МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота № 4

з дисципліни

«Дискретна математика»

Виконав:
студент групи КН-114
Долінський А.Г.
Викладач:
Мельникова Н.І.

Львів – 2019р.

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, ..., $n = \upsilon$; $V = \{\upsilon\}$, 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 (оргграф)* – 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 e *інцидентне* двом вершинам (v', v''), які воно з'єднує. У свою чергу вершини (v', v'') інцендентні до ребра e . 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.

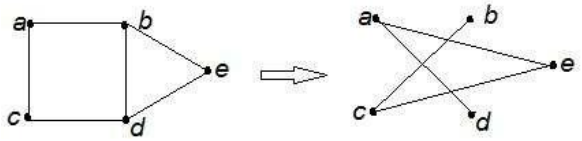
Граф, який не має ребер називається *пустим графом*, *нуль- графом*. Вершина графа, яка не інцедентна до жодного ребра, називається *ізольованою*.

Вершина графа, яка інцедентна тільки до одного ребра, називається *звисаючою*.

Операції над графами

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

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 $(v_i, v_j) \notin E$).



G — $\bar{G}$

Об'єднання 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 $E = E_1 \cup E_2$.

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 $E = E_1 \Delta E_2 = (E_1 \cup E_2) \setminus (E_1 \cap E_2)$.

Розщеплення (роздвоєння) вершини графа. Нехай υ – вершина графа $G = (V, E)$.

Множину усіх суміжних з нею вершин довільним чином розділимо на дві множини $N_1(\upsilon)$ та $N_2(\upsilon)$, таких що $N_1(\upsilon) \cup N_2(\upsilon) = V$. Видалимо вершину υ, разом з інцидентними їй ребрами, додамо дві нові вершини υ_1 та υ_2, які з'єднані ребром (υ_1, υ_2).

Вершину υ_1 з'єднаємо ребром із кожною вершиною множини $N_1(\upsilon)$, а вершину υ_2 – з кожною вершиною множини $N_2(\upsilon)$.

Таким чином з графа G отримаємо новий граф G_v^*.

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).

Добутком графів $G_1=(V_1,E_1)$ та $G_2=(V_2, E_2)$, називається граф $G = G_1 \times G_2$, у якого $V = V_1 \times V_2$, а множина ребер визначається таким чином: вершини $(u1, \upsilon1)$ та $(u2, \upsilon2)$ суміжні у G тоді і тільки тоді, коли $u1 = u2$, а $\upsilon1$ та $\upsilon2$ – суміжні у вершині G_2, або $\upsilon1 = \upsilon2$, а $u1$ та $u2$ – суміжні у вершині G_1.

Таблицею (матрицею) суміжності $R=[r_{ij}]$ графа $G = (V, E)$ називається квадратна матриця порядку n (n – число вершин графа), елементи якої $r_{i,j}$ (i=1, 2, …n; j=1,2, …n) визначаються наступним чином:

$$r_{ij} = \begin{cases} 1, \text{якщо існує дуга з } v_i \text{ в } v_j \\ 0, \text{в іншому випадку} \end{cases}$$

Матриця суміжності повністю визначає структуру графа.

Ексцентриситет вершини графа – відстань до максимально віддаленої від неї вершини. Для графа, для якого не визначена вага його ребер, відстань визначається у вигляді числа ребер.

Радіус графа – мінімальний ексцентриситет вершин.

Діаметр графа – максимальний ексцентриситет вершин.

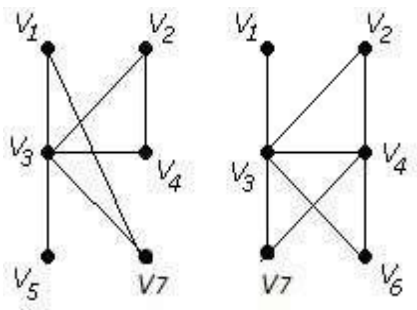
Діаметром зв'язного графа називається максимально можлива довжина між двома його вершинами.

Додаток №1

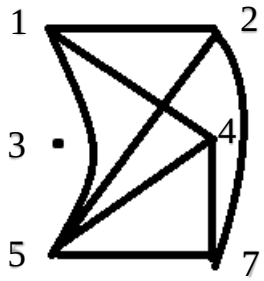
Варіант №7

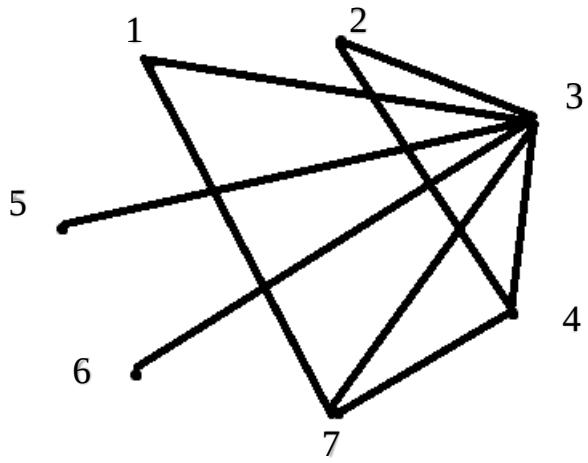
1. Виконати наступні операції над графами:

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 A, що складається з 3-х вершин в G1 і знайти стягнення A в G1 (G1\ A),
6) добуток графів.

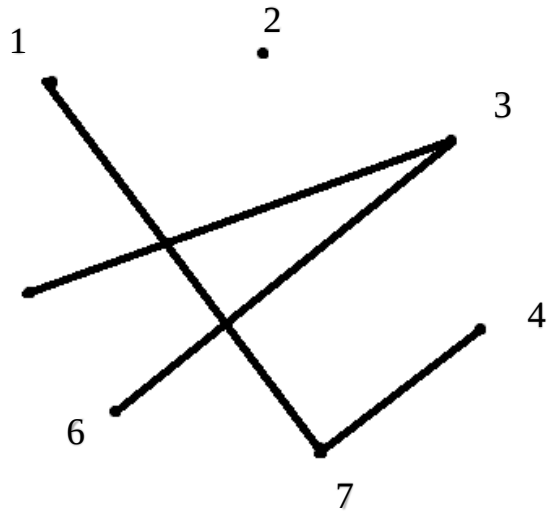


1)



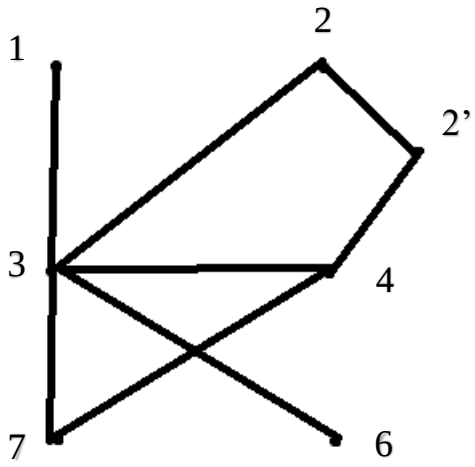


2)

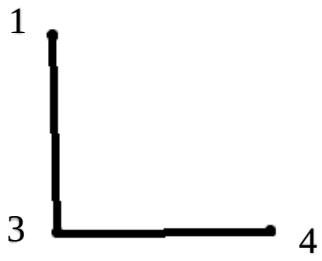


3)

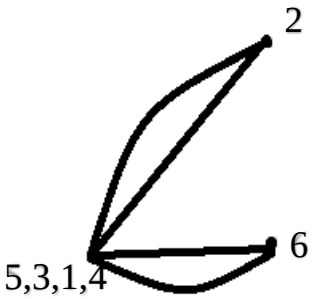
4) Розщепимо вершину 2.



5)

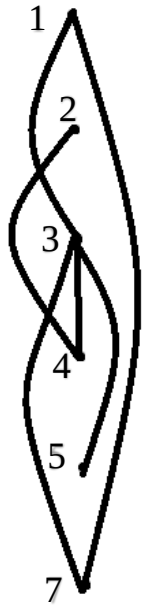


підграф А



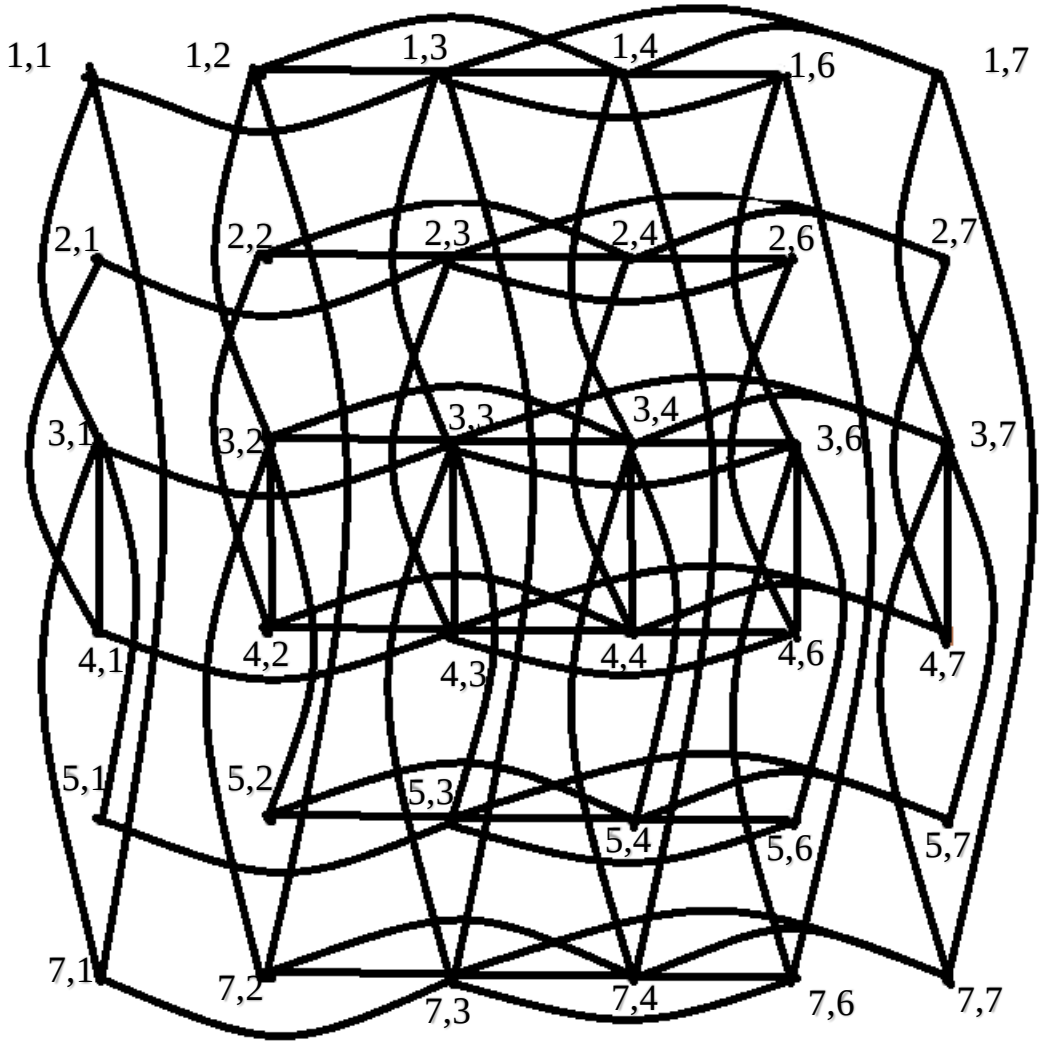
6)

G1


1
2
3
4
5
7

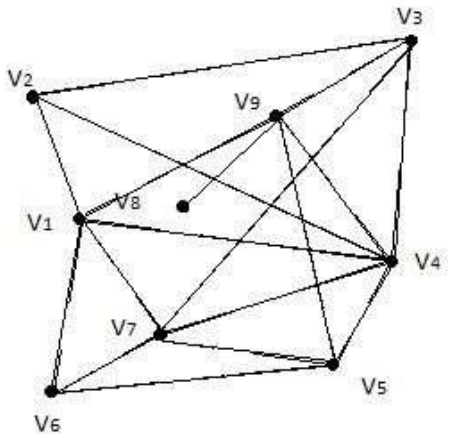
G2


1
2
3
4
6
7


1,1
1,2
1,3
1,4
1,6
1,7
2,1
2,2
2,3
2,4
2,6
2,7
3,1
3,2
3,3
3,4
3,6
3,7
4,1
4,2
4,3
4,4
4,6
4,7
5,1
5,2
5,3
5,4
5,6
5,7
7,1
7,2
7,3
7,4
7,6
7,7

G1 x G2

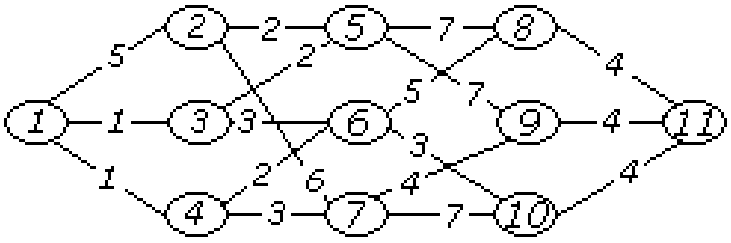
2. Знайти таблицю суміжності та діаметр графа.



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

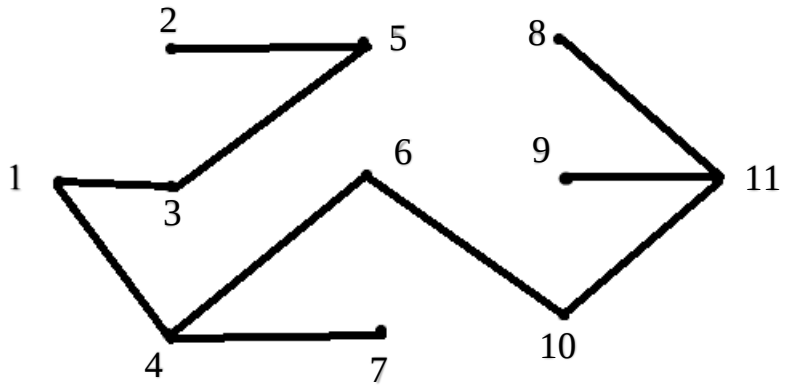
Діаметр графа = 6.

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



1) V(G) = {1, 3, 4, 5, 6, 2, 7, 10, 11, 9, 8}

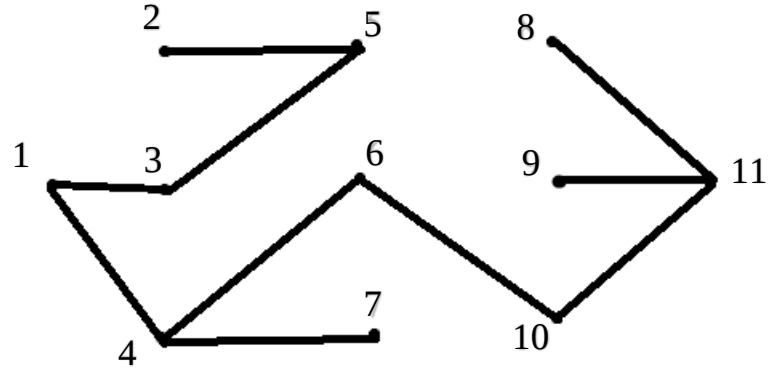
E(G) ={(1,3), (1,4), (3,5), (4,6), (5,2), (4,7), (6,10), (10, 11), (11, 9), (11, 8)}



Метод Прима.

1) E(G) = {(1,3), (1,4), (2,5), (3,5), (4,6), (4,7), (6,10), (10,11), (11,9), (11,8)}
V(G) = {1, 3, 4, 2, 5, 6, 7, 10, 11, 9, 8}

(7,9) – утв. цикл;
(1,2), (6,8) – утв. цикл;
(7,2) – утв. цикл;
(7,10), (5,9), (5,8) – утв. цикл;



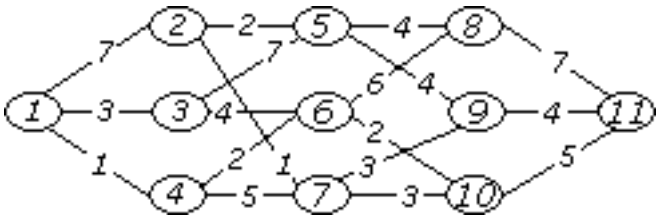
Метод Краскала.

Додаток №2

Варіант №7

Написати програму, яка реалізує алгоритм знаходження остового дерева мінімальної ваги згідно свого варіанту.

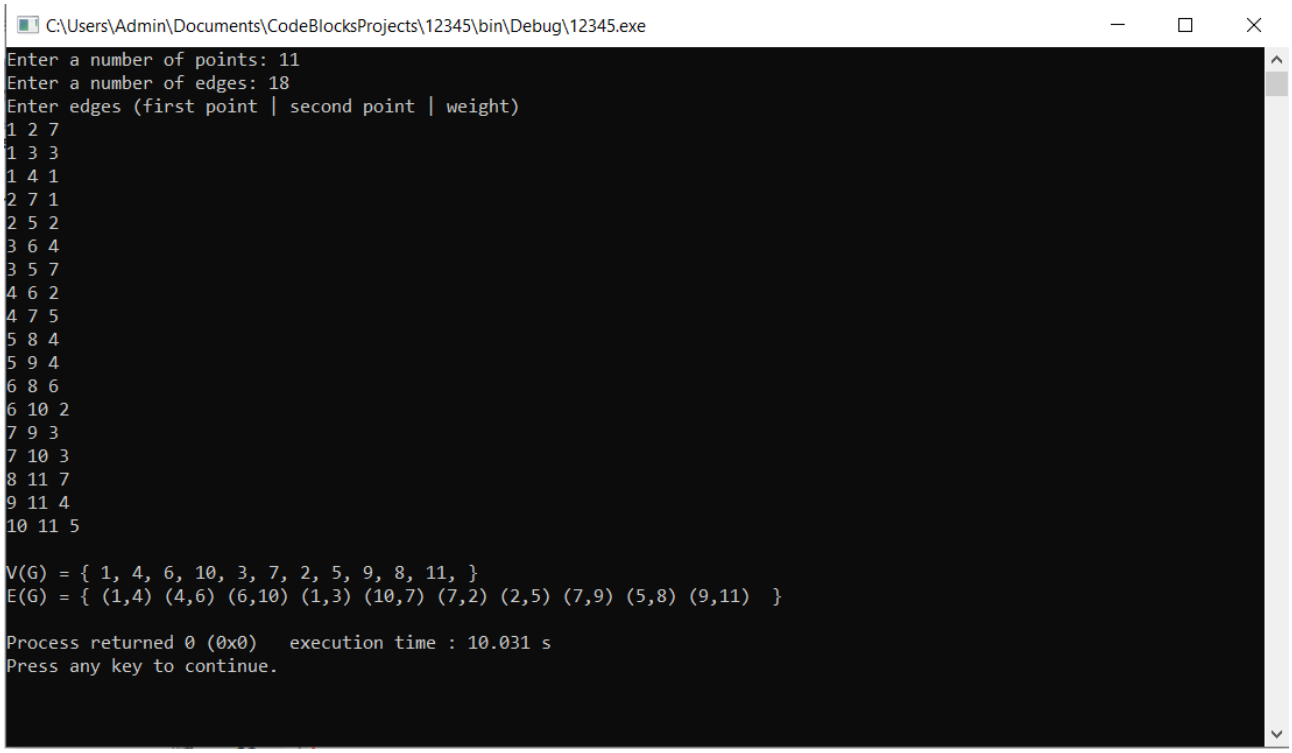
За алгоритмом Прима знайти мінімальне остове дерево графа. Етапи розв'язання задачі виводити на екран. Протестувати розроблену програму на наступному графі:




```
#include <iostream>

using namespace std;
struct edge {
    int firstPoint;
    int secondPoint;
    int weight;
};
void sortEdges(edge* a, int n);
bool isInclude(int* arr, int n, int k);
void findTheTree(edge* a, int* points, edge *tree, int &n, int &k, int &i, int &j);
bool isMinimum(int w, edge* a, int k, int* points, int n);
int main() {
    int n;
    int k;
    cout << "Enter a number of points: ";
    cin >> n;
    cout << "Enter a number of edges: ";
    cin >> k;
    edge* edges = new edge[k];
    cout << "Enter edges (first point | second point | weight) " << endl;
    for (int i = 0; i < k; i++) {
        cin >> edges[i].firstPoint >> edges[i].secondPoint >> edges[i].weight;

    }
    sortEdges(edges, k);
    for(int i = 0; i < k; i ++){
        cout << edges[i].firstPoint << " " << edges[i].secondPoint << ", " << edges[i].weight << endl;
    }
    cout << endl;
```

```
    int* points = new int[n];
    points[0] = edges[0].firstPoint;
    points[1] = edges[0].secondPoint;
    edge* tree = new edge[n-1];
    tree[0].firstPoint = points[0];
    tree[0].secondPoint = points[1];
    int i = 2;
    int j = 1;



    findTheTree(edges, points, tree, n, k, i, j);
    cout << i << " " << j << endl;
    cout << "V(G) = { ";
    for (int i = 0; i < n; i++) {
        cout << points[i] << ", ";
    }
    cout << "}" << endl;
    cout << "E(G) = { ";
    for (int i = 0; i < n-1; i++) {
        cout <<"("<< tree[i].firstPoint << "," << tree[i].secondPoint << ")"<<" ";
    }
    cout << " }" << endl;


    return 0;
}
void findTheTree(edge* a, int* points, edge *tree, int &n, int &k, int &i, int &j) {
    if(i == n){
        return;
    }
    else if (j == k) {
        j = 1;
    }

        if(isInclude(points, n, a[j].firstPoint) && isInclude(points, n, a[j].secondPoint)){
            j++;
            findTheTree(a, points, tree, n, k, i, j);
        }
        else if (!isInclude(points, n, a[j].firstPoint) && isInclude(points, n, a[j].secondPoint) && isMinimum(a[j].weight, a, k, points, n)) {
            tree[i - 1].firstPoint = a[j].secondPoint;
            tree[i - 1].secondPoint = a[j].firstPoint;
            points[i] = a[j].firstPoint;
            tree[i - 1].weight = a[j].weight;

            j++;
            i++;
            findTheTree(a, points, tree, n, k, i, j);

        }
        else if (isInclude(points, n, a[j].firstPoint) && !isInclude(points, n, a[j].secondPoint) && isMinimum(a[j].weight, a, k, points, n)) {
            tree[i - 1].firstPoint = a[j].firstPoint;
            tree[i - 1].secondPoint = a[j].secondPoint;
            points[i] = a[j].secondPoint;
            tree[i - 1].weight = a[j].weight;

            j++;
            i++;
            findTheTree(a, points, tree, n, k, i, j);
        }
        else {
            j++;
            findTheTree(a, points, tree, n, k, i, j);
        }



}
void sortEdges(edge* a, int n) {
    edge temp;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (a[j].weight > a[j+1].weight) {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}
bool isInclude(int* arr, int n, int k) {
    for (int i = 0; i < n; i++) {
        if (k == arr[i]) {
            return true;
        }

    }
    return false;
}
bool isMinimum(int w, edge* a,int k, int* points, int n) {
    for (int j = 1; j < k; j++) {
        if ((((!isInclude(points, n, a[j].firstPoint) && isInclude(points, n, a[j].secondPoint)) || (isInclude(points, n, a[j].firstPoint) && !isInclude(points, n, a[j].secondPoint))) && a[j].weight < w) {
            return false;
        }
    }
    return true;
}
```

Скрін-шот коду програми на мові C++.



Скрін-шот тесту програми.

Висновок: я набув практичних вмінь та навичок з використання алгоритмів Пріма і Краскала.