МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту

Лабораторна робота №4
з курсу “Дискретна математика ”

Виконав:
ст. гр. КН-110
Холод Ігор

Викладач:
Мельникова Н.І.

Львів – 2018

Тема: Основні операції над графами. Знаходження остова мінімальної ваги за алгоритмом Пріма-Краскала

Мета роботи: набуття практичних вмінь та навичок з використання алгоритмів Пріма і Краскала.

Теоретичні відомості:

Теорія графів дає простий, доступний і потужний інструмент побудови моделей прикладних задач, є ефективним засобом формалізації сучасних інженерних і наукових задач у різних областях
знань.

Графом G називається пара множин (V, E), де V − множина вершин, перенумерованих числами $1, 2, 3, ..., n = v; V = \{v\}$, E − множина упорядкованих або неупорядкованих пар
$e = (v', v''), v' \in V$
$v'' \in V$, називаних дугами або ребрами, $E = \{e\}$. При цьому не має примусового значення, як вершини розташовані в просторі або площині і які конфігурації мають ребра.

Неорієнтованим графом G називається граф у якого ребра не мають напрямку. Такі ребра описуються неупорядкованою парою (v′,v″). **Орієнтований граф (орграф)** – це граф ребра якого мають напрямок та можуть бути описані упорядкованою парою (v′,v″). Упорядковане ребро називають **дугою**. Граф є **змішаним**, якщо наряду з орієнтованими ребрами (дугами) є також і неорієнтовані. При розв'язку задач змішаний граф зводиться до орграфа.

Кратними (паралельними) називаються ребра, які зв'язують одні і ті ж вершини. Якщо ребро виходить та й входить у дну і ту саму вершину, то таке ребро називається **петлею**.

Мультиграф – граф, який має кратні ребра. **Псевдограф** – граф, який має петлі. **Простий граф** – граф, який не має кратних ребер та петель.

Будь яке ребро е **інцедентно** двом вершинам (v′,v″), які воно з'єднує. У свою чергу вершини (v′,v″) інцендентні до ребра е . Дві вершини (v′,v″) називають **суміжними**, якщо вони належать до

одного й того самого ребра e , і **несуміжні** у протилежному випадку. Два **ребра називають суміжними**, якщо вони мають спільну вершину. Відношення суміжності як для вершин, так і для ребер є симетричним відношенням. **Степенем вершини** графа G називається число інцидентних їй ребер.

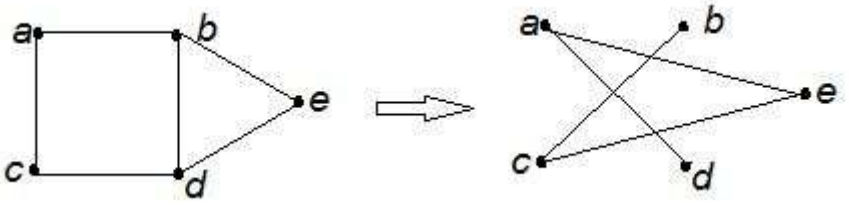
Граф, який не має ребер називається **пустим графом**, **нуль-графом**. Вершина графа, яка не інцедентна до жодного ребра, називається **ізольованою**. Вершина графа, яка інцедентна тільки до одного ребра, називається **звисаючою**.

Частина $G' = (V', E')$ графа $G = (V, E)$ називається **підграфом** графа G , якщо $V' \subseteq V$ і E' складається з тих і тільки тих ребер $e = (v', v'')$, у яких обидві кінцеві вершини $v', v'' \in V'$. Частина $G' = (V', E')$ називається **суграфом** або **остовим** підграфом графа G , якщо виконано умови: $V' = V$, $E' \subseteq E$.

Операції над графами:

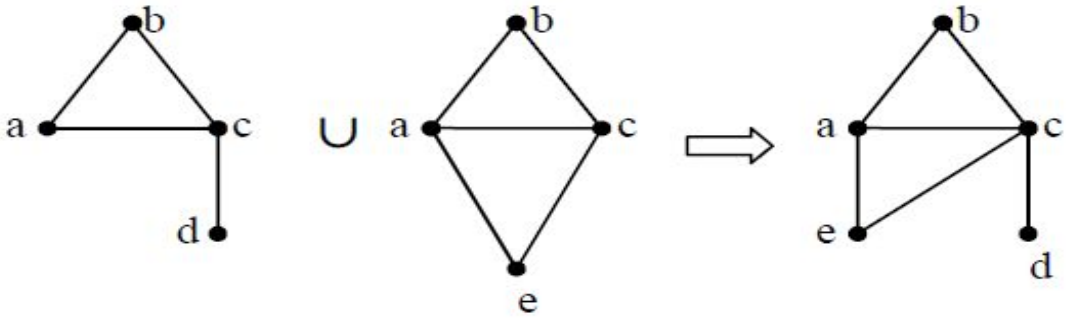
1)**Вилученням** ребра $e,\ e \in E$ з графа $G = (V, E)$ − є така операція внаслідок якої отримаємо новий граф G_1 для якого $G_1 = (V, E \setminus \{e\})$.

2)**Доповненням** графа $G = (V, E)$ називається граф $G = (V, E')$, якщо він має одну і ту саму кількість вершин та дві його вершини суміжні тоді і тільки тоді коли вони не суміжні в G (тобто ребро $(v_i, v_j) \in E'$ тоді коли $(v_i, v_j) \notin E$. Наприклад:



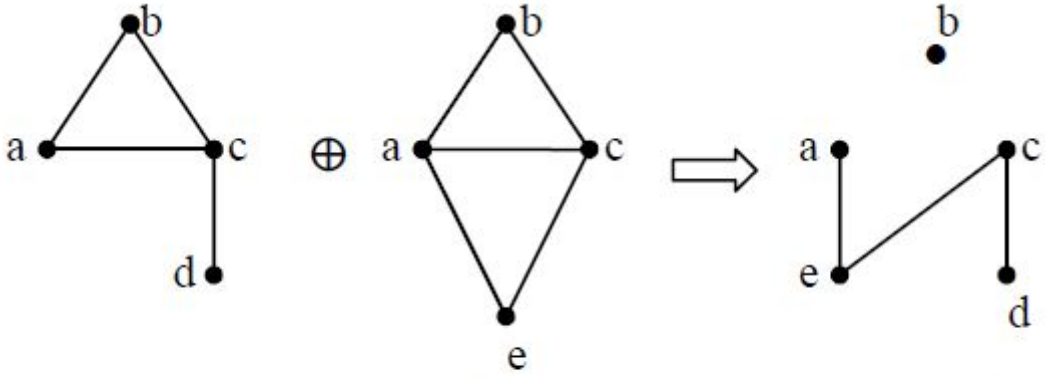
G $\overline{G}$

3)**Об’єднанням графів** $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ називається граф $G = (V, E) = G_1 \cup G_2$ у якому $V = V_1 \cup V_2$ та $E = E_1 \cup E_2$. Наприклад:



G_1 G_2 $G_1 \cup G_2$

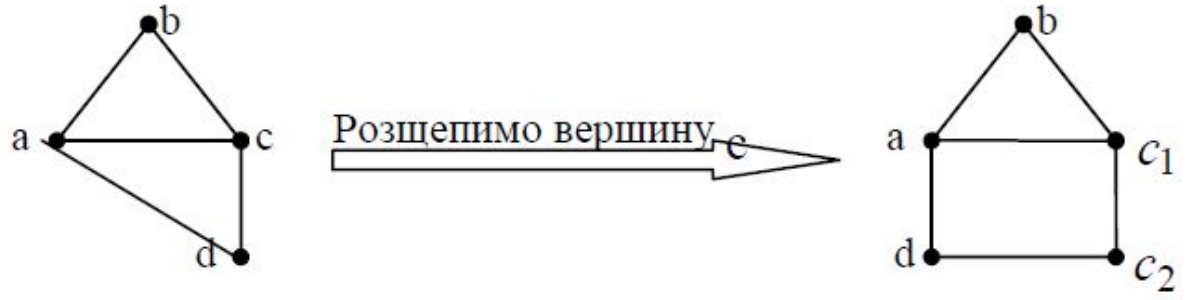
4)**Кільцевою сумою** графів $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ називається граф $G = (V, E) = G_1 \oplus G_2$ у якому $V = V_1 \cup V_2$ та $E = E_1 \Delta E_2 = (E_1 \cup E_2) \setminus (E_1 \cap E_2)$. Наприклад:



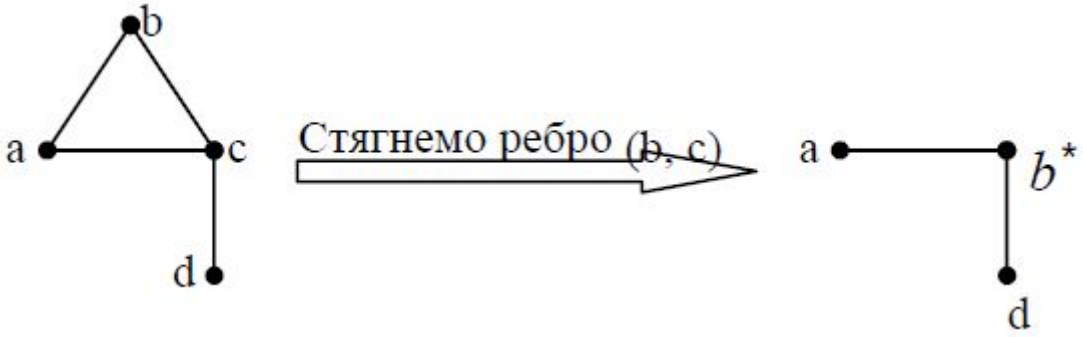
G_1 G_2 $G_1 \oplus G_2$

4)**Розщеплення (роздвоєння) вершини** графа. Нехай v – вершина графа $G = (E, V)$. Множину усіх суміжних з нею вершин довільним чином розділимо на дві множини $N_1(v)$ та $N_2(v)$, таких що $N_1(v) \cup N_2(v) = V$. Видаливши вершину v разом з інцидентними їй ребрами, додамо дві нові вершини v_1 та v_2, які з'єднані ребром (v_1, v_2). Вершину v_1 з'єднаємо ребром з кожною вершиною множини $N_1(v)$, а вершину v_2 – з кожною вершиною множини $N_2(v)$. Таким чином з графа G отримаємо новий граф G_v^*. Виконана операція називається розщепленням вершини v.

Наприклад:



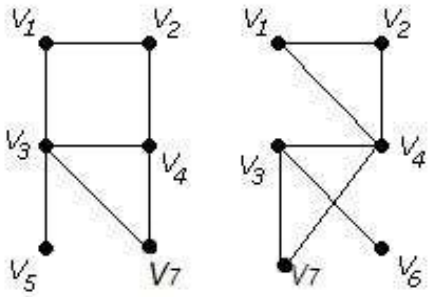
6)**Стягування ребра (дуги)**. Ця операція означає видалення ребра та ототожнення його суміжних вершин. Граф G_1 стягується до графа G_2 , якщо граф G_2 може бути отриманим з G_1 в результаті деякої послідовності стягування ребер (дуг). Наприклад



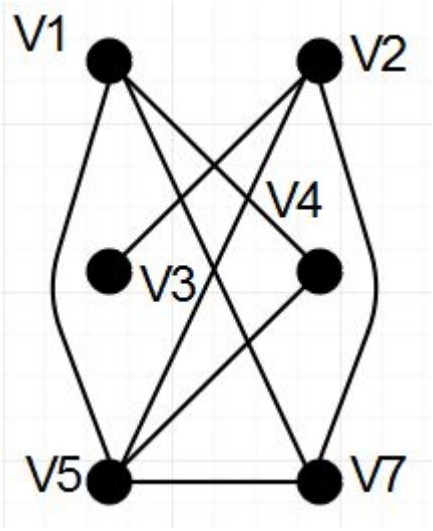
Завдання:

Варіант 13.

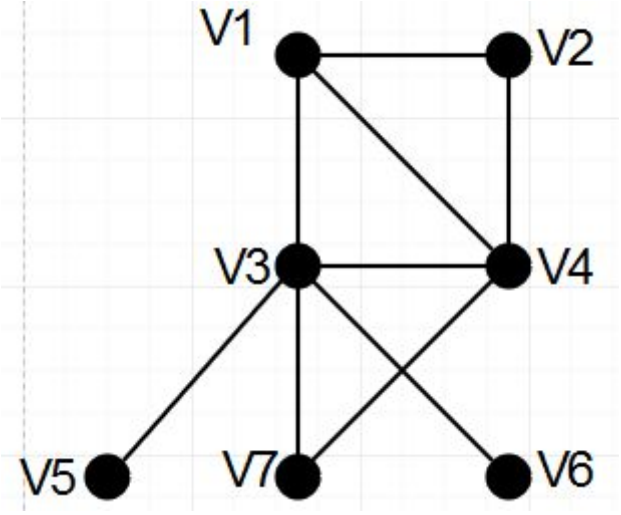
1. Виконати наступні операції над графами:
1) знайти доповнення до першого графу,
2) об’єднання графів,
3) кільцеву суму G1 та G2 (G1+G2),
4) розщепити вершину у другому графі,
5) виділити підграф А, що складається з 3-х вершин в G1 і знайти стягнення А в G1 (G1\ A), 6) добуток графів.



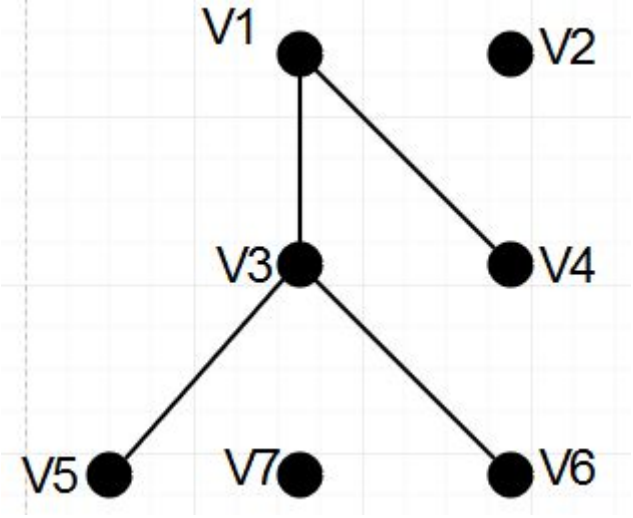
1)Доповнення:



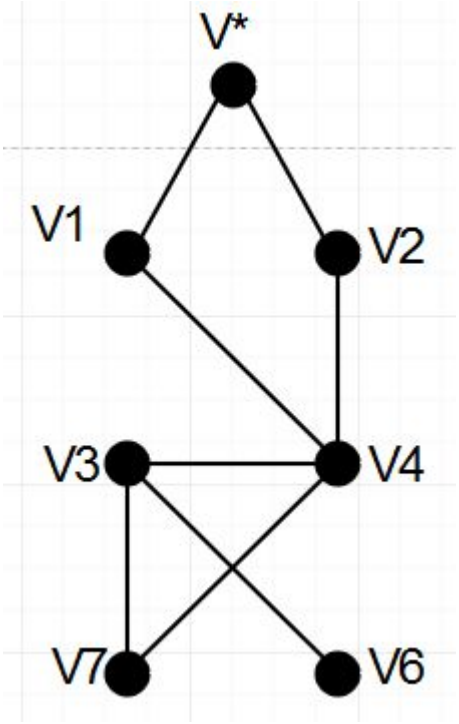
2)Об’єднання



3)Кільцева сума

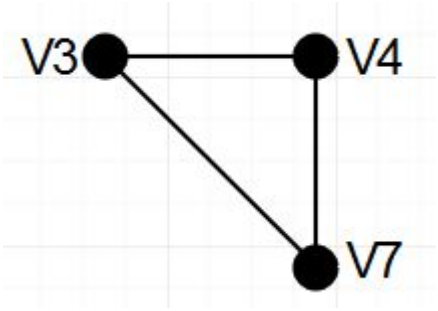


4)Розщеплення

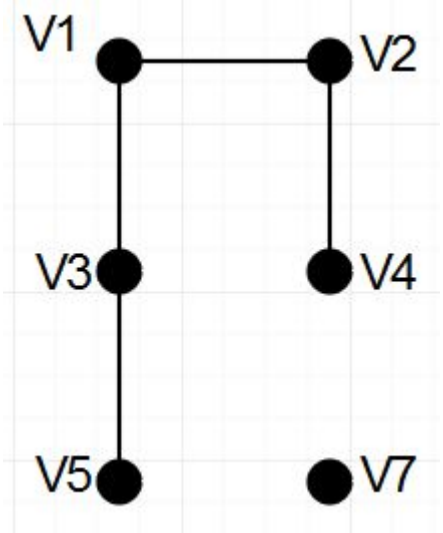


5)Стягнення

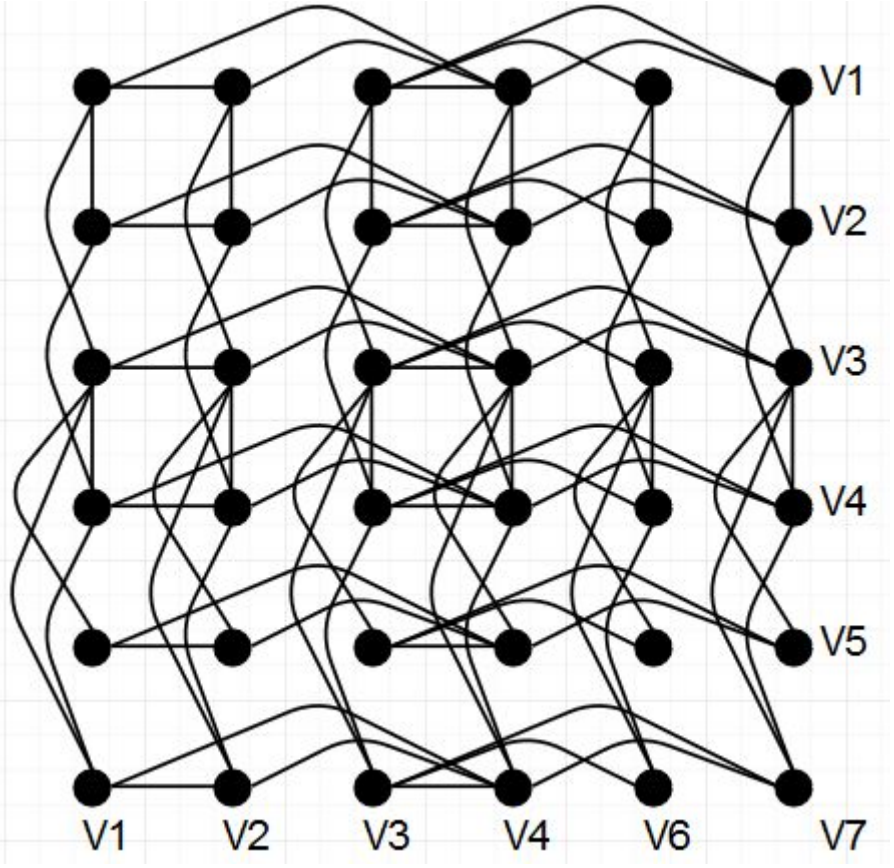
Підграф A:



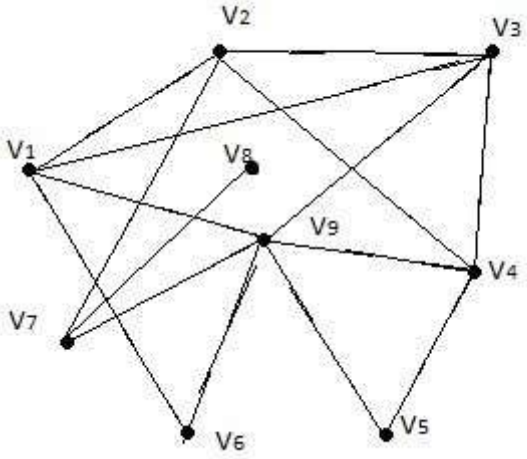
Стягнення:



6) Добуток



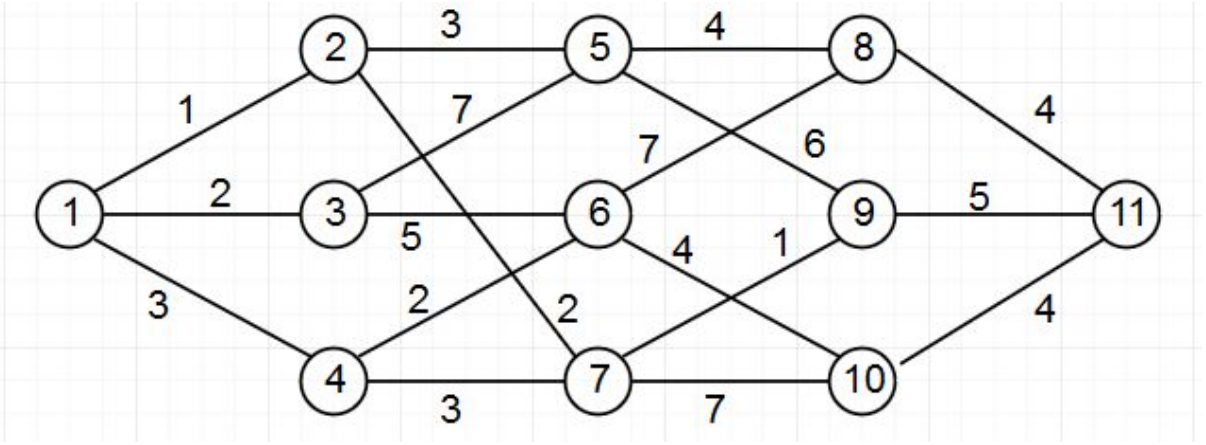
2.**Знайти таблицю суміжності та діаметр графа.**



	V1	V2	V3	V4	V5	V6	V7	V8	V9
V1	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
V2	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
V3	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
V4	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**
V5	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
V6	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
V7	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
V8	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
V9	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**

Діаметр цього графа дорівнює 3, наприклад, V1 -> V8, V3 -> V8.

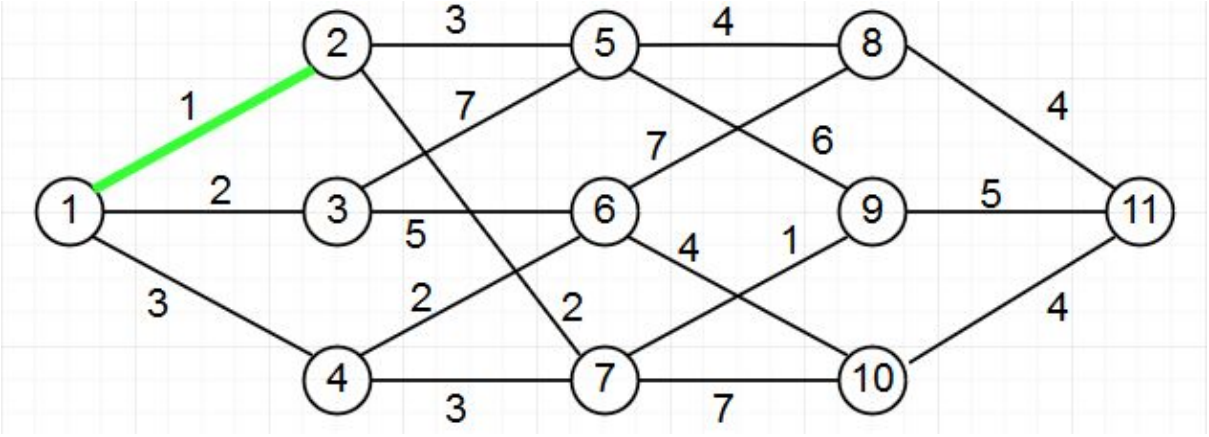
3)Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.



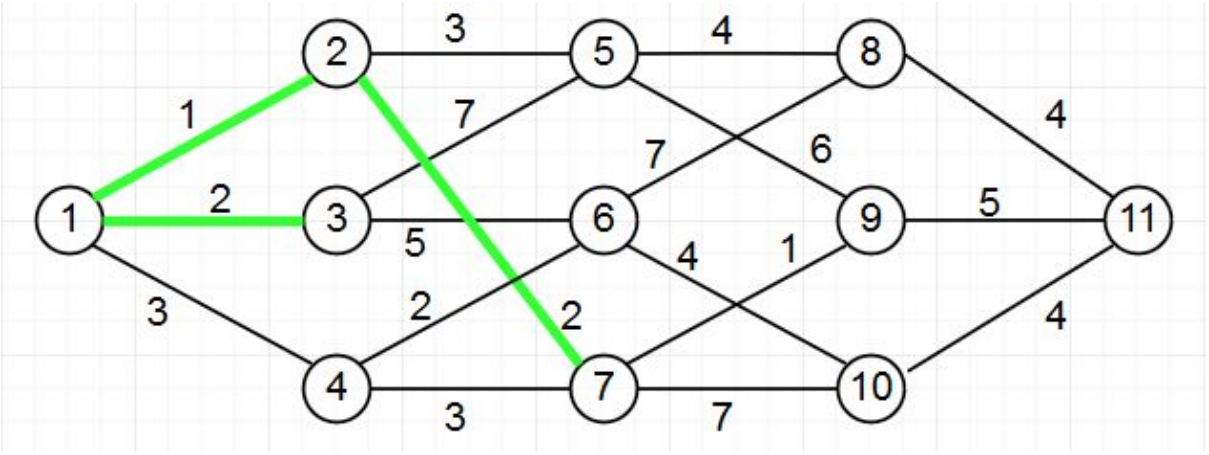
Метод Прима:

Довільно виберем якусь точку, наприклад 1.

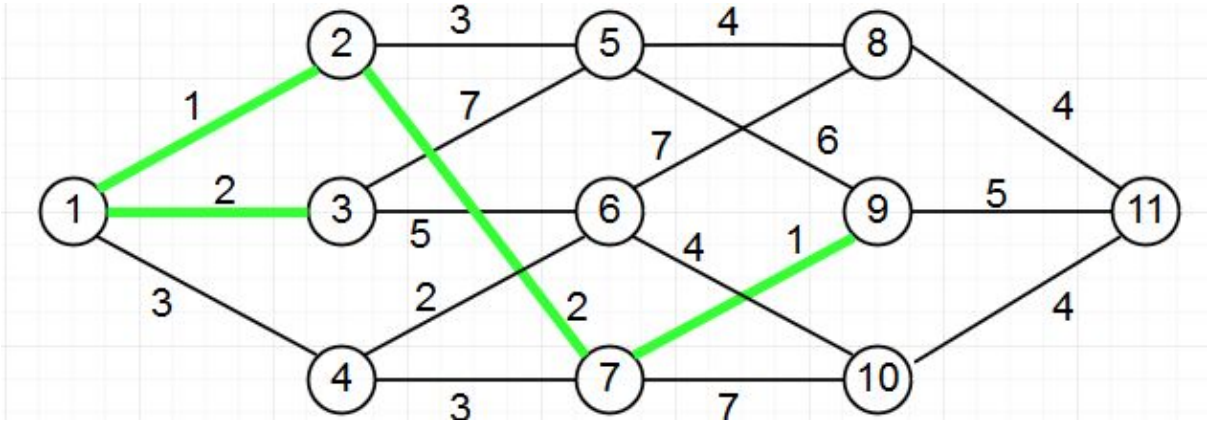
Найкоротша відстань від точки 1 - це відстань до точки 2, яка рівна 1.



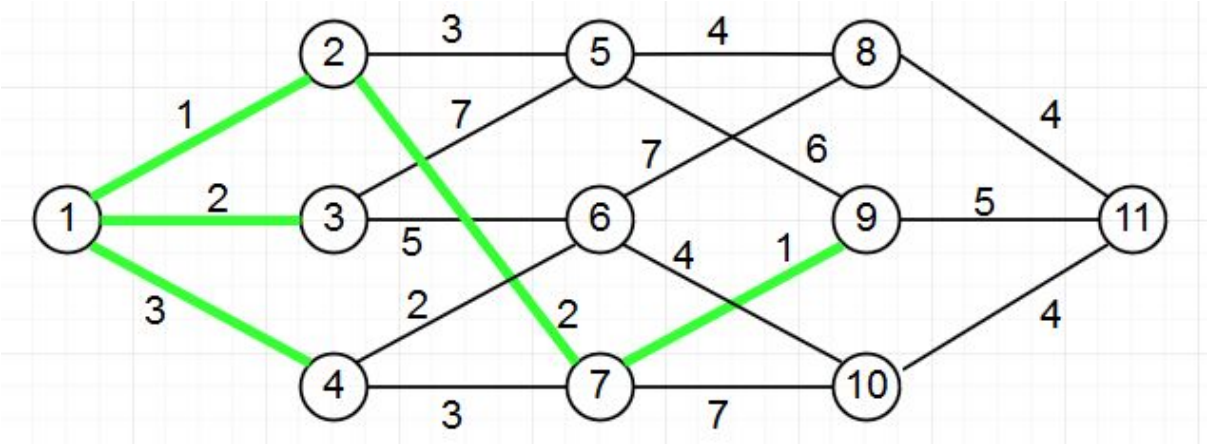
Тепер найкоротша відстань від точки 1 - це відстань до точки 3 (2), а від точки 2 - до точки 7 (також 2).



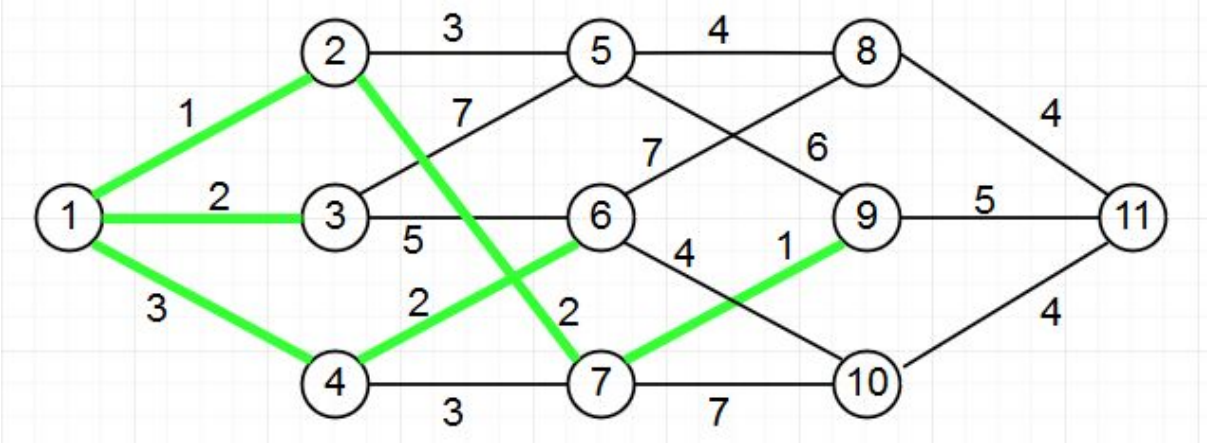
З точок 1, 2, 3 і 7 найкоротшою є відстань від точки 7 до 9 (1).



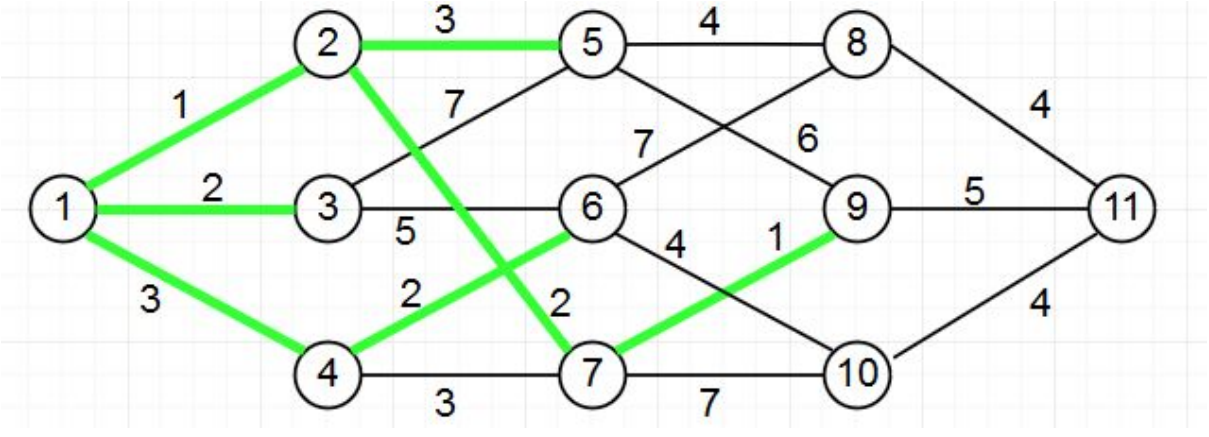
З точок 1, 2, 3, 7 і 9 найкоротшою є відстань від 1 до 4 (3), або від 7 до 4 (3), або від 2 до 5 (3). Довільно вибираємо відстань 1-4.



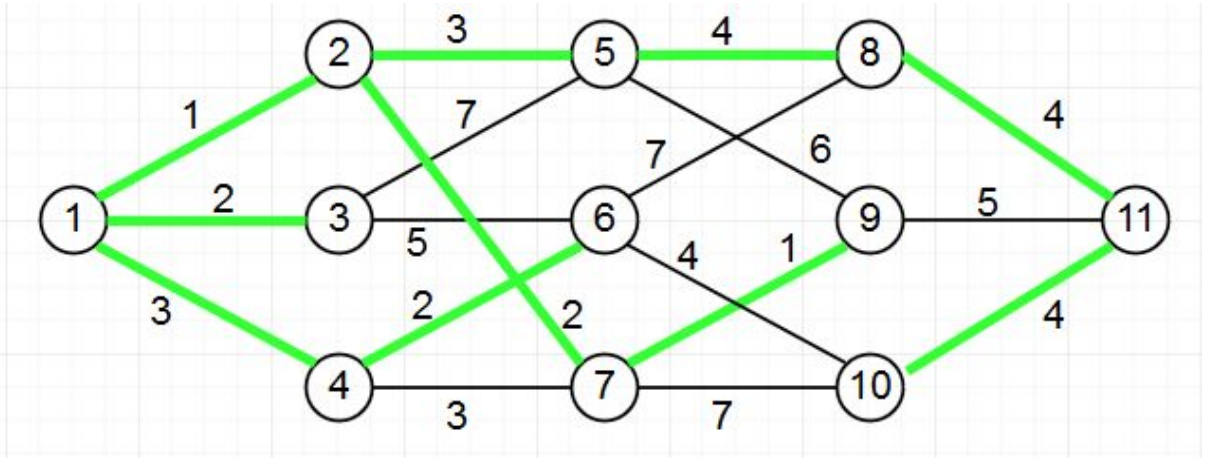
З точок 1, 2, 3, 4, 7 і 9 найкоротшою є відстань 4-6 (2).



З точок 1, 2, 3, 4, 6, 7 і 9, відстань 2-5 (3) - найкоротша, а відстань 4-7 хоч і рівна відстані 2-5, при її виборі призведе до утворення циклу.

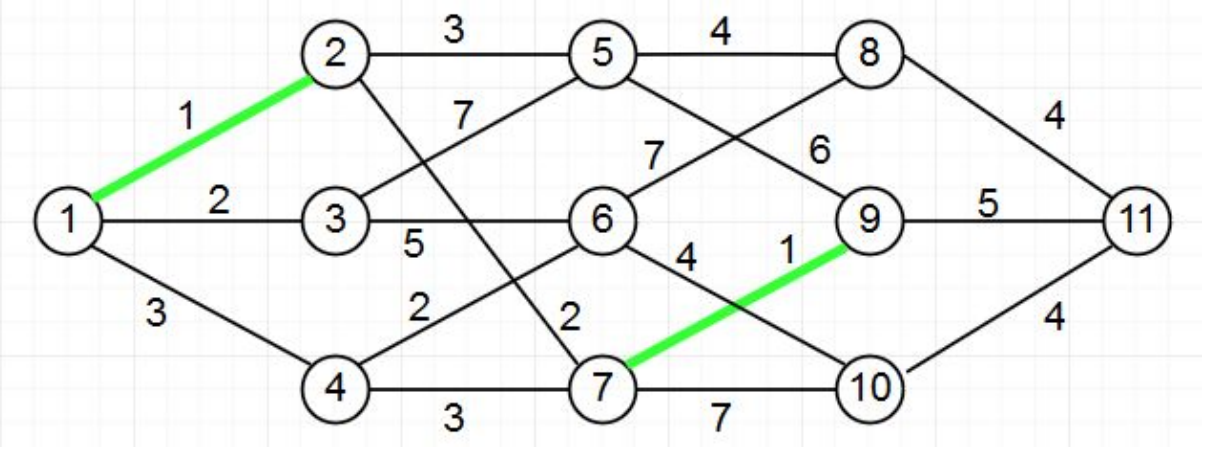


Далі найкоротшими відстанями є 5-8 (4) або 6-10 (4). Довільно вибираємо 5-8, потім 8-11 (4) і 11-10 (4). Мінімальне остове дерево побудоване.

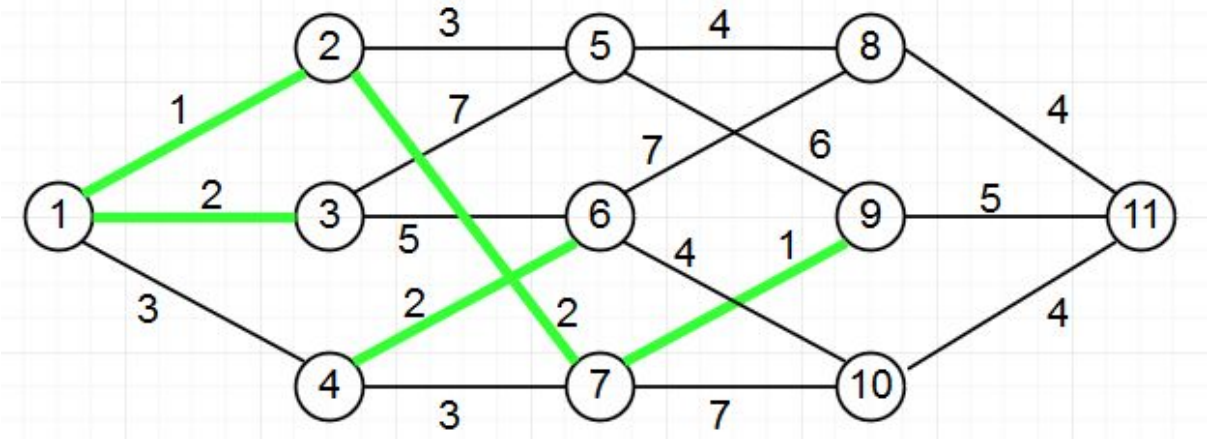


Метод Краскала.

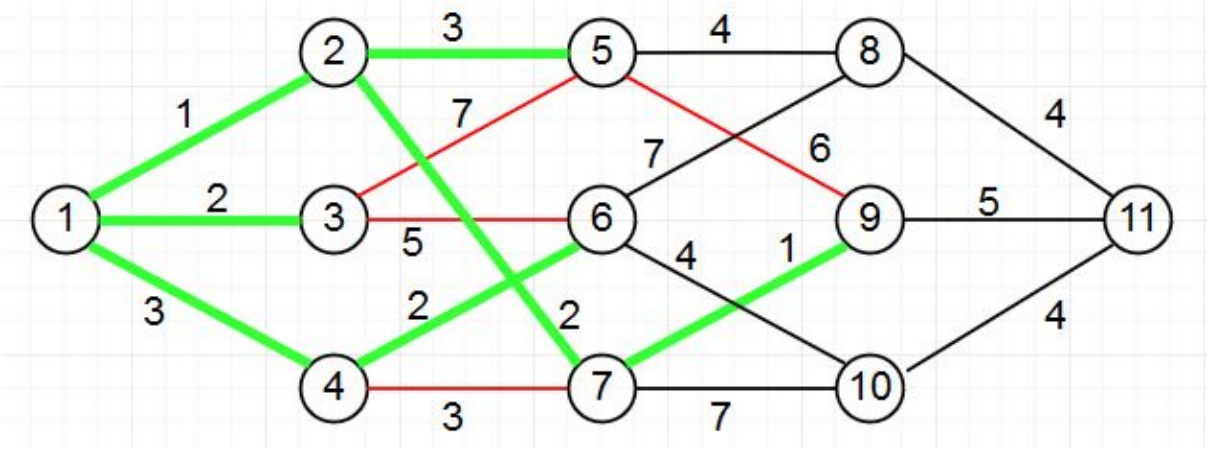
Додамо до остового дерева ребра з вагою 1, тобто 1-2 та 7-9.



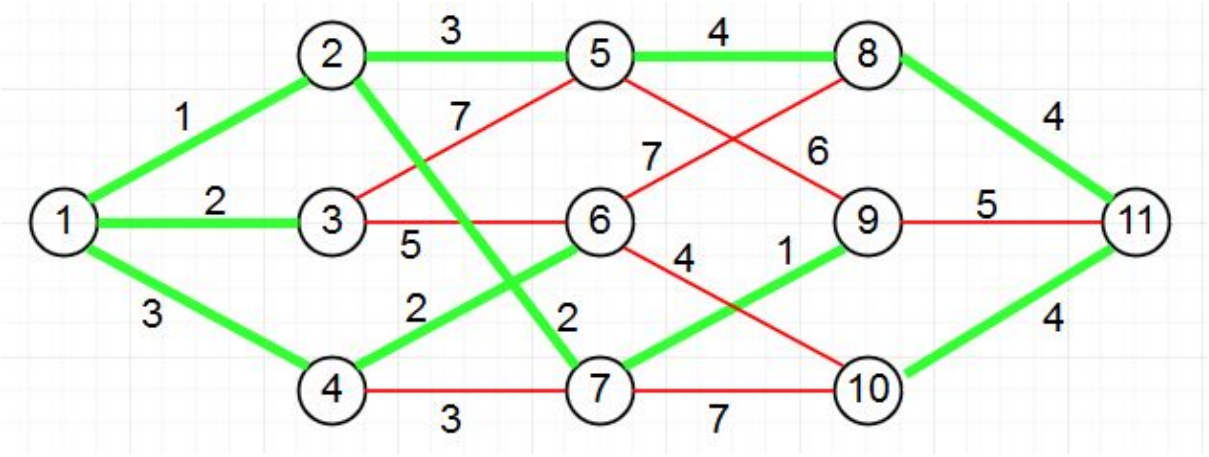
Далі те ж саме з ребрами 1-3, 2-7 та 4-6 (їх ваги 2).



Ребра з вагою 3: Довільно серед таких ребер обираємо 1-4 та 2-5. Тоді ребра 4-7, 3-6, 5-9 та 3-5 призведуть до утворення циклу (їх зафарбуємо червоним).

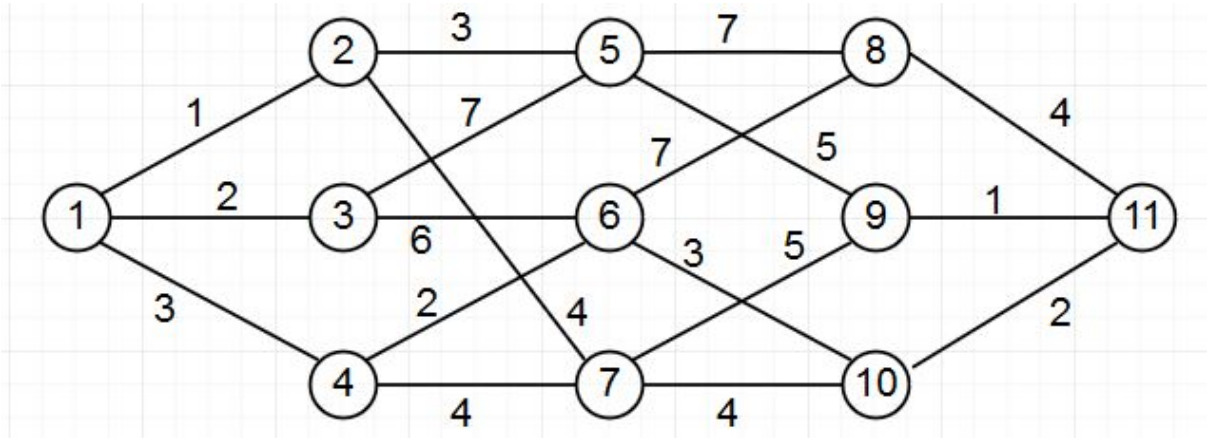


Так само обираємо ребра з вагою 4: 5-8, 8-11 та 11-10. Мінімальне остове дерево побудоване.



Завдання №2. Написати програму, яка реалізує алгоритм знаходження остового дерева мінімальної ваги згідно свого варіанту.

За алгоритмом Прима знайти мінімальне остове дерево графа. Етапи розв'язання задачі виводити на екран. Протестувати розроблену програму на наступному графі:



Код програми:

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.
4.  #define infinity 9999
5.  #define MAX 20
6.
7.  int G[MAX][MAX],spanning[MAX][MAX],n;
8.
9.  int prims();
10.
11. int main()
12. {
13.         int i,j,total_cost;
14.         printf("Enter no. of vertices:"); //21 - maximum
15.         scanf("%d",&n);
16.
17.         printf("\nEnter the adjacency matrix:\n");
18.
19.         for(i=0;i<n;i++)
20.                 for(j=0;j<n;j++)
21.                 {
22.                         printf("G[%i][%i]=", i+1, j+1);
23.                         scanf("%d",&G[i][j]);
24.                 }
25.         printf("Your matrix:\n");
26.         for(i=0; i<n; i++)
27.         {
28.                 for(j=0; j<n; j++)
29.                         printf("%i ", G[i][j]);
30.                 printf("\n");
31.         }
32.
33.         total_cost=prims();
```

```
34.         printf("\nspanning tree matrix:\n");
35.
36.         for(i=0;i<n;i++)
37.         {
38.                 printf("\n");
39.                 for(j=0;j<n;j++)
40.                         printf("%d ",spanning[i][j]);
41.         }
42.
43.         printf("\n\nTotal cost of spanning tree=%d",total_cost);
44.         return 0;
45. }
46.
47. int prims()
48. {
49.         int cost[MAX][MAX];
50.         int u,v,min_distance,distance[MAX],from[MAX];
51.         int visited[MAX],no_of_edges,i,min_cost,j;
52.
53.         //create cost[][] matrix,spanning[][]
54.         for(i=0;i<n;i++)
55.                 for(j=0;j<n;j++)
56.                 {
57.                         if(G[i][j]==0)
58.                                 cost[i][j]=infinity;
59.                         else
60.                                 cost[i][j]=G[i][j];
61.                         spanning[i][j]=0;
62.                 }
63.
64.         //initialise visited[],distance[] and from[]
65.         distance[0]=0;
66.         visited[0]=1;
67.
68.         for(i=1;i<n;i++)
69.         {
70.                 distance[i]=cost[0][i];
71.                 from[i]=0;
72.                 visited[i]=0;
73.         }
74.
75.         min_cost=0;    //cost of spanning tree
76.         no_of_edges=n-1;      //no edges to be added
77.
78.         while(no_of_edges>0)
79.         {
```

```
80.         //find the vertex at minimum distance from the tree
81.                 min_distance=infinity;
82.                 for(i=1;i<n;i++)
83.                         if(visited[i]==0&&distance[i]<min_distance)
84.                         {
85.                                 v=i;
86.                                 min_distance=distance[i];
87.                         }
88.
89.                 u=from[v];
90.
91.         //insert the edge in spanning tree
92.                 spanning[u][v]=distance[v];
93.                 spanning[v][u]=distance[v];
94.                 no_of_edges--;
95.                 visited[v]=1;
96.
97.         //updated the distance[] array
98.                 for(i=1;i<n;i++)
99.                         if(visited[i]==0&&cost[i][v]<distance[i])
100.                        {
101.                                distance[i]=cost[i][v];
102.                                from[i]=v;
103.                        }
104.
105.                min_cost=min_cost+cost[u][v];
106.                }
107.
108.        return(min_cost);
109.    }
```

Користувач вводить матрицю суміжності графа, а програма виводить матрицю суміжності мінімального остового дерева.

Перевірка:

Побудуємо матрицю суміжності для заданого графа:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
V1	0	1	2	3	0	0	0	0	0	0	0
V2	1	0	0	0	3	0	4	0	0	0	0
V3	2	0	0	0	7	6	0	0	0	0	0
V4	3	0	0	0	0	2	4	0	0	0	0

V5	0	3	7	0	0	0	0	7	5	0	0
V6	0	0	6	2	0	0	0	7	0	3	0
V7	0	4	0	4	0	0	0	0	5	4	0
V8	0	0	0	0	7	7	0	0	0	0	4
V9	0	0	0	0	5	0	5	0	0	0	1
V10	0	0	0	0	0	3	4	0	0	0	2
V11	0	0	0	0	0	0	0	4	1	2	0

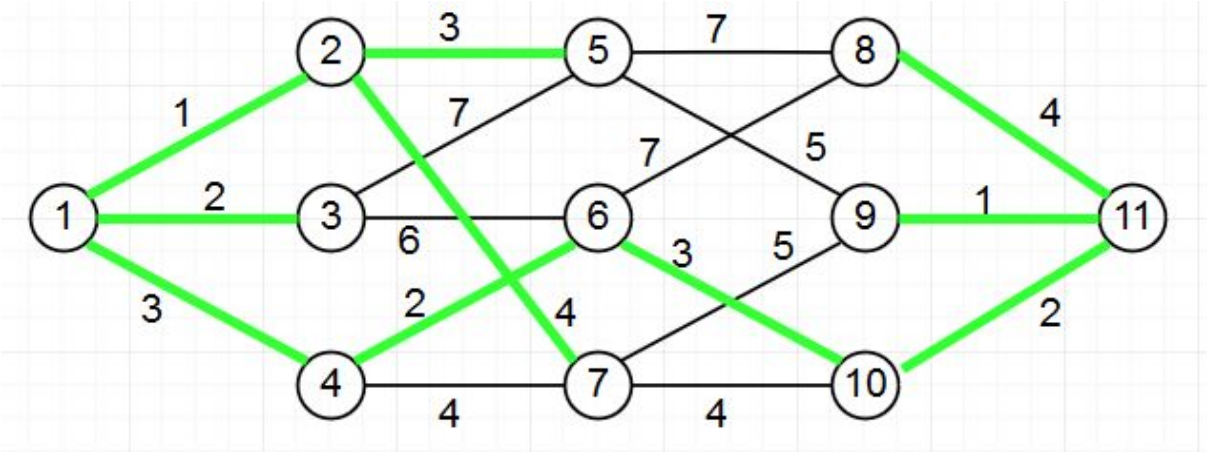
Результат виконання програми:

```
Your matrix:
0 1 2 3 0 0 0 0 0 0 0
1 0 0 0 3 0 4 0 0 0 0
2 0 0 0 7 6 0 0 0 0 0
3 0 0 0 0 2 4 0 0 0 0
0 3 7 0 0 0 0 7 5 0 0
0 0 6 2 0 0 0 7 0 3 0
0 4 0 4 0 0 0 0 5 4 0
0 0 0 0 7 7 0 0 0 0 4
0 0 0 0 5 0 5 0 0 0 1
0 0 0 0 0 3 4 0 0 0 2
0 0 0 0 0 0 0 4 1 2 0

spanning tree matrix:

0 1 2 3 0 0 0 0 0 0 0
1 0 0 0 3 0 4 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0
3 0 0 0 0 2 0 0 0 0 0
0 3 0 0 0 0 0 0 0 0 0
0 0 0 2 0 0 0 0 0 3 0
0 4 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 4
0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 3 0 0 0 0 2
0 0 0 0 0 0 0 4 1 2 0
```

Матриця-результат програми відповідає такому остовому дереву:



А таке дерево - це мінімальне остове дерево заданого графа.

Висновок: Я навчився використовувати на практиці і програмно реалізовувати алгоритми Пріма і Краскала для пошуку мінімального остового дерева заданого графа. Програмна реалізація була перевірена і працювала коректно.