МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра систем штучного інтелекту

Лабораторна робота №4

З дисципліни «Дискретна математика»

Основні операції над графами. Знаходження кістякового дерева за алгоритмом Прима та Краскела

Виконав:

студент групи КН-110

Єлечко Олег Андрійович

Викладач:

Мельникова Наталія Іванівна

Львів – 2018р.

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

1. Теорія

Графом G називається пара множин (V, E), де V – множина вершин v, а E – множина упорядкованих або неупорядкованих пар $e = \{v', v''\}$, що $v' \in V$ & $v'' \in V$. **Неорієнтований та орієнтований граф** (орграф) відрізняються упорядкованістю пар $e \in E$.

Кратними ребрами називають ребра, що зв'язують одні і ті ж множини. Ребро, що входить у ту ж вершину, звідки й виходить, називається **петлею**.

Мультиграф – граф, що має кратні ребра. **Псевдографи** мають петлі, а **прості графи** не мають ні кратних ребер, ні петель.

Будь-яке ребро є **інцидентним** вершинам (v', v''), які воно з'єднує. Дві вершини називають **суміжними**, якщо вони належать до спільного ребра й несуміжними у протилежному випадку. **Степенем** вершини v називається кількість інцидентних їй ребер.

Граф, що не має ребер, називається **пустим**. Граф, що не має вершин – **нульграфом**. Вершина графа, що не інцидентна до жодного з ребер є **ізольованою**.

Вершина із одиничним степенем є **листком**.

Граф $G' = (V', E')$ є **підграфом** графа $G = (V, E)$, якщо $V' \subseteq V$ і $E = \{(v', v'') \mid v' \in V$ & $v'' \in V$ & $(v', v'') \in E\}$. G' також називається **кістяковим підграфом**, якщо виконано умову $V = V$ & $E \subseteq E$.

Над графами можна виконувати деякі **операції**.

1. **Вилученням ребра** $e \in E$ можна отримати новий граф $G' = (V, E \setminus \{e\})$.

2. **Доповненням графа** $G = (V, E)$, можна отримати новий граф $G' = (V', E')$, де $E = \{(v_1, v_2) | (v_1, v_2) \notin E\}$;
3. **Об'єднанням графів** $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ є новий граф $G = (V, E)$, в якому $V = V_1 \cup V_2$, $E = E_1 \cup E_2$;
4. **Кільцевою сумою** графів $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ є граф $G = (V, E)$, в якому $V = V_1 \cup V_2$, а $E = E_1 \Delta E_2$;
5. **Розщепленням вершин** є операція, за якої на місці однієї вершини з'являється дві, інцидентні одна до одної. Нові вершини довільно успадковують інцидентність зі старими вершинами.
6. **Стягненням ребра** $(\boldsymbol{a}, \boldsymbol{b})$ є операція, при якій це ребро зникає, і замість точок a та b виникає c, що успадковує їхню інцидентність.
7. **Добутком графів** $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ є граф $G = G_1 \times G_2 = (V, E)$, в якого $V = V_1 \times V_2$, а вершини (v_1', v_2') та (v_1'', v_2'') суміжні тільки якщо $(v_{1'} = v_{1''})$ & $(v_{2'}, v_{2''}) \in E_2$ або $(v_{2'} = v_{2''})$ & $(v_{1'}, v_{1''}) \in E_1$.

Матрицею суміжності $R = [r_{ij}]$ графа $G = (V, E)$ є квадратна матриця порядку

$|V|$, елементи r_{ij} якої визначаються за формулою $r_{ij} = \{1, \ (v^i, v^j) \in E.$
$0, (v_i, v_j) \notin E$

Діаметром зв'язного графа є максимально можлива довжина між двома його вершинами.

В неорієнтованому графі $G = (V, E)$ **маршрутом** довжини j є послідовність $M = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{j-2}, v_{j-1}), (v_{j-1}, v_{j-2})\}$, що складається з ребер $e \in E$. При цьому кожні два сусідні ребра мають спільну кінцеву вершину. Маршрут називається **ланцюгом**, якщо всі його ребра різні. Відкритий ланцюг називається **шляхом**, якщо всі його вершини різні. Замкнений ланцюг називається **циклом**, якщо всі його вершини, за винятком кінцевих, є різними. Шлях і цикл називаються **гамільтоновими**, якщо вони проходять через усі вершини графа.

Для пошуку кістякового дерева мінімальної ваги у зв’язному графі можна використовувати алгоритми Прима та Краскела.

1.1 Алгоритм Прима

Нехай будуємо дерево $G' = (V',E')$, де $V' = E' = \emptyset$. Над графом $G = (V, E)$ алгоритм Прима буде мати такий порядкодій:

1. Вибрати довільну точку $v \in V$, і додати її у V'.
2. Вибрати перше найлегше ребро $e = (a, b) \in E$, де $a \in V'$, $b \in V$, що не утворює циклів.
   a. Якщо такого ребра не знайдено – завершити роботу, повернувши дерево (V', E').
3. Додати точку b та ребро e у G'.
4. Повернутись до кроку 2.

Якщо граф заданий матрицею суміжності, то ціну цього алгоритму можна оцінити як $O(|V|^2)$. Якщо задати граф списком суміжності, ціною алгоритму буде $O(|E| \log|V|)$.

1.2 Алгоритм Краскела

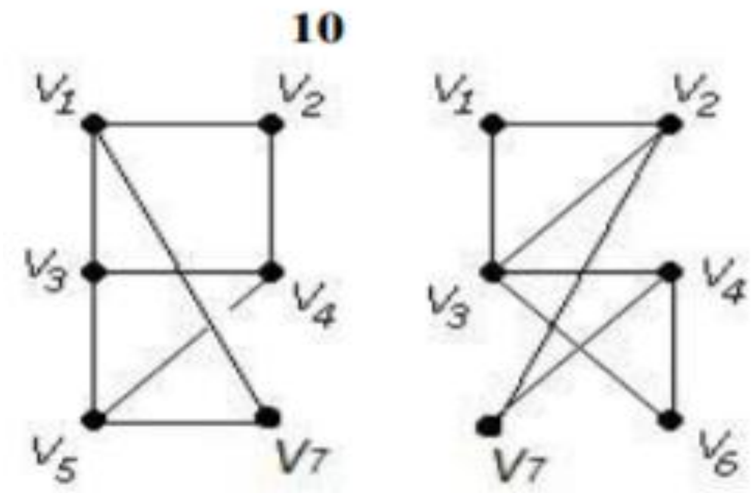
Нехай будуємо дерево $G' = (V',E')$, де $V' = E' = \emptyset$. Над графом $G = (V, E)$ алгоритм Краскела буде мати такий порядкодій:

1. Вибрати найлегше ребро $e = (u, v) \in E$ таке, щоби воно не утворювало циклів.
   a. Якщо таких ребер немає – завершити роботу, повернувши дерево (V', E')
2. Додати точки u і v та ребро (u, v) у G'.
3. Повернутись до кроку 1.

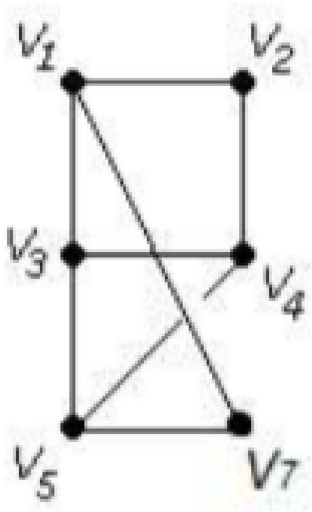
Ціна цього алгоритму - $O(|E| \log|E|)$

2. Практична частина

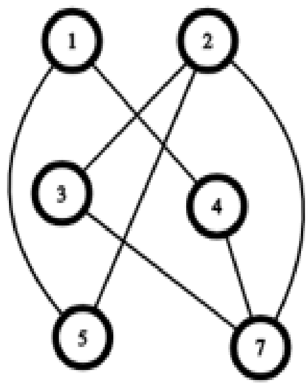
2.1 Виконати операції над графами:



а) Знайти доповнення до першого графу

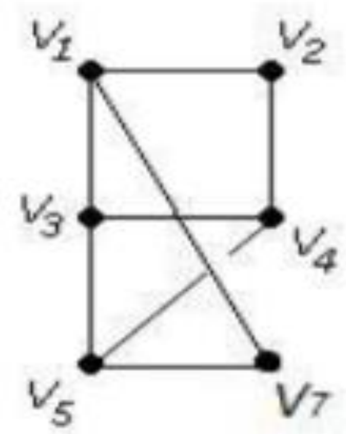


G_1

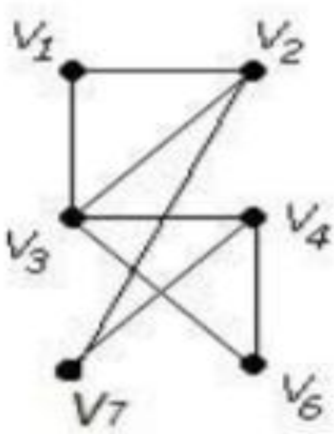


б) Знайти об’єднання графів G_1 та G_2

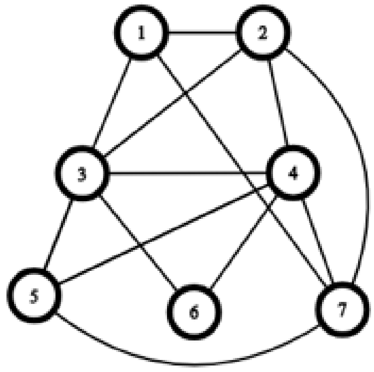
10



G_1



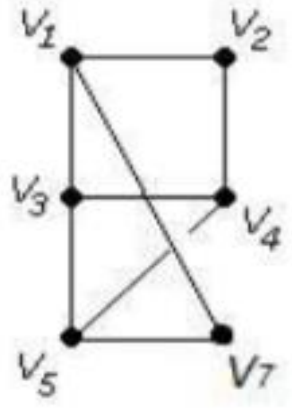
G_2



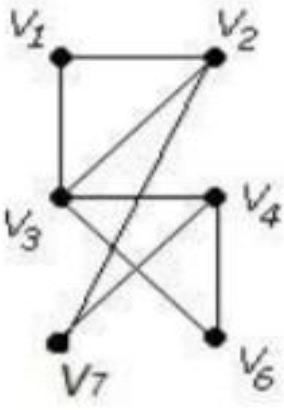
$G_1 \cup G_2$

в) Знайти кільцеву суму G_1 та G_2

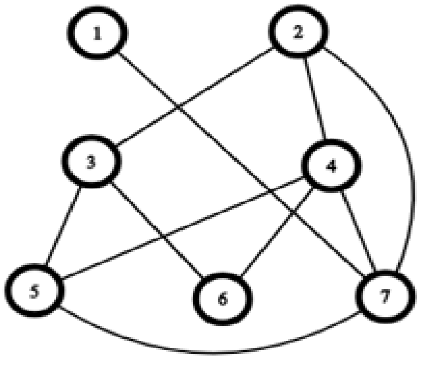
10



G_1

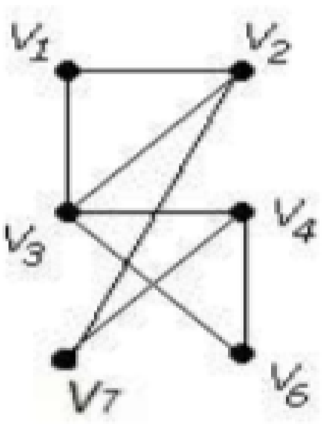


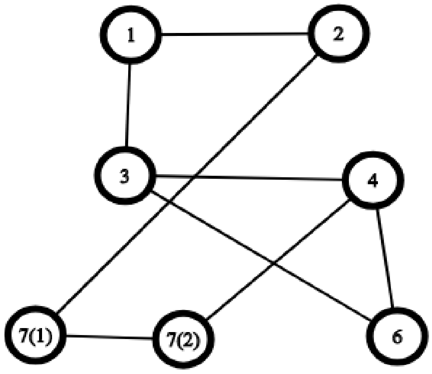
G_2



$G_1 \oplus G_2$

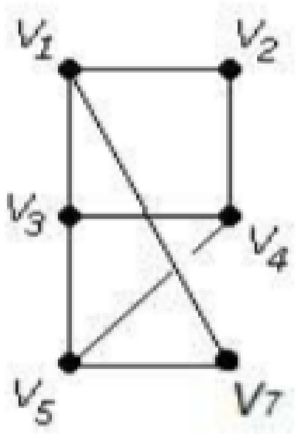
г) Розщепити довільну вершину в другому графі

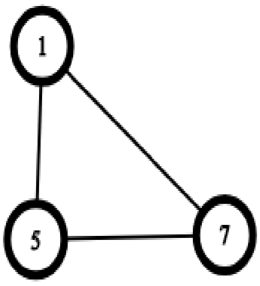


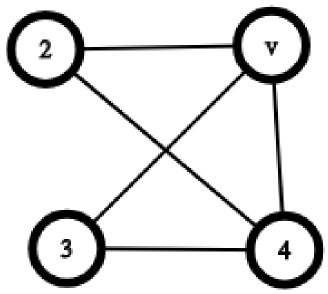


G_2 $\qquad$ $G_2\ (V_7 \to V_{7(2)})$

ґ) Виділити підграф $\mathrm{V} \subset G_1$, що $|V_A| = 3$, і знайти різницю $G_1 \backslash \mathrm{V}$.

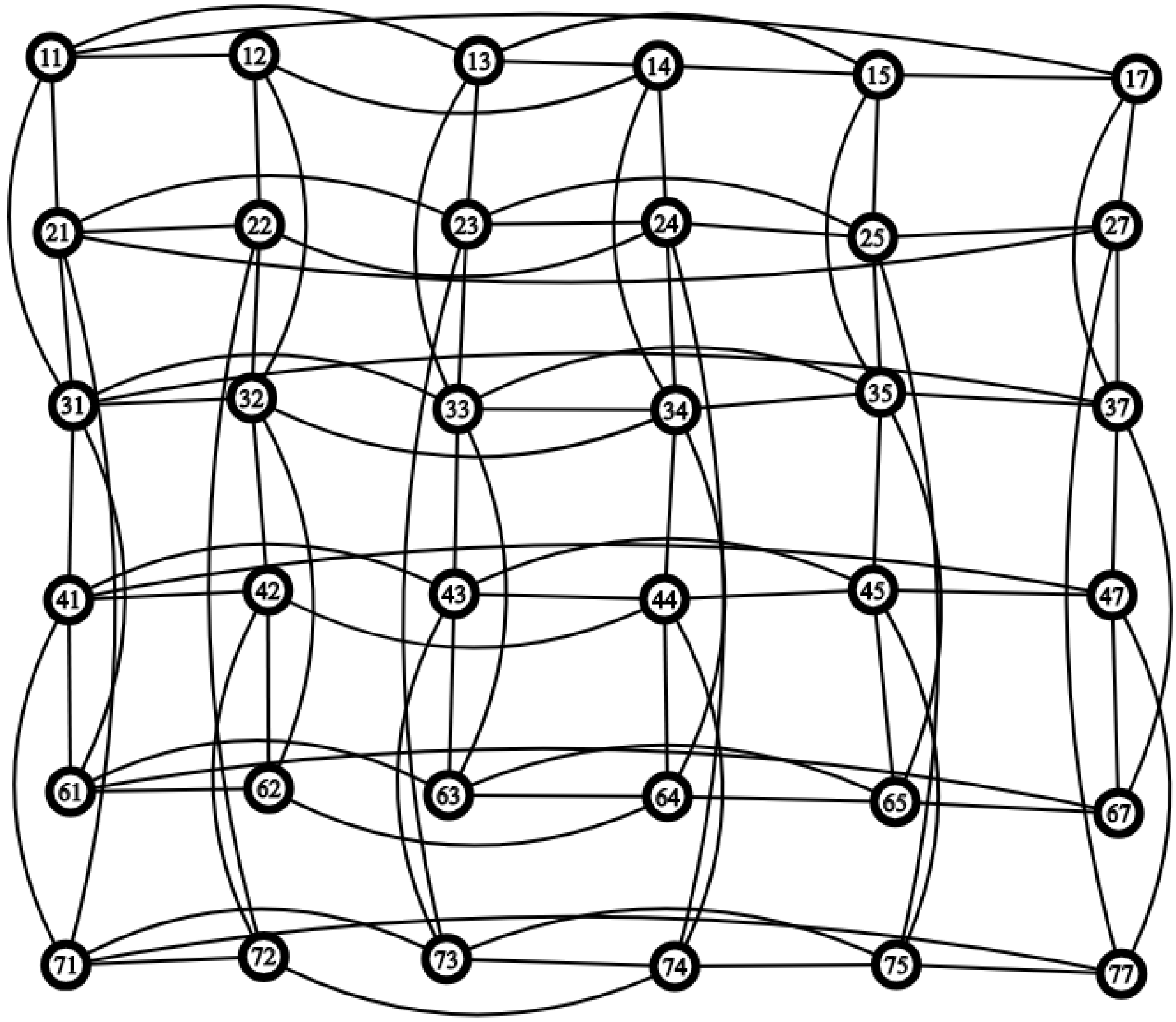




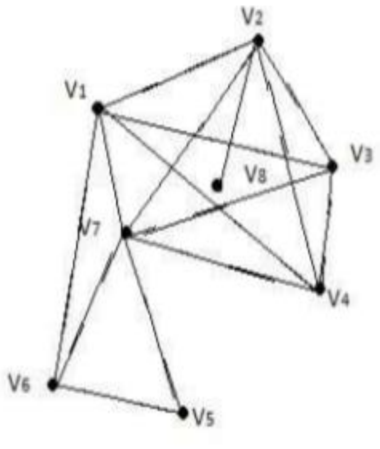


G_1 $\qquad$ $\mathrm{V} \subset G_1$ $\qquad$ $G_1 \backslash A$

д) Знайти добуток графів G_1 та G_2



2.2 Таблиця суміжності та діаметр



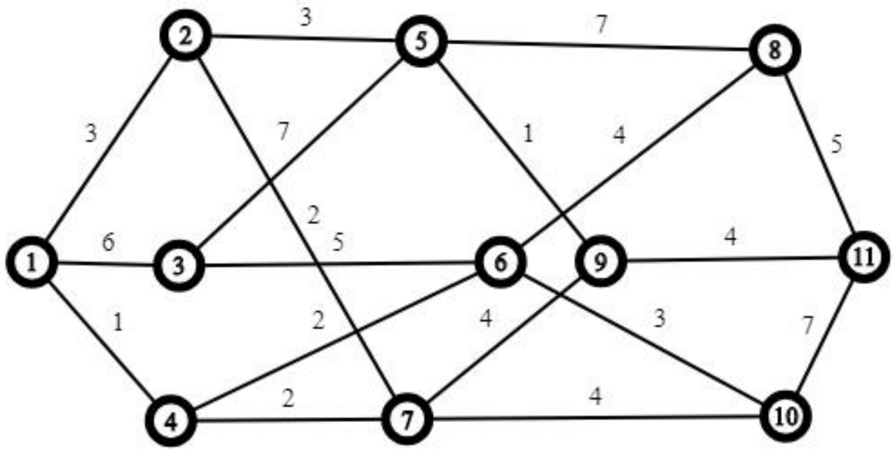
$$R = \begin{array}{c|cccccccc} & V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 & V_8 \\ \hline V_1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ V_2 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ V_3 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ V_4 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ V_5 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ V_6 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ V_7 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ V_8 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

У графі існує найкоротший шлях довжиною в 4 ребра. Тому діаметр графа:

$\max(V6V7, V7V3, V3V2, V2V8) = 4$

$G = (V, E)$

2.3 Знайти кістякове дерево алгоритмом Прима



Крок 0. Задано граф $G = (V, E)$. Потрібно утворити кістякове дерево $G' = (V',$ $E')$.

Крок 1. Розпочати алгоритм з вибору довільної точки з V, і додати її у V'.

Точка, яку розглядаємо – 1.

Крок 2. Вибрати найлегше ребро $e = (u, v) \in E, u \in V', v \in V$, помістити його у E', вершину v помістити у V'. Точка, яку розглядаємо тепер – 4.

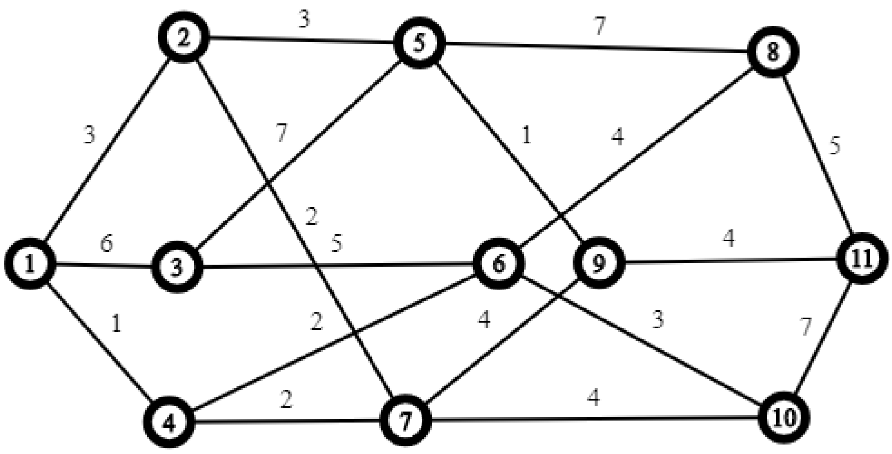
Крок 3. Наступне найлегше ребро з’єднує точки 4 і 7.

Крок 4. Наступне найлегше ребро з’єднує точки 4 та 6.

Кроки 5-10. Повторити дії аналогічно попереднім крокам, уникаючи ребер, що створять зациклення. Після останнього відгалуження у дереві, у V не залишиться жодної вершини. Вихід з алгоритму. Ціна утвореного дерева: 27.

2.4 Побудувати кістякове дерево алгоритмом Крескала

Крок 0. Задано граф $G = (V, E)$. Потрібно утворити кістякове дерево $G' = (V', E')$. Вершини та ребра, що було додано до G', буде обведено товстішою лінією.



Кроки 1-2. Посортувати ребра $e \in E$, і почергово додавати їх у E'. На перших двох кроках буде вибрано ребра вагою в 1 од.

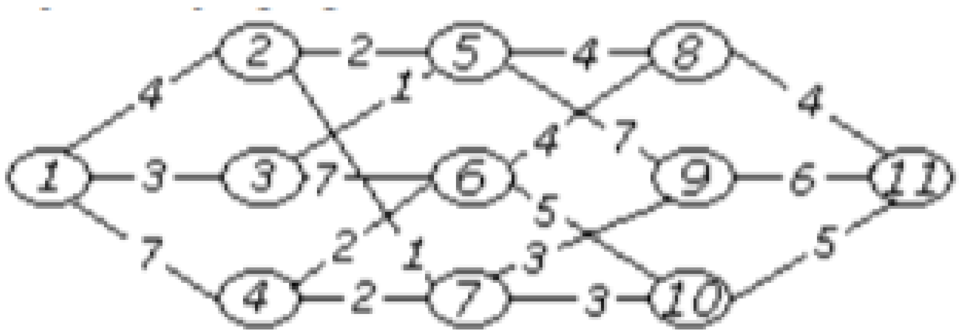
Кроки 2-5. На наступних 3 кроках буде вибрано ребра з вагою в 2 од.

Кроки 5-10. На наступних 5 кроках буде вибрано наступні ребра, окрім тих, що утворюють цикли.

Крок 11. При подальшому виборі ребер не знайдено жодного такого, що не утворює цикли. Додати в множину V' всі з'єднані вершини. Побудова закінчена. Ціна утвореного дерева: 27.

3. Комп'ютерна програма

Потрібно написати комп'ютерна програму, що, використовуючи алгоритм Крускала, знайде мінімальне кістякове дерево для такого графа:



Код реалізації:

```
#include <stdio.h>
#define MAX 30
#define VERT 11

typedef struct edge
{
    int u,v,w;
}edge;

typedef struct edgelist
{
    edge data[MAX];
    int n;
}edgelist;

edgelist elist;
//matrix of adjacency
int G[11][11] = { {0,4,3,2,0,0,0,0,0,0,0},
                  {4,0,0,0,2,0,1,0,0,0,0},
                  {3,0,0,0,6,7,0,0,0,0,0},
                  {2,0,0,0,0,2,7,0,0,0,0},
                  {0,2,6,0,0,0,0,4,7,0,0},
                  {0,0,7,2,0,0,0,4,0,3,0},
                  {0,1,0,7,0,0,0,0,5,5,0},
```

```
                  {0,0,0,0,4,4,0,0,0,0,4},
                  {0,0,0,0,7,0,5,0,0,0,1},
                  {0,0,0,0,0,3,5,0,0,0,3},
                  {0,0,0,0,0,0,0,4,1,3,0}};
int n;
edgelist spanlist;

void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();

int main()
{
    int i,j,total_cost;
    printf("         Kruskal Algorithm         \n");
    n = VERT;
    kruskal();
    print();
    }

void kruskal()
{
    int belongs[MAX],i,j,cno1,cno2;
```

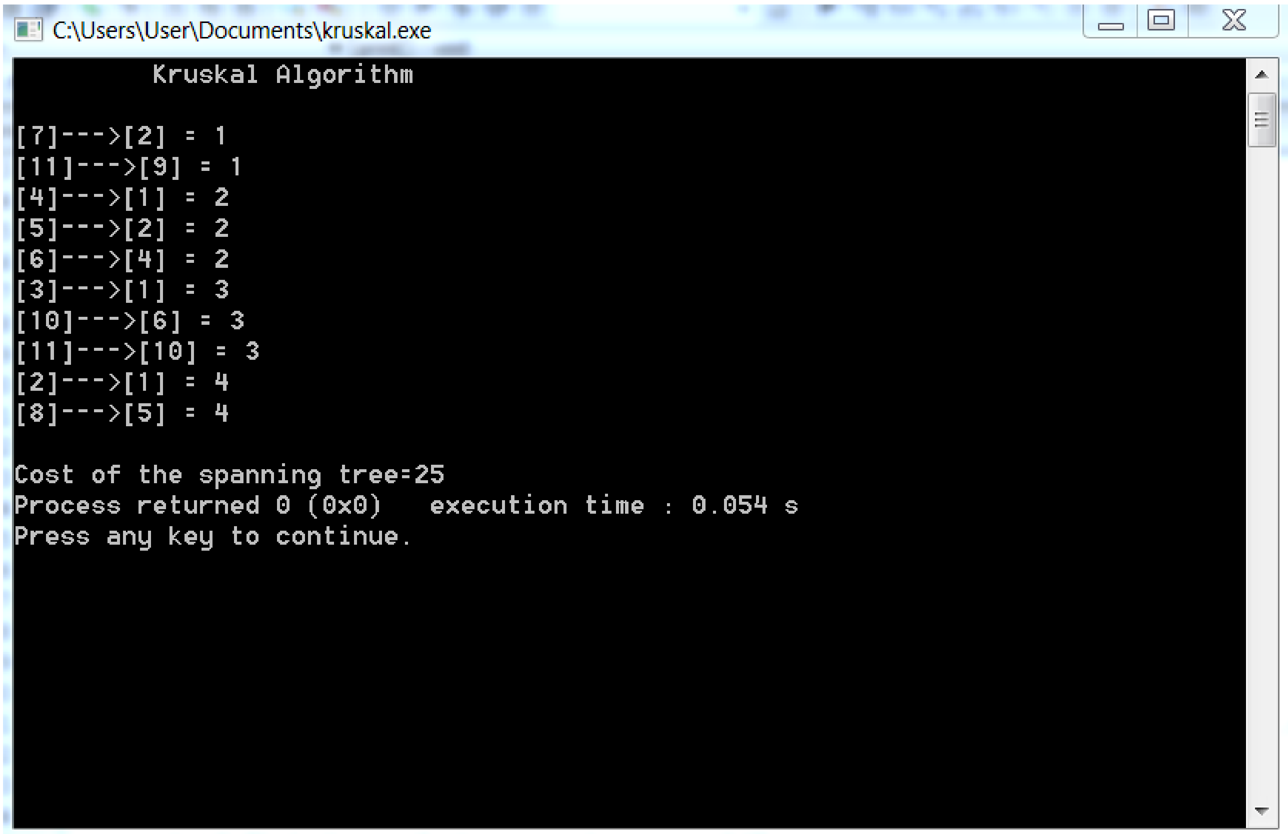
```
elist.n=0;

for(i=1;i<n;i++)
    for(j=0;j<i;j++)
    {
        if(G[i][j]!= 0)
        {
            elist.data[elist.n].u=i;
            elist.data[elist.n].v=j;
            elist.data[elist.n].w=G[i][j];
            elist.n++;
        }
    }

sort();

for(i=0;i<n;i++)
    belongs[i]=i;

spanlist.n=0;

for(i=0;i<elist.n;i++)
{
    cno1=find(belongs,elist.data[i].u);
    cno2=find(belongs,elist.data[i].v);
```

```
        if(cno1!=cno2)
        {
            spanlist.data[spanlist.n]=elist.data[i];
            spanlist.n=spanlist.n+1;
            union1(belongs,cno1,cno2);
        }
    }
}

int find(int belongs[],int vertexno)
{
    return(belongs[vertexno]);
}

void union1(int belongs[],int c1,int c2)
{
    int i;

    for(i=0;i<n;i++)
        if(belongs[i]==c2)
            belongs[i]=c1;
}

void sort()
```

```
{
    int i,j;
    edge temp;

    for(i=1;i<elist.n;i++)
        for(j=0;j<elist.n-1;j++)
            if(elist.data[j].w>elist.data[j+1].w)
            {
                temp=elist.data[j];
                elist.data[j]=elist.data[j+1];
                elist.data[j+1]=temp;
            }
}

void print()
{
    int i,cost=0;

    for(i=0;i<spanlist.n;i++)
    {
        printf("\n[%d]--->[%d] =
%d",spanlist.data[i].u+1,spanlist.data[i].v+1,spanlist.data[i]
.w);
        cost=cost+spanlist.data[i].w;
    }
```

```
    printf("\n\nCost of the spanning tree=%d",cost);
}
```

3.2 Приклад виконання



4. Висновок

Під час виконання цієї лабораторної роботи я навчився використовувати алгоритми Прима та Краскела, ознайомився з графовою структурою.