Міністерство освіти і науки України

Національний університет
“Львівська політехніка”

Кафедра систем штучного інтелекту

Лабораторна робота № 4

з дисципліни

«Дискретна математика»

Виконав:

студент групи КН-115
Поставка Маркіян

Викладач:

Мельникова Н. І.

Львів – 2019р.

Тема: “Основні операції над графами. Знаходження остова мінімальної ваги за алгоритмом Прима-Краскала”

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

Додаток 1:

Варіант 11.

Постановка задачі:

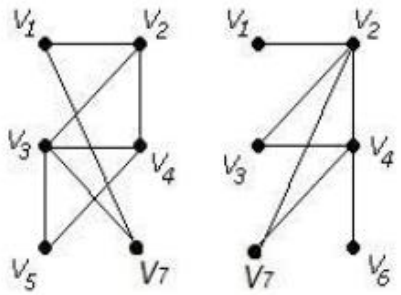
ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Завдання № 1. Розв'язати на графах наступні задачі:

1. Виконати наступні операції над графами:

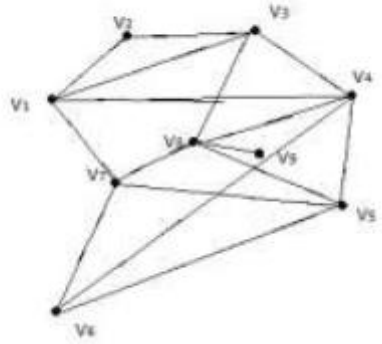
1) знайти доповнення до першого графу,
2) об’єднання графів,
3) кільцеву суму G1 та G2 (G1+G2),
4) розщепити вершину у другому графі,
5) виділити підграф А, що складається з 3-х вершин в G1 і знайти стягнення А в G1 (G1\ А), 6) добуток графів.

11



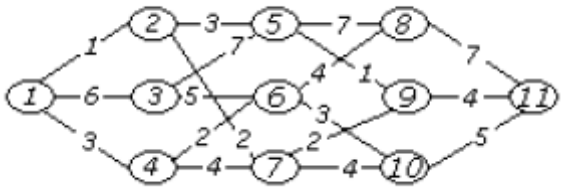
2. Знайти таблицю суміжності та діаметр графа.

11



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

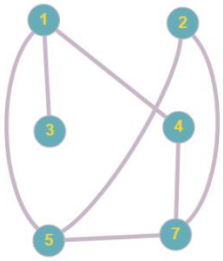
11



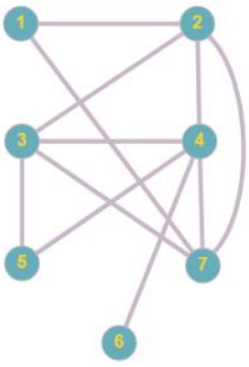
Рішення:

1.

1) Доповнення до першого графа



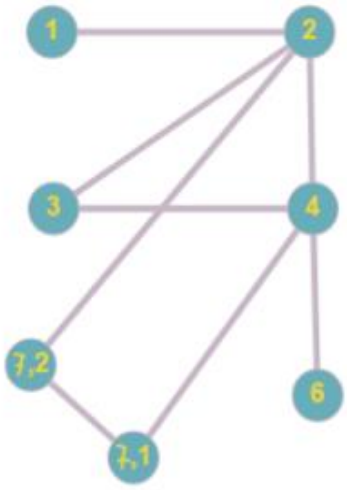
2) Об’єднання графів



3) Кільцева сума графів

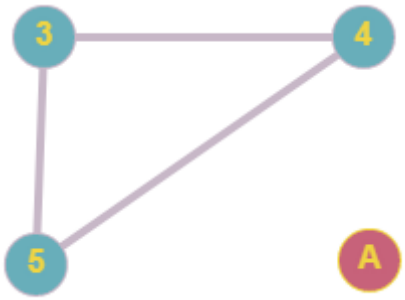
4) Розщеплення вершини у другому графі

Я розщепив вершину 7:

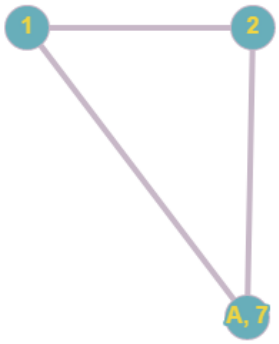


5) Виділити підграф А з 3х вершин у першому графі і знайти його стягнення до цього графа

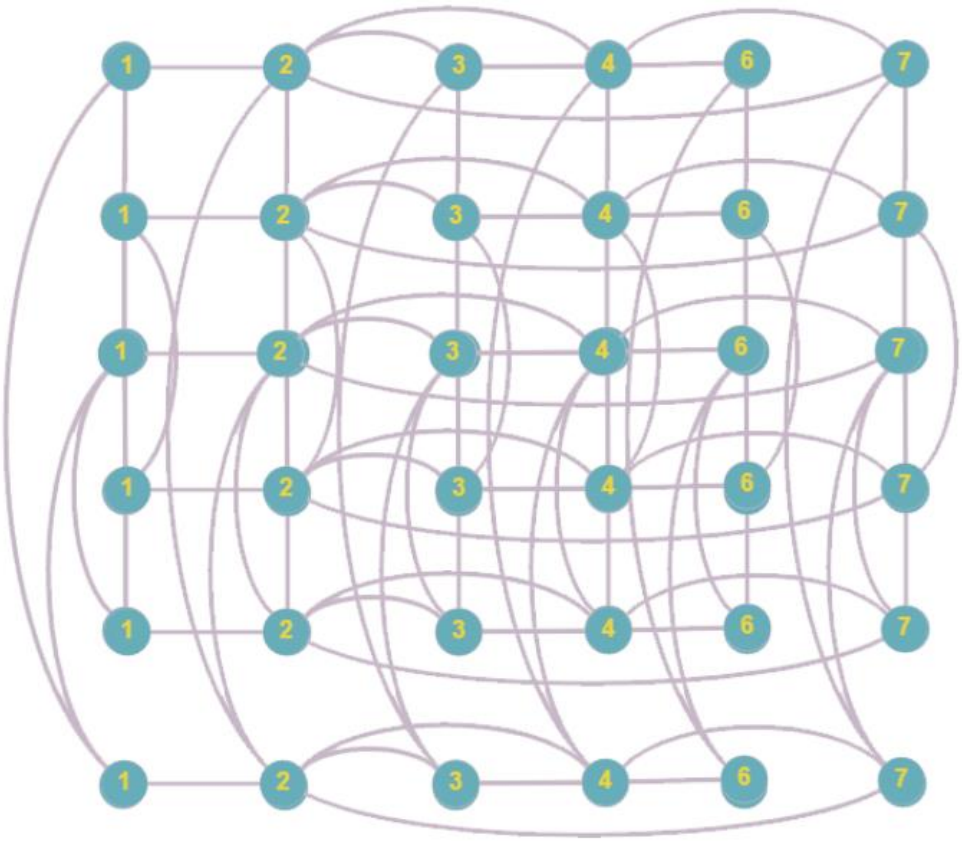
Стягнемо вершини 3, 4, 5 в А.



А стягнемо до першого графа в вершину 7.



6) Добуток графів



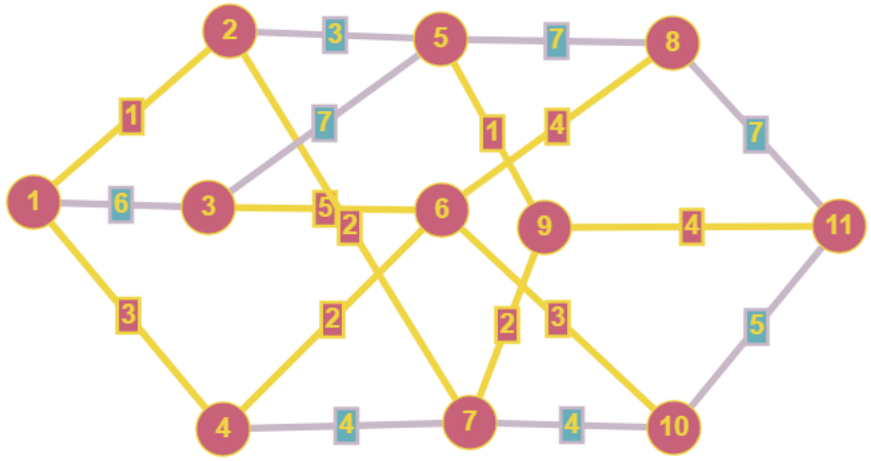
2.

Таблиця суміжності:

Вершини	1	2	3	4	5	6	7	8	9
1	0	1	1	1	0	0	1	0	0
2	1	0	1	0	0	0	0	0	0
3	1	1	0	1	0	0	0	1	0
4	1	0	1	0	1	1	0	1	0
5	0	0	0	1	0	1	1	1	0
6	0	0	0	1	1	0	1	0	0
7	1	0	0	0	1	1	0	1	0
8	0	0	1	1	1	0	1	0	1
9	0	0	0	0	0	0	0	1	0

Діаметр графа: 3

3.



Краскала:

1) Вибираємо будь-яку вершину і знаходимо найкоротше ребро, що з’єднує її з іншою вершиною. Помічаємо вершину як додану до графу.
2) Послідовно проходимося по всіх вершинах і для кожної знаходимо найменше ребро з сусідньою, записуючи їх доданими до дерева.
3) Робимо це поки всі вершини не будуть пройдені.
4) Отриманий дерево і буде кістяковим деревом.

Прима:

1) Сортуємо ребра по зростанню.
2) Беремо найменше ребро і помічаємо вершини як додані до графа.
3) Перевіряємо суміжні вершини циклом по зростанню ваг ребер з ними на приналежність до графа. Якщо одна вершина належить графу а інша ні, то додаємо її.
4) Отриманий дерево і буде кістяковим деревом.

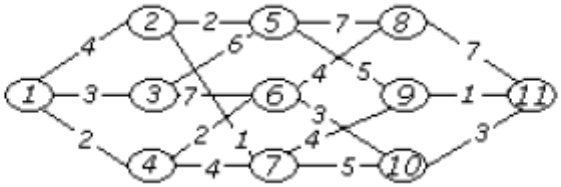
Додаток 2:

Постановка задачі:

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

Варіант № 11

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



Код програми:

```
// Laba4.cpp: определяет точку входа для консольного приложения.
//

#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;

struct graph
{
	int a;
	int b;
	int w;
	bool ck_a;
	bool ck_b;

	graph()
	{
	}

	graph(int a, int b, int w) : a{ a }, b{ b }, w{ w }
	{
		this->ck_a = 0;
		this->ck_b = 0;
	}
};

//Initialisation of graph
void initgraph(graph *g, int nodes, int sides)
{
	g[0] = { 1,2,4 };
	g[1] = { 2,5,2 };
	g[2] = { 5,8,7 };
	g[3] = { 8,11,7 };
	g[4] = { 11,10,3 };
	g[5] = { 10,7,5 };
	g[6] = { 7,4,4 };
	g[7] = { 4,1,2 };
	g[8] = { 1,3,3 };
	g[9] = { 3,5,6 };
```

```
	g[10] = { 5,9,5 };
	g[11] = { 9,11,1 };
	g[12] = { 9,7,4 };
	g[13] = { 2,7,1 };
	g[14] = { 3,6,7 };
	g[15] = { 6,4,2 };
	g[16] = { 6,10,3 };
	g[17] = { 6,8,4 };
}

void sort(graph *g, int sides)
{
	graph t;
	for (int j = 0; j < sides - 1; j++)
	{
		for (int i = 0; i < sides - 1; i++)
		{
			if (g[i + 1].w < g[i].w)
			{
				t = g[i];
				g[i] = g[i + 1];
				g[i + 1] = t;
			}
		}
	}

	for (int i = 0; i < sides; i++)
	{
		cout << i<< "\t" << g[i].a << "\t" << g[i].b << "\t" << g[i].w << "\t" <<
endl;
	}
	cout << endl;
}


void MOD_P(graph *g, int nodes, int sides)
{
	sort(g, sides);

	int *ck_node = new int[nodes]; // Масив перевірки на приналежність

	int ck_i = 0; // Кількість записаних

	for (int i = 0; i < nodes; i++)
	{
		ck_node[i] = 0;
	}
	cout << "Min ost tree" << endl;
	cout << "A\t" << "B\t" << "Weight\t" << endl;
	cout << endl;
	cout << g[0].a << "\t" << g[0].b << "\t" << g[0].w << "\t" << endl;
	g[0].ck_a = 1;	g[0].ck_b = 1;
	ck_node[ck_i] = g[0].a;
	ck_i++;
	ck_node[ck_i] = g[0].b;
	ck_i++;

	int active_i = 0; // Значення яке провіряється

	while (ck_i < nodes)
	{
		for (int i = 1; i < sides; i++) // Записуємо якшо наша вершина входить в
провірені
		{
			for (int j = 0; j < ck_i; j++)
			{
				if (g[i].a == ck_node[j])
				{
```

```
                                g[i].ck_a = 1;
                                break;
                        }
                }

                for (int j = 0; j < ck_i; j++)
                {
                        if (g[i].b == ck_node[j])
                        {
                                g[i].ck_b = 1;
                                break;
                        }
                }
        }

        for (int i = 1; i < sides; i++) // рахуємо кількість суміжних вершин
        {
                if ( (g[i].ck_a == 1 && g[i].ck_b == 0) || (g[i].ck_a == 0 && g[i].ck_b
== 1) )
                {
                        if (g[i].ck_a == 1)
                        {
                                cout << g[i].a << "\t" << g[i].b << "\t" << g[i].w << "\t"
<< endl;
                                g[i].ck_b = 1;
                                ck_node[ck_i] = g[i].b;
                                ck_i++;
                                break;
                        }

                        if (g[i].ck_b == 1)
                        {
                                cout << g[i].a << "\t" << g[i].b << "\t" << g[i].w << "\t"
<< endl;
                                g[i].ck_a = 1;
                                ck_node[ck_i] = g[i].a;
                                ck_i++;
                                break;
                        }
                }
        }
    }

    delete[] ck_node;
}


int main()
{
    int nodes = 11;
    int sides = 18;
    graph *g = new graph[sides];
    cout << "Initialized graph" << endl;
    cout <<"Side\t" << "A\t" << "B\t" << "Weight\t" << endl;
    cout << endl;
    initgraph(g, nodes, sides);

    MOD_P(g, nodes, sides);

    delete[] g;
    _getch();
    return 0;
}
```

Результат роботи програми:

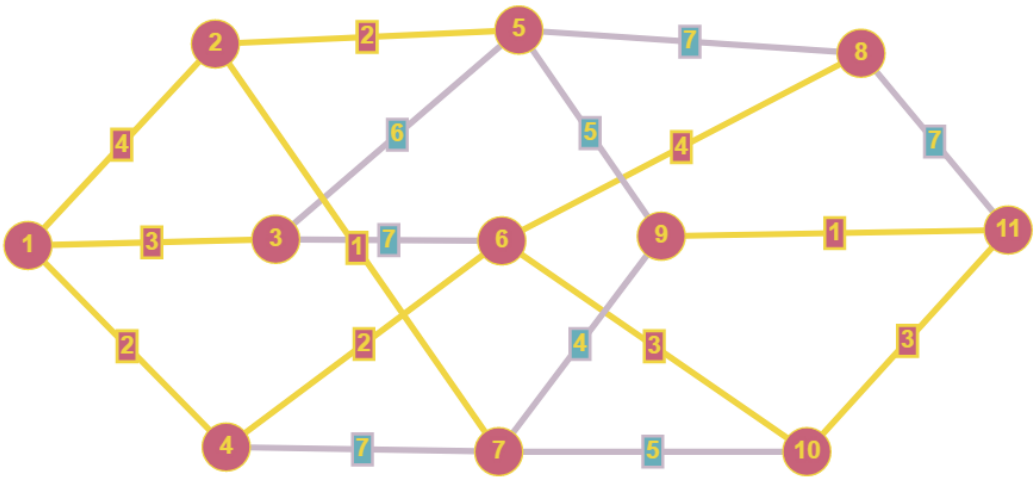
```
C:\Windows\system32\cmd.exe
Initialized graph
Side    A       B       Weight

0       9       11      1
1       2       7       1
2       2       5       2
3       4       1       2
4       6       4       2
5       11      10      3
6       1       3       3
7       6       10      3
8       1       2       4
9       7       4       4
10      9       7       4
11      6       8       4
12      10      7       5
13      5       9       5
14      3       5       6
15      5       8       7
16      8       11      7
17      3       6       7

Min ost tree
A       B       Weight

9       11      1
11      10      3
6       10      3
6       4       2
4       1       2
1       3       3
1       2       4
2       7       1
2       5       2
6       8       4
```

Візуальний вигляд остового дерева з цього завдання.



Висновок: Я набув практичних вмінь та навичок з використання алгоритмів Прима і Краскала.