**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний**

**інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 3 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

**«Прикладні задачі теорії графів ч.1»**

**Виконав** ІП-11Трикош Іван Володимирович

**Перевірила** Халус Олена Андріївна

Київ 2022

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні прикладні алгоритми на графах та

способи їх імплементації.

2 ЗАВДАННЯ

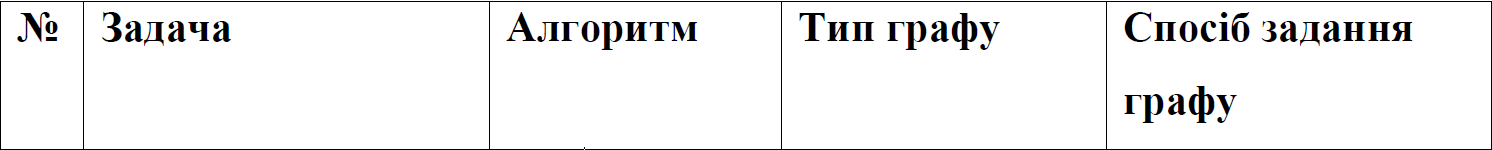
Згідно варіанту розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором).

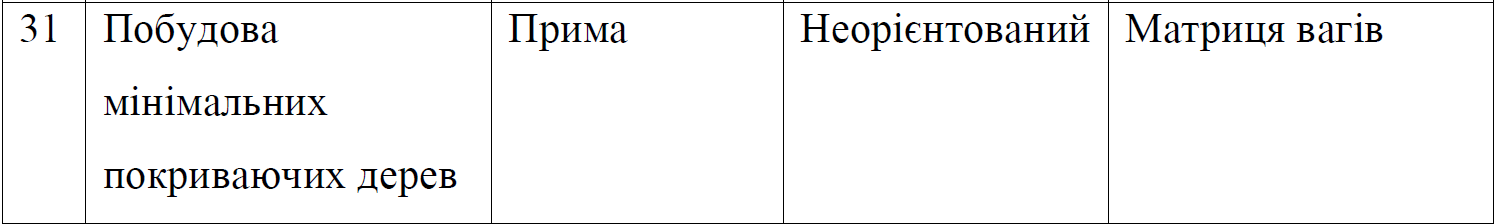
Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа

та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа (розмірності не менше 9 вершин) розв’язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв’язання задачі.





3 ВИКОНАННЯ

Будемо вважати, що в нас простий граф (не містить петель та кратних ребер).

3.1 Псевдокод алгоритму

A[n][n] – матриця ваг

T[n][n] – вихідний граф

e – ребро

indexes – список вершин, які є у вихідному графі

M – копія матриці А

M = A

indexes = [1]

while indexes.length != M[0].length do

k = p = 0

e = +infinity

for i = 1 in indexes do

for j = 1 in M[i].length do

if i != j and e > M[i][j] then

e = M[i][j]

k = i

p = j

end if

end for

end for

for i = 1 in indexes do

M[i][p] = M[p][i] = +infinity

end for

T[k][p] = T[p][k] = e

indexes += [p]

end while

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

import random

def matrix(): # Вводимо кількість вершин та заповнюємо випадковими числами матрицю

n = int(input("Введіть кількість вершин: "))

A = []

for i in range(n):

temp = []

for j in range(n):

temp += ["inf"]

A += [temp]

for i in range(n):

for j in range(n):

if i < j:

k = random.randint(0,2)

if k // 2 == 0:

A[i][j] = random.randint(1, n)

A[j][i] = A[i][j]

return A

def pryma(A):

# Створюємо матрицю Т та заповнюємо її позначенням "inf"

T = []

for i in range(len(A)):

temp = []

for j in range(len(A)):

if (i != j):

temp += [float("inf")]

else:

temp += [0]

T += [temp]

M = A[:][:] # Робимо копію матриці А

indexes = [0] # Список вершин, які є у графі(матриці) Т

while len(indexes) != len(A[0]): # Поки у нас є нерозглянуті вершини

k = None

p = None

e = float("inf")

for i in indexes: # Проходимо по вже розглянутих вершинах

for j in range(len(M[i])): # Проходимо по ребрах цієї вершини

if i != j and e > M[i][j]: # Якщо число є цілим та меншим за "е", то присвоюємо його числу "е"

e = M[i][j]

k = i

p = j

for i in indexes: # Видаляємо всі ребра між розглянутими вершинами

M[i][p] = M[p][i] = float("inf")

T[k][p] = T[p][k] = e # Записуємо ребро у матрицю Т

indexes += [p] # Додаємо індекс розглянутої вершини

return T

def output(M, name): # Виводимо матрицю

print("\nMatrix", name, end = ":")

for i in M:

print()

for j in i:

print(f'{j:3}', end = " ")

print()

def main():

A = matrix()

output(A, "A")

T = pryma(A)

output(T, "T")

if \_\_name\_\_ == "\_\_main\_\_":

main()

3.2.2 Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для графів на 10 і 15 вершин відповідно.

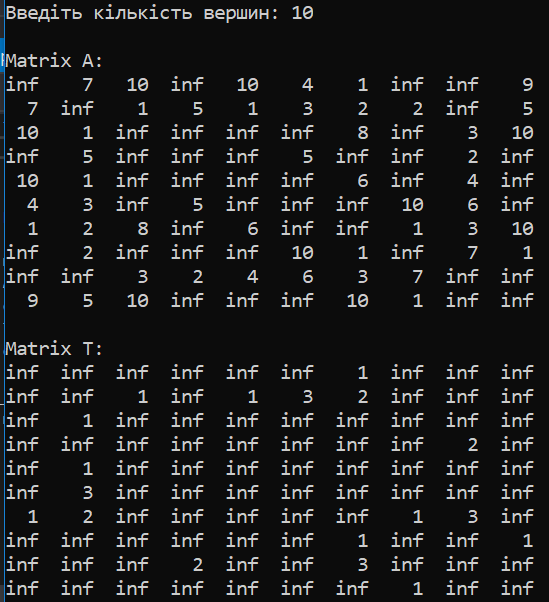


Рисунок 3.1

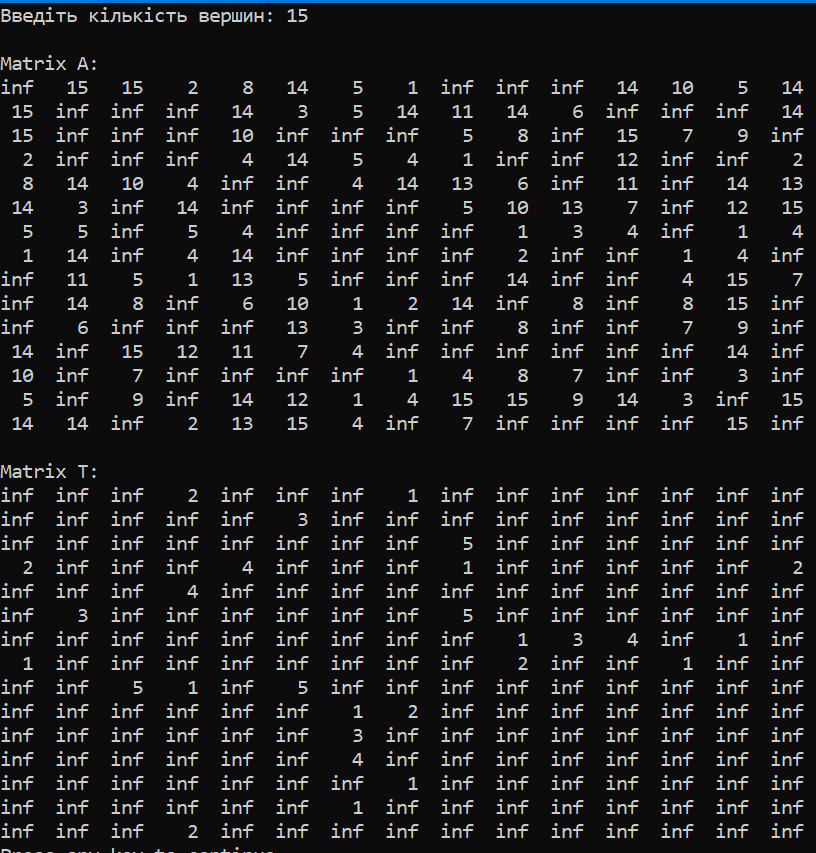
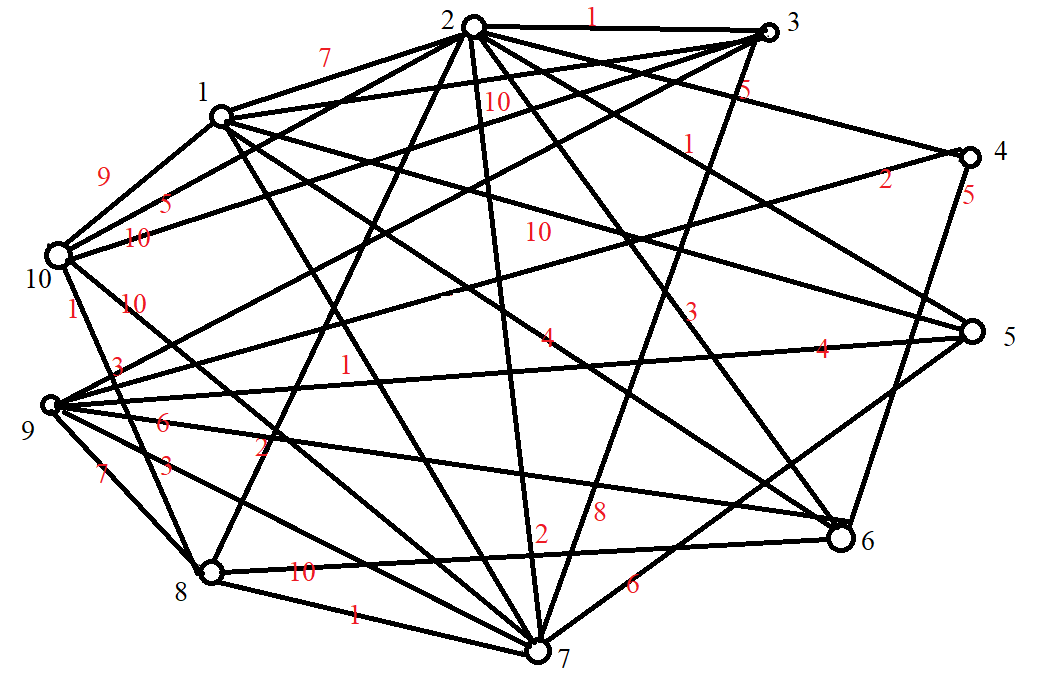


Рисунок 3.2

3.3 Розв’язання задачі вручну

На рисунку 3.3 наведено розв’язання задачі вручну для 10-ти вершин.

Граф такий самий, як і в попередньому прикладі на 10 вершин



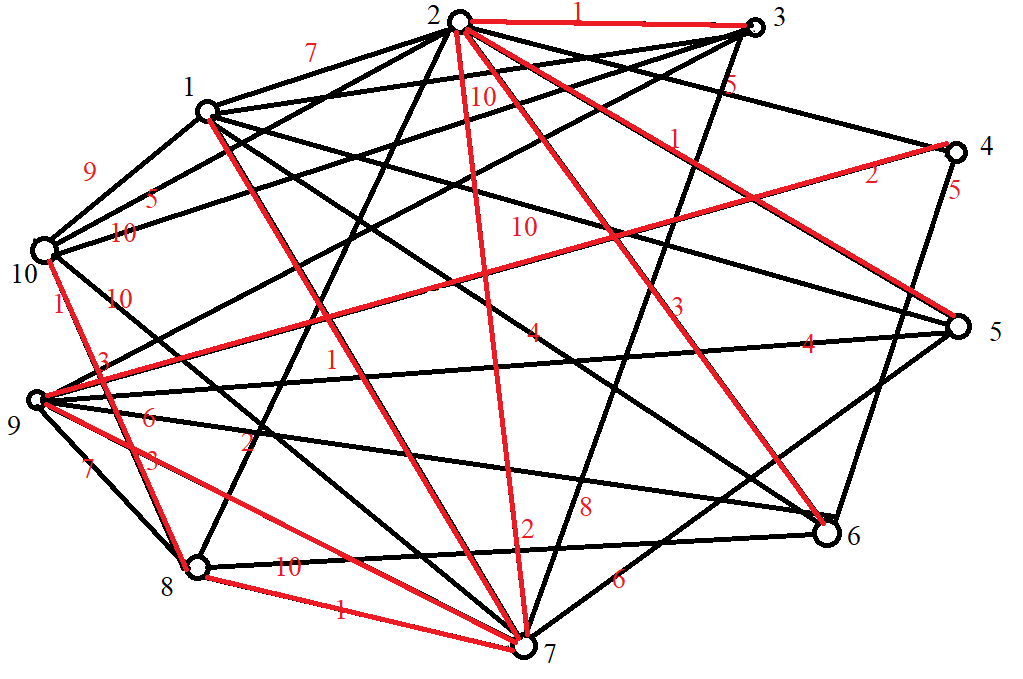


Рисунок 3.1 – Розв’язання задачі вручну

ВИСНОВОК

При виконанні даної лабораторної роботи я опанував алгоритм Прима для пошуку мінімального покриваючого дерева в простому неорієнтованому зваженому графі. Із результатів розв'язку бачимо, що результат роботи програми та роботи вручну збігається, отже робота виконана правильно.