МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського

«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп’ютерних систем та інфокомунікацій

Кафедра комп’ютерних систем, мереж і кібербезпеки (503)

Лабораторна робота № 7

|  |  |
| --- | --- |
|  | *Дослідження алгоритмів визначення мінімального остовного дерева* |
|  | (назва лабораторної роботи) |
| з дисципліни | *Моделі та структури даних* |
|  | (шифр)  ХАІ**.**503**.**525a**.**03О**.**123-Комп'ютерна інженерія**,** ПЗ №9629619 |

|  |  |  |
| --- | --- | --- |
| Виконав студент гр. | 525а | *Литвиненко А.В.* |
| 07.12.2022 | (№ групи) | (П.І.Б.) |
| (підпис, дата) |  |  |
| Перевірив | канд. техн. наук, доцент | |
|  |  | *А. В. Шостак* |
| (підпис, дата) |  | (П.І.Б.) |

Харків – 2022

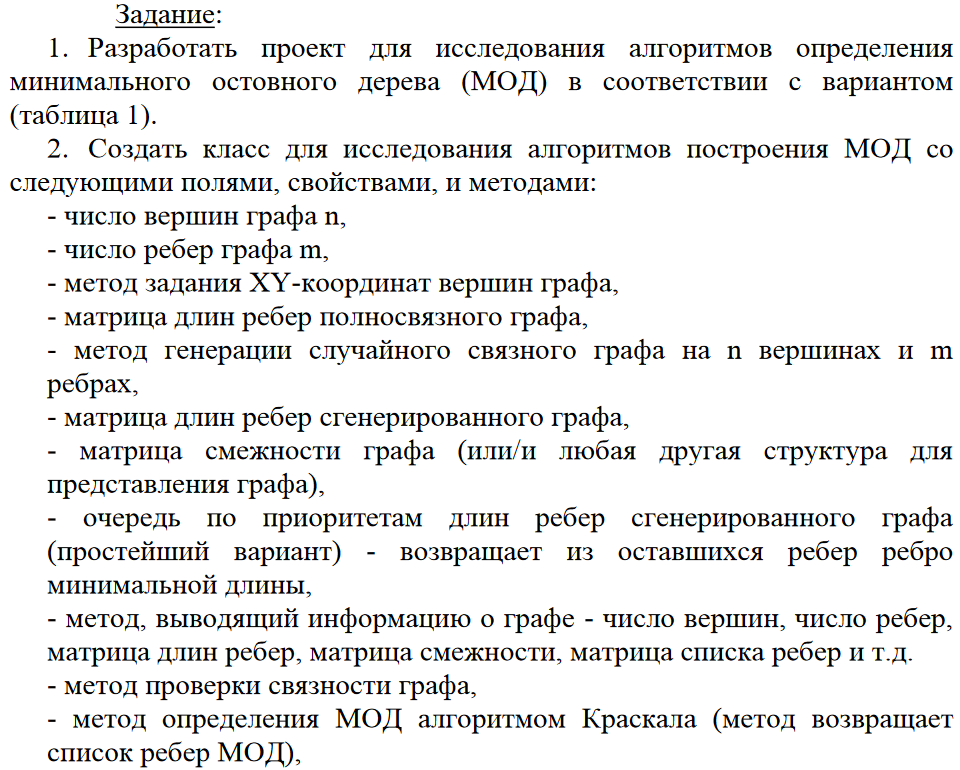
**Тема роботи:** дослідження алгоритмів мінімального остовного дерева.

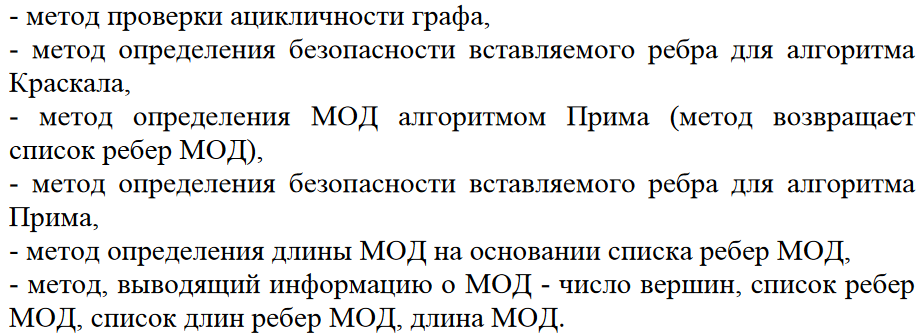
**Варіант 5**

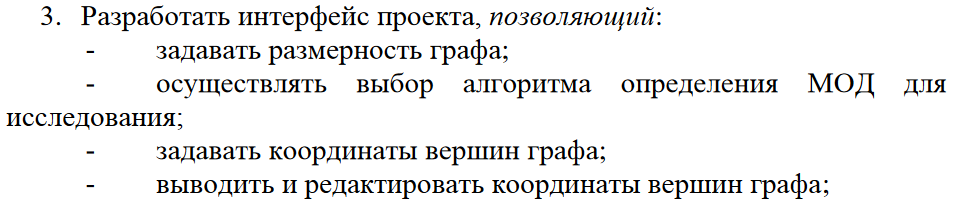
**Задача 1**

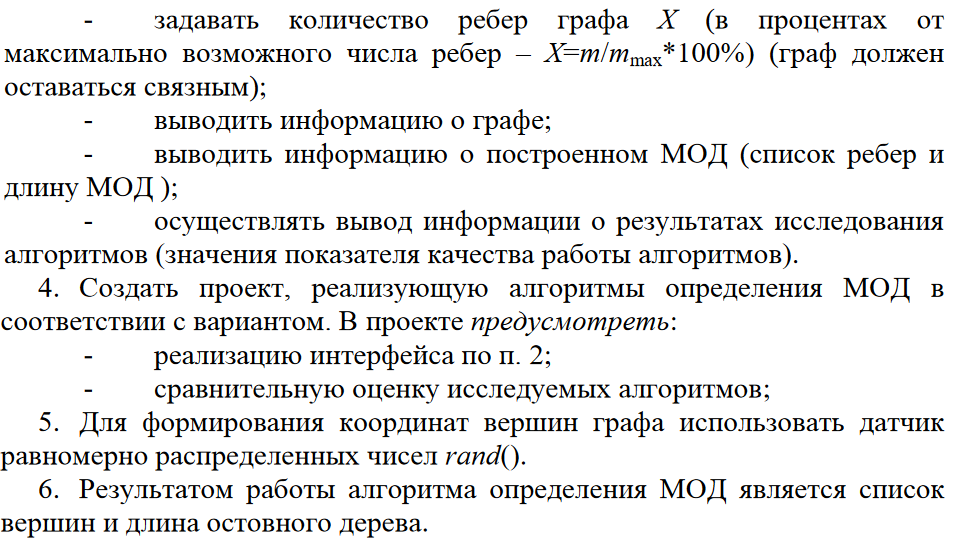
**Частина 1**. Постановка завдання

**Умова:**

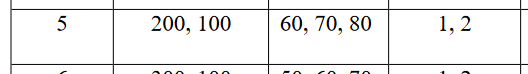


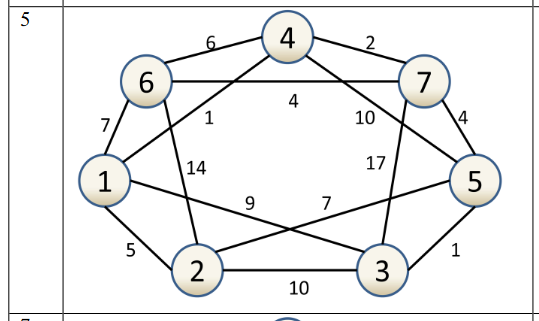




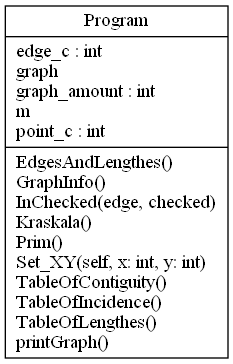


**Умова з додатка:**





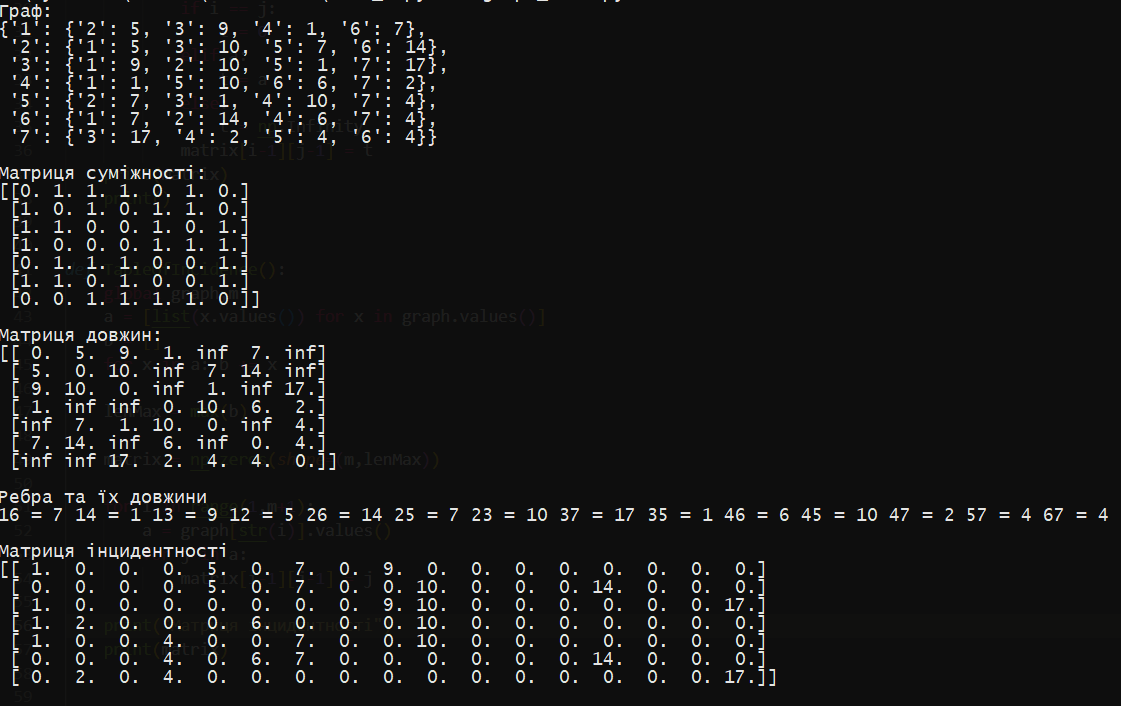
**Частина 2**. Діаграма класів



Малюнок 1 - Діаграма класів

**Частина 3**. Покрокова робота програми

|  |  |
| --- | --- |
| Алгоритм Прима | Алгоритм Краскала |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |



Малюнок 2 – Інформація стосовно графу

**Частина 4**. Текст програми

Відповідно до розробленого алгоритму в середовищі Microsoft Visual Studio була написана програма, яка наведена нижче.

Main.py

# MODULES

# =================================

import math

import numpy as np

from json import loads

from pprint import pprint

# =================================

# VARIANT 5

# =================================

X = 200

Y = 100

X\_percents = [60,70,80]

# =================================

*class* Program:

*def* \_\_init\_\_(*self*):

        # Initialized graphs from file

        with open("graph.json") as f:

*self*.graph = loads(f.read())

*self*.m = len(*self*.graph.keys())

*self*.point\_c    = 0

*self*.edge\_c     = 0

*self*.graph\_amount = 0

    @staticmethod

*def* Set\_XY(*self*, *x*: int, *y*: int):

        # Метод задания XY-координат вершин графа

        global X,Y

        X = *x*

        Y = *y*

*def* Prim(*self*):

*def* get\_min(*R*, *U*):

            rm = (math.inf, -1, -1)

            for v in *U*:

                rr = min(*R*, *key*=*lambda* *x*: *x*[0] if (*x*[1] == v or *x*[2] == v) and (*x*[1] not in *U* or *x*[2] not in *U*) else math.inf)

                if rm[0] > rr[0]:

                    rm = rr

            return rm

        graph = *self*.graph

        # список ребер графа (длина, вершина 1, вершина 2)

        R = []

        for key in graph.keys():

            for point in graph[key].keys():

                R.append((int(graph[key][point]), int(key), int(point)))

        print("Метод прима (МОД)")

        N = 7     # число вершин в графе

        U = {1}   # множество соединенных вершин

        T = []    # список ребер остова

        while len(U) < N:

            r = get\_min(R, U)       # ребро с минимальным весом

            if r[0] == math.inf:    # если ребер нет, то остов построен

                break

            T.append(r)             # добавляем ребро в остов

            U.add(r[1])             # добавляем вершины в множество U

            U.add(r[2])

        print(T)

*def* Kraskala(*self*):

        print("Метод Краскала (МОД)")

        graph = *self*.graph

        R = []

        for key in graph.keys():

            for point in graph[key].keys():

                R.append((int(graph[key][point]), int(key), int(point)))

        Rs = sorted(R, *key*=*lambda* *x*: *x*[0])

        U = set()   # список соединенных вершин

        D = {}      # словарь списка изолированных групп вершин

        T = []      # список ребер остова

        for r in Rs:

            if r[1] not in U or r[2] not in U:  # проверка для исключения циклов в остове

                if r[1] not in U and r[2] not in U: # если обе вершины не соединены, то

                    D[r[1]] = [r[1], r[2]]          # формируем в словаре ключ с номерами вершин

                    D[r[2]] = D[r[1]]               # и связываем их с одним и тем же списком вершин

                else:                           # иначе

                    if not D.get(r[1]):             # если в словаре нет первой вершины, то

                        D[r[2]].append(r[1])        # добавляем в список первую вершину

                        D[r[1]] = D[r[2]]           # и добавляем ключ с номером первой вершины

                    else:

                        D[r[1]].append(r[2])        # иначе, все то же самое делаем со второй вершиной

                        D[r[2]] = D[r[1]]

                T.append(r)             # добавляем ребро в остов

                U.add(r[1])             # добавляем вершины в множество U

                U.add(r[2])

        for r in Rs:    # проходим по ребрам второй раз и объединяем разрозненные группы вершин

            if r[2] not in D[r[1]]:     # если вершины принадлежат разным группам, то объединяем

                T.append(r)             # добавляем ребро в остов

                gr1 = D[r[1]]

                D[r[1]] += D[r[2]]      # объединем списки двух групп вершин

                D[r[2]] += gr1

        print(T)

*def* printGraph(*self*):

        graph = *self*.graph

        pprint(graph)

*def* TableOfContiguity(*self*):

        graph = *self*.graph

        m     = *self*.m

        print("Матриця суміжності: ")

        matrix = np.zeros(*shape*=(m,m))

        for i in range(1, m+1):

            for j in range(1, m+1):

                if graph[str(j)].get(str(i), 0):

                    matrix[i-1][j-1] = 1

                else:

                    matrix[i-1][j-1] = 0

        print(matrix)

        print()

*def* TableOfLengthes(*self*):

        graph = *self*.graph

        m     = *self*.m

        print("Матриця довжин: ")

        matrix = np.zeros(*shape*=(m,m))

        for i in range(1, m+1):

            for j in range(1, m+1):

                a = graph[str(j)].get(str(i), 0)

                t = 0

                if i == j:

                    t = 0

                elif a:

                    t = a

                else:

                    t = np.Infinity

                matrix[i-1][j-1] = t

        print(matrix)

        print()

*def* TableOfIncidence(*self*):

        graph = *self*.graph

        m     = *self*.m

        a = [list(x.values()) for x in graph.values()]

        b = []

        for x in a: b += x

        lenMax = max(b)

        matrix = np.zeros(*shape*=(m,lenMax))

        for i in range(1,m+1):

            a = graph[str(i)].values()

            for j in a:

                matrix[i-1][j-1] = j

        print("Матриця інцидентності")

        print(matrix)

        print()

*def* InChecked(*self*, *edge*, *checked*):

        if *edge* in *checked* or *edge*[::-1] in *checked*:

            return True

        return False

*def* EdgesAndLengthes(*self*):

        graph = *self*.graph

        m     = *self*.m

        print("Ребра та їх довжини")

        checked = []

        for point in graph.keys():

            for key in graph[point].keys():

                edge = point + str(key)

                if not *self*.InChecked(edge, checked):

                    print("%s = %d" % (edge, graph[point][key]), *end*=" ")

                    checked.append(edge)

        print("\n")

*def* GraphInfo(*self*):

*self*.printGraph()

*self*.TableOfContiguity()

*self*.TableOfLengthes()

*self*.TableOfIncidence()

*self*.EdgesAndLengthes()

m = Program()

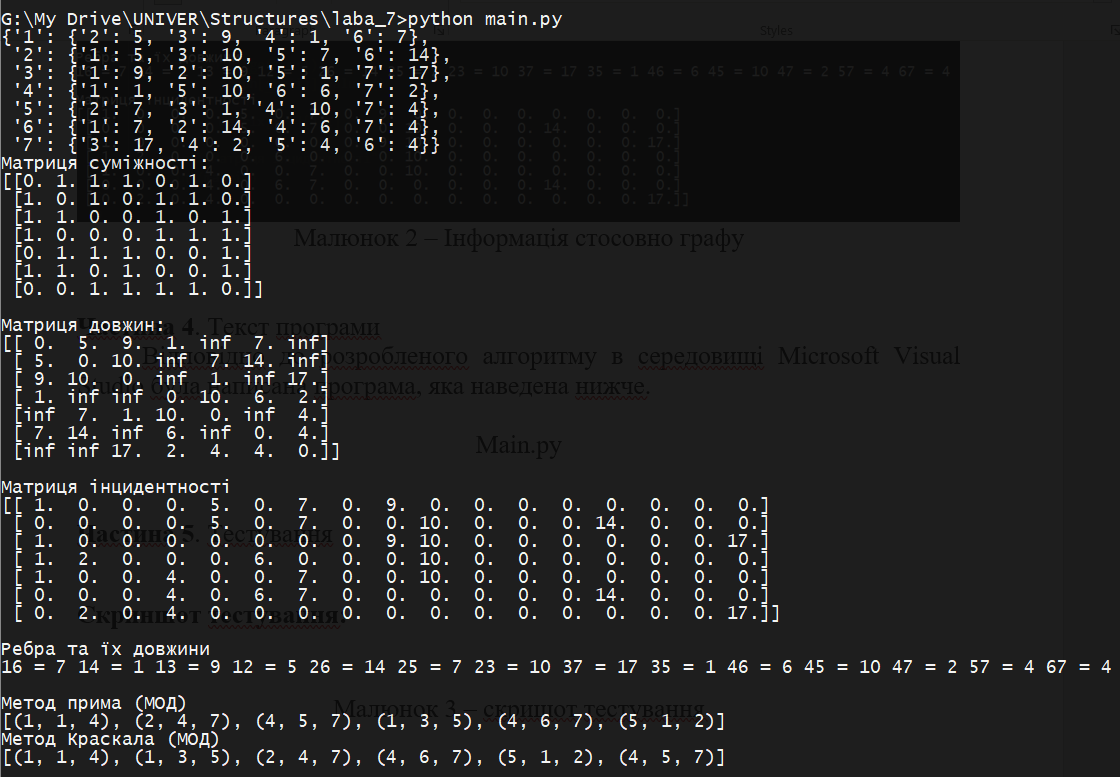
m.GraphInfo()

m.Prim()

m.Kraskala()

**Частина 5**. Тестування

**Скриншот тестування:**



Малюнок 3 – скришот тестування

**Висновки**

Під час цієї лабораторної роботи я працював з графами та навчився використовувати їх у мові програмування. Експерементував з різними методами для дослідження МОД і намагався сам реалізувати алгоритма Прима та Краскала.