

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Кафедра систем штучного інтелекту

Звіт

Розрахункова робота з дисципліни «Дискретна математика»

Варіант № 19

Виконав:

Студент групи КН-113
Іванюшенко Нестор

Викладач:

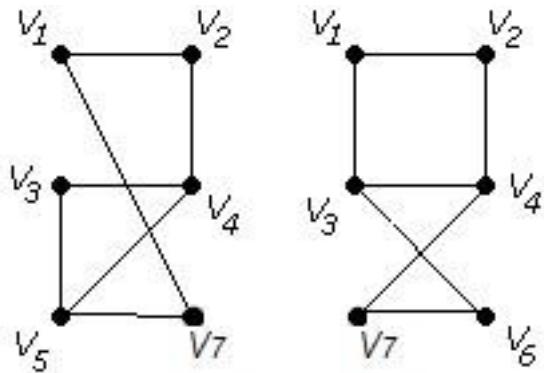
Мельникова Н. І.

Індивідуальні завдання

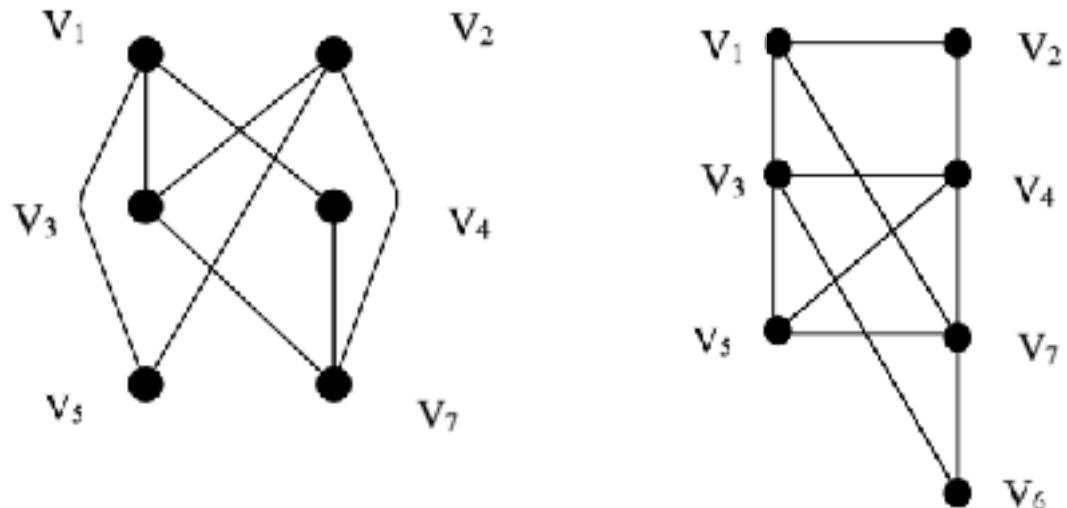
Завдання № 1.

Виконати наступні операції над графами

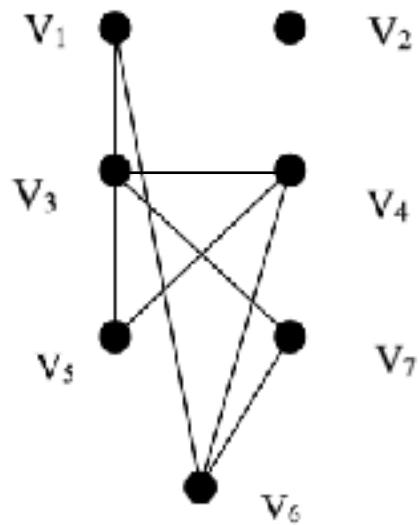
- 1) знайти доповнення до першого графу;
- 2) об'єднання графів;
- 3) кільцеву сумму G_1 та G_2 (G_1+G_2);
- 4) розмножити вершину у другому графі;
- 5) виділити підграф А - що складається з 3-х вершин в G_1 ;
- 6) добуток графів.



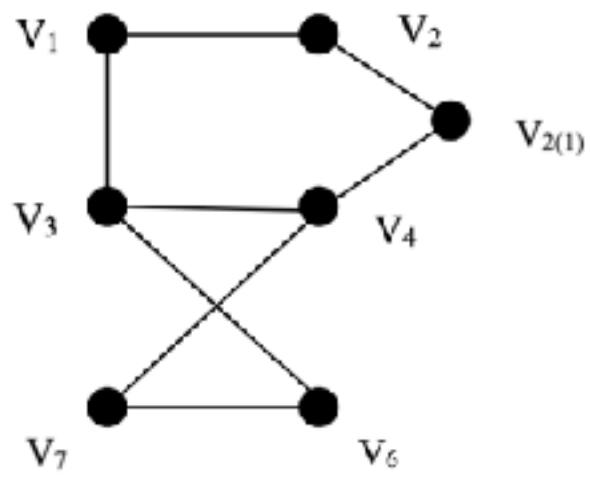
- 1) Доповнення до першого графу: 2) Об'єднання графів:



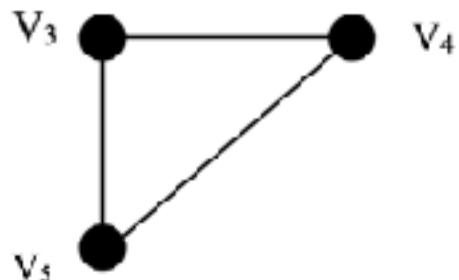
3) Кільцева сума:



4) Розмноження вершини у другому графі:



5) Виділити підграф А, що складається з 3 вершин в першому підграфі:



6) Добуток графів:

Добуток графів матиме 36 вершин, оскільки кожен із графів містить по 6 вершин.

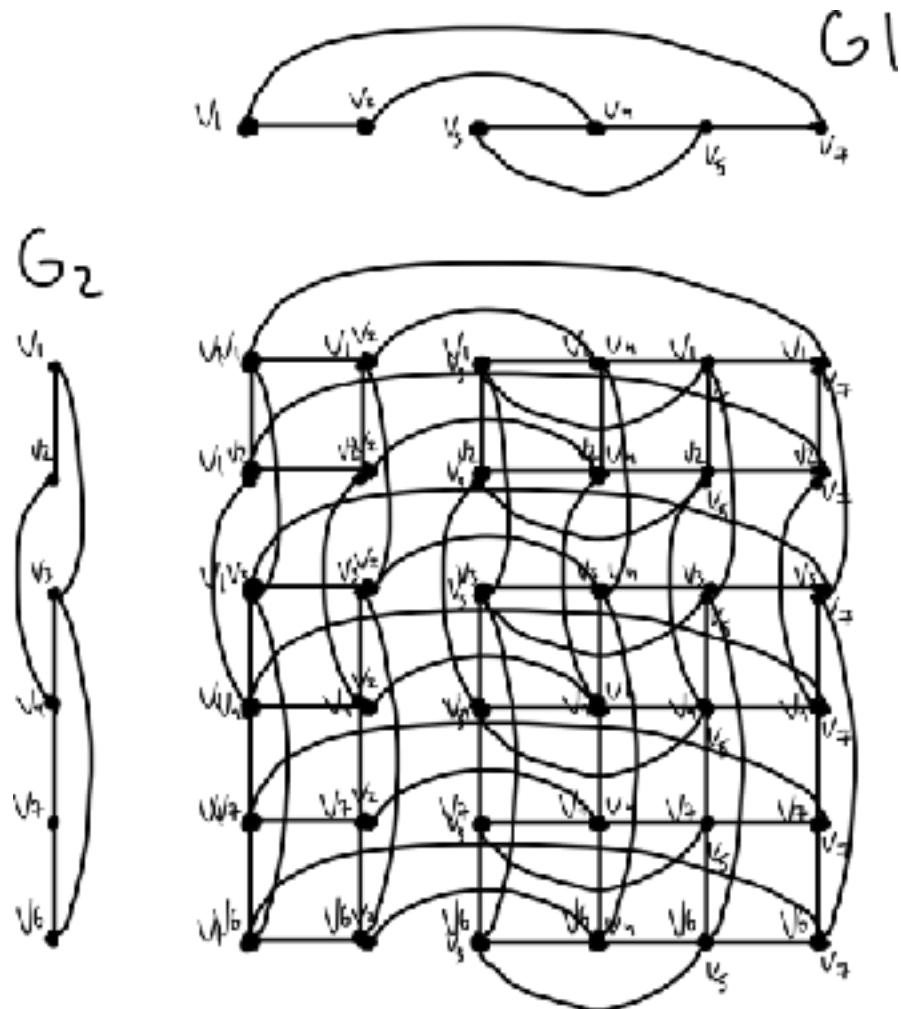
$$G = G_1 \times G_2$$

$$V(G) = V(G_1) \times V(G_2)$$

$$E(G) = E(G_1) \times E(G_2)$$

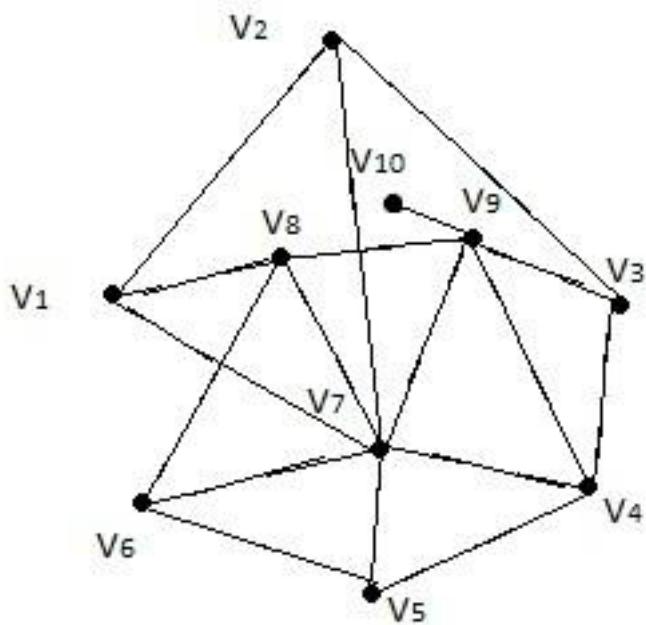
$$V(G) = \{(1, 1), (1, 2), (1, 3), (1, 4), (1, 7), (1, 6), (2, 1), (2, 2), (2, 3), (2, 4), (2, 7), (2, 6), (3, 1), (3, 2), (3, 3), (3, 4), (3, 7), (3, 6), (4, 1), (4, 2), (4, 3), (4, 4), (4, 7), (4, 6), (5, 1), (5, 2), (5, 3), (5, 4), (5, 7), (5, 6), (6, 1), (6, 2), (6, 3), (6, 4), (6, 7), (6, 6), (7, 1), (7, 2), (7, 3), (7, 4), (7, 7), (7, 6)\}$$

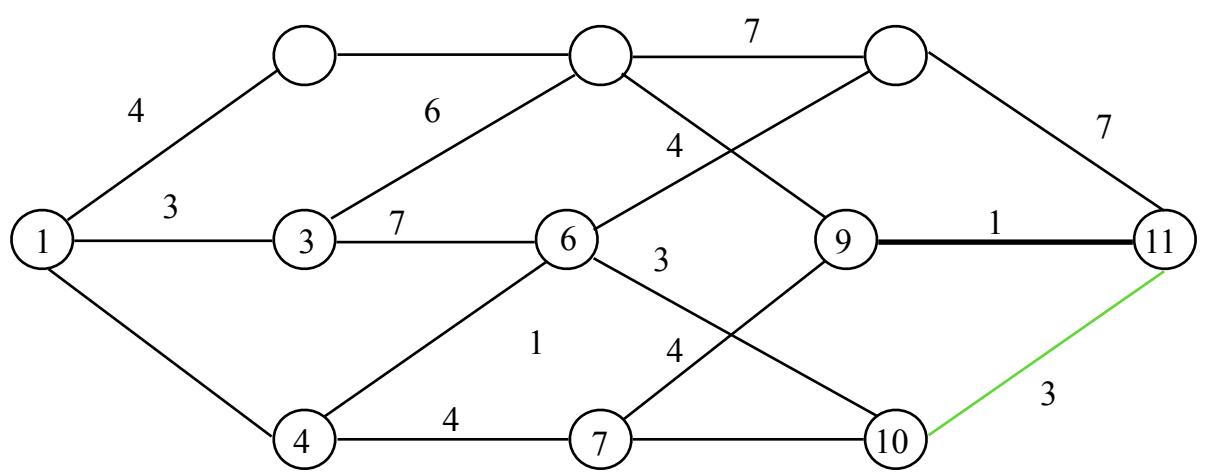
Отже, добуток графів буде виглядати так:



Завдання № 2.

Скласти таблицю суміжності для орграфа.





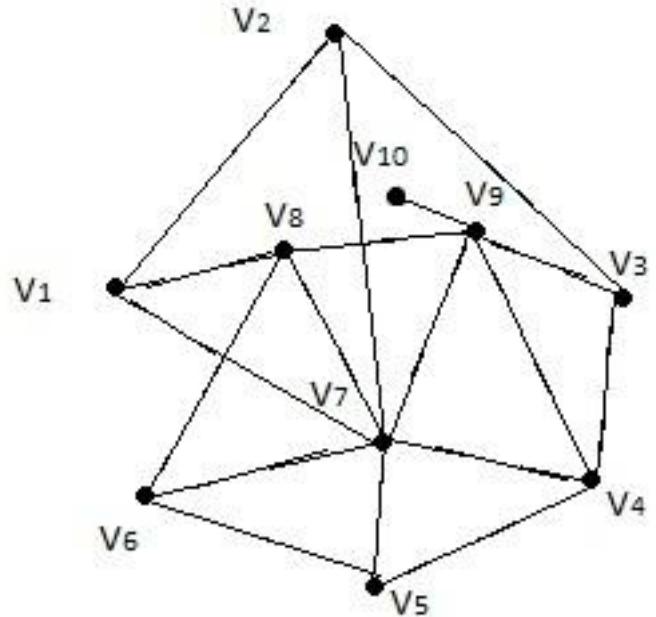
Завдання № 3.

Для графа з другого завдання знайти діаметр.

Діаметр даного графа дорівнює 3, оскільки цьому дорівнює максимальна відстань найкоротшого шлях між двома вершинами (у цьому випадку V_6 і V_{10}).

Завдання № 4.

Для графа з другого завдання виконати обхід дерева вглиб.



Завдання № 5.

Знайти двома методами (Краскала і Прима) мінімальне остеове дерево графа.



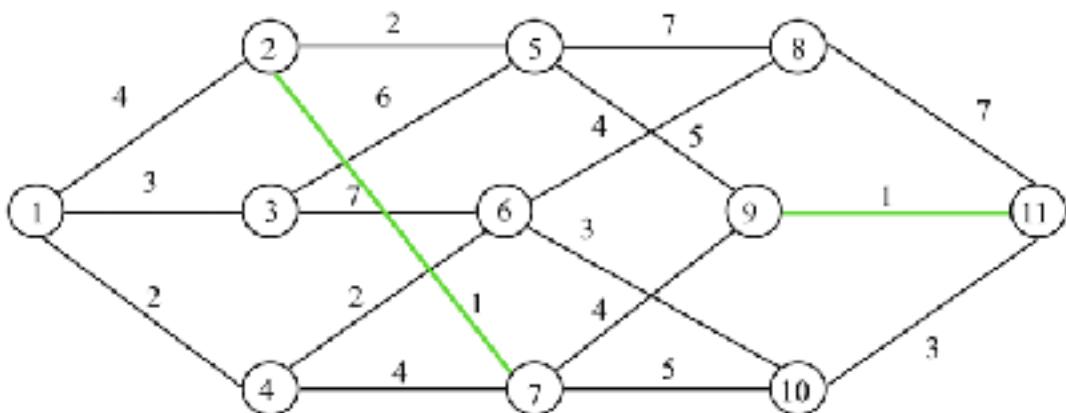
Вершина	DFS номер	Вміст стеку
1	1	1
2	2	12
3	3	123
9	4	1239
10	5	123910
-	-	1239
8	6	12398
7	7	123987
6	8	1239876
5	9	12398765
4	10	123987654
-	-	12398765
-	-	1239876
-	-	123987
-	-	12398
-	-	1239
-	-	123
-	-	12
-	-	1
-	-	\emptyset

- **Метод Краскала:**

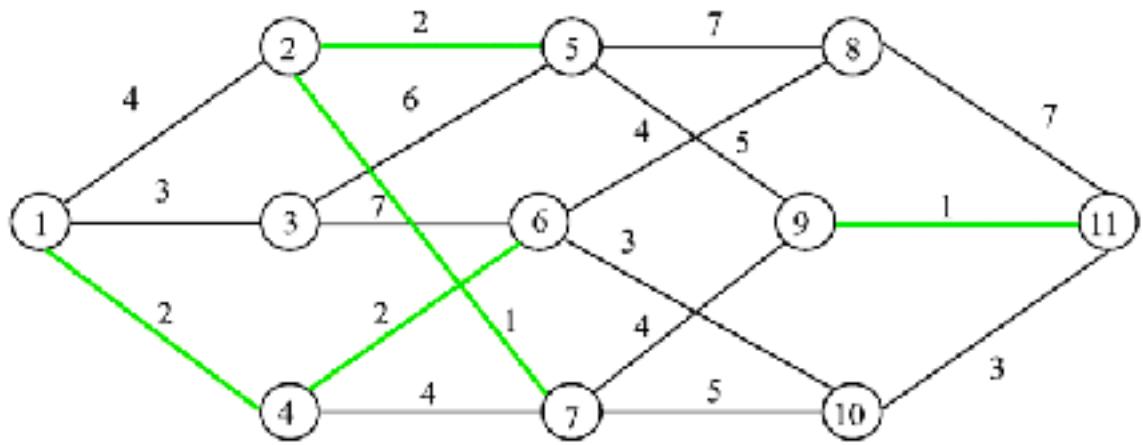
сортуємо для даного графа ребра у порядку неспадання ваг цих ребер.

- (2, 7) - 1;
- (9, 11) - 1;
- (1, 4) - 2;
- (2, 5) - 2;
- (4, 6) - 2;
- (1, 3) - 3;
- (6, 10) - 3;
- (10, 11) - 3;
- (1, 2) - 4;
- (4, 7) - 4;
- (6, 8) - 4;
- (7, 9) - 4;
- (5, 9) - 5;
- (7, 10) - 5;
- (3, 5) - 6;
- (5, 8) - 7;
- (8, 11) - 7;

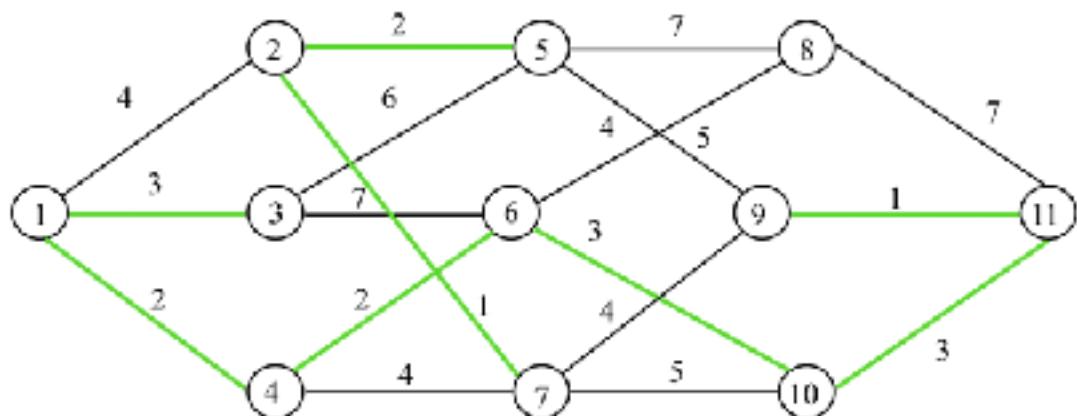
вибираємо ребро графа з найменшою вагою і додаємо до дерева. В даному випадку це ребра (2, 7) і (9, 11) з вагою 1.



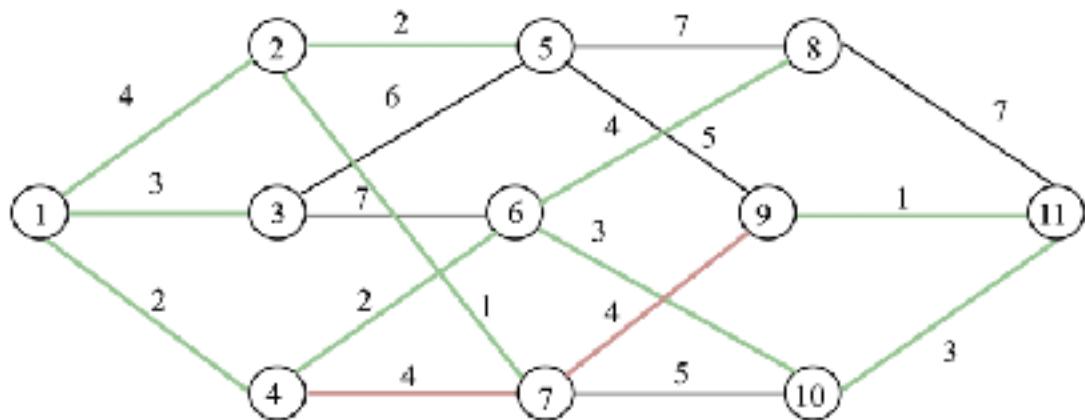
беремо ребро з наступною найменшою вагою. Це ребра $(4, 6)$, $(1, 4)$, $(2, 5)$ з вагою 2.
Додаємо до дерева.



беремо ребра з наступною найменшою вагою - $(1, 3)$, $(6, 10)$, $(10, 11)$ з вагою 3.
Додаємо до кістякового дерева.

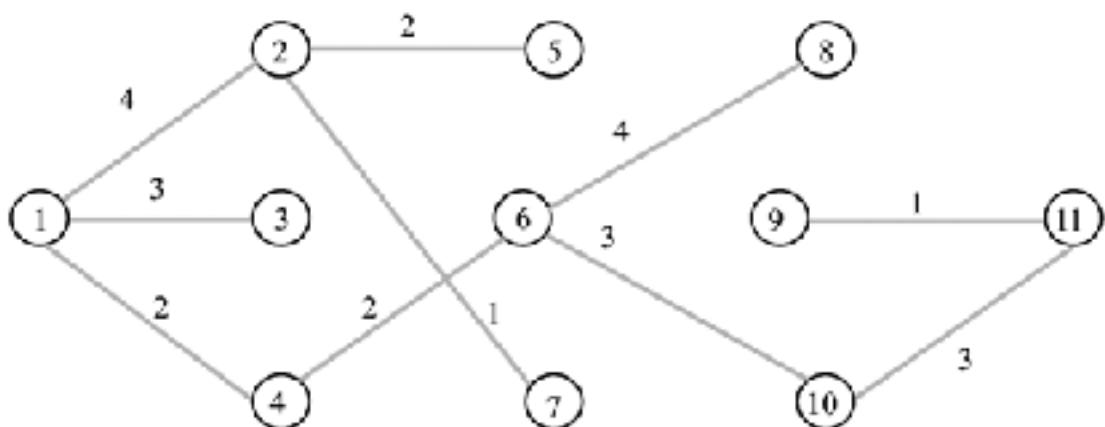


беремо ребра з наступною найменшою вагою - $(1, 2), (4, 7), (6, 8), (7, 9)$ з вагою 4. Відкидаємо ребра $(4, 7)$ і $(7, 9)$, оскільки вони утворяють цикл.

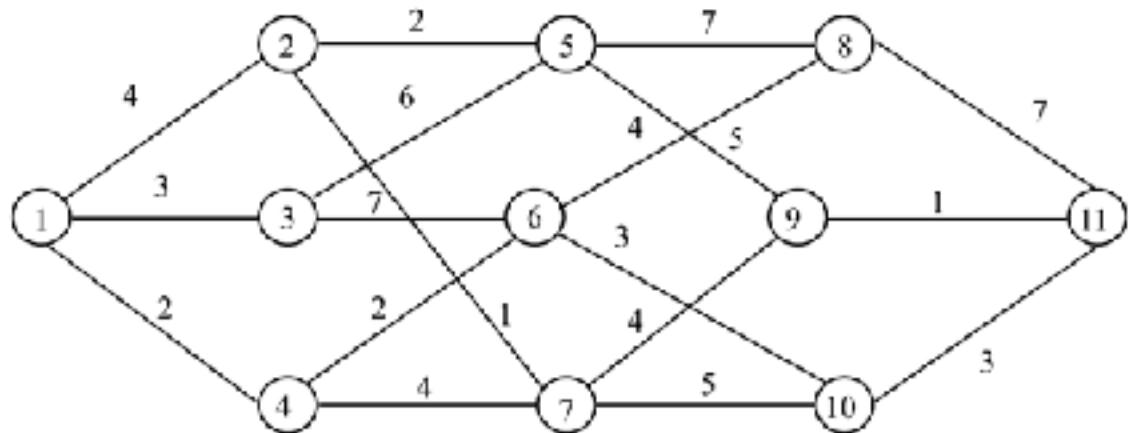


продовжуємо виконувати алгоритм за аналогією і отримуємо дерево після того як перевірили всі ребра із посортированого списку або ж всі вершини стали з'єднаними

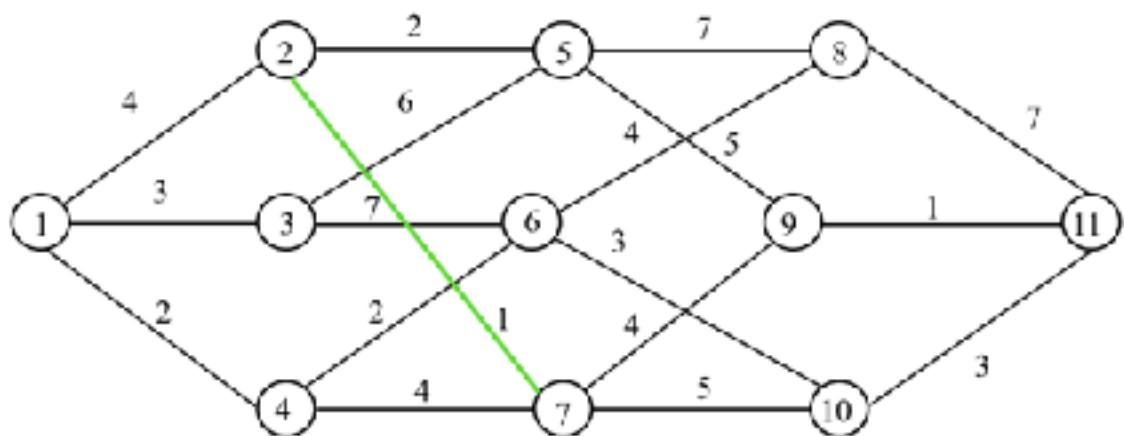
мінімальне остове дерево за алгоритмом Краскала буде виглядати так:



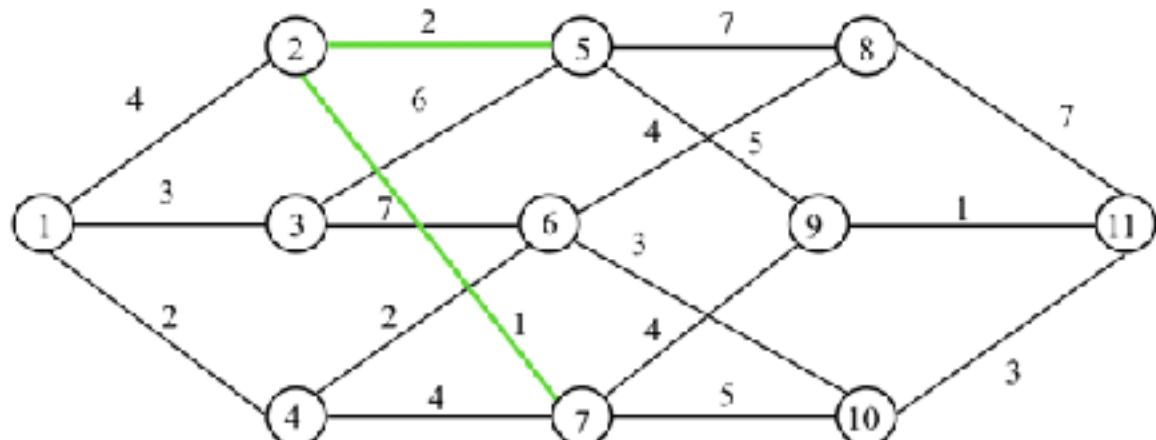
- **Метод Прима:**



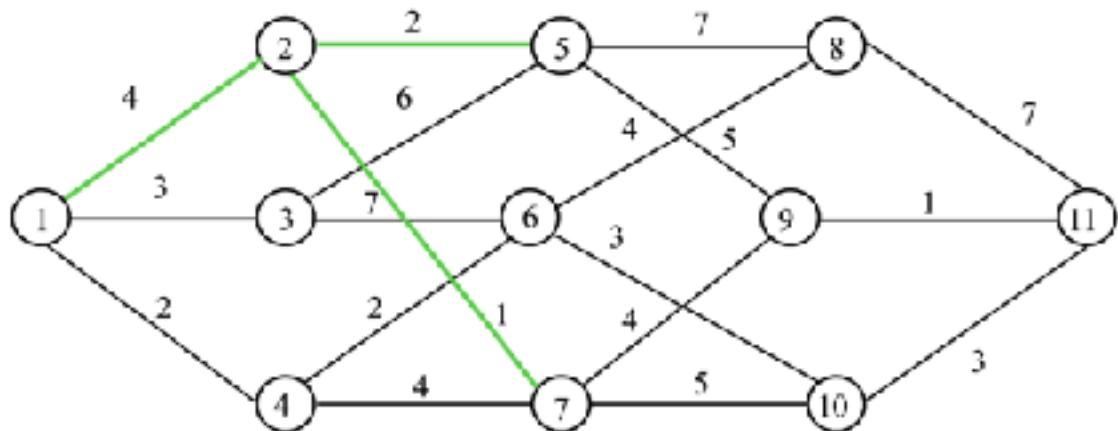
спочатку фіксуємо довільну вершину, з якої розпочнемо пошук мінімального дерева. У даному випадку це вершина 2. Знаходимо ребро, інцидентне цій вершині з найменшою вагою і це $(2, 7)$ з вагою 1. Додаємо до кістякового дерева ребро $(2, 7)$.



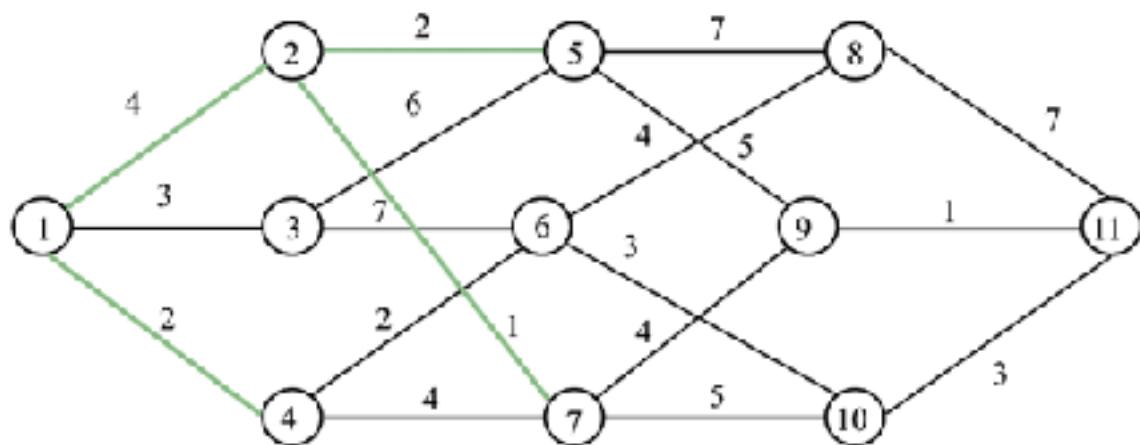
розглядаємо ребра, інцидентні вершинам 2 і 7, і вибираємо те, яке має найменшу вагу. І це $(2, 5)$ з вагою 2. Додаємо його до кістякового дерева.



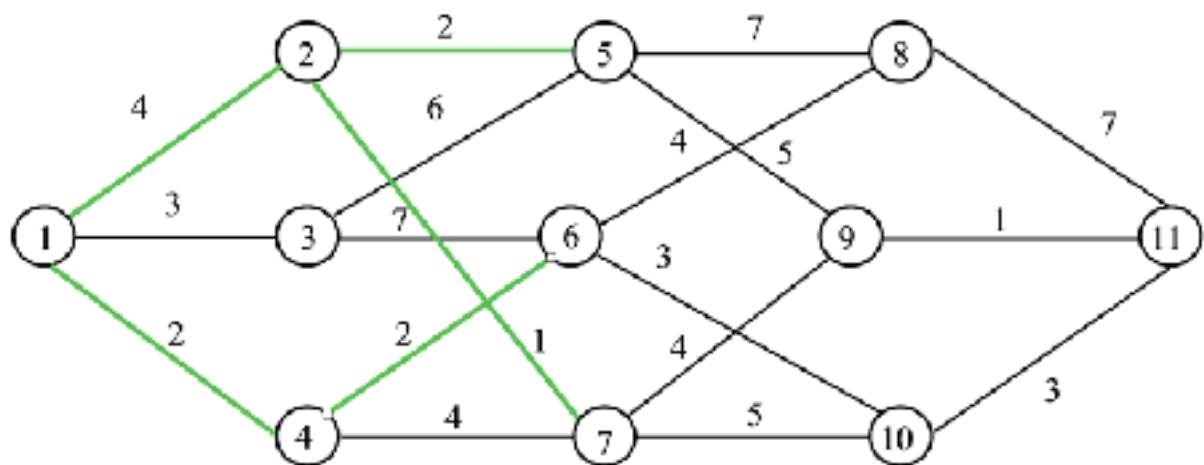
розглядаємо ребра, інцидентні вершинам 2, 7, 5 і вибираємо з найменшою вагою - і це (1, 2) з вагою 4. Додаємо до дерева.



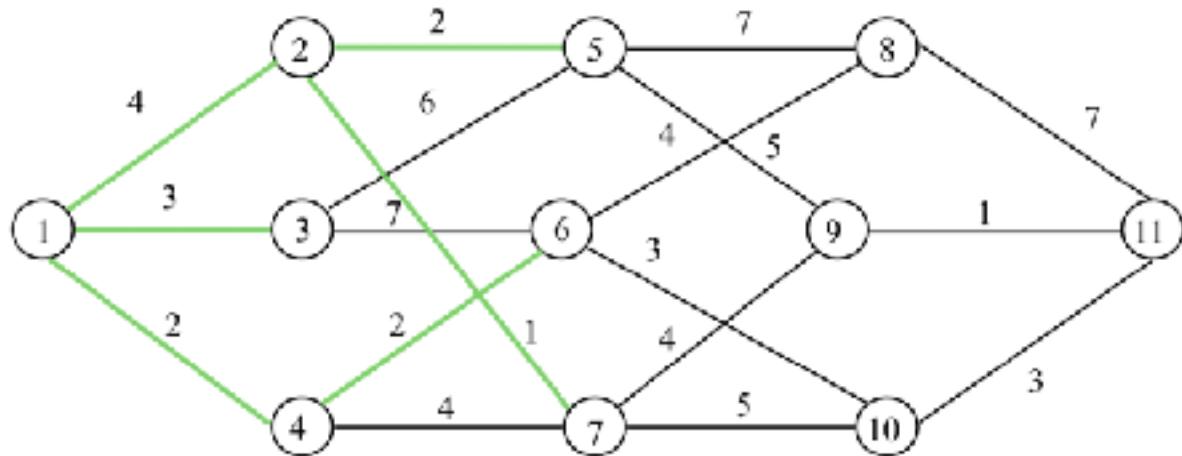
розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1 і вибираємо з найменшою вагою - і це (1, 4) з вагою 2. Додаємо до дерева.



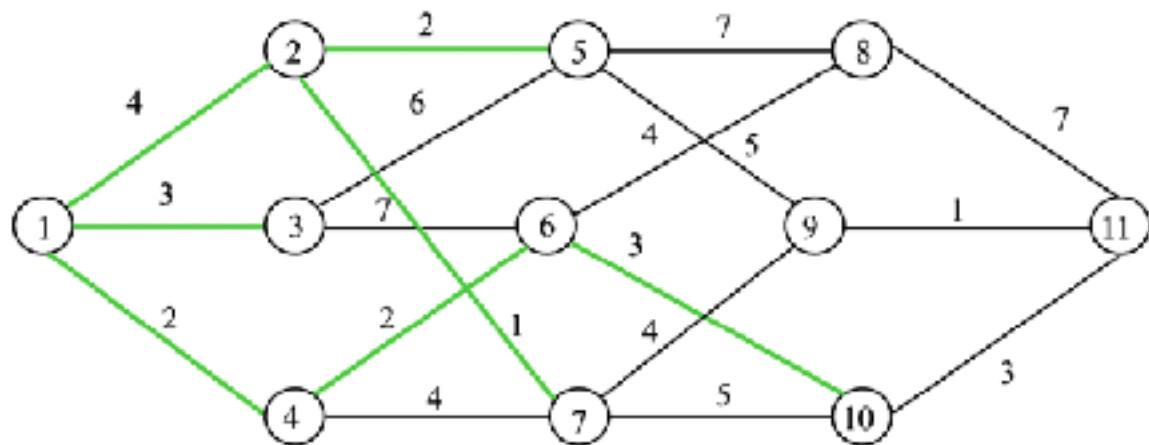
розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4 і вибираємо з найменшою вагою - і це (4, 6) з вагою 2. Додаємо до кістякового дерева.



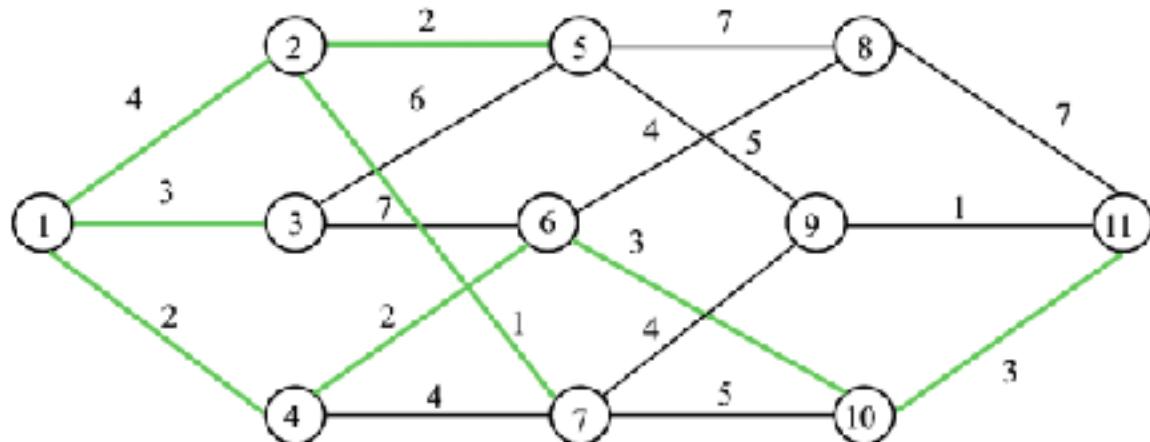
Етап 6: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6 і вибираємо з найменшою вагою - і це (1, 3) з вагою 3. Додаємо до кістякового дерева.

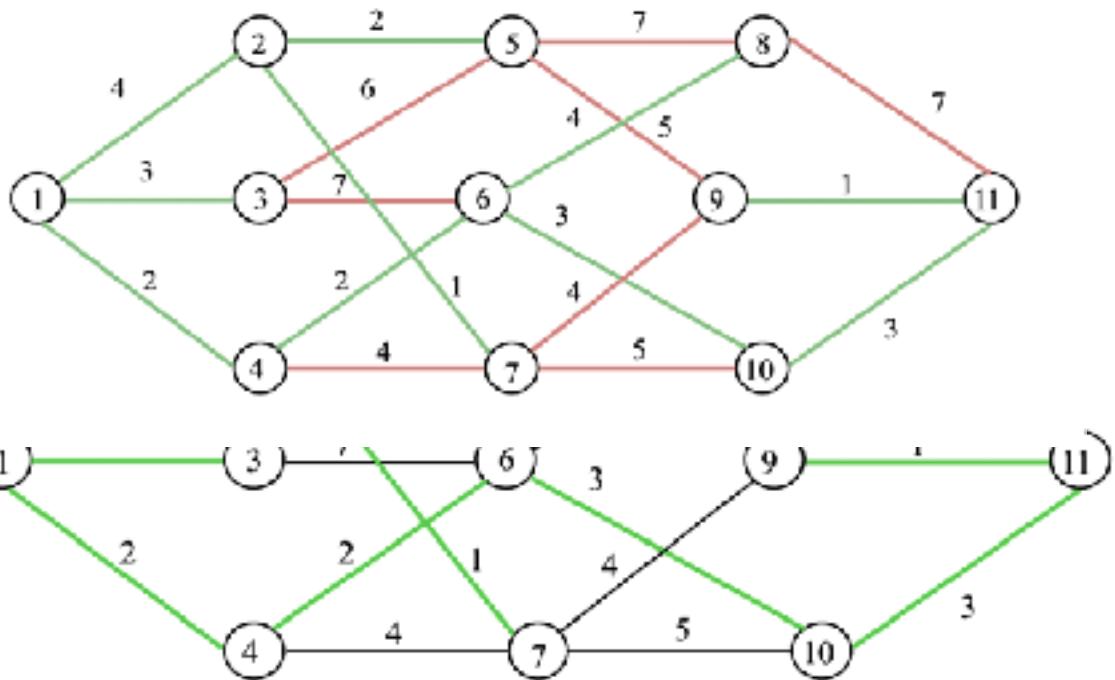


Етап 7: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3 і вибираємо з найменшою вагою - і це (6, 10) з вагою 3. Додаємо до кістякового дерева.



Етап 8: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3, 10 і вибираємо з найменшою вагою - і це (10, 11) з вагою 3. Додаємо до кістякового дерева.

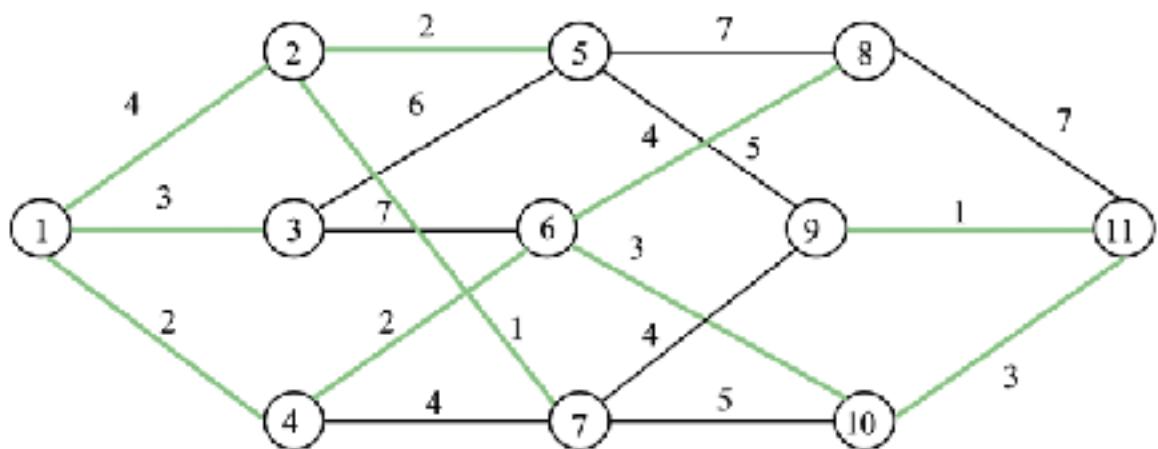




Етап
9:

розділяємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3, 10, 11 і вибираємо з найменшою вагою - і це (9, 11) з вагою 1. Додаємо до дерева.

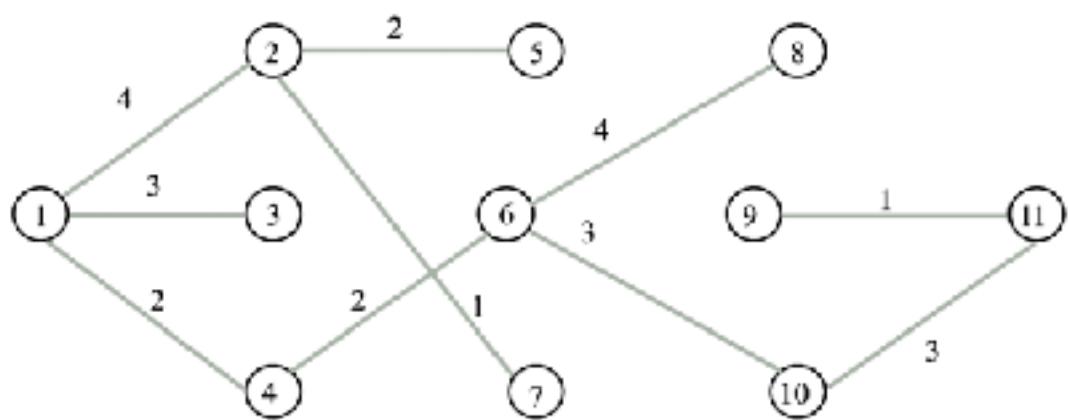
Етап 10: розглядаємо ребра, інцидентні вершинам 2, 7, 5, 1, 4, 6, 3, 10, 11 і вибираємо



з найменшою вагою - і це (6, 8) з вагою 4. Додаємо до дерева.

бачимо що всі вершини з'єднані і дерево утворене.

Отже, мінімальне остеове дерево за алгоритмом Прима виглядає так:



Завдання № 6.

Розв'язати задачу комівояжера для повного 8-ми вершин-ного графа методом «іди у найближчий», матриця вагів якого має вигляд:

19)

	1	2	3	4	5	6	7	8
1	∞	2	2	2	2	3	2	2
2	2	∞	5	1	2	3	2	4
3	2	5	∞	6	6	5	1	5
4	2	1	6	∞	6	6	6	6
5	2	2	6	6	∞	5	1	5
6	3	3	5	6	5	∞	2	1
7	2	2	1	6	1	2	∞	5
8	2	4	5	6	5	1	5	∞

вибираємо з першого рядка мінімальне значення 2. Викреслюємо 1 рядок, 1 стовпець.

	2	3	4	5	6	7	8
2	∞	5	1	2	3	2	4
3	5	∞	6	6	5	1	5
4	1	6	∞	6	6	6	6
5	2	6	6	∞	5	1	5
6	3	5	6	5	∞	2	1
7	2	1	6	1	2	∞	5
8	4	5	6	5	1	5	∞

вибираємо з восьмого рядка мінімальне значення 1. Викреслюємо 8 рядок, 8 стовпець.

	2	3	4	5	6	7
2	∞	5	1	2	3	2
3	5	∞	6	6	5	1
4	1	6	∞	6	6	6
5	2	6	6	∞	5	1
6	3	5	6	5	∞	2
7	2	1	6	1	2	∞

вибираємо з 6 рядка значення 5. Викреслюємо 6 рядок, 6 стовпець.

	2	3	4	5	7
2	∞	5	1	2	2
3	5	∞	6	6	1
4	1	6	∞	6	6
5	2	6	6	∞	1
7	2	1	6	1	∞

вибираємо з 3 рядка значення 1. Викреслюємо 3 рядок, 3 стовпець.

	2	4	5	7
2	∞	1	2	2
4	1	∞	6	6
5	2	6	∞	1
7	2	6	1	∞

вибираємо з 7 рядка значення 1. Викреслюємо 7 рядок, 7 стовпець.

	2	4	5
2	∞	1	2
4	1	∞	6
5	2	6	∞

вибираємо з 5 рядка значення 2. Викреслюємо 5 рядок, 5 стовпець.

	2	4
2	∞	1
4	1	∞

$1 \rightarrow 8 \rightarrow 6 \rightarrow 3 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 1$

$$2 + 1 + 5 + 1 + 1 + 2 + 1 + 2 = 15$$

Це найкоротший маршрут між вершинами та мінімальна вага.

Проходимо через ще один цикл:

з першого рядка вибираємо найменше значення 2. Викреслюємо 1 рядок, 1 стовпець.

	2	3	4	5	6	7	8
2	∞	5	1	2	3	2	4
3	5	∞	6	6	5	1	5
4	1	6	∞	6	6	6	6
5	2	6	6	∞	5	1	5
6	3	5	6	5	∞	2	1
7	2	1	6	1	2	∞	5
8	4	5	6	5	1	5	∞

з 8 рядка вибираємо значення 1. Викреслюємо 8 рядок, 8 стовпець.

	2	3	4	5	6	7
2	∞	5	1	2	3	2
3	5	∞	6	6	5	1
4	1	6	∞	6	6	6
5	2	6	6	∞	5	1
6	3	5	6	5	∞	2
7	2	1	6	1	2	∞

з 6 рядка вибираємо 2. Викреслюємо 6 рядок, 6 стовпець.

	2	3	4	5	7
2	∞	5	1	2	2
3	5	∞	6	6	1
4	1	6	∞	6	6
5	2	6	6	∞	1
7	2	1	6	1	∞

з 7 рядка вибираємо 1. Викреслюємо 7 рядок, 7 стовпець.

	2	3	4	5
2	∞	5	1	2
3	5	∞	6	6
4	1	6	∞	6
5	2	6	6	∞

з 5 рядка вибираємо 2. Викреслюємо 5 рядок, 5 стовпець.

	2	3	4
2	∞	5	1
3	5	∞	6
4	1	6	∞

з 2 рядка вибираємо 1. Викреслюємо 2 рядок, 2 стовпець.

	3	4
3	∞	6
4	6	∞

$$1 \rightarrow 8 \rightarrow 6 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$$

$$2 + 1 + 2 + 1 + 2 + 1 + 6 + 2 = 17$$

за аналогією знаходимо такі цикли:

$$1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 8 \rightarrow 7 \rightarrow 5 \rightarrow 1 \text{ (18)}$$

$$1 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 6 \rightarrow 3 \rightarrow 1 \text{ (20)}$$

$$3 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 8 \rightarrow 6 \rightarrow 3 \text{ (15)}$$

$$1 \rightarrow 5 \rightarrow 7 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 2 \rightarrow 4 \rightarrow 1 \text{ (17)}$$

$$1 \rightarrow 7 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 1 \text{ (22)}$$

$$2 \rightarrow 4 \rightarrow 1 \rightarrow 3 \rightarrow 7 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 2 \text{ (17)}$$

$$2 \rightarrow 4 \rightarrow 1 \rightarrow 3 \rightarrow 7 \rightarrow 5 \rightarrow 8 \rightarrow 6 \rightarrow 2 \text{ (16)}$$

$$8 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 8 \text{ (16)}$$

$$8 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 8 \text{ (17)}$$

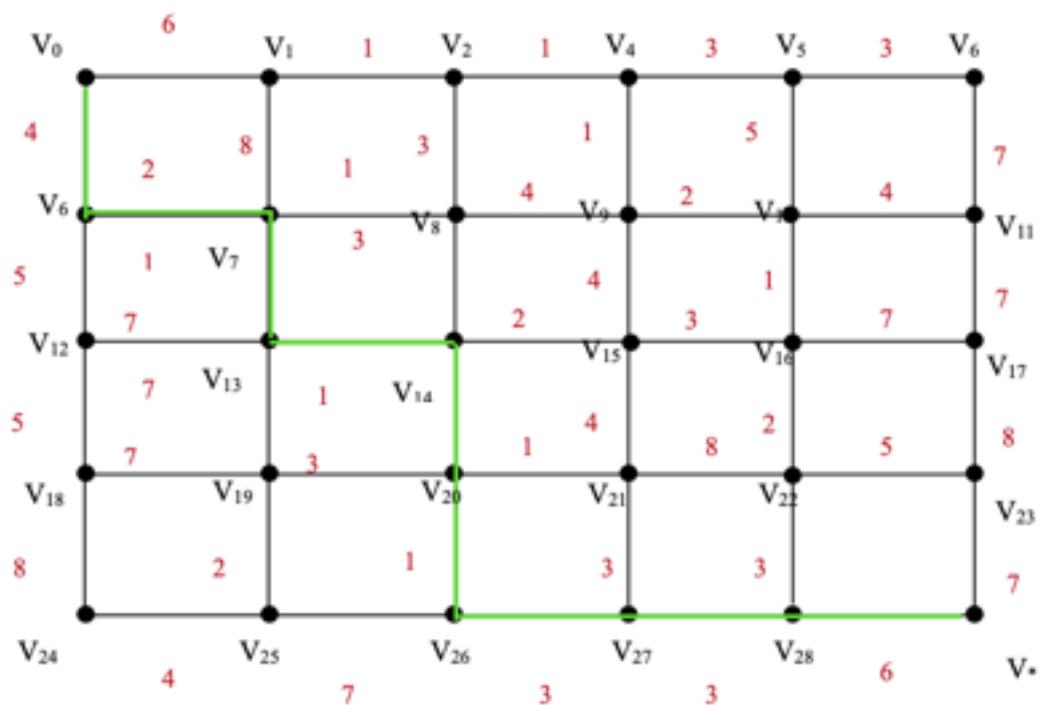
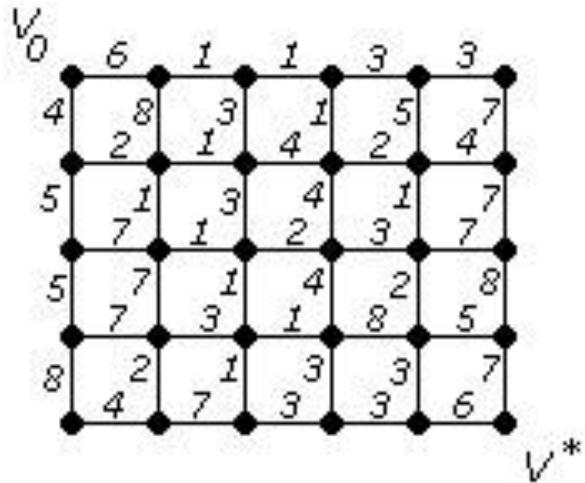
$$6 \rightarrow 8 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 3 \rightarrow 6 \text{ (15)}$$

$$7 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 8 \rightarrow 6 \rightarrow 3 \rightarrow 7 \text{ (15)}$$

Отже, вага найкоротшого маршруту дорівнює 15.

Завдання №7.

За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V_0 і V^* .



Виконання:

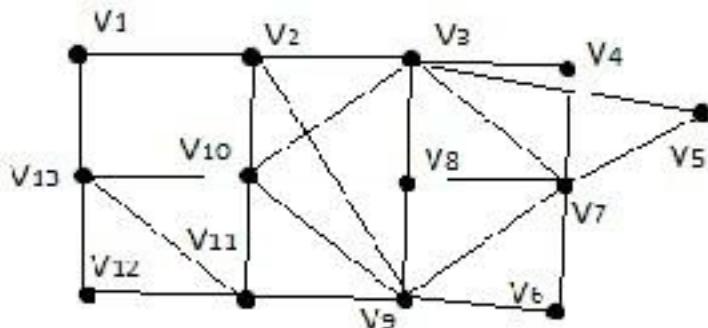
- 1) Розпочинаємо з початкової вершини, яку будемо вважати поточною. В даний момент це вершина V_0 . Переглядаємо її шлях до суміжних вершин - V_1 і V_6 . До першої вершини відстань дорівнює 6, до шостої - 4.
- 2) Проходимо через $V_0 \rightarrow V_1 \rightarrow V_2$, відстань дорівнює 7.
- 3) Проходимо через $V_1 \rightarrow V_2 \rightarrow V_3$, відстань дорівнює 7
- 4) Проходимо через $V_2 \rightarrow V_3 \rightarrow V_4$, відстань дорівнює 8.
- 5) Проходимо через $V_3 \rightarrow V_4 \rightarrow V_5$, відстань дорівнює 11.
- 6) Проходимо через $V_4 \rightarrow V_5 \rightarrow V_{11}$, відстань дорівнює 14.
- 7) Проходимо через $V_1 \rightarrow V_7$, відстань дорівнює 14.
- 8) Проходимо через $V_2 \rightarrow V_8$, відстань дорівнює 10.
- 9) Проходимо через $V_3 \rightarrow V_9$, відстань дорівнює 9.
- 10) Проходимо через $V_4 \rightarrow V_{10}$, відстань дорівнює 16.
- 11) Проходимо через $V_6 \rightarrow V_7$, відстань дорівнює 6, що є меншою за минулу відстань від вершини V_1 , то викреслюєм минулу і робимо отриману відстань поточною.
- 12) Проходимо через $V_7 \rightarrow V_8$, відстань дорівнює 7, що є меншою за минулу відстань від вершини V_2 , то викреслюєм минулу і робимо отриману відстань поточною.
- 13) Проходимо через $V_8 \rightarrow V_9$, відстань дорівнює 11, тому залишаємо минулу відстань без змін.
- 14) Проходимо через $V_9 \rightarrow V_{10}$, відстань дорівнює 11, що є меншою за минулу відстань від вершини V_4 , то викреслюєм минулу і робимо отриману відстань поточною.
- 15) Проходимо через $V_{10} \rightarrow V_{11}$, відстань дорівнює 15, що є меншою за минулу відстань від вершини V_5 , то викреслюєм минулу і робимо отриману відстань поточною.
- 16) Проходимо через $V_{12} \rightarrow V_{13}$, відстань дорівнює 16.
- 17) Проходимо через $V_{13} \rightarrow V_{14}$, відстань дорівнює 17, тому залишаємо минулу відстань без змін.
- 18) Проходимо через $V_{14} \rightarrow V_{15}$, відстань дорівнює 15, тому залишаємо минулу відстань без змін.
- 19) Проходимо через $V_{15} \rightarrow V_{16}$, відстань дорівнює 16, що є меншою за минулу відстань від вершини V_{10} , то викреслюєм минулу і робимо отриману відстань поточною.
- 20) Проходимо через $V_{16} \rightarrow V_{17}$, відстань дорівнює 23, тому залишаємо минулу відстань без змін.
- 21) Проходимо через $V_{18} \rightarrow V_{19}$, відстань дорівнює 21, тому залишаємо минулу відстань без змін.
- 22) Проходимо через $V_{19} \rightarrow V_{20}$, відстань дорівнює 17, що є меншою за минулу відстань від вершини V_{13} , то викреслюєм минулу і робимо отриману відстань поточною.
- 23) Проходимо через $V_{20} \rightarrow V_{21}$, відстань дорівнює 15, що є меншою за минулу відстань від вершини V_{15} , то викреслюєм минулу і робимо отриману відстань поточною.

- 24) Проходимо через $V_{21} \rightarrow V_{22}$, відстань дорівнює 23, тому залишаємо минулу відстань без змін.
- 25) Проходимо через $V_{22} \rightarrow V_{23}$, відстань дорівнює 24, тому залишаємо минулу відстань без змін.
- 26) Проходимо через $V_{24} \rightarrow V_{25}$, відстань дорівнює 26, тому залишаємо минулу відстань без змін.
- 27) Проходимо через $V_{25} \rightarrow V_{26}$, відстань дорівнює 32, тому залишаємо минулу відстань без змін.
- 28) Проходимо через $V_{26} \rightarrow V_{27}$ відстань дорівнює 18, що є меншою за минулу відстань від вершини V_{21} , то викреслюєм минулу і робимо отриману відстань поточною.
- 29) Проходимо через $V_{27} \rightarrow V_{28}$ відстань дорівнює 21, що є меншою за минулу відстань від вершини V_{22} , то викреслюєм минулу і робимо отриману відстань поточною.
- 30) Проходимо через $V_{28} \rightarrow V^*$ відстань дорівнює 22.

Отже, у даному графі за алгоритмом Дейкстри існує один мінімальний шлях від вершини V_0 до V^* , який дорівнює 22.

Завдання №8.

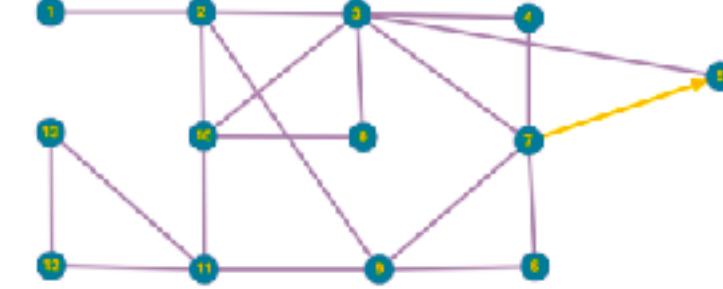
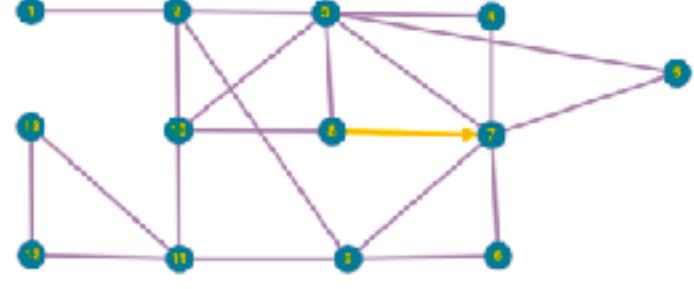
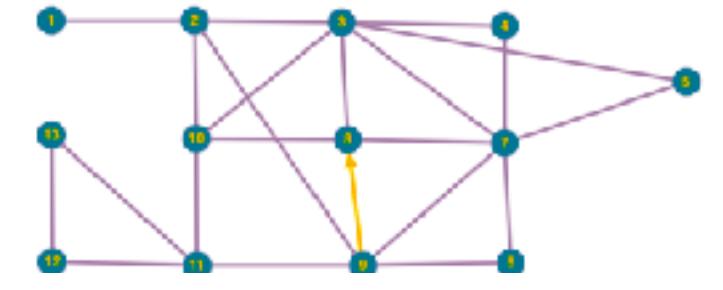
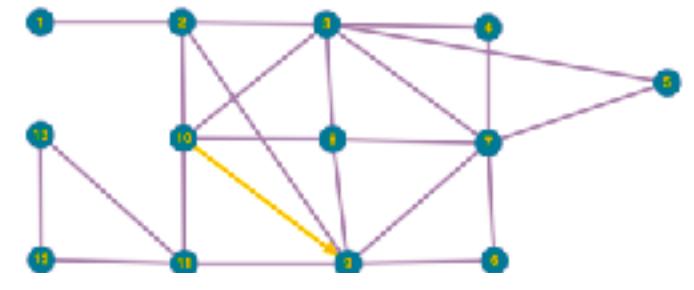
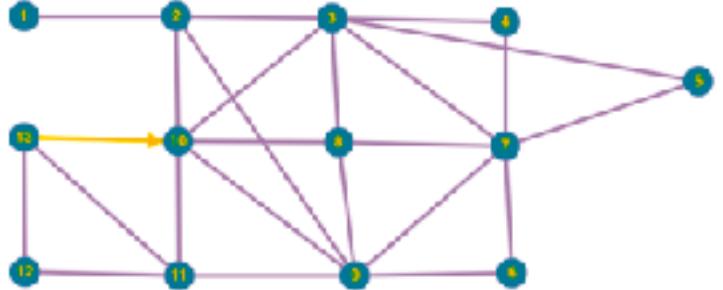
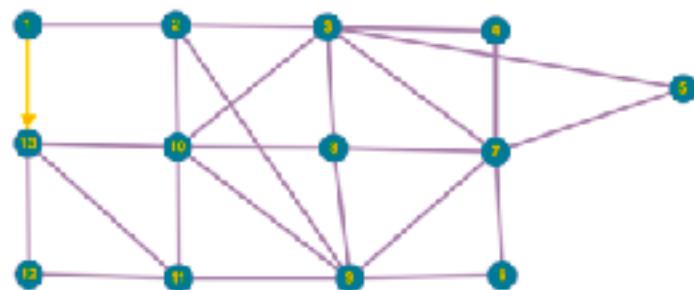
Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері; б) елементарних циклів.

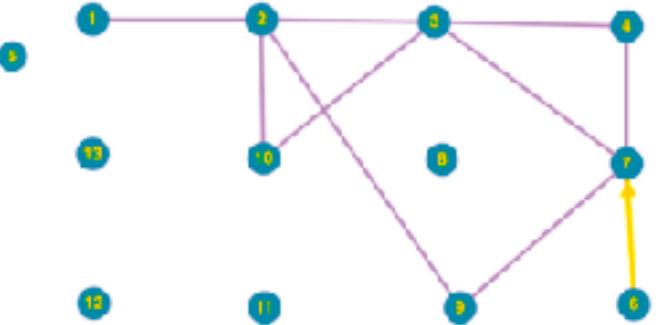
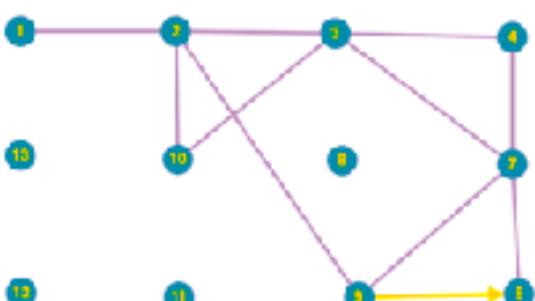
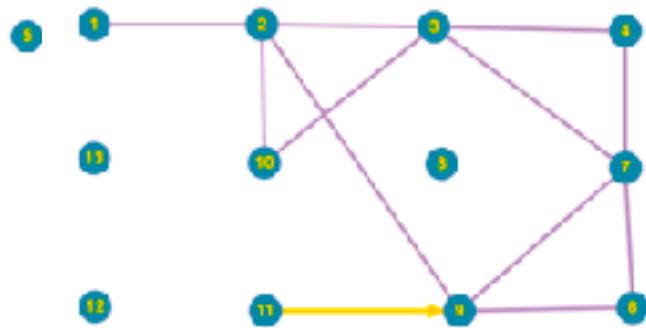
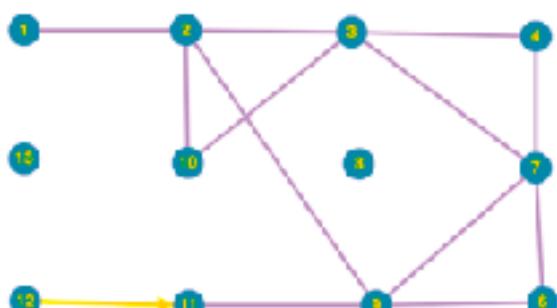
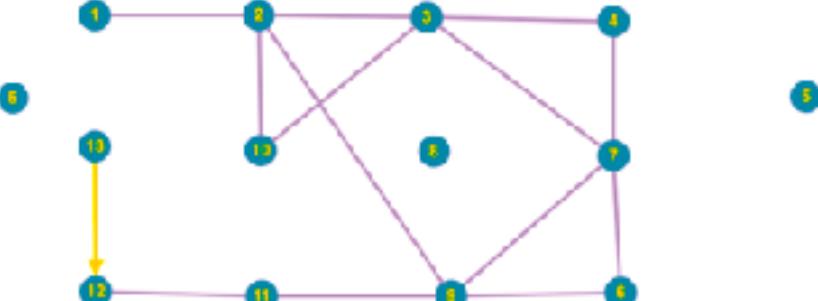
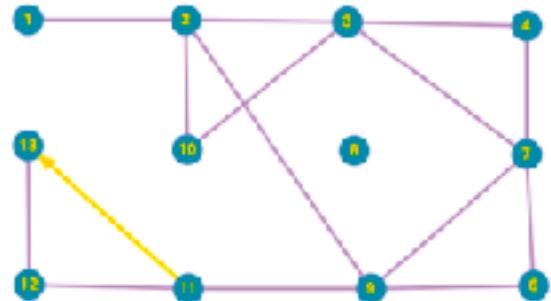
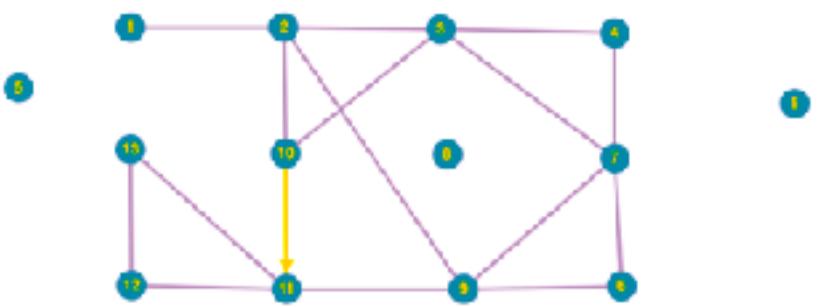
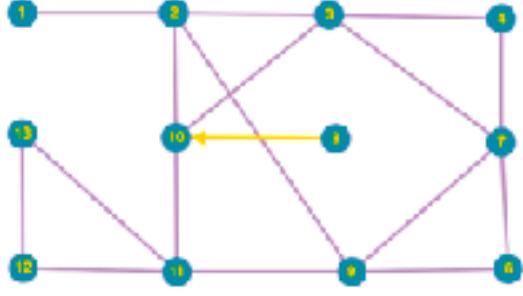


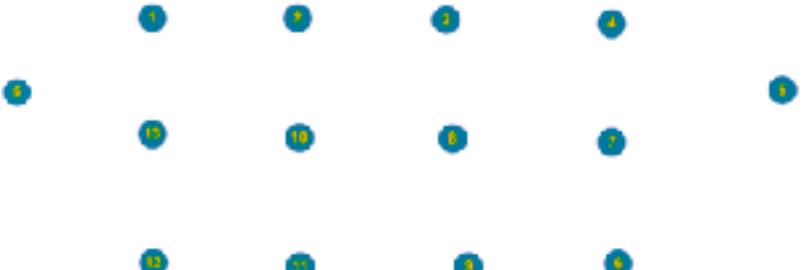
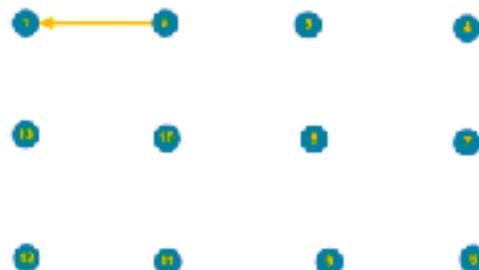
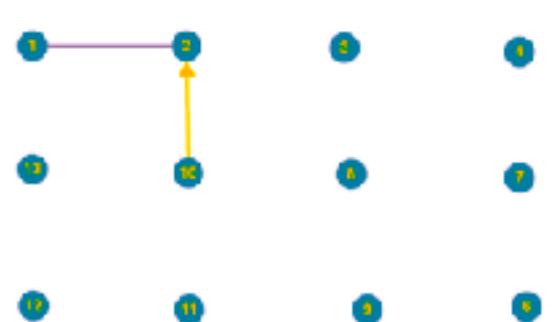
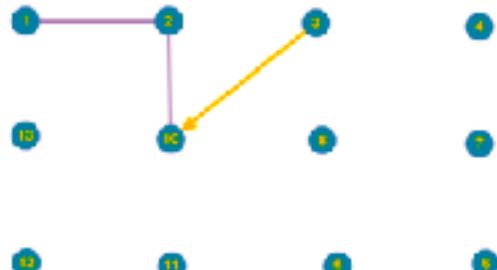
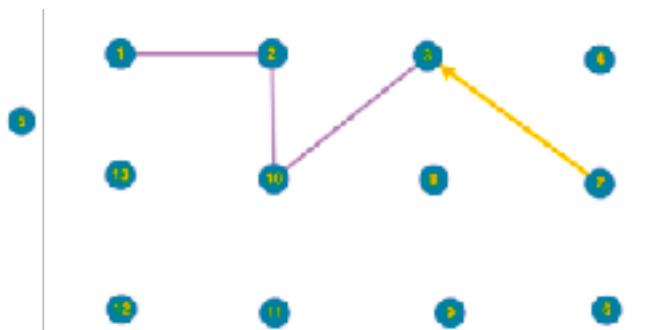
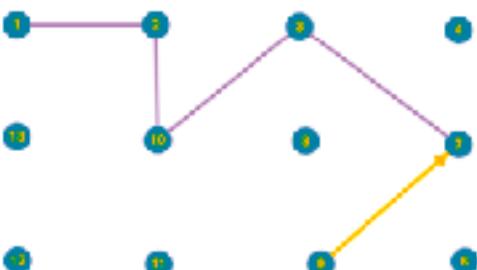
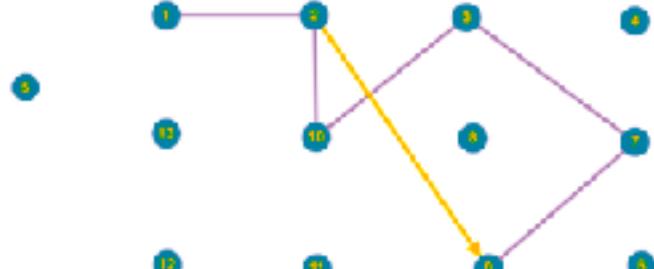
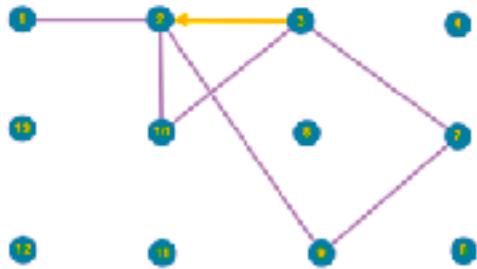
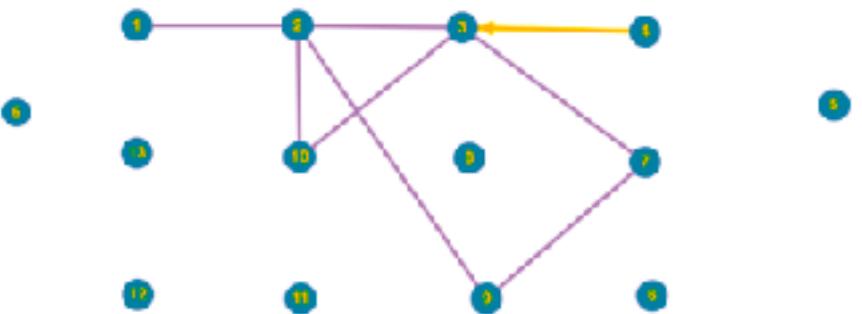
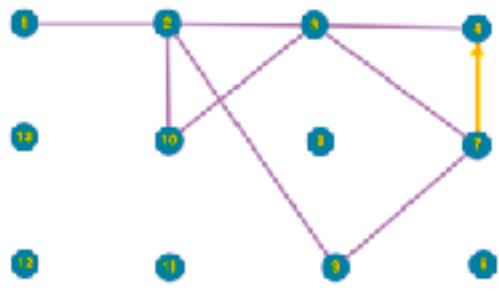
а) метод Флері:

Алгоритм Флері полягає у тому, щоб починаючи з деякої вершини, викреслювати кожне проайдене ребро. Не проходимо по ребру, якщо видалення цього ребра призводить до розбиття графа на дві зв'язні компоненти, тобто, необхідно перевіряти, чи є ребро мостом чи ні.

реф. Не проходимо по ребру, якщо видалення цього ребра призводить до розбиття графа на дві зв'язні компоненти, тобто, необхідно перевіряти, чи є ребро мостом чи ні.







Отже, ейлеровий цикл даного графа: $1 \rightarrow 13 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 7 \rightarrow 5 \rightarrow 3 \rightarrow 8 \rightarrow 10 \rightarrow 11 \rightarrow 13 \rightarrow 12 \rightarrow 11 \rightarrow 9 \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 9 \rightarrow 7 \rightarrow 2 \rightarrow 10 \rightarrow 2 \rightarrow 1$.

б) метод елементарних циклів:

Виділимо кольоровим елементарні цикли та об'єднаємо їх у один:

$V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_7 \rightarrow V_6 \rightarrow V_9 \rightarrow V_{11} \rightarrow V_{12} \rightarrow V_{13}$

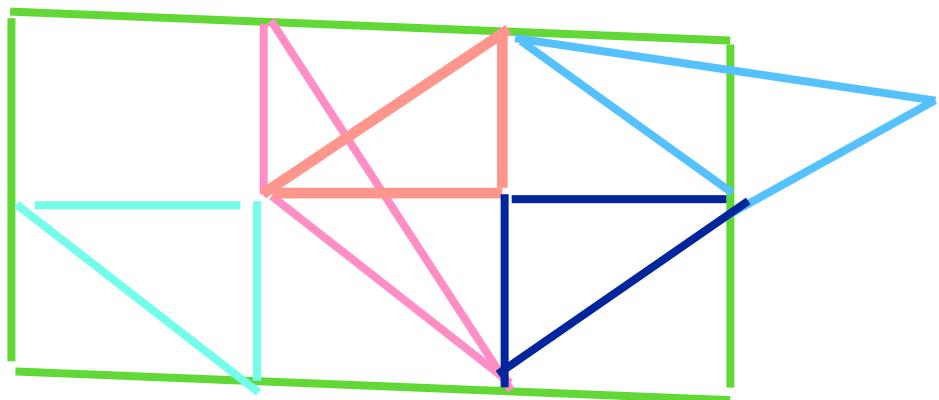
$V_2 \rightarrow V_9 \rightarrow V_{10}$

$V_3 \rightarrow V_5 \rightarrow V_7$

$V_3 \rightarrow V_8 \rightarrow V_{10}$

$V_7 \rightarrow V_8 \rightarrow V_9$

$V_{13} \rightarrow V_{11} \rightarrow V_{10}$



Завдання №9.

Спростити формули (привести їх до скороченої ДНФ).

$$\overline{\overline{x}y(\bar{x}\bar{y}z \vee \bar{x}y)}$$

спочатку запишем таблицю істинності для даної формули:

x	y	z	$\neg(\neg(x \wedge y) \wedge (x \wedge \neg y \wedge z \vee \neg x \wedge y))$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	1	1

запишемо кон'юнкцію змінних за яких формула дає 1:

$$\neg x \neg y \neg z$$

$$\neg x \neg y z$$

$$x \neg y \neg z$$

$$xy \neg z$$

$$xyz$$

об'єднаємо диз'юнкцію і отримаємо ДДНФ:

$$\neg x \neg y \neg z \vee \neg x \neg y z \vee x \neg y \neg z \vee xy \neg z \vee xyz$$

після перетворень отримаємо скорочену ДНФ:

$$\neg x \neg y \vee x \neg z \vee xy$$

ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ

1. Обхід вглиб і вшир

```
1 // class Node {
2 //     int value;
3 //     Node left;
4 //     Node right;
5 // }
6
7 class Solution {
8
9     public int heightBalanced (Node root) {
10
11         if (root == null) {
12             return 0;
13         }
14
15         int left = heightBalanced (root.left);
16         int right = heightBalanced (root.right);
17
18         if (Math.abs (left - right) >= 2) {
19             return -1;
20         }
21
22         return Math.max (left, right) + 1;
23     }
24
25     public boolean isBalanced (Node root) {
26
27         if (root == null) {
28             return true;
29         }
30
31         int left = heightBalanced (root.left);
32         int right = heightBalanced (root.right);
33
34         if (Math.abs (left - right) >= 2) {
35             return false;
36         }
37
38         if (left < 0 || right < 0) {
39             return false;
40         }
41
42         return true;
43     }
44
45     public void print (Node root) {
46
47         if (root == null) {
48             return;
49         }
50
51         System.out.print (root.value + " ");
52         print (root.left);
53         print (root.right);
54     }
55 }
```

```
117
118         if (l < r) {
119             count++;
120         }
121     }
122
123     return count;
124 }
125
126
127     for (int i = 0; i < n; i++) {
128
129         if (vec[i].count == 0) {
130             vec[i].value = min[i];
131             min[i] = min[i] + 1;
132         }
133
134         else {
135             vec[i].value = max[i];
136             max[i] = max[i] + 1;
137         }
138
139         count++;
140     }
141
142     System.out.println ("Count = " + count);
143
144     for (int i = 0; i < n; i++) {
145
146         if (vec[i].value == min[i]) {
147             min[i] = min[i] + 1;
148         }
149
150         else if (vec[i].value == max[i]) {
151             max[i] = max[i] + 1;
152         }
153
154         count++;
155     }
156
157     System.out.println ("Count = " + count);
158
159     for (int i = 0; i < n; i++) {
160
161         if (vec[i].value == min[i]) {
162             min[i] = min[i] + 1;
163         }
164
165         else if (vec[i].value == max[i]) {
166             max[i] = max[i] + 1;
167         }
168
169         count++;
170     }
171
172     System.out.println ("Count = " + count);
173
174     for (int i = 0; i < n; i++) {
175
176         if (vec[i].value == min[i]) {
177             min[i] = min[i] + 1;
178         }
179
180         else if (vec[i].value == max[i]) {
181             max[i] = max[i] + 1;
182         }
183
184         count++;
185     }
186
187     System.out.println ("Count = " + count);
188
189     for (int i = 0; i < n; i++) {
190
191         if (vec[i].value == min[i]) {
192             min[i] = min[i] + 1;
193         }
194
195         else if (vec[i].value == max[i]) {
196             max[i] = max[i] + 1;
197         }
198
199         count++;
200     }
201
202     System.out.println ("Count = " + count);
203
204     for (int i = 0; i < n; i++) {
205
206         if (vec[i].value == min[i]) {
207             min[i] = min[i] + 1;
208         }
209
210         else if (vec[i].value == max[i]) {
211             max[i] = max[i] + 1;
212         }
213
214         count++;
215     }
216
217     System.out.println ("Count = " + count);
218
219     for (int i = 0; i < n; i++) {
220
221         if (vec[i].value == min[i]) {
222             min[i] = min[i] + 1;
223         }
224
225         else if (vec[i].value == max[i]) {
226             max[i] = max[i] + 1;
227         }
228
229         count++;
230     }
231
232     System.out.println ("Count = " + count);
233
234     for (int i = 0; i < n; i++) {
235
236         if (vec[i].value == min[i]) {
237             min[i] = min[i] + 1;
238         }
239
240         else if (vec[i].value == max[i]) {
241             max[i] = max[i] + 1;
242         }
243
244         count++;
245     }
246
247     System.out.println ("Count = " + count);
248
249     for (int i = 0; i < n; i++) {
250
251         if (vec[i].value == min[i]) {
252             min[i] = min[i] + 1;
253         }
254
255         else if (vec[i].value == max[i]) {
256             max[i] = max[i] + 1;
257         }
258
259         count++;
260     }
261
262     System.out.println ("Count = " + count);
263
264     for (int i = 0; i < n; i++) {
265
266         if (vec[i].value == min[i]) {
267             min[i] = min[i] + 1;
268         }
269
270         else if (vec[i].value == max[i]) {
271             max[i] = max[i] + 1;
272         }
273
274         count++;
275     }
276
277     System.out.println ("Count = " + count);
278
279     for (int i = 0; i < n; i++) {
280
281         if (vec[i].value == min[i]) {
282             min[i] = min[i] + 1;
283         }
284
285         else if (vec[i].value == max[i]) {
286             max[i] = max[i] + 1;
287         }
288
289         count++;
290     }
291
292     System.out.println ("Count = " + count);
293
294     for (int i = 0; i < n; i++) {
295
296         if (vec[i].value == min[i]) {
297             min[i] = min[i] + 1;
298         }
299
300         else if (vec[i].value == max[i]) {
301             max[i] = max[i] + 1;
302         }
303
304         count++;
305     }
306
307     System.out.println ("Count = " + count);
308
309     for (int i = 0; i < n; i++) {
310
311         if (vec[i].value == min[i]) {
312             min[i] = min[i] + 1;
313         }
314
315         else if (vec[i].value == max[i]) {
316             max[i] = max[i] + 1;
317         }
318
319         count++;
320     }
321
322     System.out.println ("Count = " + count);
323
324     for (int i = 0; i < n; i++) {
325
326         if (vec[i].value == min[i]) {
327             min[i] = min[i] + 1;
328         }
329
330         else if (vec[i].value == max[i]) {
331             max[i] = max[i] + 1;
332         }
333
334         count++;
335     }
336
337     System.out.println ("Count = " + count);
338
339     for (int i = 0; i < n; i++) {
340
341         if (vec[i].value == min[i]) {
342             min[i] = min[i] + 1;
343         }
344
345         else if (vec[i].value == max[i]) {
346             max[i] = max[i] + 1;
347         }
348
349         count++;
350     }
351
352     System.out.println ("Count = " + count);
353
354     for (int i = 0; i < n; i++) {
355
356         if (vec[i].value == min[i]) {
357             min[i] = min[i] + 1;
358         }
359
360         else if (vec[i].value == max[i]) {
361             max[i] = max[i] + 1;
362         }
363
364         count++;
365     }
366
367     System.out.println ("Count = " + count);
368
369     for (int i = 0; i < n; i++) {
370
371         if (vec[i].value == min[i]) {
372             min[i] = min[i] + 1;
373         }
374
375         else if (vec[i].value == max[i]) {
376             max[i] = max[i] + 1;
377         }
378
379         count++;
380     }
381
382     System.out.println ("Count = " + count);
383
384     for (int i = 0; i < n; i++) {
385
386         if (vec[i].value == min[i]) {
387             min[i] = min[i] + 1;
388         }
389
390         else if (vec[i].value == max[i]) {
391             max[i] = max[i] + 1;
392         }
393
394         count++;
395     }
396
397     System.out.println ("Count = " + count);
398
399     for (int i = 0; i < n; i++) {
400
401         if (vec[i].value == min[i]) {
402             min[i] = min[i] + 1;
403         }
404
405         else if (vec[i].value == max[i]) {
406             max[i] = max[i] + 1;
407         }
408
409         count++;
410     }
411
412     System.out.println ("Count = " + count);
413
414     for (int i = 0; i < n; i++) {
415
416         if (vec[i].value == min[i]) {
417             min[i] = min[i] + 1;
418         }
419
420         else if (vec[i].value == max[i]) {
421             max[i] = max[i] + 1;
422         }
423
424         count++;
425     }
426
427     System.out.println ("Count = " + count);
428
429     for (int i = 0; i < n; i++) {
430
431         if (vec[i].value == min[i]) {
432             min[i] = min[i] + 1;
433         }
434
435         else if (vec[i].value == max[i]) {
436             max[i] = max[i] + 1;
437         }
438
439         count++;
440     }
441
442     System.out.println ("Count = " + count);
443
444     for (int i = 0; i < n; i++) {
445
446         if (vec[i].value == min[i]) {
447             min[i] = min[i] + 1;
448         }
449
450         else if (vec[i].value == max[i]) {
451             max[i] = max[i] + 1;
452         }
453
454         count++;
455     }
456
457     System.out.println ("Count = " + count);
458
459     for (int i = 0; i < n; i++) {
460
461         if (vec[i].value == min[i]) {
462             min[i] = min[i] + 1;
463         }
464
465         else if (vec[i].value == max[i]) {
466             max[i] = max[i] + 1;
467         }
468
469         count++;
470     }
471
472     System.out.println ("Count = " + count);
473
474     for (int i = 0; i < n; i++) {
475
476         if (vec[i].value == min[i]) {
477             min[i] = min[i] + 1;
478         }
479
480         else if (vec[i].value == max[i]) {
481             max[i] = max[i] + 1;
482         }
483
484         count++;
485     }
486
487     System.out.println ("Count = " + count);
488
489     for (int i = 0; i < n; i++) {
490
491         if (vec[i].value == min[i]) {
492             min[i] = min[i] + 1;
493         }
494
495         else if (vec[i].value == max[i]) {
496             max[i] = max[i] + 1;
497         }
498
499         count++;
500     }
501
502     System.out.println ("Count = " + count);
503
504     for (int i = 0; i < n; i++) {
505
506         if (vec[i].value == min[i]) {
507             min[i] = min[i] + 1;
508         }
509
510         else if (vec[i].value == max[i]) {
511             max[i] = max[i] + 1;
512         }
513
514         count++;
515     }
516
517     System.out.println ("Count = " + count);
518
519     for (int i = 0; i < n; i++) {
520
521         if (vec[i].value == min[i]) {
522             min[i] = min[i] + 1;
523         }
524
525         else if (vec[i].value == max[i]) {
526             max[i] = max[i] + 1;
527         }
528
529         count++;
530     }
531
532     System.out.println ("Count = " + count);
533
534     for (int i = 0; i < n; i++) {
535
536         if (vec[i].value == min[i]) {
537             min[i] = min[i] + 1;
538         }
539
540         else if (vec[i].value == max[i]) {
541             max[i] = max[i] + 1;
542         }
543
544         count++;
545     }
546
547     System.out.println ("Count = " + count);
548
549     for (int i = 0; i < n; i++) {
550
551         if (vec[i].value == min[i]) {
552             min[i] = min[i] + 1;
553         }
554
555         else if (vec[i].value == max[i]) {
556             max[i] = max[i] + 1;
557         }
558
559         count++;
560     }
561
562     System.out.println ("Count = " + count);
563
564     for (int i = 0; i < n; i++) {
565
566         if (vec[i].value == min[i]) {
567             min[i] = min[i] + 1;
568         }
569
570         else if (vec[i].value == max[i]) {
571             max[i] = max[i] + 1;
572         }
573
574         count++;
575     }
576
577     System.out.println ("Count = " + count);
578
579     for (int i = 0; i < n; i++) {
580
581         if (vec[i].value == min[i]) {
582             min[i] = min[i] + 1;
583         }
584
585         else if (vec[i].value == max[i]) {
586             max[i] = max[i] + 1;
587         }
588
589         count++;
590     }
591
592     System.out.println ("Count = " + count);
593
594     for (int i = 0; i < n; i++) {
595
596         if (vec[i].value == min[i]) {
597             min[i] = min[i] + 1;
598         }
599
600         else if (vec[i].value == max[i]) {
601             max[i] = max[i] + 1;
602         }
603
604         count++;
605     }
606
607     System.out.println ("Count = " + count);
608
609     for (int i = 0; i < n; i++) {
610
611         if (vec[i].value == min[i]) {
612             min[i] = min[i] + 1;
613         }
614
615         else if (vec[i].value == max[i]) {
616             max[i] = max[i] + 1;
617         }
618
619         count++;
620     }
621
622     System.out.println ("Count = " + count);
623
624     for (int i = 0; i < n; i++) {
625
626         if (vec[i].value == min[i]) {
627             min[i] = min[i] + 1;
628         }
629
630         else if (vec[i].value == max[i]) {
631             max[i] = max[i] + 1;
632         }
633
634         count++;
635     }
636
637     System.out.println ("Count = " + count);
638
639     for (int i = 0; i < n; i++) {
640
641         if (vec[i].value == min[i]) {
642             min[i] = min[i] + 1;
643         }
644
645         else if (vec[i].value == max[i]) {
646             max[i] = max[i] + 1;
647         }
648
649         count++;
650     }
651
652     System.out.println ("Count = " + count);
653
654     for (int i = 0; i < n; i++) {
655
656         if (vec[i].value == min[i]) {
657             min[i] = min[i] + 1;
658         }
659
660         else if (vec[i].value == max[i]) {
661             max[i] = max[i] + 1;
662         }
663
664         count++;
665     }
666
667     System.out.println ("Count = " + count);
668
669     for (int i = 0; i < n; i++) {
670
671         if (vec[i].value == min[i]) {
672             min[i] = min[i] + 1;
673         }
674
675         else if (vec[i].value == max[i]) {
676             max[i] = max[i] + 1;
677         }
678
679         count++;
680     }
681
682     System.out.println ("Count = " + count);
683
684     for (int i = 0; i < n; i++) {
685
686         if (vec[i].value == min[i]) {
687             min[i] = min[i] + 1;
688         }
689
690         else if (vec[i].value == max[i]) {
691             max[i] = max[i] + 1;
692         }
693
694         count++;
695     }
696
697     System.out.println ("Count = " + count);
698
699     for (int i = 0; i < n; i++) {
700
701         if (vec[i].value == min[i]) {
702             min[i] = min[i] + 1;
703         }
704
705         else if (vec[i].value == max[i]) {
706             max[i] = max[i] + 1;
707         }
708
709         count++;
710     }
711
712     System.out.println ("Count = " + count);
713
714     for (int i = 0; i < n; i++) {
715
716         if (vec[i].value == min[i]) {
717             min[i] = min[i] + 1;
718         }
719
720         else if (vec[i].value == max[i]) {
721             max[i] = max[i] + 1;
722         }
723
724         count++;
725     }
726
727     System.out.println ("Count = " + count);
728
729     for (int i = 0; i < n; i++) {
730
731         if (vec[i].value == min[i]) {
732             min[i] = min[i] + 1;
733         }
734
735         else if (vec[i].value == max[i]) {
736             max[i] = max[i] + 1;
737         }
738
739         count++;
740     }
741
742     System.out.println ("Count = " + count);
743
744     for (int i = 0; i < n; i++) {
745
746         if (vec[i].value == min[i]) {
747             min[i] = min[i] + 1;
748         }
749
750         else if (vec[i].value == max[i]) {
751             max[i] = max[i] + 1;
752         }
753
754         count++;
755     }
756
757     System.out.println ("Count = " + count);
758
759     for (int i = 0; i < n; i++) {
760
761         if (vec[i].value == min[i]) {
762             min[i] = min[i] + 1;
763         }
764
765         else if (vec[i].value == max[i]) {
766             max[i] = max[i] + 1;
767         }
768
769         count++;
770     }
771
772     System.out.println ("Count = " + count);
773
774     for (int i = 0; i < n; i++) {
775
776         if (vec[i].value == min[i]) {
777             min[i] = min[i] + 1;
778         }
779
780         else if (vec[i].value == max[i]) {
781             max[i] = max[i] + 1;
782         }
783
784         count++;
785     }
786
787     System.out.println ("Count = " + count);
788
789     for (int i = 0; i < n; i++) {
790
791         if (vec[i].value == min[i]) {
792             min[i] = min[i] + 1;
793         }
794
795         else if (vec[i].value == max[i]) {
796             max[i] = max[i] + 1;
797         }
798
799         count++;
800     }
801
802     System.out.println ("Count = " + count);
803
804     for (int i = 0; i < n; i++) {
805
806         if (vec[i].value == min[i]) {
807             min[i] = min[i] + 1;
808         }
809
810         else if (vec[i].value == max[i]) {
811             max[i] = max[i] + 1;
812         }
813
814         count++;
815     }
816
817     System.out.println ("Count = " + count);
818
819     for (int i = 0; i < n; i++) {
820
821         if (vec[i].value == min[i]) {
822             min[i] = min[i] + 1;
823        }
824
825         else if (vec[i].value == max[i]) {
826             max[i] = max[i] + 1;
827        }
828
829         count++;
830     }
831
832     System.out.println ("Count = " + count);
833
834     for (int i = 0; i < n; i++) {
835
836         if (vec[i].value == min[i]) {
837             min[i] = min[i] + 1;
838        }
839
840         else if (vec[i].value == max[i]) {
841             max[i] = max[i] + 1;
842        }
843
844         count++;
845     }
846
847     System.out.println ("Count = " + count);
848
849     for (int i = 0; i < n; i++) {
850
851         if (vec[i].value == min[i]) {
852             min[i] = min[i] + 1;
853        }
854
855         else if (vec[i].value == max[i]) {
856             max[i] = max[i] + 1;
857        }
858
859         count++;
860     }
861
862     System.out.println ("Count = " + count);
863
864     for (int i = 0; i < n; i++) {
865
866         if (vec[i].value == min[i]) {
867             min[i] = min[i] + 1;
868        }
869
870         else if (vec[i].value == max[i]) {
871             max[i] = max[i] + 1;
872        }
873
874         count++;
875     }
876
877     System.out.println ("Count = " + count);
878
879     for (int i = 0; i < n; i++) {
880
881         if (vec[i].value == min[i]) {
882             min[i] = min[i] + 1;
883        }
884
885         else if (vec[i].value == max[i]) {
886             max[i] = max[i] + 1;
887        }
888
889         count++;
890     }
891
892     System.out.println ("Count = " + count);
893
894     for (int i = 0; i < n; i++) {
895
896         if (vec[i].value == min[i]) {
897             min[i] = min[i] + 1;
898        }
899
900         else if (vec[i].value == max[i]) {
901             max[i] = max[i] + 1;
902        }
903
904         count++;
905     }
906
907     System.out.println ("Count = " + count);
908
909     for (int i = 0; i < n; i++) {
910
911         if (vec[i].value == min[i]) {
912             min[i] = min[i] + 1;
913        }
914
915         else if (vec[i].value == max[i]) {
916             max[i] = max[i] + 1;
917        }
918
919         count++;
920     }
921
922     System.out.println ("Count = " + count);
923
924     for (int i = 0; i < n; i++) {
925
926         if (vec[i].value == min[i]) {
927             min[i] = min[i] + 1;
928        }
929
930         else if (vec[i].value == max[i]) {
931             max[i] = max[i] + 1;
932        }
933
934         count++;
935     }
936
937     System.out.println ("Count = " + count);
938
939     for (int i = 0; i < n; i++) {
940
941         if (vec[i].value == min[i]) {
942             min[i] = min[i] + 1;
943        }
944
945         else if (vec[i].value == max[i]) {
946             max[i] = max[i] + 1;
947        }
948
949         count++;
950     }
951
952     System.out.println ("Count = " + count);
953
954     for (int i = 0; i < n; i++) {
955
956         if (vec[i].value == min[i]) {
957             min[i] = min[i] + 1;
958        }
959
960         else if (vec[i].value == max[i]) {
961             max[i] = max[i] + 1;
962        }
963
964         count++;
965     }
966
967     System.out.println ("Count = " + count);
968
969     for (int i = 0; i < n; i++) {
970
971         if (vec[i].value == min[i]) {
972             min[i] = min[i] + 1;
973        }
974
975         else if (vec[i].value == max[i]) {
976             max[i] = max[i] + 1;
977        }
978
979         count++;
980     }
981
982     System.out.println ("Count = " + count);
983
984     for (int i = 0; i < n; i++) {
985
986         if (vec[i].value == min[i]) {
987             min[i] = min[i] + 1;
988        }
989
990         else if (vec[i].value == max[i]) {
991             max[i] = max[i] + 1;
992        }
993
994         count++;
995     }
996
997     System.out.println ("Count = " + count);
998
999     for (int i = 0; i < n; i++) {
1000
1001         if (vec[i].value == min[i]) {
1002             min[i] = min[i] + 1;
1003        }
1004
1005         else if (vec[i].value == max[i]) {
1006             max[i] = max[i] + 1;
1007        }
1008
1009         count++;
1010     }
1011
1012     System.out.println ("Count = " + count);
1013
1014     for (int i = 0; i < n; i++) {
1015
1016         if (vec[i].value == min[i]) {
1017             min[i] = min[i] + 1;
1018        }
1019
1020         else if (vec[i].value == max[i]) {
1021             max[i] = max[i] + 1;
1022        }
1023
1024         count++;
1025     }
1026
1027     System.out.println ("Count = " + count);
1028
1029     for (int i = 0; i < n; i++) {
1030
1031         if (vec[i].value == min[i]) {
1032             min[i] = min[i] + 1;
1033        }
1034
1035         else if (vec[i].value == max[i]) {
1036             max[i] = max[i] + 1;
1037        }
1038
1039         count++;
1040     }
1041
1042     System.out.println ("Count = " + count);
1043
1044     for (int i = 0; i < n; i++) {
1045
1046         if (vec[i].value == min[i]) {
1047             min[i] = min[i] + 1;
1048        }
1049
1050         else if (vec[i].value == max[i]) {
1051             max[i] = max[i] + 1;
1052        }
1053
1054         count++;
1055     }
1056
1057     System.out.println ("Count = " + count);
1058
1059     for (int i = 0; i < n; i++) {
1060
1061         if (vec[i].value == min[i]) {
1062             min[i] = min[i] + 1;
1063        }
1064
1065         else if (vec[i].value == max[i]) {
1066             max[i] = max[i] + 1;
1067        }
1068
1069         count++;
1070     }
1071
1072     System.out.println ("Count = " + count);
1073
1074     for (int i = 0; i < n; i++) {
1075
1076         if (vec[i].value == min[i]) {
1077             min[i] = min[i] + 1;
1078        }
1079
1080         else if (vec[i].value == max[i]) {
1081             max[i] = max[i] + 1;
1082        }
1083
1084         count++;
1085     }
1086
1087     System.out.println ("Count = " + count);
1088
1089     for (int i = 0; i < n; i++) {
1090
1091         if (vec[i].value == min[i]) {
1092             min[i] = min[i] + 1;
1093        }
1094
1095         else if (vec[i].value == max[i]) {
1096             max[i] = max[i] + 1;
1097        }
1098
1099         count++;
1100     }
1101
1102     System.out.println ("Count = " + count);
1103
1104     for (int i = 0; i < n; i++) {
1105
1106         if (vec[i].value == min[i]) {
1107             min[i] = min[i] + 1;
1108        }
1109
1110         else if (vec[i].value == max[i]) {
1111             max[i] = max[i] + 1;
1112        }
1113
1114         count++;
1115     }
1116
1117     System.out.println ("Count = " + count);
1118
1119     for (int i = 0; i < n; i++) {
1120
1121         if (vec[i].value == min[i]) {
1122             min[i] = min[i] + 1;
1123        }
1124
1125         else if (vec[i].value == max[i]) {
1126             max[i] = max[i] + 1;
1127        }
1128
1129         count++;
1130     }
1131
1132     System.out.println ("Count = " + count);
1133
1134     for (int i = 0; i < n; i++) {
1135
1136         if (vec[i].value == min[i]) {
1137             min[i] = min[i] + 1;
1138        }
1139
1140         else if (vec[i].value == max[i]) {
1141             max[i] = max[i] + 1;
1142        }
1143
1144         count++;
1145     }
1146
1147     System.out.println ("Count = " + count);
1148
1149     for (int i = 0; i < n; i++) {
1150
1151         if (vec[i].value == min[i]) {
1152             min[i] = min[i] + 1;
1153        }
1154
1155         else if (vec[i].value == max[i]) {
1156             max[i] = max[i] + 1;
1157        }
1158
1159         count++;
1160     }
1161
1162     System.out.println ("Count = " + count);
1163
1164     for (int i = 0; i < n; i++) {
1165
1166         if (vec[i].value == min[i]) {
1167             min[i] = min[i] + 1;
1168        }
1169
1170         else if (vec[i].value == max[i]) {
1171             max[i] = max[i] + 1;
1172        }
1173
1174         count++;
1175     }
1176
1177     System.out.println ("Count = " + count);
1178
1179     for (int i = 0; i < n; i++) {
1180
1181         if (vec[i].value == min[i]) {
1182             min[i] = min[i] + 1;
1183        }
1184
1185         else if (vec[i].value == max[i]) {
1186             max[i] = max[i] + 1;
1187        }
1188
1189         count++;
1190     }
1191
1192     System.out.println ("Count = " + count);
1193
1194     for (int i = 0; i < n; i++) {
1195
1196         if (vec[i].value == min[i]) {
1197             min[i] = min[i] + 1;
1198        }
1199
1200         else if (vec[i].value == max[i]) {
1201             max[i] = max[i] + 1;
1202        }
1203
1204         count++;
1205     }
1206
1207     System.out.println ("Count = " + count);
1208
1209     for (int i = 0; i < n; i++) {
1210
1211         if (vec[i].value == min[i]) {
1212             min[i] = min[i] + 1;
1213        }
1214
1215         else if (vec[i].value == max[i]) {
1216             max[i] = max[i] + 1;
1217        }
1218
1219         count++;
1220     }
1221
1222     System.out.println ("Count = " + count);
1223
1224     for (int i = 0; i < n; i++) {
1225
1226         if (vec[i].value == min[i]) {
1227             min[i] = min[i] + 1;
1228        }
1229
1230         else if (vec[i].value == max[i]) {
1231             max[i] = max[i] + 1;
1232        }
1233
1234         count++;
1235     }
1236
1237     System.out.println ("Count = " + count);
1238
1239     for (int i = 0; i < n; i++) {
1240
1241         if (vec[i].value == min[i]) {
1242             min[i] = min[i] + 1;
1243        }
1244
1245         else if (vec[i].value == max[i]) {
1246             max[i] = max[i] + 1;
1247        }
1248
1249         count++;
1250     }
1251
1252     System.out.println ("Count = " + count);
1253
1254     for (int i = 0; i < n; i++) {
1255
1256         if (vec[i].value == min
```


2. Метод Прима

```

#include <iostream>
using namespace std;

int main()
{
    int N, K;
    int m, n;
    int number;
    int L, R;
    int m1 = 1;
    int max1 = 100;
    int min1 = 0;
    int cost[100][100];
    int cost1[100][100];
    int path[100][100];
    int pathIndex = 0;

    cout << "Input: Number of nodes: ";
    cin >> number;
    cout << "Input: Number of edges: ";
    cin >> edgeNumber;

    for (int i = 1; i <= number; i++)
        for (int j = 1; j <= number; j++)
        {
            cost[i][j] = cost1[i][j];
            if (cost[i][j] == 0)
                cost1[i][j] = 999;
        }
    max1 = 3;
    cout << endl;

    while (true)
    {
        for (int i = 1; i <= number; i++)
            for (int j = 1; j <= number; j++)
            {
                if (cost[i][j] < max1)
                    if (cost[i][j] != 0)
                    {
                        min1 = cost[i][j];
                        L = i + 1;
                        R = j + 1;
                    }
            }
        cout << endl;
    }

    vector<Edge> edges(edgeNumber);
    vector<int> mst1;
    mst1.push_back(1);

    cout << "Input: Edge number: ";
    cin >> edgeNumber;
    cout << endl;

    for (int i = 0; i < edgeNumber - 1; i++)
    {
        for (int j = 4; j <= 4 * (i + 1) - 1; j++)
        {
            if (mst1[j].weight <= mst1[i].weight) { swap(mst1[j].weight, mst1[i].weight); swap(mst1[j].id, mst1[i].id); }
        }
    }

    cout << endl;
    cout << "Input: Edge number: ";
    cin >> edgeNumber;
    cout << endl;

    for (int i = 0; i < edgeNumber; i++)
    {
        for (int j = 4; j <= 4 * (i + 1) - 1; j++)
        {
            for (int k = 0; k <= vertexNumber; k++)
            {
                if (cost1[i][k] == cost1[j][k] && cost1[i][k] != 0) { temp1 = j; }
                else if (cost1[i][k] == cost1[k][j] && cost1[i][k] != 0) { temp2 = j; }
            }
        }
    }
}

for (int i = 0; i < edgeNumber; i++)
{
    for (int j = 0; j <= vertexNumber; j++)
    {
        if (cost1[i][j] == temp1 || cost1[i][j] == temp2) { cost1[i][j] = 0; }
    }
}
cout << endl;
if (temp1 < temp2) { temp1 = temp1 + 100 * i; }
else { temp2 = temp2 + 100 * i; }

if (temp1 < temp2) { cost1[temp1].id = cost1[temp2].id; cost1[temp2].id++; }
else { cost1[temp2].id = cost1[temp1].id; cost1[temp1].id++; }

if (cost1[temp1].id <= cost1[temp2].id)
    cout << cost1[temp1].id << endl;
else
    cout << cost1[temp2].id << endl;
}

```

3. Метод Краскала

```

#include <iostream>
#include <vector>

using namespace std;

class AdjList {
public:
    vector<vector<int>> adjList;
    int vCount = 0;
};

class Edge {
public:
    int length;
    int u;
    int v;
    bool dir = false;
};

AdjList initGraph(vector<Edge> edges) {
    for (int i = 0; i < edges.size(); i++) {
        cout << "Length of edge " << i + 1 << " is " << edges[i].length << endl;
        cout << "edge " << i + 1 << " is connected to vertex " << edges[i].v << endl;
        cout << "edge " << i + 1 << " is directed " << edges[i].dir << endl;
        cout << "edge " << i + 1 << " is undirected " << !edges[i].dir << endl;
        cout << endl;
    }
    return adjList;
}

int main() {
    int EdgesNumber = 6, Vertices = 100, TotalE = 600;

    cout << "Number of edges in a graph: " << EdgesNumber << endl;
    cout << "Vertices number: " << Vertices << endl;
    cout << "Number of vertices in a graph: " << Vertices << endl;
}

```

```

13 if (temp1 < temp2) {
14     for (int i = n - 1; i < lastTemp2Index; i--) {
15         lastTemp3[i].arr[1] = lastTemp3[i].arr[0];
16         lastTemp3[i].arr[0] = 0;
17     }
18     lastTemp2Index = lastTemp2Index - 1;
19     lastTemp2Index = 0;
20 }
21
22 if (temp1 > temp2) {
23     for (int i = 0; i < (lastTemp3_index - lastTemp2_index) - 1; i++) {
24         lastTemp2[i].arr[1] = lastTemp2[i].arr[0];
25         lastTemp2[i].arr[0] = 0;
26     }
27     lastTemp3_index = lastTemp3_index - 1;
28     lastTemp3_index = 0;
29 }
30
31 if (temp1 == lastTemp3_index & temp2 == lastTemp2_index & lastTemp2_index != resultIndex & lastTemp3_index != resultIndex) {
32     resultIndex = true;
33 }
34
35 if (temp1 == lastTemp3_index & temp2 == lastTemp2_index) {
36     res[1].is = false;
37 }
38
39 temp2 = 100; temp3 = 100;
40
41 }
42
43 int temp = 0;
44
45 for (int i = 0; i < lastTemp2Index; i++) {
46     if (res[i].is == true) {
47         cout << "Edge " << i + connectingVertices << " --- vert " << i + << res[i].id << endl;
48         temp += res[i].weight;
49     }
50 }
51
52 cout << "minimum spanning tree: " << temp;
53
54 return 0;

```

```

Ввести матрице смежности:
0 4 3 2 0 0 0 0 0 0
4 0 0 2 2 3 0 0 0
3 0 0 0 0 7 0 0 0 0
2 0 0 0 0 2 4 0 0 0
0 2 0 0 0 0 7 2 0 0
0 0 7 2 0 0 4 0 3 0
0 1 0 4 0 0 0 9 4 0 0
0 0 0 0 7 4 0 0 0 0 7
0 0 0 0 5 0 4 0 0 0 1
0 0 0 0 0 5 0 0 0 0 2
0 0 0 0 0 0 7 1 0 0

1 1 42
2 4 62
3 1 33
4 8 103
5 10 113
6 11 91
7 1 24
8 2 71
9 2 52
10 6 84
1 -> 4 -> 6 -> 3 -> 10 -> 11 -> 9 -> 2 -> 7 -> 5 -> 8
Minimal costs:
29

```

```

11 if (temp1 < temp2 && temp1 < temp3) {
12     if (temp1 < temp2) {
13         for (int i = 0; i < len[temp2].edges; i++) {
14             if (len[temp2].edges[i] == i) {
15                 len[temp2].edges[i] = len[temp1].edges[i];
16                 len[temp2].edges[i+1] = 0;
17             }
18         }
19         len[temp1].edges = len[temp2].edges;
20         len[temp2].edges = 0;
21     }
22     if (temp2 < temp3) {
23         for (int i = 0; i < len[temp3].edges; i++) {
24             if (len[temp3].edges[i] == i) {
25                 len[temp3].edges[i] = len[temp2].edges[i];
26                 len[temp3].edges[i+1] = 0;
27             }
28         }
29     }
30 }
31 if (temp1 == temp2 && temp2 == temp3) {
32     len[temp1].edges = relative;
33     len[temp2].edges = relative;
34     len[temp3].edges = relative;
35     rel[0].is = true;
36 }
37 if (temp1 == temp2 && temp1 >= 100) { mst[0].is = false; }
38 temp1 = 100; temp2 = 100;
39 }
40 }
41 int temp = 0;
42 for (int i = 0; i < segments; i++) {
43     if (rel[i].is == false) {
44         cout << "Edge" << i << " connecting verticles " << mst[i].x << " " << mst[i].y << endl; temp += mst[i].length;
45     }
46 }
47 cout << "Minimal spinning tree: " << temp;
48 return 0;

```

Length of 12 edges: 1
First verticle connected to edge: 12 2
Second verticle connected to edge: 12 7

Length of 13 edges: 5
First verticle connected to edge: 13 5
Second verticle connected to edge: 13 9

Length of 14 edges: 3
First verticle connected to edge: 14 6
Second verticle connected to edge: 14 10

Length of 15 edges: 9
First verticle connected to edge: 15 3
Second verticle connected to edge: 15 5

Length of 16 edges: 4
First verticle connected to edge: 16 6
Second verticle connected to edge: 16 8

Length of 17 edges: 2
First verticle connected to edge: 17 4
Second verticle connected to edge: 17 6

Length of 18 edges: 4
First verticle connected to edge: 18 7
Second verticle connected to edge: 18 9

Edge connecting verticles M9 11
Edge connecting verticles M2 7
Edge connecting verticles M1 4
Edge connecting verticles M2 5
Edge connecting verticles M4 6
Edge connecting verticles M1 3
Edge connecting verticles M10 11
Edge connecting verticles M6 10
Edge connecting verticles M1 2
Edge connecting verticles M6 8
Minimal spinning tree: 25

Edge connecting verticles M9 11
Edge connecting verticles M2 7
Edge connecting verticles M1 4
Edge connecting verticles M2 5
Edge connecting verticles M4 6
Edge connecting verticles M1 3
Edge connecting verticles M10 11
Edge connecting verticles M6 10
Edge connecting verticles M1 2
Edge connecting verticles M6 8
Minimal spinning tree: 25

4. Алгоритм Дейкстри

```

#include <iostream>
using namespace std;

int N;
int graph[400][400];
int path[400];
bool check[400];
int Formation[400];

void makegrid()
{
    int i, j, rows, cols, rmg;
    cout << "Enter the number of nodes ";
    cin >> N;
    cout << "Enter the number of columns ";
    cin >> cols;
    cout << "Enter the number of rows ";
    cin >> rows;

    for(i = 0; i < N; i++) {
        for(j = i + 1; j < N; j++) {
            if ((j - i) == 1 || (j - i) == cols - 1)
                cout << "bridge ";
            else
                cout << "valley ";
            cout << j + 1 << " " << i + 1 << endl;
            cin >> graph[i][j];
        }
    }
}

int pathfind()
{
    int bits = 10000, maxsize;
    for (int v = 0; v < N; v++)
        if (check[v] == false && path[v] < maxsize)
    {

```

```

    v1 = path(v);
    v10tst = v;
}
return v10tst;
}

void printgraph(int j) {
    if (formation[j] == -5)
        return;
    printgraph(formation[j]);
    cout << "vertice " << j << " + ";
}

void pathformation() {
    int scr;
    cout << "From which node do you want to start search? ";
    cin >> scr;
    cout << endl;
    cout << "Your choice is accepted... " << endl;

    for (int i = 0; i < N; i++) {
        formation[i] = -5;
        path[i] = 00000;
        check[i] = false;
    }
    path[scr] = 1 = 1;
    for (int count = 0; count < N - 1; count++) {
        int j = path(i);
        check[j] = true;
        for (int v = 0; v < N; v++) {
            if ((check[v] && graph[i][v] &&
                path[v] + graph[i][v] < path[j])) {
                formation[v] = i;
                path[v] = path[i] + graph[i][v];
            }
        }
    }
}

cout << "Finding the least way... " << endl;
cout << "Counting the weight of the least way... " << endl;
cout << "Total weight: " << endl;
cout << "Here is the most optimal way: " << endl;
cout << "vertice 00 " << endl;
printgraph(0);
cout << endl;
cout << "The weight of way: " << endl;

```

```
cout << "Finding the least way..." << endl;
90 cout << "Counting the weight of the least way..." << endl;
91 cout << "Eureka!" << endl;
92 cout << "Here is the most optimal way: " << endl;
93 cout << "Verticle M1 -> ";
94 printgraph1(); 29];
95 cout << endl;
96 cout << "The weight of way: ";
97 cout << path[29] << endl;
98 }
99
100 int main()
101 {
102     makegraph();
103     pathformation();
104     return 0;
105 }
```

```
Enter the number of nodes: 30
Enter the number of columns: 6
Enter the number of rows: 5
Verticle % 1 -- verticle % 2: 6
Verticle % 1 -- verticle % 7: 4
Verticle % 2 -- verticle % 3: 1
Verticle % 2 -- verticle % 8: 8
Verticle % 3 -- verticle % 4: 7
Verticle % 3 -- verticle % 9: 3
Verticle % 4 -- verticle % 5: 3
Verticle % 4 -- verticle % 10: 7
Verticle % 5 -- verticle % 6: 3
Verticle % 5 -- verticle % 11: 5
Verticle % 6 -- verticle % 7: 9
Verticle % 6 -- verticle % 12: 7
Verticle % 7 -- verticle % 8: 2
Verticle % 7 -- verticle % 13: 5
Verticle % 8 -- verticle % 9: 1
Verticle % 8 -- verticle % 14: 1
Verticle % 9 -- verticle % 10: 4
Verticle % 9 -- verticle % 15: 3
Verticle % 10 -- verticle % 11: 2
Verticle % 10 -- verticle % 16: 4
Verticle % 11 -- verticle % 12: 4
Verticle % 11 -- verticle % 17: 1
Verticle % 12 -- verticle % 13: 0
Verticle % 12 -- verticle % 18: 7
Verticle % 13 -- verticle % 14: 7
Verticle % 13 -- verticle % 19: 5
Verticle % 14 -- verticle % 15: 1
Verticle % 14 -- verticle % 20: 7
Verticle % 15 -- verticle % 16: 2
Verticle % 15 -- verticle % 21: 1
Verticle % 16 -- verticle % 17: 3
Verticle % 16 -- verticle % 22: 1
```

Verticle N 10 -- verticle N 16: 4
Verticle N 11 -- verticle N 12: 4
Verticle N 11 -- verticle N 17: 1
Verticle N 12 -- verticle N 13: 3
Verticle N 12 -- verticle N 18: 7
Verticle N 13 -- verticle N 14: 7
Verticle N 13 -- verticle N 19: 5
Verticle N 14 -- verticle N 15: 1
Verticle N 14 -- verticle N 20: 7
Verticle N 15 -- verticle N 16: 2
Verticle N 15 -- verticle N 21: 1
Verticle N 16 -- verticle N 17: 2
Verticle N 16 -- verticle N 22: 1
Verticle N 17 -- verticle N 18: 7
Verticle N 17 -- verticle N 23: 2
Verticle N 18 -- verticle N 19: 8
Verticle N 18 -- verticle N 24: 8
Verticle N 19 -- verticle N 20: 7
Verticle N 19 -- verticle N 25: 8
Verticle N 20 -- verticle N 21: 3
Verticle N 20 -- verticle N 26: 2
Verticle N 21 -- verticle N 22: 1
Verticle N 21 -- verticle N 27: 1
Verticle N 22 -- verticle N 23: 3
Verticle N 22 -- verticle N 28: 3
Verticle N 23 -- verticle N 24: 5
Verticle N 23 -- verticle N 29: 3
Verticle N 24 -- verticle N 25: 8
Verticle N 24 -- verticle N 30: 7
Verticle N 25 -- verticle N 26: 4
Verticle N 26 -- verticle N 27: 7
Verticle N 27 -- verticle N 28: 3
Verticle N 28 -- verticle N 29: 3
Verticle N 29 -- verticle N 30: 5

```
Verticle & 19 -> verticle & 28: 4
Verticle & 29 -> -> verticle & 29: 2
Verticle & 28 -> -> verticle & 25: 2
Verticle & 21 -> -> verticle & 20: 2
Verticle & 21 -> -> verticle & 27: 2
Verticle & 21 -> -> verticle & 29: 2
Verticle & 23 -> -> verticle & 24: 2
Verticle & 23 -> -> verticle & 29: 2
Verticle & 24 -> -> verticle & 25: 2
Verticle & 24 -> -> verticle & 36: 2
Verticle & 25 -> -> verticle & 25: 4
Verticle & 26 -> -> verticle & 39: 2
Verticle & 27 -> -> verticle & 29: 2
Verticle & 25 -> -> verticle & 29: 2
Verticle & 29 -> -> verticle & 38: 2
From which node do you want to start search? 2
Your choice is accepted...
Finding the least way...
Counting the weight of the least way...
Turelal!
Here is the most optimal ways:
Verticle A1 -> Verticle A7 -> Verticle A8 -> Verticle A14 -> Verticle A15 -> Verticle A21 -> Verticle A23 -> Verticle A29 -> Verticle A38 ->
The weight of way: 21
```

5. «Іди в найближчий» для розв'язання задачі комівояжера

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <fstream>
5
6 using namespace std;
7
8 int main()
9 {
10     string path = "MyFile.txt";
11
12     vector<array<int, 3>> arr;
13
14     ifstream file(path);
15
16     if (!file)
17     {
18         cout << "File does not exist" << endl;
19         return -1;
20     }
21
22     int check = 0;
23     string str;
24     str = "";
25
26     file >> str;
27     arr.push_back(str);
28
29     for (int i = 0; i < check; i++)
30     {
31         arr[i] = new int[check];
32     }
33
34     for (int i = 0; i < check; i++)
35     {
36         for (int j = 0; j < check; j++)
37         {
38             arr[i][j] = 0;
39         }
40     }
41
42     for (int i = 0; i < check; i++)
43     {
44         for (int j = i + 1; j < check; j++)
45         {
46             getLine(&arr[i], &arr[j]);
47             arr[i][j] = idk(arr[i], arr[j]);
48             arr[j][i] = idk(arr[j], arr[i]);
49         }
50     }
51
52     file.close();
53
54     return 0;
55 }
```

```

42     {
43         bool isCycle(int arr[], int count)
44     {
45         int max = arr[0], min = arr[0];
46
47         for (int i = 0; i < count; i++)
48         {
49             max = max(max, arr[i]);
50             min = min(min, arr[i]);
51         }
52
53         for (int i = 0; i < count; i++)
54         {
55             if (max(arr[i], arr[i + 1]) > min(arr[i], arr[i + 1]))
56             {
57                 return false;
58             }
59             else
60             {
61                 continue;
62             }
63         }
64     }
65
66     bool cyclicSort(int arr[], int size)
67     {
68         sort(arr);
69
70         for (int i = 0; i < size; i++)
71         {
72             for (int j = i; j < size; j++)
73             {
74                 if (arr[i] == arr[j] && i != j)
75                 {
76                     return false;
77                 }
78             }
79         }
80     }
81
82     return true;
83 }
84
85 int way(int arr[], int arr[])
86 {

```

```
55    set way(int&arr, int&size)
56    {
57        int count = 0;
58
59        for (int i = 0; i < 7; i++)
60        {
61            count += arr[i] - 1 | arr[i + 1] - 1;
62        }
63
64        count += size[arr[7] - 1] | arr[0] - 1;
65
66        return count;
67    }
68
69
70    int main()
71    {
72        int const count = 8;
73        int *arr;
74        arr = new int[8];
75        int var = count - 1;
76        bool b = true;
77        int mas = new int[count];
78
79        arr[mas] = new int[6];
80        int min = 1000;
81        int long = 0;
82
83        int n = 0;
84
85        for (int i = 0; i < count; i++)
86        {
87            mas[i] = 1;
88            min[mas[i]] = 1;
89        }
90
91        while (whichway(mas, count))
92        {
93            while (mas[var] != count)
94            {
95                mas[var]++;
96
97                if (CycleMatrix(mas, count))
98                {
99                    if (var == mas[0] - mas[1])
100                }
101            }
102        }
103    }
104
```

```

134
135     if (CyclicMatrix(max, count))
136     {
137         long = waylare, mas;
138
139         for (int i = 0; i < count; i++)
140         {
141             cout << mas[i] << " ";
142         }
143         cout << mas[0] << "(" << long << ") ";
144         cout << endl;
145
146         if (long < min)
147         {
148             min = long;
149             m = i;
150         }
151         if (long == min)
152         {
153             m++;
154         }
155     }
156
157     while (mas(var) == count)
158     {
159         mas[var] = 1;
160         var--;
161     }
162     mas[var] = 2;
163
164     if (CyclicMatrix(max, count))
165     {
166         for (int i = 0; i < count; i++)
167         {
168             cout << mas[i] << " ";
169         }
170         cout << mas[0] << "(" << long << ") ";
171         cout << endl;
172
173         long = waylare, mas;

```

```

69         mas[i] = mas[i];
70         i++;
71     } if (long == min)
72     {
73         mas[i];
74     }
75 }
76 var = count - 1;
77 }
78
79 for (int i = 0; i < count; i++)
80 {
81     mas[i] = i;
82     minmas[i] = i;
83 }
84 kray rez = new Array[8];
85 int iter = 0;
86
87 while (whichway(mas, count))
88 {
89     while (mas[var] != count)
90     {
91         mas[var]++;
92
93         if (CycleMatrix(mas, count))
94         {
95             long = way(krr, mas);
96
97             if (long == min)
98             {
99                 for (int i = 0; i < count; i++)
100                 {
101                     rez[iter].papa[i] = mas[i];
102                 }
103                 rez[iter].papa[count] = mas[0];
104                 iter++;
105             }
106         }
107     }
108     while (mas[var] == count)
109     {
110         mas[var] = 1;
111     }

```

```

1
2         for (int i = 0; i < count; i++)
3         {
4             rez[iter].papa[i] = mas[i];
5         }
6         rez[iter].papa[count] = mas[0];
7         iter++;
8     }
9
10    var = count - 1;
11
12
13    cout << "Изменение: " << endl;
14
15    for (int i = 0; i < iter - 1; i++)
16    {
17        for (int j = 0; j <= count; j++)
18        {
19            if (j != 0)
20            {
21                cout << " ";
22            }
23            cout << rez[i].papa[j] << " ";
24        }
25        cout << endl;
26        if (i == 0)
27        {
28            for (int j = 0; j <= count; j++)
29            {
30                if (j != 0)
31                {
32                    cout << " ";
33                }
34                cout << rez[i].papa[count - j] << " ";
35            }
36        }
37    }
38    cout << "Минимальная шага: " << min;
39

```

Вывод:

1 → 4 → 2 → 5 → 7 → 3 → 6 → 8 → 1
 1 → 8 → 6 → 3 → 7 → 5 → 2 → 4 → 1
 1 → 8 → 6 → 3 → 7 → 5 → 2 → 4 → 1
 2 → 4 → 1 → 8 → 6 → 3 → 7 → 5 → 2
 2 → 5 → 7 → 3 → 6 → 8 → 1 → 4 → 2
 3 → 6 → 8 → 1 → 4 → 2 → 5 → 7 → 3
 3 → 7 → 5 → 2 → 4 → 1 → 8 → 6 → 3
 4 → 1 → 8 → 6 → 3 → 7 → 5 → 2 → 4
 4 → 2 → 5 → 7 → 3 → 6 → 8 → 1 → 4
 5 → 2 → 4 → 1 → 8 → 6 → 3 → 7 → 5
 5 → 7 → 3 → 6 → 8 → 1 → 4 → 2 → 5
 6 → 3 → 7 → 5 → 2 → 4 → 1 → 8 → 6
 6 → 8 → 1 → 4 → 2 → 5 → 7 → 3 → 6
 7 → 3 → 6 → 8 → 1 → 4 → 2 → 5 → 7
 7 → 5 → 2 → 4 → 1 → 8 → 6 → 3 → 7
 8 → 1 → 4 → 2 → 5 → 7 → 3 → 6 → 8
 Минимальна шага: 15

8 → 7 → 6 → 5 → 1 → 2 → 4 → 3 → 8 (28)
 8 → 7 → 6 → 5 → 1 → 3 → 2 → 4 → 8 (28)
 8 → 7 → 6 → 5 → 1 → 3 → 4 → 2 → 8 (27)
 8 → 7 → 6 → 5 → 1 → 4 → 2 → 3 → 8 (27)
 8 → 7 → 6 → 5 → 1 → 4 → 3 → 2 → 8 (31)
 8 → 7 → 6 → 5 → 2 → 1 → 3 → 4 → 8 (30)
 8 → 7 → 6 → 5 → 2 → 1 → 4 → 3 → 8 (29)
 8 → 7 → 6 → 5 → 2 → 3 → 1 → 4 → 8 (29)
 8 → 7 → 6 → 5 → 2 → 3 → 4 → 1 → 8 (29)
 8 → 7 → 6 → 5 → 2 → 4 → 1 → 3 → 8 (24)
 8 → 7 → 6 → 5 → 2 → 4 → 3 → 1 → 8 (24)
 8 → 7 → 6 → 5 → 3 → 1 → 2 → 4 → 8 (29)
 8 → 7 → 6 → 5 → 3 → 1 → 4 → 2 → 8 (27)
 8 → 7 → 6 → 5 → 3 → 2 → 1 → 4 → 8 (33)
 8 → 7 → 6 → 5 → 3 → 2 → 4 → 1 → 8 (33)
 8 → 7 → 6 → 5 → 3 → 4 → 1 → 2 → 8 (32)
 8 → 7 → 6 → 5 → 3 → 4 → 2 → 1 → 8 (32)
 8 → 7 → 6 → 5 → 4 → 1 → 2 → 3 → 8 (32)
 8 → 7 → 6 → 5 → 4 → 1 → 3 → 2 → 8 (31)
 8 → 7 → 6 → 5 → 4 → 2 → 1 → 3 → 8 (28)
 8 → 7 → 6 → 5 → 4 → 2 → 3 → 1 → 8 (28)
 8 → 7 → 6 → 5 → 4 → 3 → 1 → 2 → 8 (32)
 8 → 7 → 6 → 5 → 4 → 3 → 2 → 1 → 8 (32)

8-> 7-> 6-> 1-> 3-> 4-> 5-> 2-> 8 (38)
 8-> 7-> 6-> 1-> 3-> 5-> 2-> 4-> 9 (27)
 8-> 7-> 6-> 1-> 3-> 5-> 4-> 2-> 8 (29)
 8-> 7-> 8-> 1-> 4-> 2-> 3-> 5-> 8 (20)
 8-> 7-> 6-> 1-> 4-> 2-> 5-> 3-> 8 (26)
 8-> 7-> 8-> 1-> 4-> 3-> 2-> 5-> 8 (38)
 8-> 7-> 6-> 1-> 4-> 3-> 5-> 2-> 9 (30)
 8-> 7-> 6-> 1-> 4-> 5-> 2-> 3-> 8 (38)
 8-> 7-> 8-> 1-> 4-> 5-> 3-> 2-> 9 (33)
 8-> 7-> 6-> 1-> 5-> 2-> 3-> 4-> 8 (31)
 8-> 7-> 8-> 1-> 5-> 2-> 4-> 3-> 8 (26)
 8-> 7-> 6-> 1-> 5-> 3-> 2-> 4-> 8 (30)
 8-> 7-> 6-> 1-> 5-> 3-> 4-> 2-> 8 (29)
 8-> 7-> 6-> 1-> 5-> 4-> 2-> 3-> 8 (29)
 8-> 7-> 6-> 1-> 5-> 4-> 3-> 2-> 8 (33)
 8-> 7-> 6-> 2-> 1-> 3-> 4-> 5-> 8 (31)
 8-> 7-> 6-> 2-> 1-> 3-> 5-> 4-> 8 (32)
 8-> 7-> 8-> 2-> 1-> 4-> 3-> 5-> 8 (31)
 8-> 7-> 6-> 2-> 1-> 4-> 5-> 3-> 8 (31)
 8-> 7-> 6-> 2-> 1-> 5-> 3-> 4-> 8 (32)
 8-> 7-> 6-> 2-> 1-> 5-> 4-> 3-> 8 (31)
 8-> 7-> 6-> 2-> 1-> 5-> 4-> 3-> 8 (31)
 8-> 7-> 6-> 2-> 3-> 1-> 5-> 4-> 8 (31)
 8-> 7-> 6-> 2-> 3-> 4-> 1-> 5-> 8 (30)
 8-> 7-> 6-> 2-> 3-> 4-> 5-> 1-> 8 (38)
 8-> 7-> 6-> 2-> 3-> 5-> 1-> 4-> 8 (31)
 8-> 7-> 6-> 2-> 3-> 5-> 4-> 1-> 8 (31)
 8-> 7-> 6-> 2-> 4-> 1-> 3-> 5-> 8 (26)
 8-> 7-> 6-> 2-> 4-> 1-> 5-> 3-> 8 (26)
 8-> 7-> 6-> 2-> 4-> 3-> 1-> 5-> 8 (26)

7-> 4-> 5-> 3-> 2-> 8-> 6-> 1-> 7 (34)
 7-> 4-> 5-> 3-> 6-> 1-> 2-> 8-> 7 (37)
 7-> 4-> 5-> 3-> 6-> 1-> 8-> 2-> 7 (34)
 7-> 4-> 5-> 3-> 6-> 2-> 1-> 8-> 7 (35)
 7-> 4-> 5-> 3-> 6-> 2-> 8-> 1-> 7 (35)
 7-> 4-> 5-> 3-> 6-> 8-> 1-> 2-> 7 (30)
 7-> 4-> 5-> 3-> 6-> 8-> 2-> 1-> 7 (30)
 7-> 4-> 5-> 3-> 8-> 1-> 2-> 6-> 7 (32)
 7-> 4-> 5-> 3-> 8-> 1-> 6-> 2-> 7 (33)
 7-> 4-> 5-> 3-> 8-> 2-> 1-> 6-> 7 (34)
 7-> 4-> 5-> 3-> 8-> 2-> 6-> 1-> 7 (34)
 7-> 4-> 5-> 3-> 8-> 6-> 1-> 2-> 7 (31)
 7-> 4-> 5-> 3-> 8-> 6-> 2-> 1-> 7 (31)
 7-> 4-> 5-> 6-> 1-> 2-> 3-> 8-> 7 (37)
 7-> 4-> 5-> 6-> 1-> 2-> 8-> 3-> 7 (32)
 7-> 4-> 5-> 6-> 1-> 3-> 2-> 8-> 7 (36)
 7-> 4-> 5-> 6-> 1-> 3-> 8-> 2-> 7 (33)
 7-> 4-> 5-> 6-> 1-> 8-> 2-> 3-> 7 (32)
 7-> 4-> 5-> 6-> 1-> 8-> 3-> 2-> 7 (34)
 7-> 4-> 5-> 6-> 2-> 1-> 3-> 8-> 7 (34)
 7-> 4-> 5-> 6-> 2-> 1-> 8-> 3-> 7 (30)
 7-> 4-> 5-> 6-> 2-> 3-> 1-> 8-> 7 (34)
 7-> 4-> 5-> 6-> 2-> 3-> 8-> 1-> 7 (34)

6. Алгоритм Флері та елементарних циклів знаходження ейлерового ланцюга в ейлеровому графі.

6. Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері; б) елементарних циклів.

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#ifndef __STDC__
#define __STDC__
#endif
using namespace std;

// A class that represents an undirected graph
class Graph
{
public:
    int V; // No. of vertices
    list<int> adj[]; // A dynamic array of adjacency lists
    Graph(int V)
    {
        this->V = V;
        adj = new list<int*>[V];
        for (int i = 0; i < V; ++i)
            adj[i].clear();
    }
    // Functions to add and remove edge
    void addEdge(int u, int v)
    {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    // Methods to print Eulerian tour
    void printEulerTour();
    void printEulerUtil(int u);

    // This function returns count of vertices reachable from v. It does BFS
    int BFSCount(int v, bool visited[])
    {
        // Create a queue for BFS
        queue<int> queue;
        // Mark the current node as visited and enqueue it
        visited[v] = true;
        queue.push(v);

        int count = 1;
        while (!queue.empty())
        {
            int u = queue.front();
            queue.pop();
            // Get all adjacent vertices of the current vertex u
            list<int>::iterator it;
            for (it = adj[u].begin(); it != adj[u].end(); ++it)
                if (!visited[*it])
                {
                    visited[*it] = true;
                    queue.push(*it);
                    count++;
                }
        }
        return count;
    }

    // The main function that prints Eulerian Tour. It first finds an odd
    // degree vertex (if there is any) and then calls printEulerUtil()
    void printEulerTour()
    {
        // Find a vertex with odd degree
        int u = 0;
        for (int i = 0; i < V; i++)
            if (adj[i].size() & 1)
            {
                u = i;
                break;
            }

        // Print tour starting from odd
        printEulerUtil(u);
        cout << endl;
    }
};
```

```
// If count is greater, then edge (u, v) is a bridge
// returns (count > COUNT) false, true
bool Graph::isBridge(int u, int v)
{
    // This function removes edge v-u from graph. It removes the edge by
    // replacing adjacent vertex value with -1.
    void Graph::removeEdge(int u, int v)
    {
        // Find v in adjacency list of u and replace it with -1
        list<int>::iterator it = find(adj[u].begin(), adj[u].end(), v);
        if (it != adj[u].end())
            *it = -1;

        // Find u in adjacency list of v and replace it with -1
        list<int>::iterator it = find(adj[v].begin(), adj[v].end(), u);
        if (it != adj[v].end())
            *it = -1;
    }

    // A BFS based function to count reachable vertices from u
    int Graph::BFSCount(int v, bool visited[])
    {
        // Mark the current node as visited
        visited[v] = true;
        int count = 1;

        // Recur for all vertices adjacent to this vertex
        list<int>::iterator it;
        for (it = adj[v].begin(); it != adj[v].end(); ++it)
            if ((*it) != -1 && !visited[*it])
                count += BFSCount(*it, visited);

        return count;
    }
}
```

```
#include <iostream>
void Graph::printEulerUtil(int u)
{
    // Print Euler tour starting from vertex u
    list<int>::iterator it;
    for (it = adj[u].begin(); it != adj[u].end(); ++it)
    {
        int v = *it;
        // If edge u-v is not removed and it's a valid next edge
        if (*it != -1 && !isBridge(u, v))
        {
            cout << u << "-" << v << " ";
            printEulerUtil(v);
        }
    }
}

// The function to check if edge u-v can be considered as next edge in
// Euler tour
bool Graph::isValidNextEdge(int u, int v)
{
    // If the edge u-v is valid in any of the following two cases
    // (i) If v is the only adjacent vertex of u
    int count = 0;
    for (it = adj[u].begin(); it != adj[u].end(); ++it)
        if (*it == v)
            count++;

    if (count == 1)
        return true;

    // (ii) If there are multiple adjacents, then u-v is not a bridge
    // In following steps to check if u-v is a bridge
    // (a) count of vertices reachable from u
    bool visited[V];
    memset(visited, false, sizeof(visited));
    int count1 = BFSCount(u, visited);

    // (b) Remove edge (u, v) and after removing the edge, count
    // vertices reachable from u
    removeEdge(u, v);
    memset(visited, false, sizeof(visited));
    int count2 = BFSCount(u, visited);

    // (c) add the edge back to the graph
    addEdge(u, v);
    if (count1 == count2)
        return true;
    else
        return false;
}
```

```
int main()
{
    Graph g1(10);
    g1.addEdge(0, 1, 0);
    g1.addEdge(0, 2, 0);
    g1.addEdge(1, 3, 0);
    g1.addEdge(2, 3, 0);
    g1.addEdge(2, 4, 0);
    g1.addEdge(3, 4, 0);
    g1.addEdge(3, 5, 0);
    g1.addEdge(3, 6, 0);
    g1.addEdge(4, 5, 0);
    g1.addEdge(4, 6, 0);
    g1.addEdge(4, 7, 0);
    g1.addEdge(5, 6, 0);
    g1.addEdge(5, 7, 0);
    g1.addEdge(5, 8, 0);
    g1.addEdge(6, 7, 0);
    g1.addEdge(6, 8, 0);
    g1.addEdge(6, 9, 0);
    g1.addEdge(7, 8, 0);
    g1.addEdge(7, 9, 0);
    g1.addEdge(8, 9, 0);
    g1.addEdge(8, 10, 0);
    g1.addEdge(9, 10, 0);
    g1.printEulerTour();
    return 0;
}
```

2-1	1-11	13-18	18-3	3-2	2-9	9-6	6-7	7-3	3-4	4-7	7-5	5-3	3-8	8-7	7-9	9-8	8-10	10-6	6-9-11	11-12	12-13	13-11	11-10	10-3
-----	------	-------	------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	--------	-------	-------	-------	-------	------

Process finished with exit code 0