мінімальної відмінності сусідніх породжуваних підмножин.

Для цього поставимо у відповідність кожному елементу x_i множини X такий же за номером розряд числа $\left(g_1,...,g_n\right)$ у коді Грея. Формування чергової підмножини множини X відбувається шляхом додавання до пустої множини B тих елементів множини X, для яких відповідні розряди числа у коді Грея дорівнюють 1.
Модифікуємо другий алгоритм генерації чисел у коді Грея .

3.2.14. Блок-схема другого алгоритму генерації підмножин з умовою мінімальної відмінності елементів.

Блок 1. Ввод початкових параметрів алгоритму.
$n-$ кількість розрядів чисел у коді Грея, які потрібно згенерувати.
$A=\left(00,01,11,10\right)$ - початкова послідовність чисел, яка відповідає двохрозрядним числам у коді Грея.
$\left\{x_1,x_2,...,x_n\right\}$ - базова множина для формування підмножин з умовою мінімальної відмінності елементів.
Блок 2. Початкова установка лічильника поточної кількості розрядів чисел, що генеруються алгоритмом у коді Грея.
Блок 3. Лічильник кількості розрядів. Алгоритм починає свою роботу з початкової кількості розрядів, яка дорівнює 2.
Блок 4. Цикл генерації $\left(i+1\right)$-розрядної послідовності чисел у коді Грея на основі $i-$ розрядної послідовності.
Блок 5. Зсув вліво на один розряд чисел початкової послідовності у коді Грея. Операція аналогічна множенню даного числа на 2.
Наприклад: $0100*0010=1000$ та $0100\mathrm{shl}1=1000$
Ця операція дозволяє сформувати першу половину $\left(i+1\right)-$розрядної послідовності чисел у коді Грея.
Блок 6. До числа, сформоравоного у попередньому блоці додаємо 1 в молодший розряд. Одержані числа записуємо, починаючи з кінця $\left(i+1\right)-$ розрядної послідовності. Таким чином формуємо другу половину $\left(i+1\right)-$розрядної послідовності чисел у коді Грея.
Блок 7. Перевірка ознаки досягнення розрядності n послідовності чисел у коді Грея.
Блок 8. Цикл формування підмножини шляхом послідовного аналізу розрядів числа $\left(g_1g_2...g_n\right)$ к коді Грея
Блок 9. Перевірка значення розряду числа у коді Грея

Блок 10. Операція виконується, якщо значення поточного розряду симла у коді Грея дорівнює 1. У цьому випадку до підмножини додаєтьтся відповідний елемент x_k множини $\left\{x_1,x_2,...,x_k,...,x_n\right\}$.
Блок 11. Роздруківка послідовності n-розрядних чисел у коді Грея.

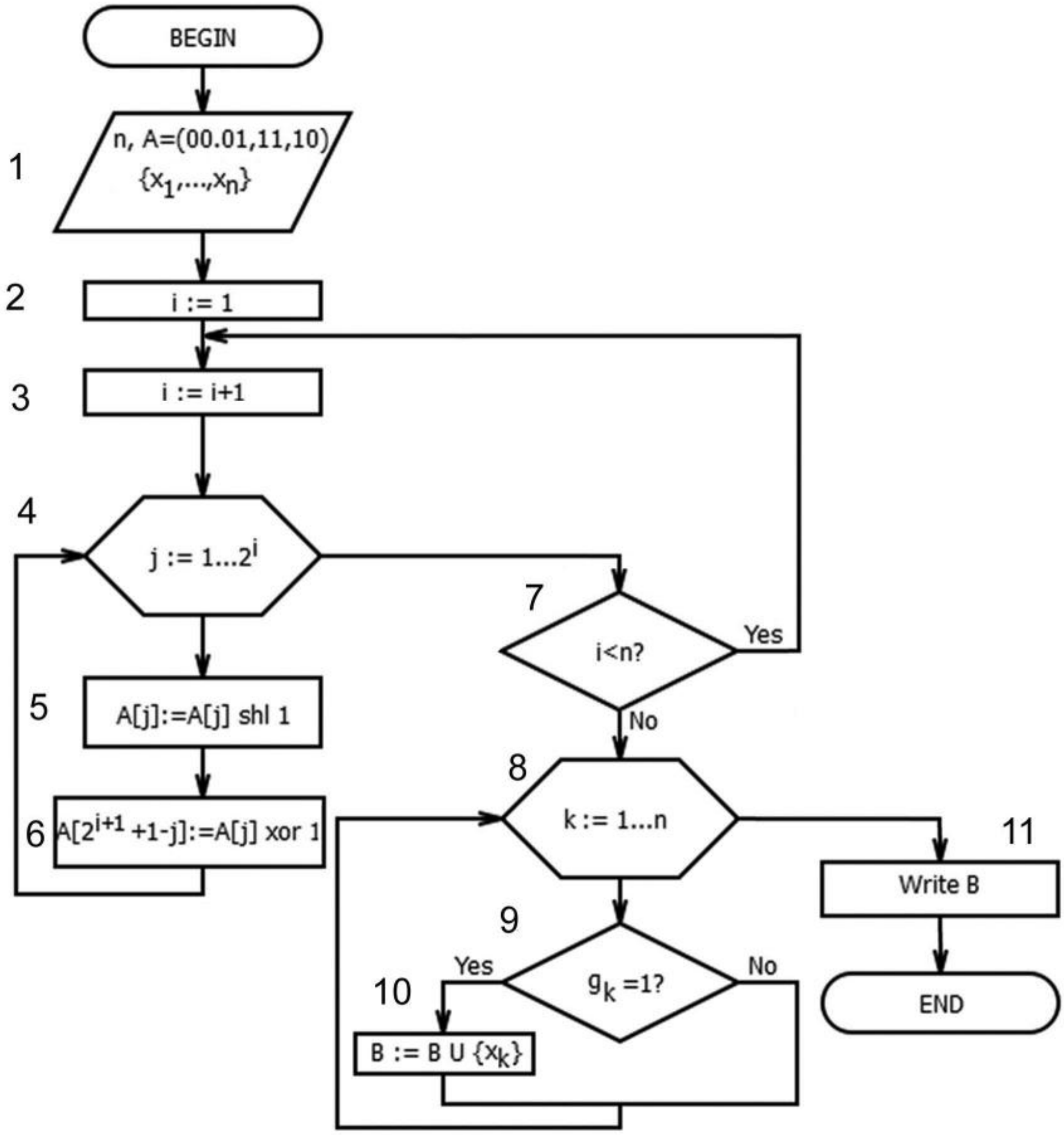


Рис.3.7. Блок-схема другого алгоритму генерації підмножин з умовою мінімальної відмінності елементів.

3.2.15. Алгоритм генерації сполучень з n по k в лексикографічному порядку.

Сполученням з *n* елементів по *k* називається неупорядкована вибірка *k* елементів із заданих *n* елементів. Ми будемо генерувати всі сполучення з *n* по *k* для заданої *n*-елементної множини *A*. Число сполучень з *n* по *k* дорівнює біномальному коефіцієнту

$$C_n^k = \frac{n!}{k!\left(n-k\right)!}$$

Тому найкраща складність, яку можна чекати для алгоритму генерації всіх сполучень дорівнює $O\left(C_n^k\right)$. Без обмеження спільності можна припускати $A=\{1,2,\ldots,n\}$. Довільне сполучення з *n* по *k* зручно представити у вигляді послідовності довжини *k* з чисел, упорядкованих за зростанням зліва направо. Всі такі послідовності, природно породжувати в лексикографічному порядку. Наприклад, при *n* = 5 і *k* = 3 послідовність всіх сполучень в лексикографічному порядку наступна:

123, 124, 125, 134, 135, 145, 234, 235, 245, 345.

Очевидно, що при генерації всіх можливих сполучень перший елемент у лексикографічному порядку є сполучення $(1,2,...,k)$, а останній- $(n-k+1,n-k....,n-1,n)$.

1.Розглянемо сполучення $(a_1,a_2,...,a_k)$.

2. Тоді наступне сполучення визначається з виразу:

$(b_1,b_2,...,b_k)=(a_1,...,a_{p-1},a_p+1,a_p+2,...,a_p+k-p+1)$, де

$p=\max\{i|a_i<n-k+1\}$

3. Наступне сполучення, яке генерується на основі сполучення $(b_1,b_2,...,b_k)$, має вигляд:

$(c_1,...,c_k)=(b_1,...,b_{p'-1},b_{p'}+1,b_{p'}+2,...,b_{p'}+k-p'+1)$, де

$$p'=\begin{cases} p-1, & при\, b_k=n, \\ k, & при\, b_k<n \end{cases}$$

Приклад. Нехай дано початковий терм $A=(a_1,a_2,a_3,a_4,a_5)=(1,2,3,4,5)$. Знайти сполучення з $n=5$ по $k=3$ у лексикографічному порядку.

Тоді перший елемент: $(a_1,a_2,a_3)=(1,2,3)$.

Останній елемент: $(a_3,a_4,a_5)=(3,4,5)$.

Даний алгоритм дозволяє генерувати всі сполучення C_n^k у випадку, коли в якості початкового сполучення вибрано перший елемент (1, 2, 3,…,*k*).

Приклад. Для послідовності (1,2,3,4,5) при обчисленні C_5^3 початковим елементом являється (1,2,3).

Якщо в якості початкового послідовності вибрати $(i+1,i+2,...,i+k)$, то алгоритм забезпечує генерацію сполучень починаючи з $(i+1,i+2,...,i+k)$ і закінчуючи $(n-k+1,n-k+2....,n-1,n)$.

Приклад. Для послідовності (1,2,3,4,5) при обчисленні C_5^3 початковим сполученням виберемо послідовність (2,3,5). Тоді з повної послідовності
(1,2,3), (1,2,4), (1,2,5), (1,3,4), (1,3,5), (1,4,5), (2,3,4), (2,3,5), (2,4,5), (3,4,5)
алгоритм забезпечить генерування таких сполучень: (2,3,5), (2,4,5), (3,4,5).

При необхідності кількість згенерованих сполучень може бути обмеженою.

Приклад. Для послідовності (1,2,3,4,5) при обчисленні C_5^3 початковим сполученням виберемо послідовність (1,2,5) і поставимо вимогу генерації трьох сполучень.
Тоді з повної послідовності
(1,2,3), (1,2,4), (1,2,5), (1,3,4), (1,3,5), (1,4,5), (2,3,4), (2,3,5), (2,4,5), (3,4,5)
алгоритм забезпечить генерування таких сполучень: (1,2,5), (1,3,4), (1,3,5).

3.2.16. Блок-схема алгоритма генерації сполучень з n по k в лексикографічному порядку.

Блок 1. Ввід початкових даних:

$n-$ довжина початкової послідовності.
$k-$ кількість елементів сполучення.
$r-$кількість сполучень, які необхідно згенерувати.
$\left(a_1, a_2, ..., a_k\right)$- початкове сполучення, з якого алгоритм починає генерувати наступні сполучення у лексикографічному порядку.
$a_1 < a_2 < ... < a_k$ - умова задавання початкового сполучення.

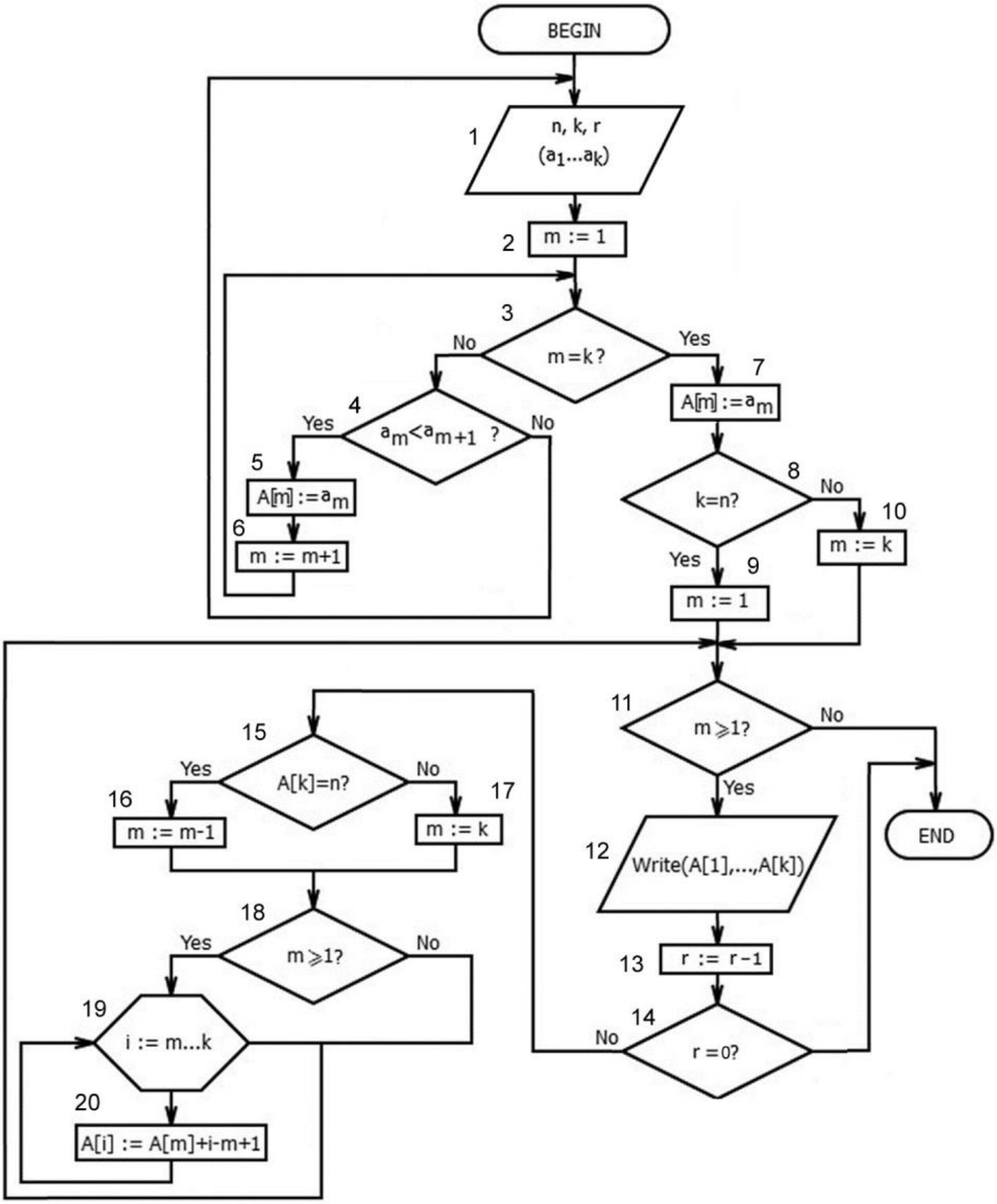


Рис.3.8. Генерація сполучень з n по k в лексикографічному порядку.

Блок 2. Установка в початковий стан лічильніка кількості елементів початкового сполучення $m := 1$.
Блок 3. Перевірка ознаки закінчення оцінки умови правильності задавання елементів початкового сполучення.
Блок 4. Блок виконується у випадку, коли ще не всі елементи початкового сполучення перевірені. В цьому випадку порівнюється поточний елемент початкового сполучення з наступним елементом. Якщо $a_m < a_{m+1}$, то елемент a_m задано правильно.
Блок 5. Значення елемента початкового сполучення a_m записуються в $m-$й елемент масиву $A[m] := a_m$
Блок 6. Інкремент лічильніка кількості елементів початкового сполучення $m := m+1$.
Блок 7. Якщо лічильник m досяг максимально допустимого значення, тобто $m = k$, записуємо цей останній елемент в масив: $A[m] := a_m$.

Увага! Далі змінна m буде використовуватия в новому смисловому значенні. Тепер вона виконує роль ознаки закінчення генерації всіх можливих сполучень з n по k.

Блок 8. Перевіряємо випадок, коли довжина сполучення дорівнює довжині початкової послідовності, тобто чи $n = k$.
Блок 9. Якщо $n = k$, то змінній m надаємо значення 1.
Блок 10. Якщо $k < n$ то $m := k$.
Блок 11. Перевіряємо ознаку закінчення роботи алгоритму m.
Якщо $m \geq 1$, то переходимо до роздруківки чергового сполучення. В протилежному випадку закінчуємо роботу алгоритму.
Блок 12. Роздруківка масиву поточного сполучення A.
Блок 13. Декремент лічильника кількості замовлених сполучень. Цей блок використовують у випадку, коли потрібно згенерувати наперед задану обмежену кількість сполучень. Якщо алгоритм повинен згенерувати всі можливі сполучення, починаючи з почаикового, то задають $r > C_n^k$.
Блок 14. Перевірка ознаки кількості замовлених сполучень. Якщо $r = 0$, то всі замовлені сполучення згенеровано і алгоритм закінчує роботу. В протилежному випадку відбувається перехід до наступної частини алгоритму, яка забезпечує формування нового сполучення.
Блоки 15-20. Алгоритм генерації чергового сполучення.

3.2.17. Алгоритм генерації сполучень з n по k на множині.

Нехай дано довільну множину $R = \{r_1, r_2, ..., r_n\}$. Необхідно створити алгоритм, який забезпечує побудову всіх сполучень з n по k на множині R. Для реалізації цього алгоритму модифікуємо попередній алгоритм генерації сполучень у лексикографічному порядку. Модифікація полягаєв у тому, що при формуванні сполучень будемо використовувати одержані числові сполучення, як індекси сполучень елементів множини R.

Приклад. Розглянемо генерацію сполучень з 5 по 3 на множині $R=\{r_1,r_2,r_3,r_4,r_5\}$. Для цього сформуємо послідовність $A=(1,2,3,4,5)$ та застосуємо до неї алгоритм генерації сполучень при $n=5$ та $k=3$. В результаті одержимо сполучення:

(1,2,3), (1,2,4), (1,2,5), (1,3,4), (1,3,5), (1,4,5), (2,3,4), (2,3,5), (2,4,5), (3,4,5)

Використавши ці сполучення як індекси елементів множини R, можемо записати сполучення з 5 по 3 даної множини:

(r_1,r_2,r_3), (r_1,r_2,r_4), (r_1,r_2,r_5), (r_1,r_3,r_4), (r_1,r_3,r_5),
(r_1,r_4,r_5),(r_2,r_3,r_4), (r_2,r_3,r_5),(r_2,r_4,r_5),(r_3,r_4,r_5)

3.2.18. Блок-схема алгоритму генерації сполучень з n по k на множині.

Блок 1. Ввід початкових даних:

$n-$ Потужнічть множини $R=\{r_1,r_2,...,r_n\}$.

$k-$ кількість елементів сполучення.

$\{r_1,r_2,...,r_n\}$-елементи множини R .

Блоки 2, 3. Генерація числової послідовності індексів елементів множини R.

Блоки 4, 5, 6. Встановлення початкового значення змінної m, нульове значення якої є ознакою закінчення роботи алгоритму.

Блок 7. Перевірка ознаки закінчення роботи алгоритму. Якщо $m\geq 1$, то алгоритм продовжує свою роботу. В протилежному випадку, тобто при $m<1$, алгоритм зупиняється.

Блок 8. Роздруківка чергового сполучення елементів множини R за умови, що індеси цих елементів дорівнюють відповідним значенням елементів масиву A.

Блок 9. Перевіряємо чи не містить останній елемент сполучення $A[k]$ максимального значення n.

Блок 10. Якщо $A[k]=n$, то виконуємо декремент змінної m: $m:=m-1$

Блок 11. Якщо $A[k]\neq n$, то змінну m встновлюємо $m:=k$.

Блок 12. Перевірка ознаки закінчення роботи алгоритму. Якщо $m\geq 1$, то відбувається генерація чергового сполучення . В протилежному випадку, тобто при $m<1$, вудбувається перехід на початок головного цилу та завершення роботи алгоритму.

Блок 13,14. Цикл генерації чергового сполучення з числових індексів, записних в масиві A.

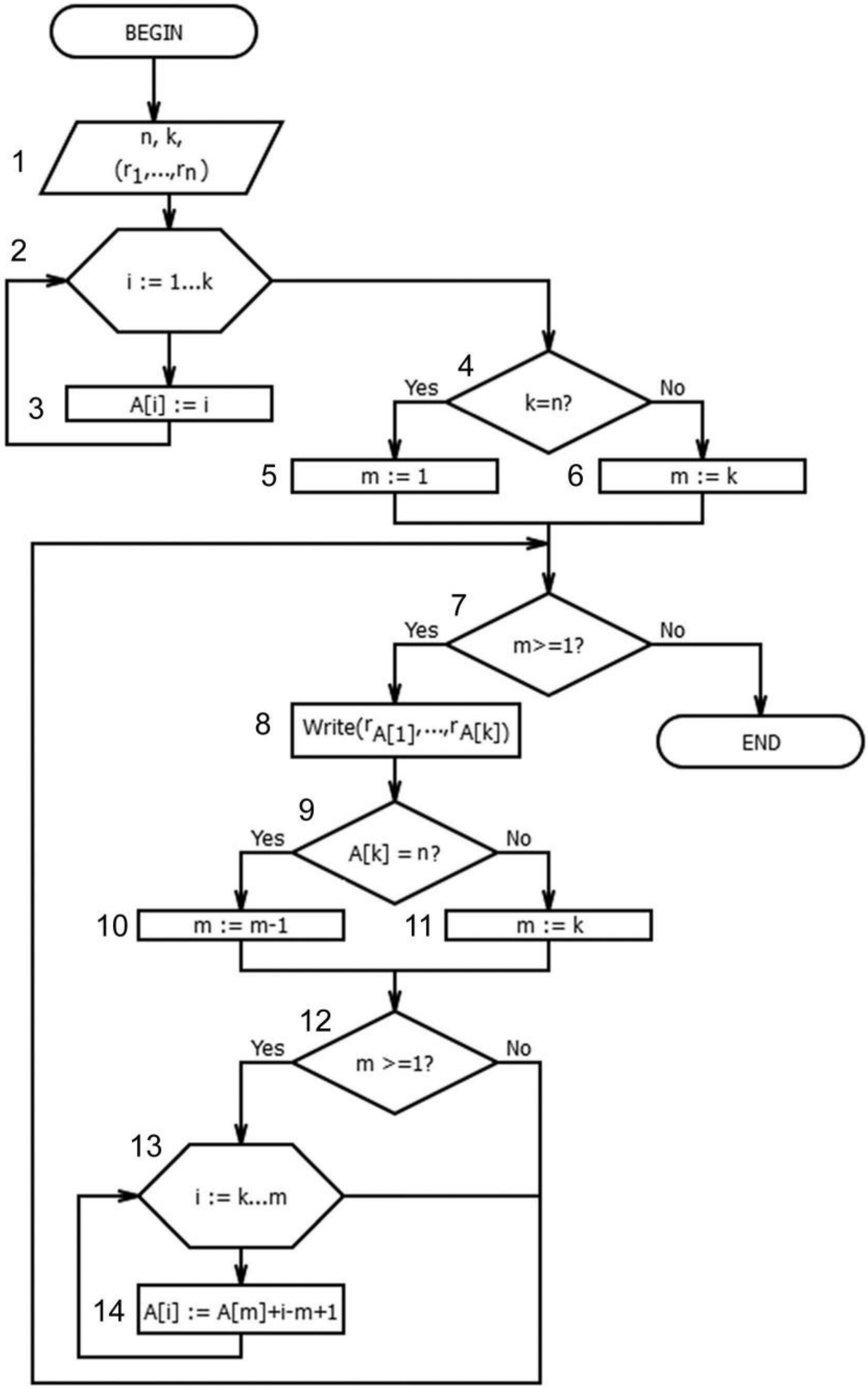


Рис.3.9. Генерація сполучень з n по k на множині.

3.2.19. Алгоритм генерації розбиття числа n у словниковому порядку

На множині цілочисельних послідовностей визначимо словниковий порядок $\leqslant$ (порівняйте визначення словарного порядку з лексикографічним). Нехай $a=\left(a_1,...,a_q\right)$ і $b=\left(b_1,...,b_s\right)$ - довільні цілочисельні послідовності, можливо різної довжини.

Визначення. $\left(a_1,...,a_q\right) \leqslant \left(b_1,...,b_s\right)$, якщо виконується хоча б одна з умов:
або $q \le s$ і $a_i = b_i$ для будь-якого $i \le q$,
або існує $p \le \min\left(s,q\right)$ таке, що $a_p < b_p$ і $a_i = b_i$ для всіх $i < p$. Бінарне відношення $\leqslant$ є лінійним порядком. Для розбиттів числа n визначення словникового порядку дещо спрощується.
Зауваження. Якщо *a, b* - розбиття числа *n,* то

$$a \prec b \Leftrightarrow \exists\, p \le \min\left(s,q\right)\left(a_p < b_p \,\&\, \forall\, i < p\left(a_i = b_i\right)\right).$$

Розбиття числа n будемо породжувати в словниковому порядку. Ясно, що перший розбивкою (найменшим елементом) в словниковому порядку буде послідовність $(1,1,...,1)$ довжини n, а останнім (найбільшим елементом) - одноелементна послідовність (n). З’ясуємо вид розбиття, яке є безпосередньо наступним за розбиттям $a=\left(a_1,...,a_q\right)$ в словниковому порядку. Знайдемо таку найбільш праву позицію $p<q$, в якій число a_p можна збільшити на 1, зберігши властивість спадання послідовності. Далі суму доданків, що залишилися за мінусом одиниці $\left(\sum\limits_{i=p+1}^{q} a_i - 1\right)$, представимо у вигляді суми одиниць.

Неважко довести, що саме таке розбиття b безпосередньо слідує за a. При переході від a до b число необхідних змін над послідовністю a є величина змінна, залежна від n. Цю залежність можна усунути, якщо перейти до іншого способу задавання розбиття $a=\left(a_1,...,a_q\right)$. Впорядкуємо всі різні числа $\left(a_{i_1},...,a_{i_k}\right)$ серед $\left(a_1,...,a_q\right)$ у порядку зменшення $a_{i_1} > ... > a_{i_k}$.

Нехай m_j - число входжень a_{i_j} в розбиття a і $m=\left(m_1,...,m_k\right)$. Ясно, що розбиття a числа n однозначно визначається парою послідовностей $\left(a_{i_1},...,a_{i_k}\right)$ і $\left(m_1,...,m_k\right)$. Тому надалі всяке розбиття a числа n будемо записувати у вигляді

$$a=\left(a_1 \cdot m_1,...,a_k \cdot m_k\right),$$

де $a_1 > a_2 > ... > a_k > 0$, $m_i > 0$, $i = 1,2,...,k$, $1 \le k \le n$, $m = \sum_{i=1}^{k} m_i a_i$.

Елемент $m_i \cdot a_i$ представляє собою послідовність $a_i, a_i, a_i, ..., a_i$ довжини m_i і називається блоком розбиття. Очевидно, що розбиття a є найменшим при $k = 1$ і $m_k = n$, а найбільшим при $k = m_k = 1$. Таке представлення розбиття числа n виключає необхідність пошуку позиції p при перегляді справа наліво поточного розбиття.

Покажемо, що для перестроювання розбиття a в безпосередньо наступне розбиття b потрібно змінити не більше двох блоків.

Теорема. Нехай $a = (a_1 \cdot m_1, ..., a_k \cdot m_k)$- розбиття числа n і розбиття b безпосередньо слідує за a в словниковому порядку. Тоді

1. Якщо $m_k = 1$, то $k \ge 2$ і
$$b = \left(m_1 \cdot a_1, ..., m_{k-2} \cdot a_{k-2}.1 \cdot (a_{k-1} + 1), S' \cdot 1\right).$$

2. Якщо $m_k \ge 2, k \ge 2$ і $a_{k-1} = a_k + 1$, то
$$b = \left(m_1 \cdot a_1, ..., m_{k-2}, a_{k-2}, (m_{k-1} + 1) \cdot a_{k-1}, S \cdot 1\right).$$

3. Якщо $m_k \ge 2, k \ge 2$ і $a_{k-1} \ne a_k + 1$, то
$$b = \left(m_1 \cdot a_1, ..., m_{k-1}, a_{k-1}, 1 \cdot (a_k + 1), S \cdot 1\right).$$

4. Якщо $k = 1$ то $b = \left(1 \cdot (a_k + 1), S \cdot 1\right)$.

Тут $S' = m_k a_k + m_{k-1} a_{k-1} - (a_{k-1} + 1)$ і $S = m_k a_k - (a_k + 1)$.

Доведення.

При переході від a до b необхідно найбільш правий можливий елемент розбиття a збільшити на 1. Такий елемент x буде першим елементом блоку, до якого він входить. Розглянемо два можливі випадки.

Випадок 1. $m_k = 1$. Елемент a_k можна збільшити, зберігши розбиття числа n. Оскільки a- не найбільше розбиття, маємо $k \ge 2$. Так як $a_{k-1} + a_k \ge a_k + 1$, то $x = a_{k-1}$ і x є першим елементом $(k-1)$ блоку. Цей елемент збільшуємо на 1, а суму доданків, що залишилися $S' = m_k a_k + m_{k-1} a_{k-1} - (a_{k-1} + 1)$, представляємо у вигляді суми одиниць.

Випадок 2. $m_k \ge 2$. Тоді $m_k a_k \ge a_k + 1$. Отже, $x = a_k$ і x є першим елементом k-го блоку. Цей елемент збільшуємо на 1, а суму доданків, що залишилися $S = m_k a_k - (a_k + 1)$ представляємо в вигляді суми одиниць. У кожному з цих випадків залишається перерахувати значення a_i і m_i для не більше, ніж двох змінених блоків. Теорема доведена.

Зауваження. Якщо вибрати значення a_0 таке, що $a_0 \neq a_1 + 1$, то в наведеній теоремі можна виключити (4) та умову $k \geq 2$ в (2), (3), зберігши справедливим твердження теореми.

Приклад. Розбиття числа 7 в словниковому порядку:

(7·1) = (1, 1, 1, 1, 1, 1, 1),

(1·2, 5·1) = (2, 1, 1, 1, 1, 1),

(2·2, 3·1) = (2, 2, 1, 1, 1),

(3·2, 1·1) = (2, 2, 2, 1),

(1·3, 4·1) = (3, 1, 1, 1, 1),

(1·3, 1·2, 2·1) = (3, 2, 1, 1),

(1·3, 2·2) = (3, 2, 2),

(2·3, 1·1) = (3, 3, 1),

(1·4, 3·1) = (4, 1, 1, 1),

(1·4, 1·2, 1·1) = (4, 2, 1),

(1·4, 1·3) = (4, 3),

(1·5, 2·1) = (5, 1, 1),

(1·5, 1·2) = (5, 2),

(1·6, 1·1) = (6, 1),

(1·7) = (7).

Розглянемо блок-схему цього алгоритму

3.2.20. Блок-схема алгоритму генерації розбиття числа *n* у словниковому порядку

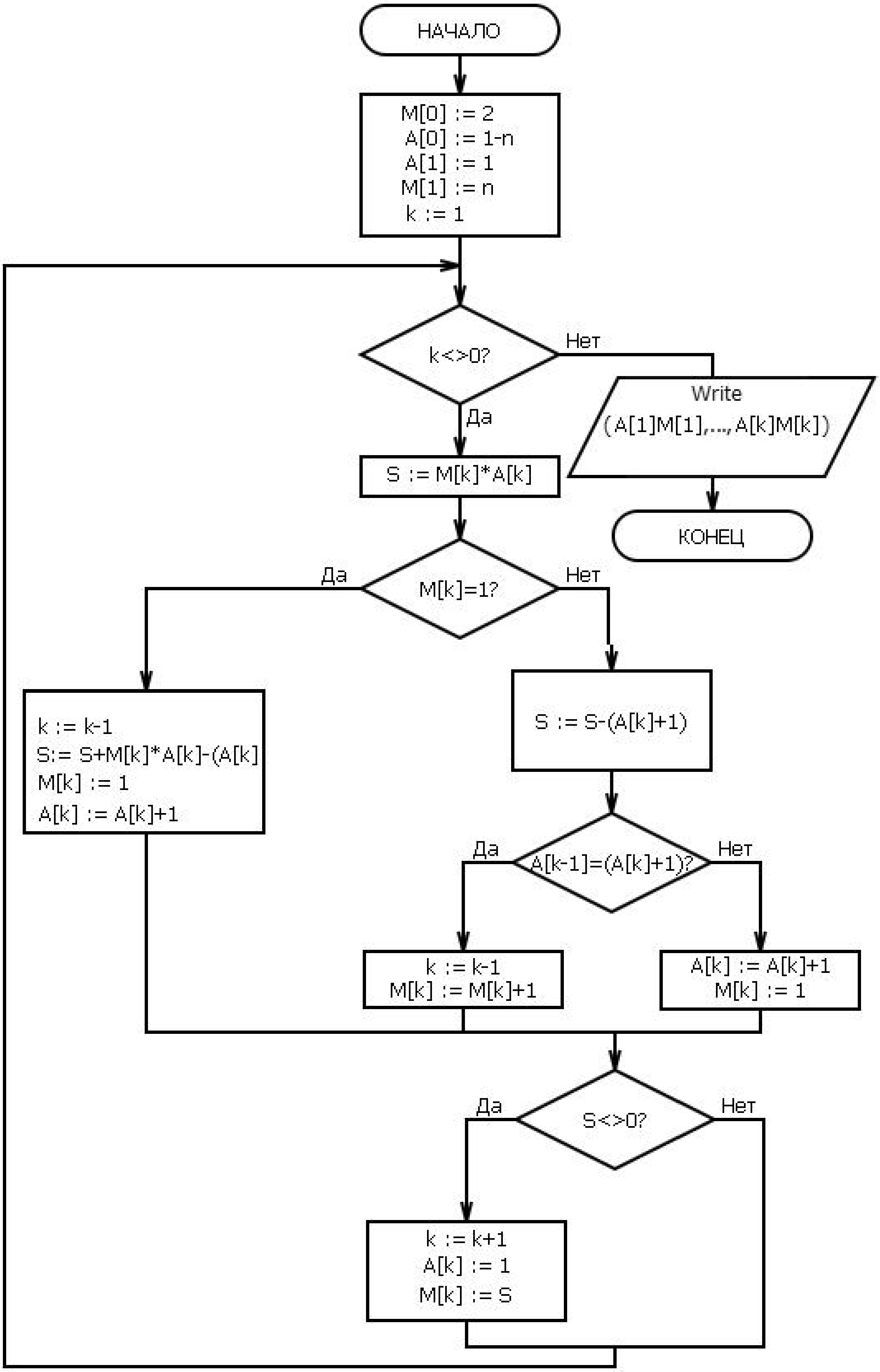


Рис.3.10. Генерація розбиття числа *n* у словниковому порядку

3.2. ЗАВДАННЯ ЛАБОРАТОРНОЇ РОБОТИ

1. Вивчити принципи роботи алгоритму перестановок у лексикографічному порядку. Використовуючи блок-схему, представлену на рис. 3.1. написати програму перестановок чисел десяткової системи числення у лексокографічному порядку.
2. Вивчити принципи роботи алгоритму генерації двійкових векторів довжини n. Написати програму генерації двійкових векторів довжини n, спираючись на блок-схему відповідного алгоритму, представлену на рис 3.2.
3. Вивчити алгоритм генерації підмножин заданої множини, який використовує принципи роботи алгоритму генерації двійкових векторів. Написати програму генерації підмножин заданої множини, застосувавши блок-схему даного алгоритму, представлену на рис 3.3.
4. Вивчити правила формування чисел у коді Грея та розглянути перший алгоритм генерації чисел у цьому коді. Написати програму, що використовує блок-схему першого алгоритму, представлену на рис. 3.4, для генерації чисел у коді Грея.
5. Вивчити правила формування чисел у коді Грея за другим алгоритмом генерації чисел у цьому коді. Написати програму, що використовує блок-схему другого алгоритму, представлену на рис. 3.5, для генерації чисел у коді Грея.
6. Використовуючи перший алгоритм генерації чисел у коді Грея вивчити спосіб формування підмножин з умовою мінімальної відмінності елементів. Написати програму, що використовує блок-схему першого алгоритму генерації підмножин з умовою мінімальної відмінності елементів, представлену на рис. 3.6, для генерації підмножин.
7. Використовуючи другий алгоритм генерації чисел у коді Грея вивчити спосіб формування підмножин з умовою мінімальної відмінності елементів. Написати програму, що використовує блок-схему другого алгоритму генерації підмножин з умовою мінімальної відмінності елементів, представлену на рис. 3.7, для генерації підмножин.
8. Вивчити принципи роботи алгоритму генерації сполучень з n по k в лексикографічному порядку. Використовуючи блок-схему, представлену на рис. 3.8., написати програму сполучень чисел десяткової системи числення у лексокографічному порядку.
9. Вивчити принципи роботи алгоритму генерації сполучень з n по k на множині. Використовуючи блок-схему, представлену на рис. 3.9., написати програму сполучень довільних елементів множини.
10. Вивичити теоретичні основи та базові принципи роботи алгоритму генерації розбиття числа n у словниковому порядку. Використовуючи блок-схему, представлену на рис. 3.10., написати програму для розбиття числа у словниковому порядку.

Вимоги до програмного забезпечення:

1. Лабораторна робота виконується в середовищі візуального програмування Lazarus.

2. В якості початкового коду використати проект LAB3_Project, попередньо скачавши його з сайту викладача.
3. Початкові данні вводяться за допомогою програмних засобів уже реалізованих в згаданому проекті.
4. Необхідно в рамках форми OperForm реалізувати алгоритм відповідно до варіанту лабораторної роботи.
5. На формі OperForm відобразити як початкові дані, так і результати роботи алгоритму.

Зміст звіту:

1. Титульний лист.
2. Тема завдання.
3. Завдання.
4. Роздруківка результатів виконання програми.
5. Роздруківка тексту програми.
7. Аналіз результатів.

Контрольні питання

1. Дати означення перестановок та вивести формулу визначення кількості (числа) перестановок.
2. Розміщення. Навести приклад розміщення.
3. Розміщення з повтореннями.
4. Сполучення.
5. Дати означення лексикографічного порядку
6. Дати означення словникового порядку.
7. Алгоритм перестановок у лексикографічному порядку.
8. Алгоритм генерації сполучень з n по k у лексикографічному порядку
9. Алгоритм генерації розбиття числа n у словниковому порядку.

Варіанти для виконання лабораторної роботи

Номер варіанту I визначається як результат операції I = NZK mod 14+1, де NZK – номер залікової книжки. Номер варіанту відповідає номеру пункту завдання до лабораторної роботи.

№	Опис варіанта
1	Виконати завдання 1 до лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Максимальне значення n дорівнює (10+NZK mod 11). 2. Значення s може змінюватися довільно від 1 до n 3. Сформувати початкову перестановку P таким чином, що кожен її елемент вибирається випадково та без повторень чисел.
2	Змінити завдання 1 до лабораторної таким чином, щоб генерація перестановок відбувалась в антилексикографічному порядку. 1. Максимальне значення n дорівнює (10+NZK mod 11). 2. Значення s може змінюватися довільно від 1 до n

	3. Сформувати початкову перестановку P таким чином, що кожен її елемент вибирається випадково та без повторень чисел..
3	Виконати завдання 2 до лабораторної роботи за умови, що що вхідні параметри приймають такі значення: 1. Максимальне значення n дорівнює номеру залікової книжки (NZK). 2. Значення m може змінюватися довільно від 1 до n 3. Сформувати початкову перестановку $\left(b[n],...,b[0]\right)$ таким чином, що кожен її елемент вибирається випадково.
4	Змінити завдання 2 до лабораторної таким чином, щоб генерація двійкових векторів відбувалась в антилексикографічному порядку. 1. Максимальне значення n дорівнює номеру залікової книжки (NZK). 2. Значення m може змінюватися довільно від 1 до n 3. Сформувати початкову перестановку $\left(b[n],...,b[0]\right)$ таким чином, що кожен її елемент вибирається випадково.
5	Виконати завдання 3 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Множину $\{a_0, a_1, ..., a_{n-1}\}$ сформувати з букв свого прізвища, імені та по батькові, виключивши повторення букв. 2. Максимальне значення n дорівнює кількості одержаних різних букв з прізвища, імені та по батькові.. 3. Значення m може змінюватися довільно від 1 до n
6	Виконати завдання 3 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Множину $\{a_0, a_1, ..., a_{n-1}\}$ сформувати з імен своїх родичів та друзів загальною кількістю не менше 20. 2. Максимальне значення n дорівнює потужності множини імен $n \geq 20$. 3. Значення m може змінюватися довільно від 1 до n
7	Виконати завдання 4 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Початкове число $\left(b_1 b_2 ... b_n\right)$ сформувати шляхом переводу у двійкову систему числення числа, яке сформоване конкатенацією чисел, що відповідають вашому дню, місяцю та року народження. *Приклад*. Дату 12 грудня 1998 перетворюємо у десяткове число 12121998, що відповідає двійковому числу 101110001111011110001110. $12121998_{10} = 101110001111011110001110_2$ 2. Значення n дорівнює кількості розрядів одержаного двійкового числа. *Приклад.* Для числа 101110001111011110001110 кількість розрядів $n = 24$ 3. Значення m може змінюватися довільно від 1 до n
8	Виконати завдання 5 лабораторної роботи. 1. Початкове число $\left(b_1 b_2 ... b_n\right)$ сформувати шляхом переводу у двійкову систему числення числа, яке сформоване конкатенацією чисел, що

	відповідають вашому року, місяцю та дню народження. *Приклад*. Дату 1999 рік, місяць січень, 01 число перетворюємо у десяткове число 19990101, що відповідає двійковому числу 1001100010000011001010101. $$19990101_{10} = 1001100010000011001010101_2$$ 2. Значення n дорівнює кількості розрядів одержаного двійкового числа. *Приклад*. Для числа 1001100010000011001010101 кількість розрядів $n = 26$ 3. Значення m може змінюватися довільно від 1 до n
9	Виконати завдання 6 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Базова множина для формування підмножин $\{a_1, a_2, ..., a_n\}$ складається з імен ваших родичів та друзів загальною кількістю не менше 20. 2. $n -$ потужність множини $\{a_1, a_2, ..., a_n\}$. 3 $m -$ кількість множин, що потрібно згенерувати, може бути довільною від 1 до n. 4. $(b_1 b_2 ... b_n)$ - початкове число у двійковій системі числення може бути довільним з кількістю розрядів n.
10	Виконати завдання 7 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. $A = (00, 01, 11, 10)$ - початкова послідовність чисел, яка відповідає двохрозрядним числам у коді Грея. 2.$\{x_1, x_2, ..., x_n\}$ - базова множина для формування підмножин з повинна складатися з не менше, ніж 20 назв міст України. 3. $n -$ потужність множини $\{x_1, x_2, ..., x_n\}$. 4. $(b_1 b_2 ... b_n)$ - початкове число у двійковій системі числення може бути довільним з кількістю розрядів n.
11	Виконати завдання 8 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Базова множина складається з послідовності чисел (1,2,3,…,n), де $n \ge 32$. 2. $(a_1, a_2, ..., a_k)$- початкове сполучення, з якого алгоритм починає генерувати наступні сполучення у лексикографічному порядку, де k може приймати довільне значення від 1 до n. 3. $r -$кількість сполучень, які необхідно згенерувати. Може приймати довільне значення від 1 до C_n^k. $a_1 < a_2 < ... < a_k$ - умова задавання початкового сполучення.
12	Виконати завдання 9 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Базова множина R складається з проіндесованих елементів, якими є

	міста України. *Приклад.* r_1=Київ, r_2=Харків, r_3=Дніпропетровськ і т.д. 2. Загальна кількість елементів множини $n \geq 16$. 3. k – кількість елементів сполучення може змінюватися довільно від 1 до n.
13	Виконати завдання 9 лабораторної роботи за умови, що вхідні параметри приймають такі значення: 1. Базова множина R складається з проіндесованих елементів, якими є імена ваших друзів та родичів. *Приклад.* r_1=Іван, r_2=Марія, r_3=Ольга і т.д. 2. Загальна кількість елементів множини $n \geq 16$. 3. k – кількість елементів сполучення може змінюватися довільно від 1 до n. 4. Передбачити можливість виводу обмеженої кількості підмножин
14	Виконати завдання 10 лабораторної роботи за умови, що в якості числа яке підлягає розбиттю, можу бути довільне число від 1 до 100.

Лабораторна робота №4

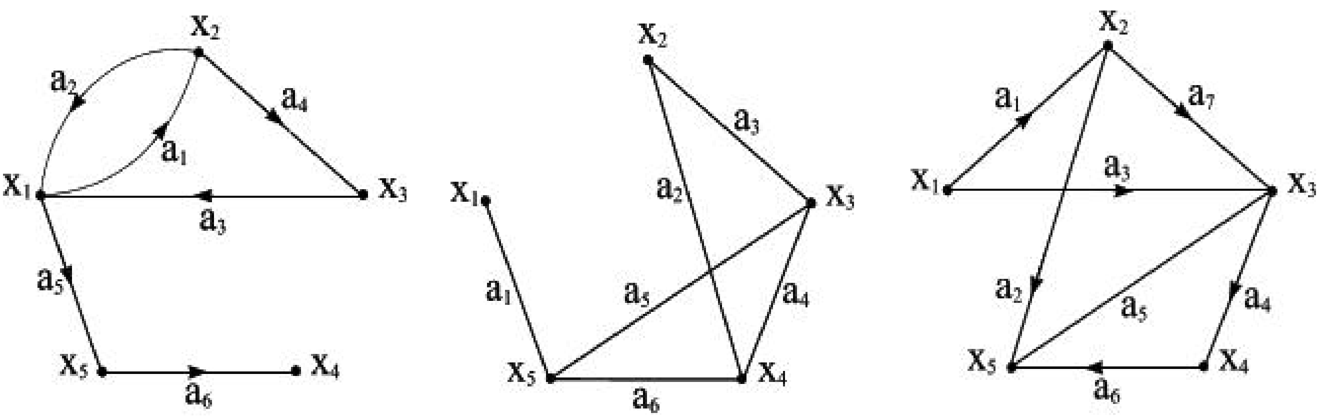
Тема: «Графи. Способи представлення графів. Остовні дерева. Пошук найкоротших шляхів»
Мета роботи: Вивчення властивостей графів, способів їх представлення та основних алгоритмів на графах.
Завдання : створити програму, яка реалізує один з алгоритмів на графах.

Теоретичні основи

4.1. ОСНОВНІ ОЗНАЧЕННЯ

Останнім часом теорія графів стала простим, доступним і потужним засобом вирішення питань, що відносяться до широкого кола проблем. Це проблеми проектування інтегральних схем і схем управління, дослідження автоматів, логічних ланцюгів, блок-схем програм, економіки та статистики, хімії та біології, теорії розкладів і дискретної оптимізації. **Граф** *G* задається множиною точок або вершин x_1, x_2, ..., x_n (яке позначається через *X*) і множиною ліній або ребер a_1, a_2, ..., (яке позначається символом *A*), що з'єднують між собою всі або частину цих точок. Таким чином, граф *G* повністю задається (і позначається) парою (*X*, *A*).

Якщо ребра з множини *A* орієнтовані, що зазвичай показується стрілкою, то вони називаються дугами, і граф з такими ребрами називається орієнтованим графом (рис. 4.1 (а)). Якщо ребра не мають орієнтації, то граф називається неорієнтованим (рис. 4.1 (б)). У разі, коли *G* = (*X*, *A*) є орієнтованим графом і ми хочемо знехтувати спрямованістю дуг з множини *A*, то неорієнтований граф, відповідний *G*, будемо позначати як G = (*X*, *A*).



а б в

Рис. 4.1 (а) – орієнтований граф; (б) – неорієнтований граф; (в) – змішаний граф

Якщо дуга позначається впорядкованою парою, що складається з початкової та кінцевої вершин (тобто двома кінцевими вершинами дуги), її напрямок передбачається заданим від першої вершини до другої. Так, наприклад, на рис.4.1 (а) позначення (x_1, x_2) відноситься до дуги a_1, а (x_2, x_1) – до дуги a_2.

Інший, вживаний частіше, опис орієнтованого графа G полягає у задаванні множини вершин X і *відповідності* Г, яка показує, як між собою пов'язані вершини. Відповідність Г називається відображенням множини X в X, а граф в цьому випадку позначається парою $G = (X, Г)$.

Для графа на рис.4.1 (а) маємо $Г (x_1) = \{x_2, x_5\}$, тобто вершини x_2 і x_5 є кінцевими вершинами дуг, у яких початковою вершиною є x_1.

$Г (x_2) = \{x_1, x_3\}$, $Г (x_3) = \{x_1\}$, $Г (x_4) =\varnothing$ - порожня множина, $Г (x_5) = \{x_4\}$

У разі неорієнтованого графа або графа, що містить і дуги, і неорієнтовані ребра (див., наприклад, графи, зображені на рис.4.1 (б) і рис.4.1 (в)), передбачається, що відповідність Г задає такий еквівалентний орієнтований граф, який отримуємо з вихідного графа заміною кожного неорієнтованого ребра двома протилежно спрямованими дугами, що з'єднують ті ж самі вершини. Так, наприклад, для графа, наведеного на рис.4.1 (б), маємо $Г (x_5) = \{x_1, x_3, x_4\}$, $Г(x_1) = \{x_5\}$ і ін.

Оскільки пряма відповідність або образ вершини $Г(x_i)$ є множиною таких $x_j\in X$, для яких в графі G існує дуга (x_i,x_j), то через $Г^{-1}(x_i)$ природно позначити множину вершин x_k, для яких в G існує дуга $(x_к,x_j)$. Таку відповідність прийнято називати *зворотною відповідністю* або *прообразом* вершини. Для графа, зображеного на рис.4.1(а), маємо

$Г^{-1}(x_1)=\{x_2,x_3\}$, $Г^{-1}(x_2)=\{x_1\}$ і т. д.

Цілком очевидно, що для неорієнтованого графа $Г^{-1}(x_i)=Г(x_i)$ для всіх $x_i\in$ X.

Коли відображення Г діє не на одну вершину, а на множину вершин $X_q=\{x_1,x_2,...,x_q\}$, то під $Г(X_q)$ розуміють об'єднання $Г(x_1)\cup Г(x_2) \cup...\cup Г(x_q)$, тобто $Г(X_q)$ є множиною таких вершин $x_j\in X$, що для кожної з них існує дуга (x_i,x_j) в G, де $x_i\in X_q$. Для графа, наведеного на рис.4.1(а),

$Г(\{x_2,x_5\})=\{x_1,x_3,x_4\}$ і $Г(\{x_1,x_3\})=\{x_2,x_5,x_1\}$.

Відображення $Г(Г(x_i))$ записується як $Г^2(x_i)$. Аналогічно "потрійне" відображення $Г(Г(Г(x_i)))$ записується як $Г^3(x_i)$ і т. д. Для графа, показаного на рис.4.1(а), маємо:

$Г^2(x_1)=Г(Г(x_1))=Г(\{x_2,x_5\})=\{x_1,x_3,x_4\}$;

$Г^3(x_1)=Г(Г^2(x_1))=Г(\{x_1,x_3,x_4\})=\{x_2,x_5,x_1\}$ і т. д

Аналогічно розуміються позначення $Г^{-2}(x_i)$, $Г^{-3}(x_i)$ і т. д.

Дуги $a=(x_i,x_j)$, $x_i\leftrightarrow x_j$,що мають загальні кінцеві вершини, називаються суміжними. Дві вершини x_i і x_j називаються суміжними, якщо яка-небудь з двох дуг (x_i,x_j) і (x_j,x_i) або обидві одночасно присутні в графі. Так, наприклад, на

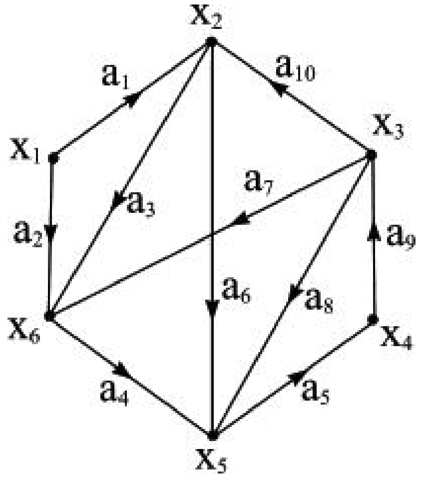


Рис.4.2.

рис.4.2 дуги a_1, a_{10}, a_3 і a_6 як і вершини x_5 і x_3, є суміжними, у той час як дуги a_1 і a_5 або вершини x_1 і x_4 не є суміжними.

Число дуг, які мають вершину x_i своєю початковою вершиною, називається **напівстепенем виходу** вершини x_i,, і, аналогічно, число дуг, які мають x_i своєю кінцевою вершиною, називається напівстепенем входу вершини x_i.

Таким чином, на рис.4.2 напівстепінь виходу вершини x_3, що позначається через $deg^+(x_3)$, дорівнює $|\Gamma(x_3)|=3$, і напівстепінь входу вершини x_3, що позначається через $deg^-(x_3)$, дорівнює $|\Gamma^{-1}(x_3)|=1$.

Очевидно, що сума напівстепенів входу всіх вершин графа, а також сума напівстепенів виходу всіх вершин дорівнюють загальному числу дуг графа G, тобто

$$\sum_{i=1}^{n} \deg^+(x_i) = \sum_{i=1}^{n} \deg^-(x_i) = m \qquad (4.1)$$

де n – число вершин і m – число дуг графа G. Для неорієнтованого графа $G=(X,\Gamma)$ степінь вершини x_i визначається аналогічно – за допомогою співвідношення $deg(x_i) =|\Gamma(x_i)|=|\Gamma^{-1}(x_i)|$.

Петлею називається дуга, початкова та кінцева вершини якої збігаються. На рис. 4.3, наприклад, дуги a_3 і a_{10} є петлями.

Одним з найбільш важливих понять теорії графів є дерево. *Неорієнтованим деревом* називається зв'язний граф, що не має циклів.

Суграфом графа G є підграф G_p, що містить всі вершини початкового графа.

Якщо $G=(X,A)$ – неорієнтований граф з n вершинами, то зв'язний суграф G_p, який не має циклів, називається *остовим деревом (остовом) графа* G.

Для остового дерева справедливе співвідношення:

$$G_p=(X_p,A_p) \subseteq G, \quad \text{где } X_p = X,\ A_p \subseteq A \qquad (4.2)$$

Легко довести, що остове дерево має наступні властивості:

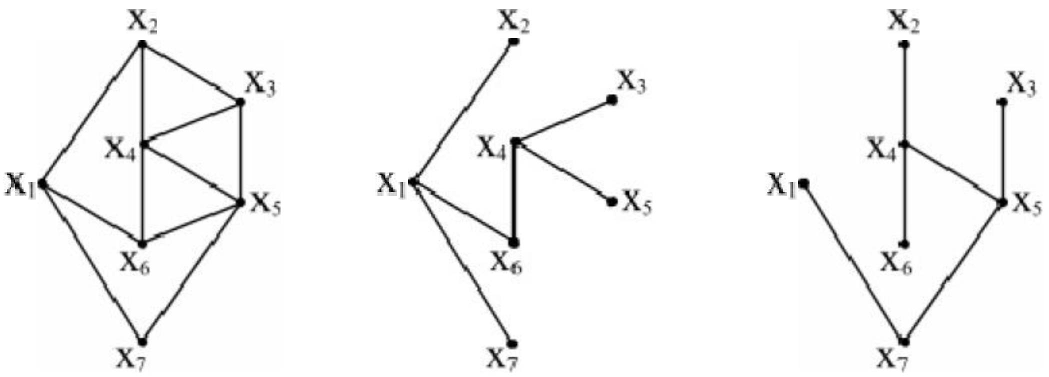
1) Остове дерево графа з n вершинами має n-1 ребро ($|X_p|=|A_p|-1$);

2) Існує єдиний шлях, що з'єднує будь-які дві вершини остова графа:

$\forall x_i,x_j \in X_p\ (i\neq j) \rightarrow \exists!$ «(x_i,x_j).

Наприклад, якщо G – граф, показаний на рис.4.3(а), то графи на рис.4.3(б, в) є остовами графа G. Зі сформульованих вище визначень випливає, що остов графа G можна розглядати як мінімальний пов'язаний остовий підграф графа G.

Поняття дерева як математичного об'єкта було вперше запропоновано Кірхгофом у зв'язку з визначенням фундаментальних циклів, застосовуваних при аналізі електричних ланцюгів. Приблизно десятьма роками пізніше Келі знову (незалежно від Кірхгофа) ввів поняття дерева і отримав більшу частину перших результатів у галузі дослідження властивостей дерев. Велику популярність здобула його знаменита теорема:



(а) (б) (в)

Рис. 4.3. Представлення графа у вигляді остових дерев

Теорема Келі. На графі з n вершинами можна побудувати n^{n-2} остови дерев.

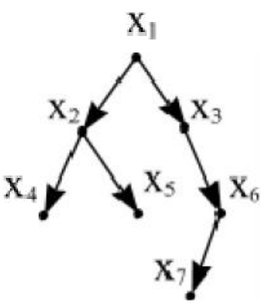


Рис. 4.4.

Орієнтоване дерево являє собою орієнтований граф без циклів, в якому напівстепінь входу кожної вершини, за винятком однієї (вершини r), дорівнює одиниці, а напівстепень входу вершини r (названої коренем цього дерева) дорівнює нулю.

На рис.4.4 показаний граф, який є орієнтованим деревом з коренем у вершині x_1. З наведеного визначення випливає, що орієнтоване дерево з n вершинами має $n-1$ дуг і є зв’язним.

Неорієнтоване дерево можна перетворити в орієнтоване: треба взяти його довільну вершину в якості кореня і ребрам приписати таку орієнтацію, щоб кожна вершина з’єднувалася з коренем (тільки однієї) простим ланцюгом.

"Генеалогічне дерево", в якому вершини відповідають особам чоловічої статі, а дуги орієнтовані від батьків до дітей, є добре відомии прикладом орієнтованого дерева. Корінь в цьому дереві відповідає "засновнику" роду (особі, народженій раніше за інших).

Шляхом (або *орієнтованим маршрутом*) орієнтованого графа називається послідовність дуг, в якій кінцева вершина будь-якої дуги, відмінної від останньої, є початковою вершиною наступної. Так, на рис. 4.5 послідовності дуг $\mu_1=\{a_6, a_5, a_9, a_8, a_4\}$, $\mu_2=\{a_1, a_6, a_5, a_9\}$, $\mu_3=\{a_1, a_6, a_5, a_9, a_{10}, a_6, a_4\}$ є шляхами.

Орієнтованим ланцюгом називається такий шлях, в якому кожна дуга використовується не більше одного разу. Так, наприклад, наведені вище шляхи μ_1 і μ_2 є орієнтованими ланцюгами, а шлях μ_3 не є таким, оскільки дуга a_6 в ньому використовується двічі.

Маршрут є неорієнтованим "двійником" шляху, і це поняття розглядається в тих випадках, коли можна знехтувати спрямованістю дуг у графі.

Таким чином, маршрут є послідовністю ребер a_1, a_2,...,a_q, в якій кожне ребро a_i, за винятком, можливо, першого і останнього ребра, пов’язане з ребрами a_{i-1} і a_{i+1} своїми двома кінцевими вершинами.

Послідовності дуг на рис. 4.6 $\mu_4=\{a_2, a_4, a_8, a_{10}\}$, $\mu_5=\{a_2, a_7, a_8, a_4, a_3\}$ і $\mu_6=\{a_{10}, a_7, a_4, a_8, a_7, a_2\}$ є маршрутами.

Контуром (*простим ланцюгом*) називається такий шлях (маршрут), в якому кожна вершина використовується не більше одного разу. Наприклад, шлях μ_2 є контуром, а шляхи μ_1 і μ_3 – ні. Очевидно, що контур є також ланцюгом, але зворотне твердження невірне. Наприклад, шлях μ_1 є ланцюгом, але не контуром, шлях μ_2 є ланцюгом і контуром, а шлях μ_3 не є ні ланцюгом, ні контуром.

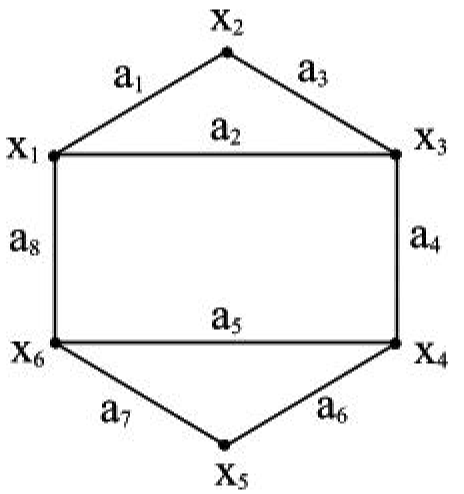


Рис. 4.5

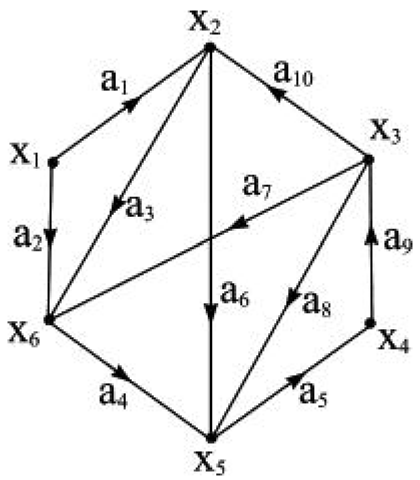


Рис. 4.6

Аналогічно визначається простий ланцюг у неорієнтованих графах. Так, наприклад, маршрут μ_4 є простим ланцюгом, маршрут μ_5 – ланцюг, а маршрут μ_6 не є ланцюгом.

Шлях або маршрут можна зображати також послідовністю вершин. Наприклад, шлях μ_1 можна представити також: $\mu_1=\{x_2, x_5, x_4, x_3, x_5, x_6\}$ і таке представлення часто виявляється більш корисним у тих випадках, коли здійснюється пошук контурів або простих ланцюгів.

Іноді дугам графа G співставляються (приписуються) числа – дузі (x_i,x_j) ставиться у відповідність деяке число c_{ij}, назване **вагою**, або довжиною, або **вартістю** (ціною) дуги. Тоді граф G називається зваженим. Іноді ваги (числа v_i) приписуються вершинам x_i графа.

При розгляді шляху μ, представленого послідовністю дуг $(a_1, a_2,...,a_q)$, за його вагу (або довжину, або вартість) приймається число $L(\mu)$, рівне сумі ваг всіх дуг, що входять до μ, тобто

$$L(«) = \sum_{(x_i,x_j)\in\mu} c_{ij} \qquad (4.3)$$

Таким чином, коли слова "довжина", "вартість", "ціна" і "вага" застосовуються до дуг, то вони еквівалентні за змістом, і в кожному конкретному випадку вибирається таке слово, яке краще відповідає змісту завдання. **Довжиною** (або потужністю) шляху називається кількість (число) дуг, що входять до нього.

4.2. МАТРИЧНІ ПРЕДСТАВЛЕННЯ

4.2.1. МАТРИЦЯ СУМІЖНОСТІ

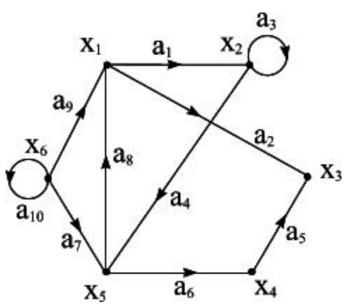


Рис. 4.7.

Нехай дано граф *G*, його матриця суміжності позначається через **A**=[a_{ij}] і визначається наступним чином:

a_{ij}=1, якщо в *G* існує дуга (x_i,x_j),

a_{ij}=0, якщо в *G* немає дуги (x_i,x_j).

Таким чином, матриця суміжності графа, зображеного на рис 4.7, має вигляд:

$$A=\begin{array}{c|cccccc|} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ \hline x_1 & 0 & 1 & 1 & 0 & 0 & 0 \\ x_2 & 0 & 1 & 0 & 0 & 1 & 0 \\ x_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_4 & 0 & 0 & 1 & 0 & 0 & 0 \\ x_5 & 1 & 0 & 0 & 1 & 0 & 0 \\ x_6 & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline \end{array}$$

Матриця суміжності повністю визначає структуру графа. Наприклад, сума всіх елементів рядка x_i матриці дає напівстепінь виходу вершини x_i, а сума елементів стовпця x_i - напівстепінь входу вершини x_i. Множина стовпців, які мають 1 у рядку x_i є множиною $\Gamma(x_i)$, а множина рядків, які мають 1 у стовпчику x_i , збігається з множиною $\Gamma^{-1}(x_i)$.

Петлі на графі являють собою елементи, що мають 1 на головній діагоналі матриці, наприклад a_{22}, a_{66} для графа, зображеного на рис.4.7.

У разі неорієнтованого графа матриця суміжності є симетричною відносно головної діагоналі (рис.4.8).

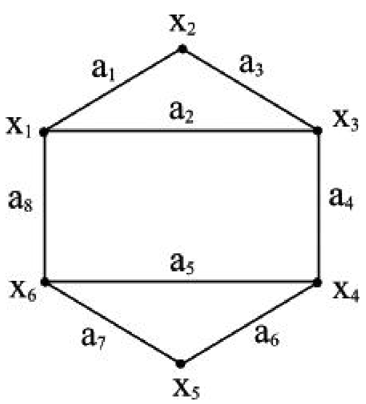


Рис. 4.8

$$A=\begin{array}{c|cccccc|} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ \hline x_1 & 0 & 1 & 1 & 0 & 0 & 1 \\ x_2 & 1 & 0 & 1 & 0 & 0 & 0 \\ x_3 & 1 & 1 & 0 & 1 & 0 & 0 \\ x_4 & 0 & 0 & 1 & 0 & 1 & 1 \\ x_5 & 0 & 0 & 0 & 1 & 0 & 1 \\ x_6 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array}$$

4.2.2. МАТРИЦЯ ІНЦИДЕНТНОСТІ

Нехай дано граф G з n вершинами і m дугами. *Матриця інцидентності* графа G позначається через $\mathbf{B}=[b_{ij}]$ і є матрицею розмірності n x m, яка визначається таким чином:

$b_{ij}=1$, якщо x_i є початковою вершиною дуги a_j;

$b_{ij}=-1$, якщо x_i є кінцевою вершиною дуги a_j;

$b_{ij}=0$, якщо x_i не є кінцевий вершиною дуги a_j.

Для графа, наведеного на рисунку 4.7, матриця інцидентності має вигляд:

B=

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
x_1	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**-1**	**-1**	**0**
x_2	**-1**	**0**	**ї1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
x_3	**0**	**-1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
x_4	**0**	**0**	**0**	**0**	**-1**	**-1**	**0**	**0**	**0**	**0**
x_5	**0**	**0**	**0**	**-1**	**1**	**1**	**-1**	**1**	**0**	**0**
x_6	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**ї1**

Оскільки кожна дуга інцидентна двом різним вершинам (за винятком випадку, коли дуга утворює петлю), то кожен стовпець містить один елемент, рівний 1, і один – рівний -1. Петля в матриці інцидентності не має адекватного математичного подання (в програмній реалізації допустимо завдання одного елемента $b_{ij}=1$).

Якщо G є неорієнтованим графом (рис. 4.8), то його матриця інцидентності визначається наступним чином:

$b_{ij}=1$, якщо x_i є кінцевий вершиною дуги a_j;

$b_{ij}=0$, якщо x_i не є кінцевий вершиною дуги a_j.

B=

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
x_1	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
x_2	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
x_3	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
x_4	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
x_5	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
x_6	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**

Матриця інцидентності як спосіб задавання графів, успішно застосовується при описі мультиграфів (графів, в яких суміжні вершини можуть з’єднуватися кількома паралельними дугами).

4.3. НАЙКОРОТШИЙ ОСТОВ ГРАФА

Розглянемо метод прямої побудови *найкоротших остових дерев* у зваженому графі (в якому ваги приписані дугам). Найкоротша остове дерево графа застосовується при прокладці доріг (газопроводів, ліній електропередач і т. д.), коли необхідно пов’язати n точок деякою мережею так, щоб загальна довжина "ліній зв’язку" була мінімальною. Якщо точки лежать на евклідовій площині, то їх можна вважати вершинами повного графа G з вагами дуг, рівними відповідним "прямолінійним" відстаням між кінцевими точками дуг. Якщо "розгалуження" доріг допускається тільки в заданих n точках, найкоротше остове дерево графа G буде якраз необхідною мережею доріг, яка має найменшу вагу.

Розглянемо зважений зв’язний неорієнтовані граф $G=(X,A)$; вагу ребра (x_i,x_j) позначимо c_{ij}.. З великого числа остовів графа потрібно знайти один, у якого сума ваг ребер найменша. Така задача виникає, наприклад, у тому випадку, коли вершини є клемами електричної мережі, які повинні бути з’єднані одна з одною за допомогою проводів найменшої загальної довжини (для зменшення рівня наведень). Інший приклад: вершини представляють міста, які потрібно пов’язати мережею трубопроводів; тоді найменша загальна довжина труб, яка повинна бути використана для будівництва (за умови, що поза межами міст "розгалуження" трубопроводів не допускаються), визначається найкоротшим остовом відповідного графа.

Завдання побудови найкоротшого остова графа є однією з небагатьох задач теорії графів, які можна вважати повністю вирішеними.

4.4. АЛГОРИТМ ПРИМА-КРАСКАЛА

Цей алгоритм породжує остове дерево за допомогою розростання тільки одного піддерева, наприклад X_p, що містить більше однієї вершини. Піддерево поступово розростається за рахунок приєднання ребер (x_i,x_j), де $x_i\in X_p$ і $x_j\notin X_p$; причому ребро, яке додається, повинно мати найменшу вагу c_{ij}. Процес продовжується доти, поки число ребер в A_p не стане рівним $n-1$. Тоді піддерево $G_p=(X_p,A_p)$ буде необхідним остовим деревом. Вперше така операція була запропонована Примом і Краскалом (з різницею – в способі побудови дерева), тому даний алгоритм отримав назву Прима-Краскала.

Алгоритм починає роботу з включення в піддерево початкової вершини. Оскільки остове дерево включає всі вершини графа G, то вибір початкової вершини не має принципового значення. Будемо кожній черговій вершині присвоювати позначку $(x_i)=1$, якщо вершина x_i належить піддереву X_p і $(x_j)=0$ – в іншому випадку.

Алгоритм має вигляд:

Крок 1. Нехай $X_p=\{x_1\}$, де x_1 - початкова вершина, і $A_p=\varnothing$ (A_p є множиною ребер, що входять в остове дерево). Вершині x_1 присвоїти позначку $(x_1)=1$. Для кожної вершини $x_i\notin X_p$ присвоїти $(x_i)=0$.

Крок 2. З усіх вершин $x_j\in\Gamma(X_p)$, для яких $(x_j)=0$, знайти вершину x_j^* таку, що

$$c(x_i,x_j^*)=\min_{x_j\in\Gamma(X_p)}\{c(x_i,x_j)\},\ \text{де } x_i\in X_p \text{ и } x_j\notin X_p. \qquad (5.2)$$

Крок 3. Оновити дані: $X_p= X_p \cup \{x_j^*\}$; $A_p= A_p \cup (x_i, x_j^*)$. Присвоїти $(x_j^*)=1$..

Крок 4. Якщо $|X_p|=n$, то зупинитися. Ребра в A_p утворюють найкоротший остов графа. Якщо $|X_p|<n$, то перейти до кроку 2.

4.5. КОНТРОЛЬНИЙ ПРИКЛАД

Для прикладу розглянемо граф, зображений на рис. 4.9 Знайдемо для нього найкоротше остове дерево, з цією метою використовуючи розглянутий вище алгоритм Прима-Краскала. Позначимо множину суміжних вершин, що не входять в породжене піддерево, як $\Gamma^*(X_p)$. Таким чином, для всіх вершин, що входять у цю множину, оцінка $(x_j)=0$, $\forall x_j \in \Gamma^*(X_p)$. Вектор B є множиною оцінок (x_i) для всіх вершин графа G: $\forall x_i \in X$.Довжина породженого піддерева позначається як L.

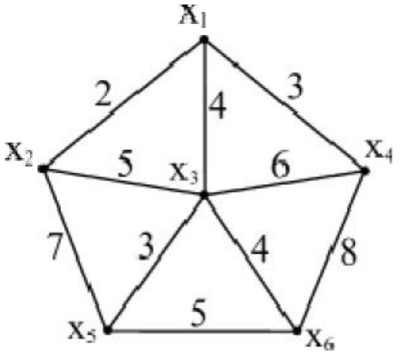


Рис. 4.9.

Ітерація 1: $X_p=\{x_1\}$; $A_p=\varnothing$; $\Gamma^*(X_p)=\{x_2, x_3, x_4\}$;

$c(x_1, x_2^*)=2$; $B=\{1,1,0,0,0,0\}$; $L=2$.

Ітерація 2: $X_p=\{x_1, x_2\}$; $A_p=\{(x_1, x_2)\}$;

$\Gamma^*(X_p)=\{x_3, x_4, x_5\}$; $c(x_1,x_4^*)=3$;
$B=\{1,1,0,10,0\}$; $L=2+3=5$.

Ітерація 3: $X_p=\{x_1, x_2, x_4\}$; $A_p=\{(x_1, x_2); (x_1, x_4)\}$;

$\Gamma^*(X_p)=\{x_3, x_5, x_6\}$; $c(x_1,x_3^*)=4$;
$B=\{1,1,1,1,0,0\}$; $L=5+4$.

Ітерація 4: $X_p=\{x_1, x_2, x_3, x_4\}$; $A_p=\{(x_1, x_2); (x_1, x_4); (x_1,x_3)\}$;
$\Gamma^*(X_p)=\{x_5, x_6\}$; $c(x_3, x_5^*)=2$; $B=\{1,1,1,1,1,0\}$; $L=9+3=12$.

Итерация 5: $X_p=\{x_1, x_2, x_3, x_4,x_5\}$; $A_p=\{(x_1, x_2); (x_1, x_4); (x_1,x_3); (x_3,x_5)\}$;
$\Gamma^*(X_p)=\{x_6\}$; $c(x_3,x_6^*)=4$; $B=\{1,1,1,1,1,1\}$; $L=12+4=16$.

Задача розв'язана. Отримані множини вершин X_p і ребер A_p складають найкоротше остове дерево:

$X_p=\{x_1, x_2, x_3, x_4,x_5,x_6\}$; $A_p =\{(x_1, x_2); (x_1, x_4); (x_1,x_3); (x_3,x_5); (x_3,x_6)\}$;

Сумарна довжина найкоротшого остового дерева $L=16$

Результат розв'язання задачі представлений на рис. 4.10.

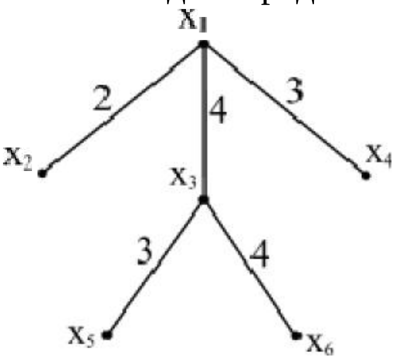


Рис. 4.10.

4.6. ЗАДАЧА ПРО НАЙКОРОТШИЙ ШЛЯХ

Нехай дано граф $G=(X,\Gamma)$, дугам якого приписані ваги (вартості), що задаються матрицею $\mathbf{C}=[c_{ij}]$. Задача про найкоротший шлях полягає в знаходженні найкоротшого шляху від заданої початкової вершини (витоку) s до заданої кінцевої вершини (стоку) t, за умови, що такий шлях існує:

Знайти $\mu(s,t)$ при $L(\mu)\rightarrow min$, $s,t\in X$, $t\in R(s)$, де $R(s)$ – множина, досяжна з вершини s.

У загальному випадку елементи c_{ij} матриці ваг $\mathbf{C}$ можуть бути додатними, від'ємними або нулями. Єдине обмеження полягає в тому, щоб в G не було циклів з сумарною від'ємною вагою. Звідси випливає, що дуги (ребра) графа G не повинні мати від'ємні ваги.

Майже всі методи, що дозволяють вирішити задачу про найкоротший (s-t)-шлях, дають також (в процесі вирішення) і всі найкоротші шляхи від s до $x_i(\forall x_i\in X)$. Таким чином, вони дозволяють вирішити задачу з невеликими додатковими обчислювальними витратами.

Допускається, що матриця ваг $\mathbf{C}$ не задовольняє умову трикутника, тобто не обов'язково $c_{ij}\leq c_{ik}+c_{kj}$ для всіх i, j і k.

Якщо в графі G дуга (x_i, x_j) відсутня, то її вага дорівнює ∞.

Ряд задач, наприклад, задача знаходження в графах шляхів з максимальною надійністю і з максимальною пропускною здатністю, пов'язані із задачею про найкоротший шлях, хоча в них характеристика шляху (скажімо, вага) є не сумою, а деякою іншою функцією характеристик (ваг) дуг, які утворюють шлях. Такі задачі можна переформулювати як задачі про найкоротший шлях і вирішувати їх відповідним чином.

Існує безліч методів вирішення даної задачі, що відрізняються областю застосування і трудомісткістю (Дейкстри, Флойда, динамічного програмування). Серед них велике поширення одержали спеціальні алгоритми, що застосовуються при вирішенні окремих задач, і мають меншу трудомісткість. Ці окремі випадки зустрічаються на практиці досить часто (наприклад, коли c_{ij} є відстанями), так що розгляд цих спеціальних алгоритмів виправданий.

4.6.1. АЛГОРИТМ ДЕЙКСТРИ

Алгоритм Дейкстри знаходить найкоротший шлях **для випадку** $c_{ij}\geq 0$.

1. Вершинам приписують **тимчасові позначки**. Нехай $l(v_i)$ – позначка вершини v_i.
2. Кожна позначка вершини **дає верхню границю довжини шляху** від s до цієї вершини.
3. Величини цих **позначок зменшуються ітераційно**.
4. На кожному кроці ітерації одна з тимчасових позначок **стає постійною**.
5. Величина цієї позначки є точною довжиною найкоротшого шляху від s до розглянутої вершини.

Теоретичний опис алгоритму Дейкстри

Присвоєння початкових значень

Крок 1. Встановити $l(s) := 0$ і вважати цю позначку постійною. Встановити $l(v_i) := \infty$ для всіх $v_i \neq s$ і вважати ці позначки тимчасовими. Встановити $p = s$.

Відновлення позначок

Крок 2. Для всіх $v_i \in \Gamma(p)$, позначки яких тимчасові, змінити позначки у відповідності з наступному виразом:

$$l(v_i) \leftarrow \min\left[l(v_i), l(p) + c(p, v_i)\right]$$

Перетворення позначки в постійну

Крок 3. Серед вершин з тимчасовими позначками знайти таку, для якої

$$l\left(v_i^*\right) = \min l(v_i),\ v_i \in \Gamma(p)$$

Крок 4. Вважати позначку $l\left(v_i^*\right)$ постійною і встановити $p = v_i^*$.

Крок 5. Коли треба знайти шлях від s до t Якщо $p = t$, то $l(p)$ є довжиною найкоротшого шляху від вершини s до вершини t. Останов.

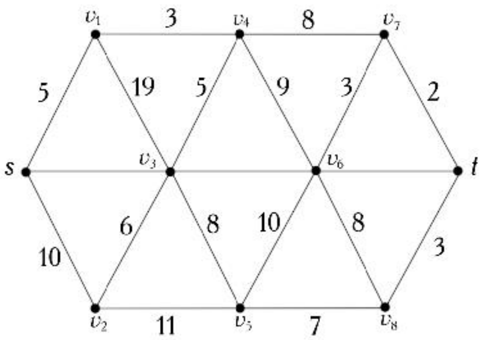
Крок 6. Якщо $p \neq t$, перейти до кроку 2.

Крок 7. **Якщо потрібно знайти шляхи від s до всіх інших вершин.** Якщо всі вершини відзначені як постійні, то ці позначки дають довжини найкоротших шляхів. Останов.

Крок 8. Якщо деякі позначки є змінними, перейти до кроку 2.

Приклад ручного розв'язку завдання

Розглянемо граф, зображений на рисунку, де кожне неорієнтоване ребро розглядається як пара протилежно орієнтованих дуг рівної ваги. Ця вага виписана на малюнку, що зображує граф, поруч із ребром.



Потрібно знайти найкоротші шляхи від вершини s до всіх інших вершин. Для цього використовуємо алгоритм Дейкстри. Постійні позначки відзначаються знаком +, інші позначки є тимчасовими.

Схема застосування алгоритму.

Крок 1. $l(s) = 0^+, l(v_i) = \infty,\ i = 1,\ldots,8,\ p = s.$

Перша ітерація.

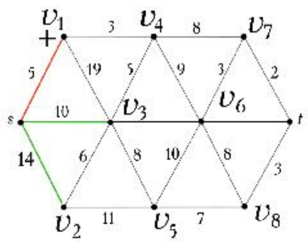
Крок 2. $\Gamma(s)=\{v_1,v_2,v_3\}-$ усі позначки тимчасові.

$l(v_1)=\min\left[\infty,0^+ +5\right]=5,$

$l(v_2)=\min\left[\infty,0^+ +14\right]=14,$

$l(v_3)=\min\left[\infty,0^+ +10\right]=10.$

Крок 3. $l(v_1)=\min\limits_{i=1,2,3} l(v_i)=5.$



Крок 4. $l(v_1)=5^+ - v_1$ одержує постійну позначку; $p=v_1$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

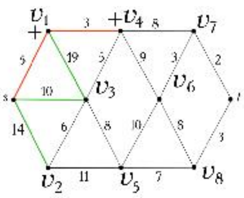
Друга ітерація.

Крок 2. $\Gamma(p)=\Gamma(v_1)=\{s,v_3,v_4\}$. Вершина s має постійну позначку;

$l(v_3)=\min\left[10,5^+ +19\right]=10,$

$l(v_4)=\min\left[\infty,5^+ +3\right]=8.$

Крок 3. $l(v_4)=\min\limits_{i=3,4} l(v_i)$.



Крок 4. Вершина v_4 одержує постійну мітку: $l(v_4)=8^+$; $p=v_4$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Третя ітерація.

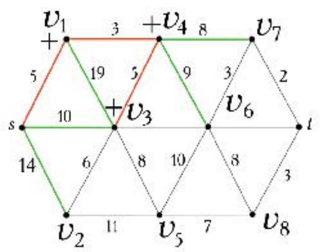
Крок 2. $\Gamma(p)=\Gamma(v_4)=\{v_1,v_3,v_6,v_7\}$. Вершина v_1 має постійну позначку;

$l(v_3)=\min\left[10,8^+ +5\right]=10,$

$l(v_7)=\min\left[\infty,8^+ +8\right]=16,$

$l(v_6)=\min\left[\infty,8^+ +9\right]=17.$

Крок 3. $l(v_3)=\min\limits_{i=3,6,7} l(v_i)=10$.



Крок 4. Вершина v_3 одержує постійну мітку: $l(v_3)=10^+$; $p=v_3$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

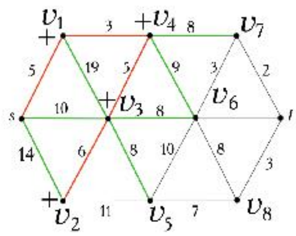
Четверта ітерація.

Крок 2. $\Gamma(p)=\Gamma(v_3)=\{s,v_1,v_2,v_4,v_5,v_6\}$. Вершини s,v_1,v_4 мають постійні позначки;

$l(v_2)=\min\left[14,10^{+}+6\right]=14,$

$l(v_5)=\min\left[\infty,10^{+}+8\right]=18,$

$l(v_6)=\min\left[17,10^{+}+8\right]=17.$



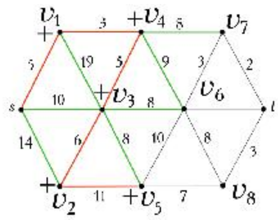
Крок 3. $l(v_2)=\min\limits_{i=2,5,6} l(v_i)=14$.

Крок 4. Вершина v_2 одержує постійну мітку: $l(v_2)=14^{+}$; $p=v_2$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

П'ята ітерація.

Крок 2. $\Gamma(p)=\Gamma(v_2)=\{s,v_3,v_5\}$. Вершини s,v_3 має постійні позначки;



$l(v_5)=\min\left[18,14^{+}+11\right]=18.$

Крок 3. $l(v_5)=\min\limits_{i=5} l(v_i)=18$.

Крок 4. Вершина v_5 одержує постійну мітку: $l(v_5)=18^{+}$; $p=v_5$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

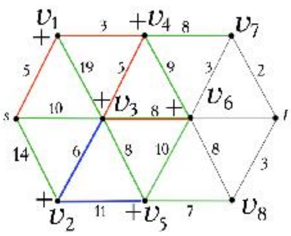
***Шоста ітерація*.**

Крок *2.* $\Gamma(p)=\Gamma(v_5)=\{v_2,v_3,v_6,v_8\}$. Вершини v_2,v_3 має постійні позначки;

$l(v_6)=\min\left[17,18^{+}+10\right]=17,$

$l(v_8)=\min\left[\infty,18^{+}+7\right]=25.$

Крок 3. $l(v_6)=\min\limits_{i=6,8} l(v_i)=17$.



Крок 4. Вершина v_6 одержує постійну мітку: $l(v_6)=17^{+}$; $p=v_6$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Сьома ітерація.

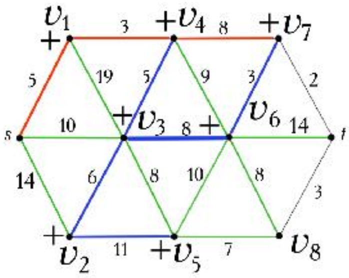
Крок 2. $\Gamma(p)=\Gamma(v_6)=\{v_3,v_4,v_4,v_7,v_8,t\}$. Вершини v_3,v_4,v_5 мають постійні позначки;

$l(v_7) = \min\left[16{,}17^+ + 3\right] = 16,$

$l(v_6) = \min\left[25{,}17^+ + 8\right] = 25,$

$l(t) = \min\left[\infty{,}17^+ + 14\right] = 31.$

Крок 3. $l(v_7) = \min\limits_{i=7,8,t} l(v_i) = 16$.



Крок 4. Вершина v_7 одержує постійну мітку: $l(v_7) = 16^+$; $p = v_7$.

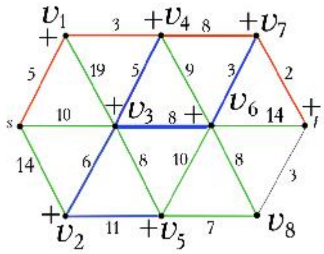
Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Восьма ітерація.

Крок *2.* $\Gamma(p) = \Gamma(v_7) = \{v_4, v_6, t\}$. Вершини v_4, v_6 мають постійні позначки;

$l(t) = \min\left[31{,}16^+ + 2\right] = 18.$

Крок 3. $l(v_t) = \min\limits_{i=t} l(v_i) = 18$.



Крок 4. Вершина t одержує постійну мітку: $l(t) = 18^+$; $p = t$.

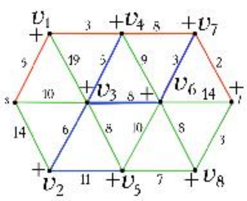
Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Дев'ята ітерація.

Крок 2. $\Gamma(p) = \Gamma(t) = \{v_6, v_7, v_8\}$. Вершини v_6, v_7 мають постійні позначки;

$l(v_8) = \min\left[25{,}18^+ + 3\right] = 21.$

Крок 3. $l(v_8) = \min\limits_{i=8} l(v_i) = 21$.



Крок 4. Вершина v_8 одержує постійну мітку: $l(v_8) = 21^+$; $p = v_8$.

Крок 5. Позначки всіх вершин постійні. Останов.

Зауваження.

Як тільки всі позначки вершин графа стають постійними, тобто знайдені довжини найкоротших шляхів від вершини s до будь-якої іншої вершини, самі шляхи можна одержати, використовуючи рівність

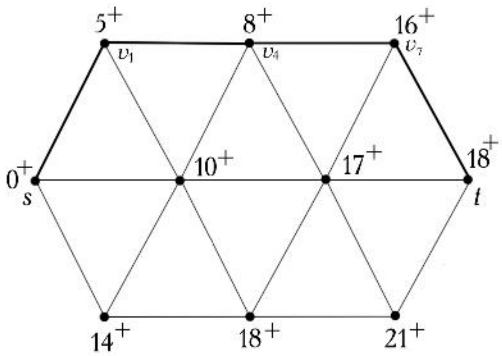
$$l\left(v_i'\right)+c\left(v_i',v_i\right)=l\left(v_i\right).$$

Тут $c\left(v_i',v_i\right)$ – вага дуги, що з'єднує вершину v_i' з вершиною v_i. Ця рівність дає можливість для кожної вершини v_i знайти попередню їй вершину v_i' й відновити весь шлях від s до v_i .

У розглянутому прикладі найкоротший шлях від вершини s до вершини t можна відновити з рівностей:

$$l\left(t\right)=l\left(v_6\right)+c\left(v_6,t\right),$$
$$l\left(v_6\right)=l\left(v_4\right)+c\left(v_4,v_7\right),$$
$$l\left(v_4\right)=l\left(v_1\right)+c\left(v_1,v_4\right),$$
$$l\left(v_1\right)=l\left(s\right)+c\left(s,v_1\right),$$

отриманих в 7-й, 2-й і 1-й ітераціях.
Йому відповідає послідовність вершин s,v_1,v_4,v_7,t .



Опис машинного алгоритму Дейкстри

Алгоритм використовує три масиви з n чисел кожний. Перший масив Visited містить мітки із двома значеннями: False (вершина ще не розглянута) і True (вершина вже розглянута); другий масив Len містить поточні найкоротші відстані від початкової до відповідної вершини; третій масив *C* містить номери вершин $-k$-й елемент С є номер передостанньої вершини на поточному найкоротшому шляху з початкової вершини в k-у. Використовується також Matrix – матриця відстаней.

Опишемо алгоритм Дейкстры:

1 (ініціалізація). У циклі від 1 до n заповнити значенням False масив Visited; заповнити числом *i* масив C (*i* – номер стартової вершини); перенести *i*-й рядок матриці Matrix у масив Len;

Visited[*i*]:=True; C[*i*]:=0;

2 (загальний крок). Знайти мінімум серед невідмічених (тобто тих *k* , для яких Visitid[*k*]=False); нехай мінімум досягається на індексі *j*, тобто Len[*j*] ≤ Len[*k*];

Потім виконувати наступні операції:
Visited[*i*]:=True;
якщо Len[*k*]>Len[*j*]+Matrix[*j*, *k*], то (Len[*k*]:=Len[*j*]+Matrix[*j*, *k*]; C[*k*]:=*j*)
{Якщо всі Visited[*k*] відзначені, то довжина шляху від v_i до v_k дорівнює C[*k*]. Тепер треба перелічити вершини, що входять у найкоротший шлях}.

3 (видача відповіді). {Шлях від v_i до v_k видається у зворотному порядку наступною процедурою:}
3.1 z:=C[*k*];
3.2 Видати z
3.3 z:=C[z]. Якщо z =0, то кінець,
інакше перейти до 3.2.

```
Program Deikstra;
Uses Crt;
Const Maxsize=10;
      Infinity=1000;
Var   Mattr: array [1..Maxsize, 1..Maxsize] of integer;
      Visited: array [1..Maxsize] of boolean;
      Len,Path: array [1..Maxsize] of integer;
      n, Start, Finish, k, i: integer;

Procedure Init;
Var  f: text;
      i, j: integer;
begin
  Assign(f, 'INPUT.MTR');
  Reset(f);
  Readln(f, n);
  For i:=1 to n do
  begin
    For j:=1 to n do Read(f, mattr[i,j]);
    Readln(f)
  end;
```

```
  Write('Початкова вершина: '); Readln(Start);
  For i:=1 to n do
  begin
    Visited[i]:=False;
    Len[i]:=Mattr[Start, i];
    Path[i]:=Start;
  end;
  Path[Start]:=0;
  Visited[Start]:=True;
end;

Function Possible: Boolean;
Var i: integer;
begin
  Possible:=True;
  For i:=1 to n do If not Visited[i] then Exit;
  Possible:=False;
end;

Function Min: Integer;
Var i, minvalue, currentmin: integer;
begin
  Minvalue:=Infinity;
  For i:=1 to n do
  If not Visited[i] then
  If Len[i]<minvalue then
  begin
    currentmin:=i;
    minvalue:=Len[i]
  end;
  min:=currentmin;
end;



begin
  Clrscr;
  Init;
  While Possible do
  begin
    k:=min;
    Visited[k]:=True;
    For i:=1 to n do
    If Len[i]>Len[k]+Mattr[i, k] then
    begin
```

```
      Len[i]:=Len[k]+Mattr[i, k];
      Path[i]:=k;
    end;
  end;
  Write('Кінцева вершина: '); Readln(Finish);
  Write(Finish);
  Finish:=Path[Finish];
  While Finish<>0 do
  begin
    Write('<-', Finish);
    Finish:=Path[Finish];
  end;
  Readkey;
end.
```

4.6.2. АЛГОРИТМ ФОРДА-БЕЛЛМАНА ЗНАХОДЖЕННЯ МІНІМАЛЬНОГО ШЛЯХУ

Передбачається, що орієнтований граф не містить контурів негативної довжини.

Алгоритм 1. (*Алгоритм Форда – Беллмана*)
Основними, величинами цього алгоритму є величини $\lambda_i\left(k\right)$, де $i = 1,2,\ldots,n$ (n – число вершин графа); $k = 1,2,\ldots,n-1$.

Для фіксованих i і k величина $\lambda_i\left(k\right)$ дорівнює довжині мінімального шляху, що веде із заданої початкової вершини v_1 у вершину v_i і складається не більше, ніж з k дуг.

Крок 1. Установка початкових умов. Ввести число вершин графа n й матрицю ваг $C = \left|c_{ij}\right|$.

Крок 2. Встановити $k = 0$. Встановити $\lambda_i\left(0\right) = \infty$ для всіх вершин, крім v_1; встановити $\lambda_1\left(0\right) = 0$.

Крок 3. У циклі по k, $k = 1,2,\ldots,n-1$, кожній вершині v_i на k-му кроці приписати індекс $\lambda_i\left(k\right)$ за наступним правилом:

$$\lambda_i\left(k\right) = \min_{1\le j\le n}\left\{\lambda_j\left(k-1\right) + c_{ji}\right\} \qquad (1)$$

для всіх вершин, крім v_1, встановити $\lambda_1\left(k\right) = 0$.

У результаті роботи алгоритму формується таблиця індексів

$$\lambda_i\left(k\right),\ i = 1,2,\ldots,n;\ k = 0,1,2,\ldots,n-1.$$

При цьому $\lambda_i\left(k\right)$ визначає довжину мінімального шляху з першої вершини в i-у, що містить не більше, ніж k дуг.

Крок 5. Відновлення мінімального шляху. Для будь-якої вершини v_s попередня їй вершина v_r визначається зі співвідношення:

$$\lambda_r\left(n-2\right)+c_{rs}=\lambda_s\left(n-1\right),\ v_r\in G^{-1}\left(v_s\right), \tag{2}$$

де $G^{-1}\left(v_s\right)$ – прообраз вершини v_s.

Для знайденої вершини v_r попередня їй вершина v_q визначається зі співвідношення:

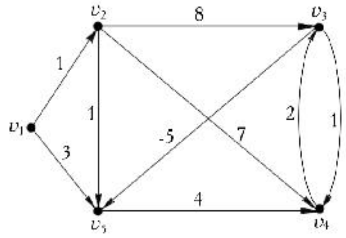
$$\lambda_q\left(n-3\right)+c_{qr}=\lambda_r\left(n-2\right),\ v_q\in G^{-1}\left(v_r\right),$$

де $G^{-1}\left(v_r\right)$ – прообраз вершини v_r, і т.д.

Послідовно застосовуючи це співвідношення, починаючи від останньої вершини v_i, знайдемо мінімальний шлях.

Приклад ручного розв'язку

За допомогою *алгоритму Форда-Беллмана* знайдемо мінімальний шлях з вершини v_1 у вершину v_3 на графі, зображеному на рисунку.



Значення індексів $\lambda_i\left(k\right)$ будемо заносити в таблицю індексів (табл. 1).

Крок 1. Введемо число вершин графа *n* = 5. Матриця ваг цього графа має вигляд:

$$C=\begin{Vmatrix}\infty & 1 & \infty & \infty & 3\\ \infty & \infty & 8 & 7 & 1\\ \infty & \infty & \infty & 1 & -5\\ \infty & \infty & 2 & \infty & \infty\\ \infty & \infty & \infty & 4 & \infty\end{Vmatrix}$$

Крок 2. Встановимо $k=0$,

$\lambda_1\left(0\right)=0,\ \lambda_2\left(0\right)=\lambda_3\left(0\right)=\lambda_4\left(0\right)=\lambda_5\left(0\right)=\infty$

Ці значення занесемо в перший стовпець табл. 1.

Крок 3. $k=1$. $\lambda_1\left(0\right)=0$.

λ1 (1) = 0.

Рівність (1) для *k* = 1 має вигляд:

$$\lambda_i\left(1\right)=\min_{1\le j\le 5}\left\{\lambda_j\left(0\right)+c_{ji}\right\}$$

$\lambda_2(1)=\min\{\lambda_1(0)+c_{12};\,\lambda_2(0)+c_{22};\,\lambda_3(0)+c_{32};\,\lambda_4(0)+c_{42};\,\lambda_5(0)+c_{52}\}=$
$=\min\{0+1;\infty+\infty;\infty+\infty;\infty+\infty;\infty+\infty\}=1.$
$\lambda_3(1)=\min\{\lambda_1(0)+c_{13};\lambda_2(0)+c_{23};\lambda_3(0)+c_{33};\lambda_4(0)+c_{43};\lambda_5(0)+c_{53};\}=$
$=\min\{0+\infty;\infty+8;\infty+\infty;\infty+2;\infty+\infty\}=\infty.$
$\lambda_4(1)=\min\{\lambda_1(0)+c_{14};\lambda_2(0)+c_{24};\lambda_3(0)+c_{34};\lambda_4(0)+c_{44};\lambda_5(0)+c_{54}\}=$
$=\min\{0+\infty;\infty+7;\infty+1;\infty+\infty;\infty+4\}=\infty.$
$\lambda_5(1)=\min\{\lambda_1(0)+c_{15};\lambda_2(0)+c_{25};\lambda_3(0)+c_{35};\lambda_4(0)+c_{45};\lambda_5(0)+c_{55}\}=$
$=\min\{0+3;\infty+1;\infty-5;\infty+\infty;\infty+\infty\}=3.$

Отримані значення $\lambda_i(1)$ занесемо в другий стовпець таблиці. Переконуємося, що другий стовпець, починаючи із другого елемента, збігається з першим рядком матриці ваг, що легко пояснюється змістом величин $\lambda_i(1)$, які дорівнюють довжині мінімального шляху з першої вершини в i-у, яка містить не більше однієї дуги.
$k=2.\ \lambda_1(2)=0.$
Рівність (1) для *k* = 2 має вигляд:

$$\lambda_i(2)=\min_{1\le j\le 5}\{\lambda_j(1)+c_{ji}\}$$

$\lambda_2(2)=\min\{0+1;1+\infty;\infty+\infty;\infty+\infty;3+\infty\}=1.$
$\lambda_3(2)=\min\{0+\infty;1+8;\infty+\infty;\infty+2;3+\infty\}=9.$
$\lambda_4(2)=\min\{0+\infty;1+7;\infty+1;\infty+\infty;3+4\}=7.$
$\lambda_5(2)=\min\{0+3;1+1;\infty-5;\infty+\infty;3+\infty\}=2.$
Отримані значення $\lambda_i(2)$ занесемо в третій стовпець таблиці. Величини $\lambda_i(2)$ дорівнюють довжині мінімального шляху з першої вершини в і-*ю*, що містить не більш двох дуг.
$k=3.\ \lambda_1(3)=0.$
Рівність (1) для $k=3$ має вигляд:

$$\lambda_i(3)=\min_{1\le j\le 5}\{\lambda_j(2)+c_{ji}\}$$

$\lambda_2(3)=\min\{0+1;1+\infty;9+\infty;7+\infty;2+\infty\}=1.$
$\lambda_3(3)=\min\{0+\infty;1+8;9+\infty;7+2;2+\infty\}=9.$
$\lambda_4(3)=\min\{0+\infty;1+7;9+1;7+\infty;2+4\}=6.$
$\lambda_5(3)=\min\{0+3;1+1;9-5;7+\infty;2+\infty\}=2.$

Отримані значення $\lambda_i(3)$ занесемо в четвертий стовпець таблиці. Величини λі (3) дорівнюють довжині мінімального шляху з першої вершини в і-у, що містить не більш трьох дуг.

$k = 4$. $\lambda_1(4) = 0$.

Рівність (1) для $k = 4$ має вигляд:

$$\lambda_i(4) = \min_{1 \le j \le 5} \{\lambda_j(3) + c_{ji}\}$$

$\lambda_2(4) = \min\{0+1;1+\infty;9+\infty;6+\infty;2+\infty\} = 1$.

$\lambda_3(4) = \min\{0+\infty;1+8;9+\infty;6+2;2+\infty\} = 8$.

$\lambda_4(4) = \min\{0+\infty;1+7;9+1;6+\infty;2+4\} = 6$.

$\lambda_5(4) = \min\{0+3;1+1;9-5;6+\infty;2+\infty\} = 2$

Отримані значення $\lambda_i(4)$ занесемо в п'ятий стовпець таблиці. Величини $\lambda_i(4)$ дорівнюють довжині мінімального шляху з першої вершини в і-у, що містить не більш чотирьох дуг.

i (номер вершини)	$\lambda_i(0)$	$\lambda_i(1)$	$\lambda_i(2)$	$\lambda_i(3)$	$\lambda_i(4)$
1	0	0	0	0	0
2	∞	1	1	1	1
3	∞	∞	9	9	8
4	∞	∞	7	6	6
5	∞	3	2	2	2

Крок 5. Відновлення мінімального шляху.

Для останньої вершини v_3 попередня їй вершина v_r визначається зі співвідношення (2), отриманого при $s = 3$:

$$\lambda_r(3) + c_{r3} = \lambda_3(4),\ v_r \in G^{-1}(v_3), \tag{3}$$

де $G^{-1}(v_3)$ – прообраз вершини v_3.

$$G^{-1}(v_3) = \{v_2, v_4\} \ .$$

Підставимо в (3) послідовно $r = 2$ та $r = 4$, щоб визначити, для якого r ця рівність виконується:

$\lambda_2(3) + c_{23} = 1 + 8 \ne \lambda_3(4) = 8$,

$\lambda_4(3) + c_{43} = 6 + 2 = \lambda_3(4) = 8$,

Таким чином, вершиною, що передує вершині v_3, є вершина v_4

Для вершини v_4 попередня їй вершина v_r визначається зі співвідношення (2) отриманого при $s = 4$:

$$\lambda_r(2) + c_{r4} = \lambda_4(3), v_r \in G^{-1}(v_4), \quad (4)$$

де $G^{-1}(v_4)$ – прообраз вершини v_4.
$G^{-1}(x4) = \{x2, x3, x5\}$.
Підставимо в (4) послідовно $r = 2$, $r = 3$ й $r = 5$, щоб визначити, для якого r ця рівність виконується:
$\lambda_2(2) + c_{24} = 1 + 7 \neq \lambda_4(3) = 6$,
$\lambda_3(2) + c_{34} = 1 + 1 \neq \lambda_4(3) = 6$,
$\lambda_5(2) + c_{54} = 2 + 4 = \lambda_4(3) = 6$
Таким чином, вершиною, що передує вершині v_4, є вершина v_5.
Для вершини v_5 попередня їй вершина v_r визначається зі співвідношення (2), отриманого при $s = 5$:

$$\lambda_r(1) + c_{r5} = \lambda_5(2), v_r \in G^{-1}(v_5), \quad (5)$$

де $G^{-1}(v_5)$ – прообраз вершини v_5.

$$G^{-1}(v_5) = \{v_1, v_2\}.$$

Підставимо в (5) послідовно $r = 1$ й $r = 2$, щоб визначити, для якого r ця рівність виконується:
$\lambda_1(1) + c_{15} = 0 + 3 \neq \lambda_5(2) = 2$,
$\lambda_2(1) + c_{25} = 1 + 1 = \lambda_5(2) = 2$.
Таким чином, вершиною, що передує вершині v_5, є вершина v_2.
Для вершини v_2 попередня їй вершина v_r визначається зі співвідношення (2), отриманого при $s = 2$.

$$\lambda_r(0) + c_{r2} = \lambda_2(1), v_r \in G^{-1}(v_2), \quad (6)$$

Де $G^{-1}(v_2)$ – прообраз вершини v_2.

$$G^{-1}(v_2) = \{v_1\}.$$

Підставимо в (6) $r = 1$, щоб визначити, чи виконується ця рівність:

$$\lambda_1(0) + c_{12} = 0 + 1 = \lambda_2(1) = 1$$

Таким чином, вершиною, що передує вершині v_2, є вершина v_1.
Отже, знайдений мінімальний шлях $- v_1, v_2, v_5, v_4, v_3$, його довжина рівна 8.

Програма по алгоритму Форда-Беллмана

(*Алгоритм Форда-Беллмана*)
Program Ford;
var a : array [1..20,1..20] of word;(*матриця суміжності*)

```
c, pred, fl, d : array [1..20] of word;
(*c – масив найкоротших відстаней
pred – масив попередніх вершин
fl – масив ознак
d – масив для запису шляху*)

i, j, k, n, first, last : byte;
f : text;(* змінна для відкриття файлу in.txt*)
(*процедура обходу графа вглиб – для пошуку всіх шляхів*)
Procedure Dfs(x : word);
var i : byte;  (*локальна змінна*)
begin
  if x=last then (*якщо кінцева вершина те виводимо шлях*)
  begin
    write(first,' ');
    for i:=1 to j do (*виводимо шлях*)
    write(d[i],' ');
    writeln;
    exit; (*виходимо із процедури*)
  end;
  fl[x]:=1; (*позначаємо, що були у вершині*)
  for i:=1 to n do
  if (fl[i]=0)and(a[x,i]<>32767) then
  begin
    inc(j);
    d[j]:=i; (*записуємо в шлях вершини*)
    dfs(i); (*викликаємося від і-й вершини*)
    dec(j);
  end;
  fl[x]:=0; (*позначаємо, що вершина вільна*)
end;
(*Основна програма*)
begin
  assign(f,'in.txt'); (*Відкриваємо файл для читання*)
  reset(f);
  readln(f, n); (*зчитуємо кількість вершин*)
  for i := 1 to n do
  for j := 1 to n do
  read(f, a[i,j]); (*зчитуємо матрицю суміжності*)
  writeln('Matrix:');
  for i:=1 to n do  (*виводимо матрицю на екран*)
  for j:=1 to n do
  if j=n then writeln(a[i,j]) else write(a[i,j],' ');
  for i:=1 to n do (*заміняємо нулі нескінченністю*)
  for j:=1 to n do
```

```
  if a[i,j]=0 then a[i,j]:=32767;
  writeln('Введіть вершину 1');
  readln(first);
  writeln('Введіть вершину 2');
  readln(last);
  close(f); (* закриваємо file in.txt*)
  for j := 1 to n do
  begin
    c[j] := a[first,j]; (* записуємо початкові значення*)
    if a[first,j] < 32767 then
    pred[j] := first;
  end;
  for i := 3 to n do
  for j := 1 to n do
  if j <> first then
  for k := 1 to n do  (*якщо не нескінченність і шлях більш вигідний*)
  if (c[k] < 32767) and (c[k] + a[k,j] < c[j]) then
  begin
    c[j] := c[k] + a[k,j];(*записывем нове значення*)
    pred[j] := k;{записуємо предвершины}
  end;
  if c[last] = 32767 then writeln('Немає шляхів') else
  begin
    writeln;
    writeln('Найкоротший шлях:');
    write(first,' ');
    i := last;
    k := 1;
    while i <> first do (*у зворотному порядку обходимо шлях*)
    begin
      d[k] := i;(*записуємо шлях у масив*)
       k := k + 1;
       i := pred[i];
    end;
    for i:= k-1 downto 1 do  (*виводимо найкоротший шлях*)
    write(d[i],' ');
    writeln;
    writeln('Усі шляхи:');
    j:=0;
    Dfs(first);(*викликаємо процедуру пошуку всіх шляхів*)
  end;
  readln; readln; (*чекаємо натискання клавіші*)
end.
```

4.6.3. АЛГОРИТМ ФЛОЙДА-УОРШЕЛЛА

Даний алгоритм іноді називають алгоритмом Флойда-Уоршелла. Алгоритм Флойда-Уоршелла є алгоритмом на графах, який розроблений в 1962 році Робертом Флойдом і Стивеном Уоршеллом. Він служить для знаходження найкоротших шляхів між усіма парами вершин графа.

Метод Флойда безпосередньо ґрунтується на тому факті, що в графі з позитивними вагами ребер усякий неелементарний (більше 1 ребра) найкоротший шлях складається з інших найкоротших шляхів.

Цей алгоритм більш загальний у порівнянні з алгоритмом Дейкстры, тому що він знаходить найкоротші шляхи між будь-якими двома вершинами графа.

В алгоритмі Флойда використовується матриця A розміру $n \times n$, у якій обчислюються довжини найкоротших шляхів. Елемент $A\left[i,j\right]$ дорівнює відстані від вершини i до вершини j, яка має кінцеве значення, якщо існує ребро $\left(i,j\right)$, і дорівнює нескінченності в протилежному випадку.

Алгоритм Флойда

Основна ідея алгоритму. Нехай є три вершини i, j, k і задані відстані між ними. Якщо виконується нерівність $A\left[i,k\right] + A\left[k,j\right] < A\left[i,j\right]$, то доцільно замінити шлях $i \to j$ шляхом $i \to k \to j$. Така заміна виконується систематично в процесі виконання даного алгоритму.

Крок 0. Визначаємо початкову матрицю відстані A_0 й матрицю послідовності вершин S_0. Кожний діагональний елемент обох матриць дорівнює 0, таким чином, показуючи, що ці елементи в обчисленнях не беруть участь. Встановимо $k = 1$.

Основний крок k. Задаємо рядок k і стовпець k як провідний рядок і провідний стовпець. Розглядаємо можливість застосування описаної вище заміни, до всіх елементів $A\left[i,j\right]$ матриці A_{k-1}. Якщо виконується нерівність

$A\left[i,k\right] + A\left[k,j\right] < A\left[i,j\right], \left(i \neq k, j \neq k, i \neq j\right)$, тоді виконуємо наступні дії:

1. створюємо матрицю A_k шляхом заміни в матриці A_{k-1} елемента $A\left[i,j\right]$ на суму $A\left[i,k\right] + A\left[k,j\right]$;

2. створюємо матрицю S_k шляхом заміни в матриці S_{k-1} елемента $S\left[j,j\right]$ на k. Встановимо $k = k + 1$ й повторюємо крок k.

Таким чином, алгоритм Флойда робить n ітерацій, після i-ї ітерації матриця А буде містити довжини найкоротших шляхів між будь-якими двома парами вершин за умови, що ці шляхи проходять через вершини від першої до i-ї. На кожній ітерації перебираються все пари вершин і шлях між ними скорочується за допомогою i-ї вершини.

```
Program Floid_Uorshel;
Uses Crt;
Const
PP=50;
Type
Graph = array[1..pp,1..pp] of integer;
Var
p:integer;
t,c,h:graph;
i,j: integer;

Procedure Floyd (var t:graph; c:graph; var h:graph);
var i,j,k:integer;
GM:real;
begin
  GM:=10000;
  for i:=1 to p do
  for j:=1 to p do t[i,j]:=c[i,j];
  if c[i,j]=GM then H[i,j]:=0  else
  begin
    H[i,j]:=j;
  end;
  for i:=1 to p do
  for j:=1 to p do
  for k:=1 to p do
  if (i<>j)and(T[j,i]<>GM)and(i<>k)and(T[i,k]<>GM)and(T[j,k]=GM) or
(T[j,k]>T[j,i]+T[i,k]) then
  begin
    H[j,k]:=H[j,i];
    T[j,k]:=T[j,i]+T[i,k]
  end;
end;

Procedure Readfilegraph (var T:graph);
var
 i,j:integer;
 f: text;
begin
  Writeln (’Reading from the text file’);
```

```
  Assign (f,'nell.txt');
  reset(f);
  Readln(f,P);
  for i:=1 to p do for j:=1 to p do
  read(f,t[i,j]); close(f);
end;

begin
  Clrscr;
  Readfilegraph(c);
  floyd(t,c,h);
  writeln('-----------------------------------------');
  for i:=1 to p do
  begin
    for j:=1 to p do write (t[i,j]:3);
    writeln
  end;
  writeln('-----------------------------------------');
  for i:=1 to p do
  begin
    for j:=1 to p do write (h[i,j]:3);
    writeln
  end;
  readln;
end.
```

4.6.4 МЕТОД ДИНАМІЧНОГО ПРОГРАМУВАННЯ

Пряма ітерація. Нехай вершини пронумеровані так, що дуга (x_i,x_j) завжди орієнтована від вершини x_i до вершини x_j, що має більший номер. Для ациклічного графа така нумерація завжди можлива і проводиться дуже легко. При цьому початкова вершина s отримує номер 1, а кінцева t – номер n.

Нехай $\lambda(x_i)$ – позначка вершини x_i, яка дорівнює довжині найкоротшого шляху від 1 до x_i,, s – початкова вершина (джерело), t – кінцева вершина (стік).

Крок 1. Нехай $\lambda(s)=0$, $\lambda(x_i) = \infty$ для всіх вершин $x_i \in X/s$; $i=1$.

Крок 2. $i=i+1$. Присвоїмо вершині x_j позначку $\lambda(x_j)$, яка дорівнює довжині найкоротшого шляху від 1 до x_j, використовуючи для цього співвідношення

$$\lambda(x_j) = \min_{x_i \in \Gamma^{-1}(x_j)} [\lambda(x_i) + c_{ij}] \qquad (6.2)$$

Крок 3. Повторювати п. 2., поки остання вершина n не отримає позначку $\lambda(t)$.

Необхідно відзначити, що якщо вершина x_j позначена, то помітки $\lambda(x_i)$ відомі для $x_i \in \Gamma^{-1}(x_j)$, тому що у відповідності зі способом нумерації це означає, що $x_i < x_j$ і, отже, вершини x_i вже позначені в процесі застосування алгоритму.

Позначка $\lambda(t)$ дорівнює довжині найкоротшого шляху від s до t. Дуги, що утворюють шлях, можуть бути знайдені способом послідовного повернення. А дуга (x_i,x_j), згідно (6.2), належить шляху тоді і тільки тоді, коли

$$\lambda(x_j)= \lambda(x_i)+c_{ij} \tag{6.3}$$

Зворотна ітерація: починаючи з вершини t, що має номер n, вважаємо на кожному кроці x_j рівною такій вершині (скажімо, x_j^*), для якої виконується співвідношення (6.3), і так продовжуємо доти, поки не буде досягнута початкова вершина (тобто поки не буде $x_j^*\equiv s$).

Цілком очевидно, що позначка $\lambda(x_j)$ вершини x_j дає довжину найкоротшого шляху μ від s до x_j.

4.6.5. АЛГОРИТМ ТОПОЛОГІЧНОГО СОРТУВАННЯ

У деяких випадках початковий граф є ациклічним, але має неправильну нумерацію – містить дуги (x_j,x_i), орієнтовані від вершини x_j до вершини x_i, яка має менший номер ($j>i$). Для успішного знаходження найкоротшого шляху за допомогою методу динамічного програмування до такого графу спочатку застосовується алгоритм топологічного сортування вершин.

Алгоритм топологічного сортування вершин дуже простий. Він дозволяє не тільки правильно перенумерувати вершини графа, але й визначити його ациклічність.

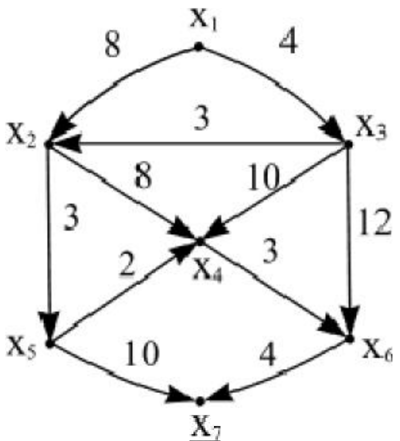
Крок 1. Нехай $i=n$, де n – число вершин графа G.

Крок 2. У графі визначається вершина x_k, для якої виконується умова $|\Gamma(x_k)|=\varnothing$ (тобто, вершина, з якої не виходить жодна дуга). Вершина x_k отримує порядковий номер i (перенумеровується) і виключається з подальшого розгляду разом з усіма вхідними в неї інцидентними дугами. $i=i–1$.

Крок 3. Повторювати п.2. доти, поки не буде виконано одну з умов: 1) $i=1$ – досягнута початкова вершина. Вершини графа отримали правильну нумерацію.

2) Неможливо визначити вершину, для якої виконувалася б умова $|\Gamma(x_k)|=\varnothing$.. У графі є цикл.

В останньому випадку алгоритм динамічного програмування непридатний. Для пошуку найкоротших шляхів на такому графі необхідно використовувати більш ефективні методи, наприклад, алгоритм Дейкстри.



4.6.6. КОНТРОЛЬНИЙ ПРИКЛАД

Для графа, зображеного на рис. 6.3, визначимо найкоротший шлях між вершинами x_1 і x_7, використовуючи метод динамічного програмування.

Оскільки граф містить дуги (x_3 ,x_2) і (x_5 ,x_4), що мають неправильну нумерацію (від більшого до меншого), необхідно перенумерувати вершини графа, застосовуючи алгоритм топологічного сортування вершин.

У нашому випадку з графа будуть послідовно виключатися вершини x_7, x_6 , x_4, x_5, x_2, x_3, x_1. Відповідно, вершини графа отримають нову нумерацію, і граф буде мати вигляд, представлений на рис. 6.4 (стара нумерація вершин збережена в дужках).

Рис. 6.3.

На першому кроці оцінка $\lambda(x_1)=0$. Переходимо до вершини x_2. Множина $\Gamma^{-1}(x_2)$ включає тільки одну вершину x_1. Отже, оцінка для вершини x_2, яка визначається за формулою (6.2), буде $\lambda(x_2)=min\{0+4\}=4$. Переходимо до вершини x_3. Для вершини x_3 множина $\Gamma^{-1}(x_3)=\{x_1, x_2\}$. У цьому випадку оцінка буде вибиратися як мінімальна з двох можливих: $\lambda(x_3)=min\{0+8, 4+3\}=7$. Для вершини x_4 оцінка визначається знову однозначно: $\lambda(x_4)=min\{7+3\}=10$. Однак при переході до вершини x_5 ми отримуємо відразу три вхідні дуги (x_2, x_5), (x_3, x_5), (x_4, x_5). Застосовуючи формулу (6.2), визначаємо оцінку для вершини x_5: $\lambda(x_5)=min\{4+10, 7+8, 10+2\}=12$.

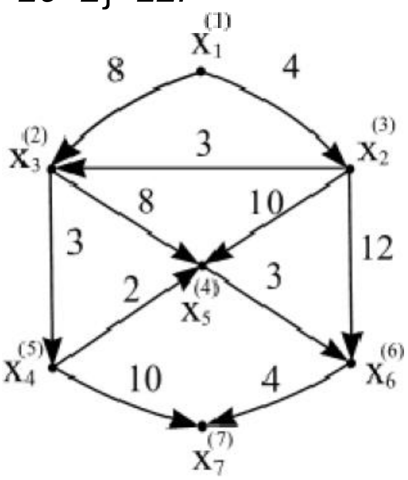


Рис. 6.4.

Далі, аналогічним чином вершина x_6 отримує оцінку $\lambda(x_6)=min\{4+12, 12+3\}=15$ і, нарешті, вершина x_7 отримує оцінку $\lambda(x_7)=min\{10+10, 15+4\}=19$.

Таким чином, кінцева вершина x_7 шляху $\mu(x_1,x_7)$ досягнута, довжина шляху дорівнює $L(\mu)=19$. Застосовуючи вираз (6.3), послідовно визначаємо вершини, які входять в найкоротший шлях. Переміщуючись від кінцевої вершини x_7, вибираємо послідовність вершин, для якої вираз (6.3) приймає значення «істинно»: x_6, x_5, x_4, x_3, x_2, x_1,, тобто найкоротший шлях проходить послідовно через усі вершини графа.

Дійсно, легко переконатися в істинності виразів

$\lambda(x_7)= \lambda(x_6)+ c_{67}$, $(19=15+4)$;

$\lambda(x_6)= \lambda(x_5)+ c_{56}$, $(15=12+3)$;

$\lambda(x_5)= \lambda(x_4)+ c_{45}$,$(12=10+2)$;

$\lambda(x_4)= \lambda(x_3)+ c_{34}$,$(10=7+3)$;

$\lambda(x_3)= \lambda(x_2)+ c_{23}$,$(7=4+3)$;

$\lambda(x_2)= \lambda(x_1)+ c_{12}$,$(4=0+4)$;

Зробивши перенумерацію вершин графа на початкову, остаточно визначаємо найкоротший шлях:

$\mu(x_1,x_7)=\{x_1, x_3, x_2, x_5, x_4, x_6, x_7\}$.

Задачу вирішено.

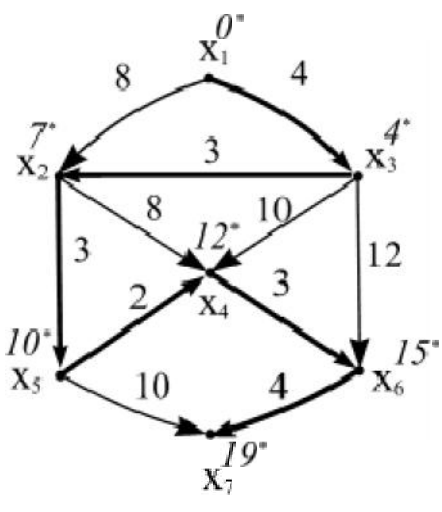


Рис. 6.5

Результат вирішення у вигляді виділеного шляху зображений на рис. 6.5. Курсивом «із зірочкою» виділені значення позначок вершин $\lambda(x_i)$.

4.7. ЗАВДАННЯ ЛАБОРАТОРНОЇ РОБОТИ

1. Вивчити способи представлення графів та написати програму для перетворення графу, задонного матрицею суміжності в граф, заданний матрицею інцидентності.
2. Вивчити способи представлення графів та написати програму для перетворення графу, задонного матрицею інцидентності в граф, заданний матрицею суміжності.
3. Вивчити основні означення та правила формування остовних дерев у графі. Написати програму побудови найкоротшого остовного дерева за допомогою алгоритму Прима-Краскала. Програма повинна вивести список ребер, що входять в найкоротше остовне дерево.
4. Розглянути основні принципи пошуку найкоротших шляхів у графі. Написати програму за алгоритмом Дейкстри для пошуку найкоротших шляхів у графі.
5. Розглянути принцип побудови алгоритму Форда-Беллмана та написати программу пошуку найкоротших шляхів у графі за цим алгоритмом.
6. Розглянути принцип побудови алгоритму Флойда-Уоршелла та написати программу пошуку найкоротших шляхів у графі за цим алгоритмом.
7. Вивчити принципи застосування динамічного програмування до побудови найкоротших шляхів у графі. Написати программу пошуку найкоротших шляхів у графі з використанням методу динамічного програмування.
8. Вивчити принципи застосування алгоритму топологічного сортування до пошуку найкоротших шляхів у графі. Написати програму пошуку найкоротших шляхів у графі з застосуванням алгоритму топологічного сортування.

Вимоги до програмного забезпечення:

1. Лабораторна робота виконується в середовищі візуального програмування Lazarus.
2. В якості початкового коду використати проект LAB4_Project, попередньо скачавши його з сайту викладача.
3. Початкові данні вводяться за допомогою програмних засобів уже реалізованих в згаданому проекті.
4. Необхідно в рамках форми OperForm реалізувати алгоритм відповідно до варіанту лабораторної роботи.
5. На формі OperForm відобразити як початкові дані, так і результати роботи алгоритму.
6. Остовне дерево або ребра, що входять до найкоротшого шляху повинні бути відображені розфарбованими у відповідності з результатами роботи вашого алгоритму.

Зміст звіту:

1. Титульний лист;
2. Тема завдання;
3. Завдання;
4. Блок-схема алгоритму;
5. Роздруківка тексту програми;
6. Контрольний приклад та результати машинного розрахунку.
7. Висновки по роботі.

Контрольні питання

1. Назвіть основні способи представлення графів.
2. Покажіть на прикладі пряму і зворотну відповідність для заданої вершини.
3. Чому дорівнює сума степенів усіх вершин неорієнтованого графа?
4. У чому відмінності матричного представлення орієнтованих і неорієнтованих графів?
5. У чому особливості представлення графа матрицею суміжності?
6. У чому особливості представлення графа матрицею інцидентності?
7. Дайте визначення дерева; орієнтованого дерева.
8. Яке дерево називається остовим?
9. Властивості остових дерев. Теорема Келі.
10. Що називається коренем дерева?
11. Як перетворити неорієнтоване дерево в орієнтоване?
12. Скільки ребер містить остове дерево графа?
13. Завдання знаходження найкоротшого остова графа.
14. Наведіть практичні приклади знаходження найкоротшого остова графа.
15. Реалізація алгоритму Прима-Краскала для знаходження найкоротшого остова графа.
16. Дайте визначення шляху, маршруту, ланцюга, контуру.
17. Який граф називається зваженим?
18. Як визначається довжина шляху графа?
19. Задача знаходження найкоротшого шляху на графі.
20. Реалізація методу динамічного програмування для знаходження найкоротшого шляху на графі.
21. Обмеження застосування методу динамічного програмування для знаходження найкоротшого шляху на графі.
22. Що називається правильною нумерацією вершин графа?
23. Застосування алгоритму топологічного сортування для перенумерації вершин графа.
24. Застосування алгоритму Дейкстри для пошуку найкоротших шляхів у графі.

25. Основні принципи роботи алгориту Форда-Белламана для пошуку найкоротших шляхів у графі.
26. Алгоритм Флойда-Уоршелла для пошуку найкоротших шляхів у графі.

Варіанти для виконання лабораторної роботи

Варіанти для виконання лабораторної роботи

Номер варіанту I визначається як результат операції I = NZK mod 8+1, де NZK – номер залікової книжки. Номер варіанту відповідає номеру пункту завдання до лабораторної роботи.

№	Опис варіанта
1	А) Виконати завдання 1 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю суміжності. В) Представити початковий граф, заданиий матрицею суміжності, та кінцевий граф, заданий матрицею інцидентності у графічній формі.
2	А) Виконати завдання 2 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю інцидентності. В) Представити початковий граф, заданиий матрицею інцидентності, та кінцевий граф, заданий матрицею суміжності, у графічній формі.
3	А) Виконати завдання 3 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю суміжності. В) Представити граф, заданий матрицею суміжності, у графічній формі та виділити найкоротше остовне дерево, сформоване за допомогою алгоритму Прима-Краскала
4	А) Виконати завдання 4 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю ваг $C=\left[c_{i,j}\right]$ графа. В) Представити граф, заданий матрицею ваг C, у графічній формі та виділити найкоротший шлях між заданими викладачем вершинами, сформований за допомогою алгоритму Дейкстри
5	А) Виконати завдання 5 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю ваг $C=\left[c_{i,j}\right]$ графа. В) Представити граф, заданий матрицею ваг C, у графічній формі та виділити найкоротший шлях між заданими викладачем вершинами, сформований за допомогою алгоритму Форда-Беллмана
6	А) Виконати завдання 6 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю ваг $C=\left[c_{i,j}\right]$ графа. В) Представити граф, заданий матрицею ваг C, у графічній формі та

	вивести матрицю найкоротших шляхів між вуршинами, сформовану за алгоритмом Флойда-Уоршелла
7	А) Виконати завдання 7 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю ваг $C=\left[c_{i,j}\right]$ графа. В) Представити граф, заданий матрицею ваг C, у графічній формі та виділити найкоротший шлях між заданими викладачем вершинами, сформований за допомогою алгоритму динамічного програмування.
8	А) Виконати завдання 8 до лабораторної роботи. Б) За правилом, наданим викладачем, сформувати матрицю ваг $C=\left[c_{i,j}\right]$ графа. В) Представити граф, заданий матрицею ваг C, у графічній формі та виділити найкоротший шлях між заданими викладачем вершинами сформований за допомогою алгоритму топологічного сортування

<ins>**Лабораторна робота №5**</ins>

<ins>**Тема:**</ins> «Розфарбовування графа, алгоритми розфарбування»
<ins>**Мета роботи**</ins>**:** вивчення способів правильного розфарбовування графа.
<ins>**Завдання**</ins>**:** створити програму для правильного розфарбовування графа на основі одного з алгоритмів розфарбування.

<ins>**Теоретичні основи**</ins>

5.1. ОСНОВНІ ОЗНАЧЕННЯ

Різноманітні завдання, що виникають при плануванні виробництва, складанні графіків огляду, зберіганні та транспортуванні товарів та ін., часто можуть бути представлені як задачі теорії графів, тісно пов’язані з так званим «завданням розфарбовування». Графи, що розглядаються в даній лабораторній роботі, є неорієнтованими і такими, що не мають петель.

Граф G називають r*-хроматичним*, якщо його вершини *можуть бути розфарбовані* з використанням r кольорів (фарб) так, що не знайдеться двох суміжних вершин одного кольору. Найменше число r, таке, що граф G є $r-$*хроматичним*, називається *хроматичним числом* графа G і позначається ¥$\left(G\right)$. Завдання знаходження хроматичного числа графа називається *задачею про розфарбовування* (або *завданням розфарбовування*) графа. Відповідне цьому числу розфарбування вершин розбиває множину вершин графа на r підмножин, кожна з яких містить вершини одного кольору. Ці множини є незалежними, оскільки в межах однієї множини немає двох суміжних вершин.

Завдання знаходження хроматичного числа довільного графа стало предметом багатьох досліджень в кінці XIX і в XX столітті. З цього питання отримано багато цікавих результатів.

Хроматичне число графа не можна знайти, знаючи тільки кількість вершин і ребер графа. Недостатньо також знати степінь кожної вершини, щоб обчислити хроматичне число графа. При відомих величинах n (кількість вершин), m (кількість ребер) і $\deg\left(x_1\right),...,\deg\left(x_n\right)$ (степені вершин графа) можна отримати тільки верхню і нижню оцінки для хроматичного числа графа.

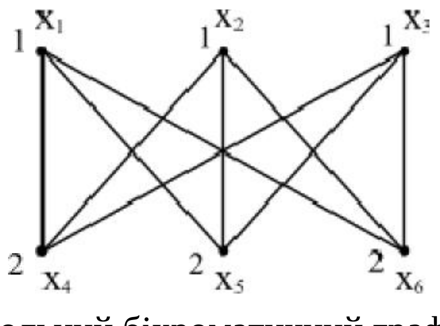


Рис. 5.1 – Дводольний біхроматичний граф Кеніга.

Приклад розфарбовування графа наведений на рисунку 5.1. Цей граф є однією із заборонених фігур, що використовуються для визначення планарності. Цифрами «1» і «2» позначені кольори вершин.

Максимальна кількість незалежних вершин графа $\sim(G)$, що дорівнює потужності найбільшої множини попарно несуміжних вершин, збігається також з потужністю найбільшої множини вершин в G, які можуть бути пофарбовані в один колір, отже:

$$¥\left(G\right) \geq \left\lceil \frac{n}{\sim\left(G\right)} \right\rceil, \tag{5.1}$$

де n - кількість вершин графа G, а $\lceil x \rceil$ позначає найбільше ціле число, яке не більше за x.

Ще одна нижня оцінка для $¥\left(G\right)$ може бути отримана наступним чином:

$$¥\left(G\right) \geq \frac{n^2}{n^2 - 2m}. \tag{5.2}$$

Верхня оцінка хроматичного числа може бути обчислена за формулою:

$$¥\left(G\right) \leq 1 + \max_{x_j \in X}\left[d\left(x_j\right) + 1\right]. \tag{5.3}$$

Застосування оцінок для хроматичного числа значно звужує межі рішення. Для визначення оцінки хроматичного числа також можуть використовуватися інші топологічні характеристики графа, наприклад, властивість планарності.

Граф, який можна зобразити на площині так, що жодні два його ребра не перетинаються між собою, називається *планарним.*

Теорема про п'ять фарб. Кожен планарний граф можна розфарбувати за допомогою п'яти кольорів так, що будь-які дві суміжні вершини будуть пофарбовані в різні кольори, тобто якщо граф G - планарний, то $¥\left(G\right) \leq 5$.

Гіпотеза про чотири фарби (недоведена). Кожен планарний граф можна розфарбувати за допомогою чотирьох кольорів так, що будь-які дві суміжні вершини будуть пофарбовані в різні кольори, тобто якщо граф G – планарний, то $¥\left(G\right) \leq 4$.

У 1852 р. про гіпотезу чотирьох фарб говорилося в листуванні Огюста де Моргана з сером Вільямом Гамільтоном. З того часу ця «теорема» стала, поряд з теоремою Ферма, однією з найзнаменитіших невирішених задач в математиці.

Повний граф K_n завжди розфарбовується в n кольорів, тобто кількість кольорів дорівнює кількості його вершин.

5.2. АГОРИТМ ПРЯМОГО НЕЯВНОГО ПЕРЕБОРУ

Алгоритм прямого неявного перебору є найпростішим алгоритмом вершинного розфарбування графів. Цей алгоритм дозволяє реалізувати правельне розфарбування графа з вибором мінімальної в рамках даного алгоритму кількості фарб.

Введемо такі структури даних:

```
Const Nmax=100; {*максимальна кількість вершин графа*}
Type V=0..Nmax;
     TS=Set of V;
    TColArr = Array (1..Nmax) of V;
    TA = Array (1..Nmax, 1..Nmax) of Integer;

Var ColArr: TColArr; {*масив номерів фарб для кожної вершини графа*}
    A:TA; {*матриця суміжності графа*}

Function Color (i): Integer;
{*функція вибору фарби для розфарбування вершини з номером i *}
Var W:TS;
    j:Byte;
Begin
  W:=[];
  For j=1 to i-1 do if A[j,i]=1 then W:=W+[ColArr[j]];
 {*формування множини фарб, що використані для розфарбування суміжних до
  вершини i вершин з номерами меншими за i*}
 j:=0; {*змінну j далі використовуємо для вибору номера фарби*}
  Repeat
     Inc(j);
   Until NOT (j In W);
   Color:=j;
End;
Begin
 <Вводимо матрицю суміжності графа>
 {*цикл по вершинах графа*}
  For i=1 to Nmax do ColArr[i]:=Color(i);
  <Виводимо результат розфарбування>
End;
```

5.3. ПРИКЛАД АГОРИТМУ ПРЯМОГО НЕЯВНОГО ПЕРЕБОРУ

3. Розглянемо граф $G(V,E)$, який показаний на рис. 5.2.

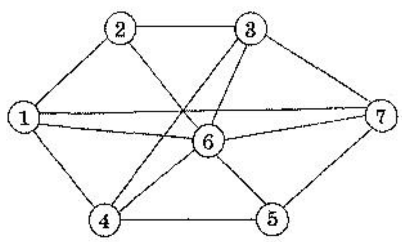


Рис.5.2

Множину вершин графа $V = \{1, 2, 3, 4, 5, 6, 7\}$ потрібно розфарбувати з використанням алгоритму послідовного розфарбування.
Сформуємо матрицю суміжності A :

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Крок1. Для вершини 1, відповідно до представленого вище алгоритму, множина розфарбованих суміжних вершин завжди є пустою. Тому функція Color(1) завжди повертатиме фарбу 1. Встановимо, що 1 кодує фарбу червоного кольору.

Крок2. Розглянемо вершину 2. Для цієї вершини єдиною меншою за номером суміжною вершиною є вершина 1. Ця вершина розфарбована червоним кольором. Тому множина W містить єдиний елемент 1. Тому функція Color(2) повертає наступну за номером фарбу 2 синього кольору.

Крок3. Вершина 3 має єдину суміжну вершину з меншим номером. Це вершина 2. Множина W містить єдиний елемент 2. Тому функція Color(3) повертає фарбу з номером 1 червоного кольру.

Крок4. Вершина 4 має дві суміжні вершини з меншими номерами:1 і 3. Оскільки обидві вершини розфарбовані в колір 1, то множина W містить єдиний елемент 1. Тому функція Color(4) повертає наступну за номером фарбу 2 синього кольору.

Крок5. Вершина 5 має єдину суміжну вершину з меншим номером. Це вершина 4. Множина W містить єдиний елемент 2. Тому функція Color(5) повертає фарбу з номером 1 червоного кольру.

Крок6. Вершина 6 має такі суміжні вершини з меншими номерами:1,3,4 і 5. Ці вершини розфарбовані в колір 1 та колір 2. Отже множина W містить два елементи: 1 і 2. Тому функція Color(6) повертає наступну за номером фарбу 3 зеленого кольору.

Крок7. Вершина 7 має такі суміжні вершини з меншими номерами:1,3, 5 і 6. Ці вершини розфарбовані в колір 1 та колір 3. Отже множина W містить два елементи: 1 і 2. Тому функція Color(7) повертає фарбу 2 синього кольору.

В результаті роботи данного алгоритму одержуємо правильно розфарбований граф, що показаний на рис. 5.3.

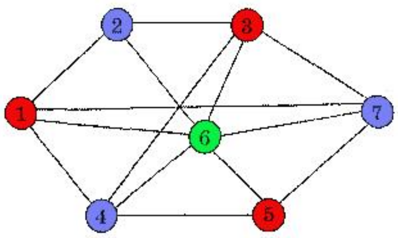


Рис. 5.3.

5.4. ЕВРИСТИЧНИЙ АЛГОРИТМ РОЗФАРБОВУВАННЯ

Точні методи розфарбовування графа складні для програмної реалізації. Однак існує багато евристичних процедур розфарбовування, які дозволяють знаходити хороші наближення для визначення хроматичного числа графа. Такі процедури також можуть з успіхом використовуватися при розфарбовуванні графів з великим числом вершин, де застосування точних методів не виправдане з огляду на високу трудомісткість обчислень.

З евристичних процедур розфарбовування слід зазначити послідовні методи, засновані на впорядкуванні множини вершин. В одному з найпростіших методів вершини спочатку розташовуються в порядку зменшення їх степенів. Перша вершина зафарбовується в колір 1, потім список вершин переглядається за зменшенням степенів, і в колір 1 зафарбовується кожна вершина, яка не є суміжною з вершинами, зафарбованими в той же колір. Потім повертаємося до першої в списку незафарбованої вершині, фарбуємо її в колір 2 і знову переглядаємо список вершин зверху вниз, зафарбовуючи в колір 2 будь-яку незафарбовану вершину, яка не з’єднана ребром з іншою, вже пофарбованою в колір 2 вершиною. Аналогічно діємо із кольорами 3, 4 і т. д., допоки не будуть пофарбовані всі вершини. Кількість використаних кольорів буде тоді наближеним значенням хроматичного числа графа.

Евристичний алгоритм розфарбовування вершин графа має наступний вигляд:

Крок 1. Сортувати вершини графа за степенями зменшення:

$$\deg(x_i) \geq \deg(x_j),\ \forall x_i, x_j \in G.$$

Встановити поточний колір $p := 1, i := 1$.

Крок 2. Вибрати чергову нерозфарбовану вершину зі списку і призначити їй новий колір $\mathrm{col}(x_i) := p;\ X = \{x_i\}$.

Крок 3. $i := i + 1$. Вибрати чергову не розфарбовану вершину x_i і перевірити умову суміжності: $x_i \cap \Gamma(X) = \varnothing$, де X - множина вершин, вже розфарбованих в колір p. Якщо вершина x_i не є суміжною з даними вершинами, то також присвоїти їй колір p: $\mathrm{col}(x_i) := p$.

Крок 4. Повторювати крок 3 до досягнення кінця списку $(i = n)$.

Крок 5. *Якщо* всі вершини графа розфарбовані, *то* – кінець алгоритму; *інакше*: $p := p + 1;\ i := 1$. Повторити крок 2.

Для роботи алгоритму можна використовувати довільну структуру даних, яка однозначно задає граф.

Розглянемо роботу алгоритму на прикладі матриці суміжності A.

```
Const Nmax=100; {*максимальна кількість вершин графа*}
```

```
Type TArr = Array (1..Nmax) of Byte;
    TA = Array (1..Nmax, 1..Nmax) of Byte;

Var ColArr: TArr; {*масив номерів фарб для кожної вершини графа*}
    DegArr: TArr {*масив ступенів вершин*}
    SortArr:TArr;{*відсортований за зменшенням ступенів масив вершин*}
     A:TA; {*матриця суміжності графа*}
    CurCol: Byte;  {*поточний номер фарби*}
    n:Byte;

Procedure DegForming;{*Процедура формування масиву ступенів вершин*}
Var i:Byte;
Begin
  For i:=1 to Nmax do
  begin
   DegArr[i]:=0; ColArr[i]:=0;
   For j:=1 to Nmax do
   DegArr[i]:= DegArr[i]+A[i,j];
  end;
End;
Procedure SortNodes;  {*Сортування вершин за ступенями*}
Var max,c,k,i:Byte;
Begin
 For k:=1 to Nmax-1 do
 begin
  max:=DegArr[k]; c:=k;
  For i:=k+1 to N do
  If DegArr[i] > max then
  begin
   max:= DegArr[ [i];
   c:=i;
  end;
  DegArr[c]:= DegArr[ [k];
  DegArr[k]:=max;
  SortArr[k]:=c;
 end;
End;
Procedure Color (i:Byte);
{*Розфарбування поточним кольром не суміжних з і вершин *}
Var j:Byte;
Begin
 For j=1 to Nmax do if A[j,i]=0 then
 begin
  If ColArr[j]=0 then ColArr[j]:=CurCol;
 end;
```

```
End;
Begin
 CurCol:=1;
 <Вводимо матрицю суміжності графа>
 DegForming; {*Формування масиву ступенів вершин*}
 SortNodes;  {*Формування массиву відсортованих вершин SortArr*}
 For n:=1 to Nmax do
 begin
  If ColArr[SortArr[n]]=0 then
  begin
   ColArr[SortArr[n]]:=CurCol;
   Color(SortArr[n]);
   Inc(CurCol);
  end;
 end;
<Виводимо результат розфарбування>
end;
```

5.5. ПРИКЛАД ЕВРИСТИЧНОГО АЛГОРИТМУ РОЗФАРБУВАННЯ

Розфарбуємо граф G, зображений на рисунку 5.4. Проміжні дані для вирішення завдання будемо записувати в таблицю. Відсортуємо вершини графа за зменшенням їх степенів. У результаті отримуємо вектор відсортованих вершин $\mathrm{SortArr} = (1,5,6,4,2,3)$

Степені, що відповідають даним вершинам, утворюють другий вектор: $D = (5,4,4,3,2,2)$

У першому рядку таблиці запишемо вектор $\mathrm{SortArr}$, у другому – ступені відповідних вершин. Наступні рядки відображають вміст вектора розфарбування $\mathrm{ColArr}[\mathrm{SortArr}]$.

Таким чином, даний граф можна розфарбувати не менш ніж у чотири кольори, тобто ¥$(G) = 4$.

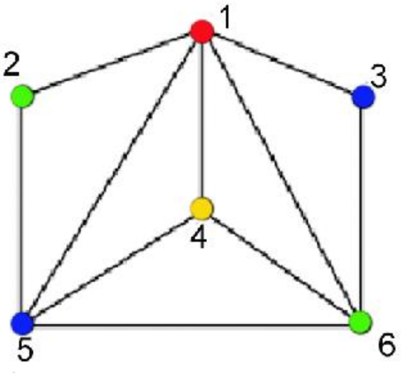


Рис. 5.4.

Номери вершин SortArr	x_1	x_5	x_6	x_4	x_2	x_3
Степені вершин DegArr	5	4	4	3	2	2
$\mathrm{CurCol} = 1$	1	-	-	-	-	-
$\mathrm{CurCol} = 2$	1	2	-	-	-	2
$\mathrm{CurCol} = 3$	1	2	3	-	3	2
$\mathrm{CurCol} = 4$	1	2	3	4	3	2

5.6. МОДИФІКОВАНИЙ ЕВРИСТИЧНИЙ АЛГОРИТМ РОЗФАРБУВАННЯ

Попередні визначення

Визначення 1. Відносний ступінь – це ступінь нерозфарбованих вершин у нерозфарбованому підграфі даного графа.

Визначення 2. Двокроковий відносний ступінь – сума відносних ступенів суміжних вершин у нерозфарбованому підграфі.

Проста модифікація описаної вище евристичної процедури базується на переупорядкуванні нерозфарбованих вершин по незростанню їх відносних ступенів.

Дана модифікація полягає у тому, що якщо дві вершини мають однакові ступені, то порядок таких вершин випадковий. Їх можна впорядкувати по двокрокових ступенях. Двокроковий ступінь визначимо як суму відносних ступенів суміжних вершин.

Крок1. Сортуємо вершини графа за ступенями зменшення:

$$\deg(x_i) \geq \deg(x_j),\ \forall x_i, x_j \in G.$$

У випадку $\deg(x_i) = \deg(x_j)$, $\forall x_i, x_j \in G$ розглянемо множини суміжності $\Gamma(x_i)$ і $\Gamma(x_j)$.

Сортуємо вершини за ознакою :

$$[\deg(x_{i1}) + \deg(x_{i2}) + ... + \deg(x_{ik})] \geq [\deg(x_{j1}) + \deg(x_{j2}) + ... + \deg(x_{jn})]\ ,$$

де $x_{i1}, x_{i2}, ..., x_{ik}$ - нерозфарбовані вршини з множини суміжності $\Gamma(x_i)$;

$x_{j1}, x_{j2}, ..., x_{jn}$ - нерозфарбовані вршини з множини суміжності $\Gamma(x_j)$;

Встановити поточний колір $p := 1, i := 1$.

Крок 2. Вибрати чергову нерозфарбовану вершину зі списку і призначити їй новий колір $\mathrm{col}(x_i) := p;\ X = \{x_i\}$.

Крок 3. $i := i + 1$. Вибрати чергову нерозфарбовану вершину x_i і перевірити умову суміжності: $x_i \cap \Gamma(X) = \varnothing$, де X - множина вершин, вже розфарбованих в колір p. Якщо вершина x_i не є суміжною з даними вершинами, то також присвоїти їй колір p: $\mathrm{col}(x_i) := p$.

Крок 4. Повторювати крок 3 до досягнення кінця списку $(i = n)$.

Крок 5. *Якщо* всі вершини графа розфарбовані, *то* – кінець алгоритму;

інакше: $p := p + 1;\ i := 1$. Повторити крок 2.

Даний алгоритм від попереднього відрізняється ускладненням процедури сортування SortNodes, яка при сортуванні вершин з однаковими ступенями враховує двукроковий ступінь.

Як і в попередньому випадку, розглянемо роботу алгоритму на прикладі матриці суміжності A.

```
Const Nmax=100; {*максимальна кількість вершин графа*}
Type TArr = Array (1..Nmax) of Integer;
    TA = Array (1..Nmax, 1..Nmax) of Byte;

Var ColArr: TArr; {*масив номерів фарб для кожної вершини графа*}
    DegArr: TArr {*масив ступенів вершин*}
    SortArr:TArr;{*відсортований за зменшенням ступенів масив вершин*}
     A:TA; {*матриця суміжності графа*}
    CurCol: Byte;  {*поточний номер фарби*}
    n:Byte;

Procedure DegForming;{*Процедура формування масиву ступенів вершин*}
Var k:Byte;
 Function DegCount(m:Byte):Integer;
 Var Deg:Iteger;
 Begin
  Deg:=0;
  For k:=1 to Nmax do Deg:= Deg+A[k,m];
  DegCount:=Deg;
 End;
Begin
 For j:=1 to Nmax do
 begin
  ColArr[i]:=0;
  DegArr[j]:= DegCount(j)*100;
  For i:=1 to Nmax do
  If A[i,j]=1 then DegArr[i]:= DegArr[i]+DegCount(i);
 end;
End;

Procedure SortNodes;  {*Сортування вершин за ступенями*}
Var max,c,k,i:Byte;
Begin
 For k:=1 to Nmax-1 do
 begin
  max:=DegArr[k]; c:=k;
  For i:=k+1 to N do
  If DegArr[i] > max then
  begin
   max:= DegArr[ [i];
```

```
   c:=i;
  end;
  DegArr[c]:= DegArr[ [k];
  DegArr[k]:=max;
  SortArr[k]:=c;
 end;
End;
Procedure Color (i:Byte);
{*Розфарбування поточним кольром не суміжних з i вершин *}
Var j:Byte;
Begin
 For j=1 to Nmax do if A[j,i]=0 then
 begin
  If ColArr[j]=0 then ColArr[j]:=CurCol;
 end;
End;
Begin
 CurCol:=1;
 <Вводимо матрицю суміжності графа>
 DegForming; {*Формування масиву ступенів вершин*}
 SortNodes;  {*Формування массиву відсортованих вершин SortArr*}
 For n:=1 to Nmax do
 begin
  If ColArr[SortArr[n]]=0 then
  begin
   ColArr[SortArr[n]]:=CurCol;
   Color(SortArr[n]);
   Inc(CurCol);
  end;
 end;
<Виводимо результат розфарбування>
end;
```

5.7. ПРИКЛАД МОДИФІКОВАНОГО ЕВРИСТИЧНОГО АЛГОРИТМУ РОЗФАРБУВАННЯ

Розфарбуємо граф G , зображений на рисунку 5.3

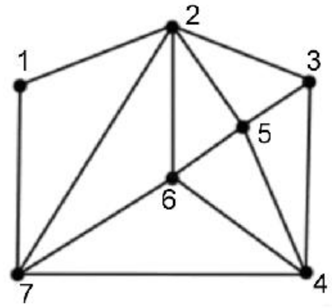


Рис. 5.5.

Відсортуємо вершини графа за зменшенням їх ступенів. У результаті отримуємо вектор відсортованих вершин $\mathrm{SortArr} = (2,6,5,4,7,3,1)$

Вектор ступенів відсортованих вершин має наступний вигляд: $D = (5,4,4,4,4,3,2)$

У першому рядку таблиці запишемо вектор SortArr, у другому – вектор D, а у третьому – вектор D^2 .

Четвертий рядок відповідає представленню ступенів D та D^2 в масиві DegArr.

Наступні рядки відображають вміст вектора розфарбування $\mathrm{col}\left(X^*\right)$.

Таким чином, даний граф можна розфарбувати не менш ніж у три кольори, тобто ¥ $(G) = 3$.

Номери вершин X^*	a_2	a_6	a_5	a_4	a_7	a_3	a_1
Ступінь вершин D	5	4	4	4	4	3	2
Двокроковий ступінь D^2		17	16	15	15		
DegArr	500	417	416	415	415	300	200
$\mathrm{CurCol} = 1$	1	-	-	1	-	-	-
$\mathrm{CurCol} = 2$	1	2	-	1	-	2	2
$\mathrm{CurCol} = 3$	1	2	3	1	3	2	2

В результаті роботи модифікованого евристичного алгоритму одержимо розфарбований граф, показаний на рис. 5.6.

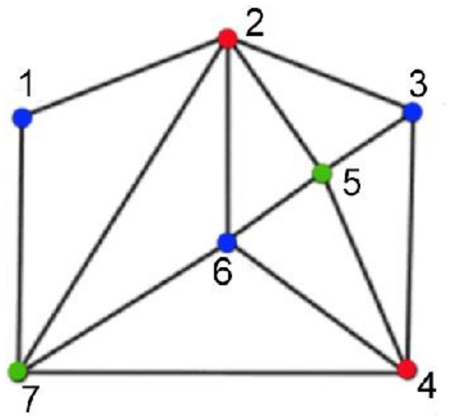


Рис. 5.6.

5.8. РОЗФАРБУВАННЯ ГРАФА МЕТОДОМ А.П. ЄРШОВА

Андрій Петрович Єршов (1931-1988 рр.), видатний вчений в області теоретичного програмування, вніс великий вклад у розвиток інформатики. Зокрема, він створив алгоритм розфарбування графа, що базується на оригінальній евристичній ідеї.
Введемо ряд визначень.

Для даної вершини $v \in V$ графа $G(V,E)$ назвемо всі суміжні з нею вершини околицею 1-го порядку — $R_1(v)$.

Всі вершини, що перебувають на відстані два від v, назвемо околицею 2-го порядку — $R_2(v)$.

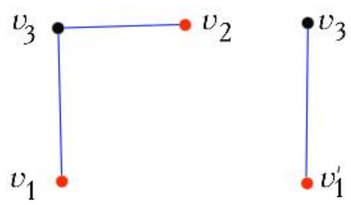
Граф $G(V,E)$, у якого для вершини $v \in V$ всі інші вершини належать околиці $R_1(v)$ назвемо граф-зіркою відносно вершини v.

Ідея алгоритму

Фарбування у фарбу $\sim$ вершини v утворює навколо неї в $R_1(v)$ «мертву зону» для фарби $\sim$. Очевидно, при мінімальному розфарбуванні кожна фарба повинна розфарбувати максимальну можливу кількість вершин графа. Для цього необхідно, щоб мертві зони, хоча б частково, перекривалися між собою. Перекриття мертвих зон двох несуміжних вершин v_1 і v_2 досягається тільки тоді, коли одна з них перебуває в околиці $R_2(v_1)$ від іншої, $v_2 \in R_2(v_1)$.

Таким чином, суть алгоритму полягає в тому, щоб на черговому кроці вибрати для розфарбування фарбою $\sim$ вершину $v_2 \in R_2(v_1)$. Цей процес повторювати доти, поки фарбою $\sim$ не будуть пофарбовані всі можливі вершини графа.

Графічно фарбування вершин v_1 і v_2 однією фарбою можна відобразити як «склеювання» цих вершин.



.

Рис. 14.1. Приклад об’єднання двох вершин: $v_1' := v_1 \cup v_2$

При цьому кількість вершин зменшується на одиницю у графі G, а також зменшується кількість ребер.

Алгоритм

1.Встановити $i := 0$.

2.Вибрати в графі G довільну незафарбовану вершину v.

3.Встановити $i := i + 1$.

4.Розфарбувати вершину v фарбою i.

5.Розфарбовувати фарбою i незабарвлені вершини графа G, вибираючи їх з $R_2(v)$, поки граф не перетвориться в граф-зірку відносно v .

6.Перевірити, чи залишилися незабарвлені вершини в графі G. Якщо так, то перейти до п. 2, інакше - до п. 7.

7.Отриманий повний граф K_i. Хроматичне число графа $X(K_i) = i$.

Кінець алгоритму.

Як випливає з алгоритму, після того, як у першу обрану вершину стягнуті всі можливі й граф перетворився в граф-зірку відносно цієї вершини, вибирається довільним чином друга вершина й процес повторюється доти, поки граф не перетвориться в повний.

Повний граф - це граф-зірка відносно будь-якої вершини. Хроматичне число повного графа дорівнює числу його вершин.

5.9. ПРИКЛАД РОЗФАРБУВАННЯ ГРАФА МЕТОДОМ А.П. ЄРШОВА

На мал.14.2 зображений граф G, на прикладі якого будемо розглядати роботу евристичного алгоритму Єршова.

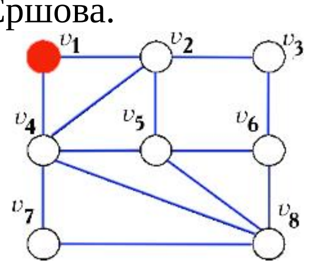


Рис.14.2. Початковий граф G

Виберемо довільну вершину, наприклад, v_1. Околиця 2-го порядку $R_2(v_1)=\{v_3,v_5,v_7,v_8\}$. Склеїмо вершину v_1 наприклад, з вершиною v_3: $v_1'=v_1\cup v_3$. Одержимо граф G_1 зображений на мал. 14.3.

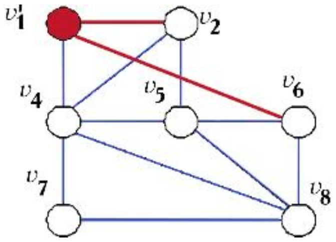


Рис.14.3. Граф G_1. Склеєні вершини v_1 та v_3

Розглянемо тепер граф G_1. Околиця другого порядку вершини v_1' визначається множиною $R_2(v_1')=\{v_5.v_7,v_8\}$. Склеїмо вершину v_1', наприклад, з вершиною v_5: $v_1'':=v_1'\cup v_5$. Одержимо граф G_2, зображений на мал. 14.4.

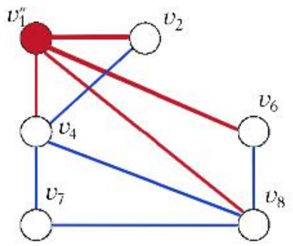


Рис.14.4. Граф G_2. Склеєні вершини v_1' й v_5

У графі G_2 визначимо околицю другого порядку для вершини v_1'' : $R_2(v_1'')=\{v_7\}$. Склеїмо вершину v_1'' з вершиною v_7: $v_1'''=v_1''\cup v_7$. Одержимо граф G_3, зображений на мал. 14.5. Цей граф є зіркою відносно вершини v_1'''.

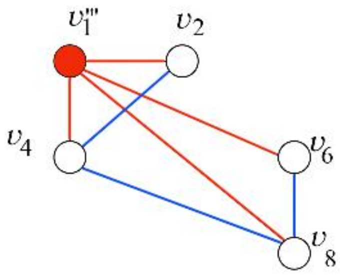


Рис.14.5. Граф G_3. Склеєні вершини v_1'' й v_7

У графі G_3 виберемо, наприклад, вершину v_2 для фарбування другою фарбою. Околиця 2-го порядку $R_2(v_2)=\{v_6,v_8\}$. Склеїмо вершину v_2, наприклад, з вершиною v_6: $v_1'''=v_1''\cup v_6$. Одержимо граф G_4, зображений на мал.14.6.

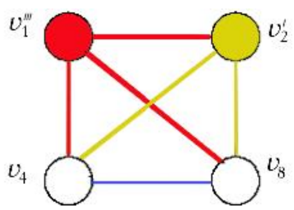


Рис.14.6. Граф G_4 еквівалентний K_4. Склеєні вершини v_1'' й v_6

Граф G_4 є повним четиривершинним графом K_4 . Отже, граф G_4 на мал.14.6 розфарбовується в чотири кольори. У перший (червоний) колір розфарбовується вершина v_1 й склеєні з нею вершини: v_3, v_5 і v_7. У другий (жовтий) колір розфарбовується вершина v_2 й склеєна з нею вершина v_6. У третій (зелений) колір розфарбовується вершина v_4 й у четвертий (білий) колір розфарбовується вершина v_8.

У підсумку одержуємо правильно розфарбований граф, показаний на мал.14.7.

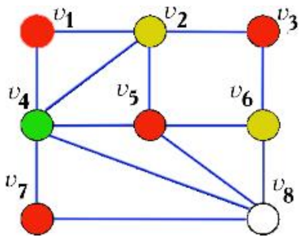


Рис.14.7.Граф G, розфарбований за допомогою алгоритму розфарбування А.П.Єршова

Програмна реалізація

Розглянемо матрицю суміжності графа G, представленого на рис. 14.2.

$$A = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\ v_1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ v_2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ v_3 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ v_4 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ v_5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ v_6 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ v_7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ v_8 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Операції склеювання відповідних вершин графа, які описні вище, виконуються на матриці суміжності шляхом виконання операції диз’юнкції між цими вершинами.

Наприклад, розглянемо склеювання вершин 1 та 3.

Результуюча матриця сумажності містить новий рядок та стовбець для вершини v_1' , але не містить v_1 та v_3 .

$$A' = \begin{array}{c} \\ v_1' \\ v_2 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{array} \begin{pmatrix} v_1' & v_2 & v_4 & v_5 & v_6 & v_7 & v_8 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Фінальна матриця суміжності має вигляд:

$$A''' = \begin{array}{c} \\ v_1''' \\ v_2' \\ v_4 \\ v_8 \end{array} \begin{pmatrix} v_1''' & v_2' & v_4 & v_8 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix},$$

що відповідає матриці повного графа K_4 .

```
Const n=10; {*максимальна кількість вершин графа*}
Type V=0..n;
     R2S=Set of V;
    TA = Array (1..n, 1..n) of Integer;

Var  A:TA; {*матриця суміжності графа*}
    MainNode:Byte;

Procedure Glue(master,slave:Byte);
{*Процедура склеювання вершин *}
Begin
  {*Склеювання рядка master і рядка slave*}
 For i=1 to n do A[i,master]:= A[i,master] OR A[i,slave];
{*Склеювання стовбця master і стовбця slave*}
 For j=1 to n do A[master,j]:= A[master,j] OR A[slave,j];
End;

Procedure Reduce(master,slave:Byte);
{*Процедура видалення стовбця і рядка в матриці суміжності*}
Begin
 For i:=1 to n do
  For j:=1 to n-1 do
  {*Видалення рядка slave*}
   If (j≥slave) then A[i,j]:= A[i,j+1];
```

```
 For j:=1 to n-1 do
  For i:=1 to n-1 do
 {*Видалення стовбця slave*}
   If (i≥slave) then A[i,j]:= A[i+1,j];
 n:=n-1;
End;
Function Check_K:Byte;
{*Процедура перевірки наявності повного графа*}
Var Gh:Byte;
Begin
   {*У повному графі всі елементи матриці, крім діагональних, повинні бути одиничними*}
 Ch:=0;
 For i:=1 to n do
 For j:=1 to n do if (i≠j) AND (A[i,j]=0) then Ch:=j;
 Check_K:=Ch;
End;

Procrdure R2(master:Byte);
{*Процедура формування околиці 2-го порядка*}
Begin
 For j:=1 to n do
 begin
  If (j≠master) AND (A[master,j]=1) then
  begin
   {*Вибрали суміжну вершину до master*}
   For i=1 to n do
   begin
    If (i≠master) AND (A[j,i]=1) then
    {* Вибрали  вершину околиці 2-го порядку  до master*}
    R2S:=R2S+[i];    {*Додали вершину до множини вершин околиці*}
   end;
  end;
 end;
End;

Begin
 MainNode:=1; {*Вибираємо першу вершину*}
 K_finded:=false; {*Ознака закінчення алгоритму*}
 While K_finded do
 begin
  R2S:=[]; {*Очистка множини околиці 2-го порядку*}
  R2(MainNode); {*Формуємо околицю 2-го порядку для MainNode*}
  For k=1 to n do {*Цикл по вершинах графа*}
  begin
   If k in R2S then
   begin {*Вершина належить околиці другого порядку*}
```

```
      {*Склеювання вершини MainNode з вершиною околиці k *}
      Glue(MainNode,k);
      {*Модифікація матриці суміжності після склеювання вершин *}
      Reduce(MainNode,k);
    end;
  end;
  MainNode:= Check_K(MainNode);
  If MainNode=0 then K_finded:=true;
 end;
End;
```

5.10. РЕКУРСИВНА ПРОЦЕДЕРА ПОСЛІДОВНГО РОЗФАРБУВАННЯ

1. Фіксуємо порядок обходу вершин.

2. Ідемо по вершинах, використовуючи такий найменший колір, який не викличе конфліктів.

3. Якщо вже використаний колір вибрати не виходить, то тільки тоді вводимо новий колір.

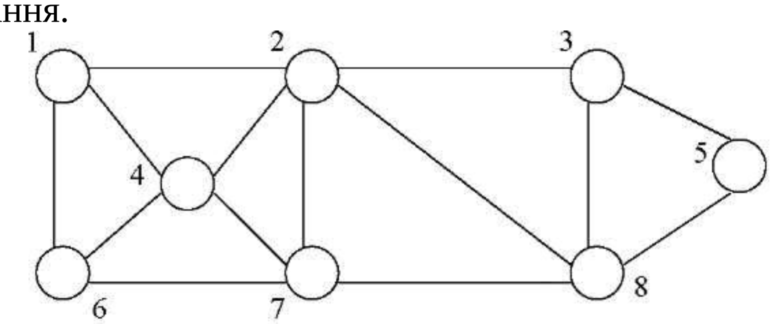
У процедурі використовується рекурсивний виклик процедури фарбування наступної вершини у випадку успішного фарбування попередньої вершини.

```
Const n=10;
      Cmax=10;
Type
    TA = Array (1..n, 1..n) of Byte;
    TArr = Array (1..n) of Byte;
Var i:Byte;
    color:TArr;
     A:TA;
     C:Byte;

procedure visit(i:Byte);
 Function Nicecolor:Boolean;
{*Функція пошуку неконфліктної фарби*}
 Var CN:Boolean;
     j:integer;
 Begin
  CN:=true;
  For j=1 to n do
  If (A[j,i]=1) AND (color[j]=c) then CN:=false;
  {*Знайдена суміжна вершин до вершини i. Ця вершина має поточний колір c .
     Тоді колір c є конфліктним*}
 End;
begin
 if i = n + 1 then Print else
{*Якщо всі вершини розфарбовано, то виводимо результат*}
 begin
```

```
  If color[i]=0 then {*Якщо поточна вершина не рофарбована*}
  begin
   for c:=color[i]+1 to Cmax do
   if Nicecolor then
   begin
    color[i]:=c;
   {*Якщо неконфліктний, то розфарбовуємо вершину і фарбою с*}
    visit(i+1);
    {Рекурсивно викликаємо процедуру для розфарбування наступної вершини}
   end;
  end;
 end;
end;

Begin
 i:=1;
 visit(i);
End;
```

5.11. ПРИКЛАД РОБОТИ РЕКУРСИВНОЇ ПРОЦЕДУРИ ПОСЛІДОВНОГО РОЗФАРБУВАННЯ

Розглянемо граф G та застосуємо рекурсивну процедуру для його розфарбування.



Матриця суміжності A має такий вигляд

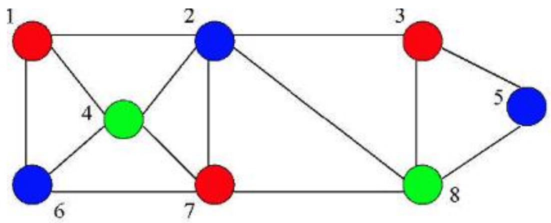
$$A = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 3 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 4 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 6 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 7 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 8 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Робота алгоритму зведена в таблицю

Перший стовбець містить виклики процедури Visit, а решта стовбців показує яка фарба була принята, а яка відхилена.

	Червоний	Синій	Зелений
Visit(1)	+		
Visit(2)	-	+	
Visit(3)	+		
Visit(4)	-	-	+
Visit(5)	-	+	
Visit(6)	-	+	
Visit(7)	+		
Visit(8)	-	-	+

Розфарбований граф має вигляд:



5.12. «ЖАДІБНИЙ » АЛГОРИТМ РОЗФАРБУВАННЯ

Нехай дано зв'язний граф $G(V,E)$.

1. Задамо множину $monochrom := \varnothing$, куди будемо записувати всі вершини, які можна пофарбувати одним кольором.

2. Переглядаємо всі вершини й виконуємо наступний «жадібний» алгоритм:

Procedure Greedy
For (для кожної незафарбованої вершини $v \in V$) **do**
If v *не суміжна з вершинами з* $monochrom$ **then**
begin
$color(v) :=$ колір;
$monochrom := monochrom \cup \{v\}$
End;

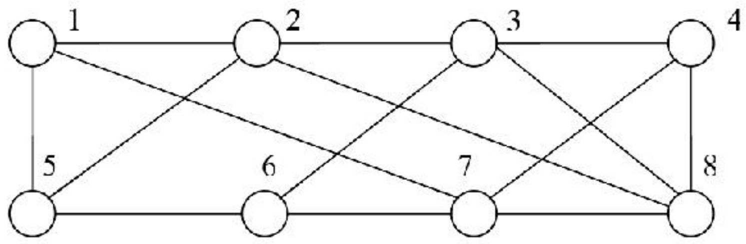
Розглянемо детальніше програмну реалізацію даного алгоритму за умови, що для представлення графа використовують матрицю суміжності.

```
Const N=10; {*максимальна кількість вершин графа*}
Type V=0..N;
      TS=Set of V;
     TColArr = Array (1..N) of V;
     TA = Array (1..N, 1..N) of Integer;
```

```
Var ColArr: TColArr; {*масив номерів фарб для кожної вершини графа*}
     A:TA; {*матриця суміжності графа*}
     Color:Byte;
      AllColored:Boolean;
      k:Byte;
Procedure Avid(i:Integer);
{*функція вибору фарби для розфарбування вершини з номером і *}
Var W:TS;
    j:Byte;
function Check(i):Boolean; {*Перевірка кольору суміжних вершин*}
var Ch:Boolean;
begin
 Ch:=true;
 For j=1 to n do
 If (A[j,i]=1)then {*Якщо вершина j суміжна з тою, що підлягає перевірці *}
 If (j in W)then Ch:=false;
 {*Якщо вершина j розфарбована поточним кольором *}
 Check:= Ch;
end;
Begin
 Inc(Color); {*Встановлюємо новий колір*}
 W:=[]; {*Очищаємо множину одноколірних вершин*}
 ColArr[i]:=Color; {*Розфарбовуємо першу вершину новою фарбою*}
 W:=W+[i]; {*Доповнюємо множину одноколірних вершин вершиною і*}
{*Перевіряємо інші вершини на можливість розфарбування цією фарбою *}
 For k:=1 to n do if ColArr[k]=0 then
 If Check(k)then begin ColArr[k]:=Color; W:=W+[k];end;
End;
Begin {*Головний цикл*}
 <Вводимо матрицю суміжності графа>
 {*цикл по вершинах графа*}
  Color:=0;
  AllColored:=false; {*Ознака того, що всі вершини розфарбовано*}
  While not AllColored do
  Begin
   AllColored:=true;
   For i=1 to N do If ColArr[i]=0 then
   begin
    {*Знайшли не розфарбовану вершину*}
    AllColored:=false;
    Avid(i); {*процедура жадібного розфарбування*}
   end;
  End;
  <Виводимо результат розфарбування>
End;
```

5.13. ПРИКЛАД РОБОТИ «ЖАДІБНОГО» АЛГОРИТМУ РОЗФАРБУВАННЯ

Розглянемо граф G та застосуєм до нього «жадібний» алгоритм розфарбування.



Матриця суміжності A має такий вигляд

$$A=\begin{array}{c@{}c} & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Для початку розфарбування вибираємо вершину з номером 1 та розфарбовуємо її в колір 1 (червоний).

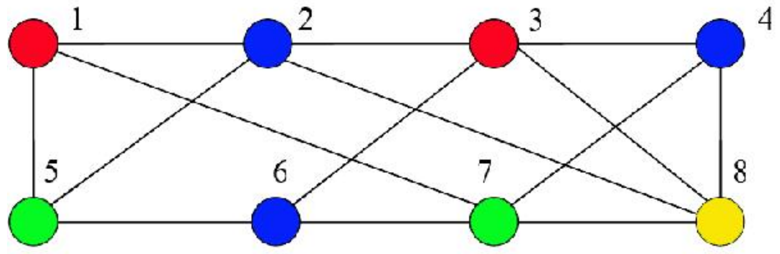
Далі відбувається пошук несуміжної вершини з вершиною 1. Якщо така вершина знайдена, то вона також розфарбовується в колір 1(червоний).

Наступна знайдена для розфорбування кольором 1 вершина повинна бути не суміжною з двома попередніми. Процес продовжується до того часу, поки всі можливості розфарбувати вершини кольором 1 будуть вичерпані.

Після цього, вибираємо фарбу кольору 2 (синя) і розфарбовуємо нею вершину з мінімальним номером, яка є не розфарбованою до цього часу. Наступна підходяща для розфарбування фарбою 2 вершина повинна бути не суміжною з вершиною, яка була розфарбована кольором 2 (синій) на попередньому кроці. Процес розфарбування фарбою 2 також продовжується до того часу, поки не будуть вичерпані всі можливості розфарбування вершин цією фарбою.

Перед вибором чергової фарби для розфарбування завжди перевіряємо, чи залишилися ще не розфарбовані вершини. Якщо такі вершини знайдено, то вибираємо чергову фарбу і продовжуємо процес розфарбування.

Якщо ж всі вершини графа розфарбовано, то процес розфарбування жадібним алгоритмом закінчується.
Результат розфарбування «жадібним» алгоритмом графа G показано на рисунку



5.14. ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №5

1. Вивчити основні означення та теореми про розфарбування графів. Створити програму розфарбування графів, що реалізує алгоритм прямого перебору.
2. Набути теоретичні знання по темі «Розфарбування графів». Створити програму розфарбування графів яка реалізує евристичний алгоритм розфарбування.
3. Вивчити основні означення та теореми про розфарбування графів. Створити програму розфарбування графів яка реалізує модифікований евристичний алгоритм розфарбування.
4. Набути теоретичні знання по темі «Розфарбування графів». Створити програму розфарбування графів яка реалізує алгоритм розфарбування методом А.П. Єршова.
5. Вивчити основні означення та теореми про розфарбування графів. Створити програму розфарбування графів за рекрсивною процедурою послідовного розфарбування.
6. Набути теоретичні знання по темі «Розфарбування графів». Створити програму розфарбування графів яка реалізує «жадібний» алгоритм розфарбування.

Вимоги до програмного забезпечення:

1. Лабораторна робота виконується в середовищі візуального програмування Lazarus.
2. В якості початкового коду використати проект LAB5Project, попередньо скачавши його з сайту викладача.
3. Початкові данні вводяться за допомогою програмних засобів уже реалізованих в згаданому проекті.
4. Необхідно в рамках форми OperForm реалізувати алгоритм відповідно до варіанту лабораторної роботи.
5. На формі OperForm відобразити як початкові дані, так і результати роботи алгоритму.
6. Вершини результуючого графу необхідно відобразити розфарбованими у відповідності з результатами роботи алгоритму.

Зміст звіту:

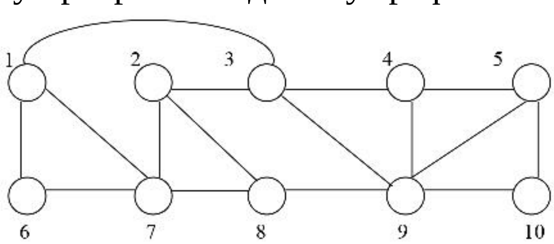
1. Титульний лист;
2. Тема завдання;
3. Завдання;
4. Блок-схема програми по п. Б
5. Роздруківка тексту програми з п. В.
7. Контрольний приклад та результати машинного розрахунку.
8. Висновки по роботі.

Контрольні питання

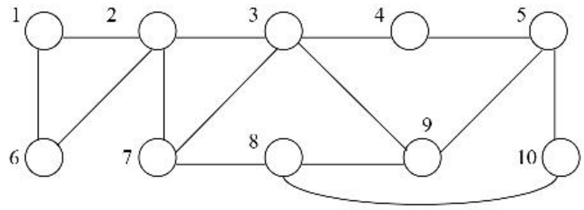
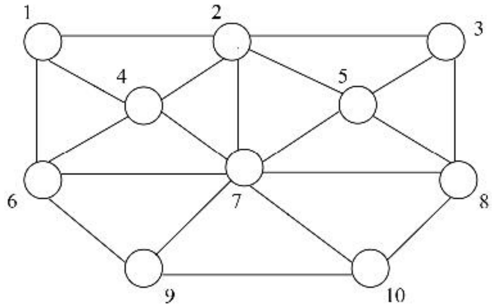
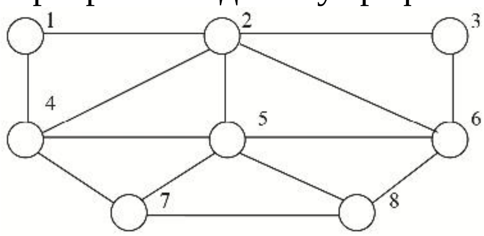
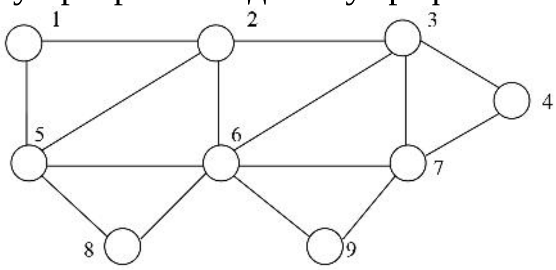
1. Сформулюйте задачу розфарбовування графа.
2. Який граф називається r-хроматичним?
3. Що називається хроматичним числом графа?
4. Як визначаються нижня і верхня оцінки хроматичного числа?
5. Який граф називається планарним? У скільки кольорів його можна розфарбувати?
6. У скільки кольорів можна розфарбувати повний граф?
7. Алгоритм простого перебору.
8. Евристичний алгоритм розфарбовування графа.
9. Модифікований истичний алгоритм розфарбовування графа.
10. Алгоритм розфарбування за методом А.П. Єршова.
11. Розфарбування за допомогою рекурсивної процедури послідовного розфарбування.
12. «Жадібний» алгоритм розфарбування графів.

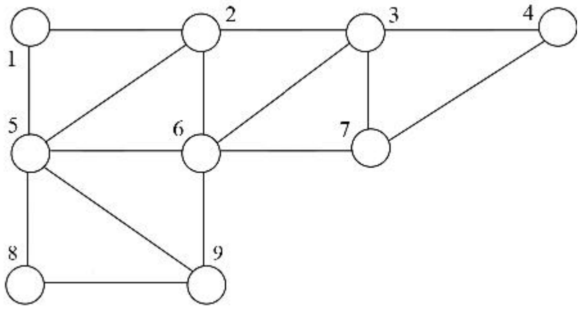
Варіанти для виконання лабораторної роботи

Номер варіанту I визначається як результат операції I = NZK mod 6+1, де NZK – номер залікової книжки. При виконанні завдань лабораторної роботи використати такі варіанти задавання графа:

<table>
<tr><td>№</td><td>Опис варіанта</td></tr>
<tr><td>1</td><td>А) Виконати завдання 1 до лабораторної роботи.
Б) Программа повинна дозволяти розфарбування довільного графа
В) Перевірити роботу програми на даному графі *G*


Вивести у графічному режимі розфарбований граф, або включити у</td></tr>
</table>

	протокол розфарбований вручну граф за результатами роботи програми.
2	А) Виконати завдання 2 до лабораторної роботи. Б) Программа повинна дозволяти розфарбування довільного графа. В) Перевірити роботу програми на даному графі G  Вивести у графічному режимі розфарбований граф, або включити у протокол розфарбований вручну граф за результатами роботи програми.
3	А) Виконати завдання 3 до лабораторної роботи. Б) Программа повинна дозволяти розфарбування довільного. В) Перевірити роботу програми на даному графі G  Вивести у графічному режимі розфарбований граф, або включити у протокол розфарбований вручну граф за результатами роботи програми.
4	А) Виконати завдання 4 до лабораторної роботи. Б) Программа повинна дозволяти розфарбування довільного. В) Перевірити роботу програми на даному графі G  Вивести у графічному режимі розфарбований граф, або включити у протокол розфарбований вручну граф за результатами роботи програми.
5	А) Виконати завдання 5 до лабораторної роботи. Б) Программа повинна дозволяти розфарбування довільного графа. В) Перевірити роботу програми на даному графі G  Вивести у графічному режимі розфарбований граф, або включити у протокол розфарбований вручну граф за результатами роботи програми.

6	А) Виконати завдання 6 до лабораторної роботи. Б) Программа повинна дозволяти розфарбування довільного графа. В) Перевірити роботу програми на даному графі G  Вивести у графічному режимі розфарбований граф, або включити у протокол розфарбований вручну граф за результатами роботи програми.

ЗМІСТ

1.	**Лабораторна робота №1.** «Множини: основні властивості та операції над ними, діаграми Венна»………………...…………………………….	3
1.1.	Властивості множин	3
1.2.	Операції над множинами	4
1.3.	Діаграми Венна	6
1.4.	Тотожності алгебри можин	7
1.5.	Вимоги до програмного забезпечення	8
1.6.	Контрольні питання	8
1.7.	Блок-схеми алгоритмів виконання операцій над множинами	9
1.8.	Завдання лабораторної роботи №1	14
2.	**Лабораторна робота №2.** «Бінарні відношення та їх основні властивості, операції над відношеннями» ……………………….………..	15
2.1.	Основні означення	15
2.2.	Властивості бінарних відношень	16
2.3.	Операції над відношеннями	17
2.4.	Завдання лабораторної роботи	18
3.	**Лабораторна робота №3.** «Комбінаторика: перестановки, розміщення, сполучення»…………………………………………………………….	22
3.1.	Основні означення	22
3.2.	Комбінаторні алгоритми	24
3.2.1.	Алгоритм побудови перестановок у лексикографічному порядку	25
3.2.2.	Блок-схема алгоритму побудови перестановок у лексикографічному порядку	29
3.2.3.	Алгоритм генерації двійкових векторів довжини n	30
3.2.4.	Блок-схема алгоритму генерації двійкових векторів довжини n	31
3.2.5.	Алгоритм генерації підмножин заданої множини	33
3.2.6.	Блок-схема алгоритму генерації підмножин заданої множини	34
3.2.7.	Перший алгоритм генерації коду Грея	35
3.2.8.	Блок-схема першого алгоритму генерації коду Грея	37
3.2.9.	Другий алгоритм генерації коду Грея	38
3.2.10.	Блок-схема другого алгоритму генерації коду Грея	38
3.2.11.	Перший алгоритм генерації підмножин з умовою мінімальної відмінності елементів	39
3.2.12.	Блок-схема першого алгоритму генерації підмножин з умовою мінімальної відмінності елементів.	40
3.2.13.	Другий алгоритм генерації підмножин з умовою мінімальної відмінності елементів	42
3.2.14.	Блок-схема другого алгоритму генерації підмножин з умовою мінімальної відмінності елементів.	42
3.2.15.	Алгоритм генерації сполучень з n по k в лексикографічному порядку.	43
3.2.16.	Блок-схема алгоритма генерації сполучень з n по k в лексикографічному порядку	44
3.2.17.	Алгоритм генерації сполучень з n по k на множині.	46
3.2.18.	Блок-схема алгоритму генерації сполучень з n по k на множині.	47
3.2.19.	Алгоритм генерації розбиття числа n у словниковому порядку	49

3.2.20. Блок-схема алгоритму генерації розбиття числа n у словниковому порядку 52

4. **Лабораторна робота №4.** «Графи. Способи представлення графів. Остовні дерева. Пошук найкоротших шляхів»……………………..……. 58
4.1. Основні означення 58
4.2. Матричні представлення 63
4.2.1. Матриця суміжності 63
4.2.2. Матриця інцидентності 64
4.3. Найкоротший остов графа 64
4.4. Алгоритм Прима-Краскала 65
4.5. Контрольний приклад 66
4.6. Задача про найкоротший шлях 67
4.6.1. Алгоритм Дейкстри 67
4.6.2. Алгоритм Форда-Беллмана знаходження мінімального шляху 75
4.6.3. Алгоритм Флойда-Уоршела 82
4.6.4. Метод динамічного програмування 84
4.6.5. Алгоритм топологічного сортування 85
4.7. Контрольний приклад 85

5. **Лабораторна робота №5.** «Розфарбовування графа, алгоритми розфарбування» …………………………………………………... 91
5.1. Основні означення 91
5.2. Алгоритм прямого неявного перебору 92
5.3. Приклад алгоритму прямого неявного перебору 93
5.4. Евристичний алгоритм разфарбування 95
5.5. Приклад евристичного алгоритму розфарбування 97
5.6. Модифікований евристичний алгоритм разфарбування 98
5.7. Приклад модифікованого евристичного алгоритму розфарбування 100
5.8. Розфарбування методом А.П. Єршова 102
5.9. Приклад розфарбування методом А.П. Єршова 103
5.10. Рекурсивна процедура послідовного розфарбування 108
5.11. Приклад роботи рекурсивної процедури послідовного розфарбування 109
5.12. «Жадібний» алгоритм розфарбування 110
5.13. Приклад роботи «жадібного» алгоритму розфарбування 112
5.14. Завдання до лабораторної роботи №5 113