**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №6**

з дисципліни  
«Алгоритми і структури даних»

Виконав: Перевірив:

студент групи ІМ-44 Сергієнко М. А.  
Мундурс Нікіта Юрійович  
номер у списку групи: 16

Київ 2025

**Завдання**

1. Представити ненапрямлений граф із заданими параметрами так само, як у лабораторній роботі №3.

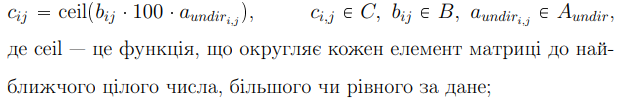
***Відмінність 1***: коефіцієнт *k* = 1.0 – *n3* \* 0.01 - *n4* \* 0.005 – 0.05

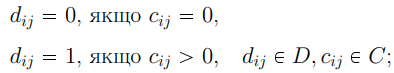
Отже, матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1. встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту *n1n2n3n4*;
2. матриця розміром *n*\**n* заповнюється згенерованими випадковими числами в діапазоні [0, 2,0);
3. обчислюється коефіцієнт *k* = 1.0 – *n3* \* 0.01 - *n4* \* 0.005 – 0.15, кожен елемент матриці множиться на коефіцієнт *k*;
4. елементи матриці округлюються: 0 – якщо елемент менший за 1.0, 1 – якщо елемент більший або дорівнює 1.0.

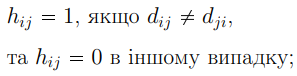
Матриця Aundir ненапрямленого графа одержується з матриці Adir так само, як у ЛР №3.

***Відмінність 2***: матриця ваг W формується таким чином:

1. матриця B розміром *n\*n* заповнюється згенерованими випадковими числами в діапазоні [0, 2,0) (параметр генератора випадкових чисел той же самий, *n1n2n3n4*);
2. одержується матриця C: 
3. одержується матриця D, у якій



1. одержується матриця H, у якій



1. Tr – верхня трикутна матриця з одиниць (*trij* при *i* < *j*)
2. матриця ваг W симетрична, і її елементи одержуються за формулою: *wij = wji = (dij + hij \* trij) \* cij.*
3. Створити програму для знаходження мінімального кістяка за алгоритмом Краскала при n4 – парному і за алгоритмом Пріма – при непарному. При цьому у програмі:
4. графи представляти у вигляді динамічних списків, обхід графа, додавання, віднімання вершин, ребер виконувати як функції з вершинами відповідних списків;
5. у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.
6. Під час обходу графа побудувати дерево його кістяка. У програмі дерево кістяка виводити покроково у процесі виконання алгоритму. Це можна виконати одним із двох способів:
7. або виділяти іншим кольором ребра графа;
8. або будувати кістяк поряд із графом.

При зображенні як графа, так і його кістяка, вказати ваги ребер.

При проєктуваннi програми також ***слiд врахувати наступне***:

1. мова програмування обирається студентом самостійно;
2. графічне зображення всіх графів має формуватися програмою з тими ж вимогами, як у ЛР №3;
3. усі графи обов’язково зображувати в графічному вікні;
4. типи та структури даних для внутрішнього представлення всіх даних у програмі слід вибрати самостійно;

**Варіант:** 4416

***k* =** 0,91

**Кількість вершин:** 11

**Розміщення вершин:** колом з вершиною в центрі

**Алгоритм**: Краскала

**Обрана мова програмування**: Python

**Текст програми**

import turtle

import random

import math

import time

class Vertex:

def \_\_init\_\_(self, number, pos\_x, pos\_y):

self.number = number

self.pos\_x = pos\_x

self.pos\_y = pos\_y

class Graph:

def \_\_init\_\_(self, num\_vertices):

self.edges\_list = []

self.vertices\_list = set(range(0, num\_vertices))

def add\_edge(self, start\_vertex, end\_vertex, weight):

self.edges\_list.append((start\_vertex, end\_vertex, weight))

def remove\_edge(self, edge):

self.edges\_list.remove(edge)

def remove\_vertex(self, vertex):

self.vertices\_list.remove(vertex)

def get\_weight(self, start\_vertex, end\_vertex):

for edge in self.edges\_list:

if (edge[0] == start\_vertex and edge[1] == end\_vertex) or (edge[0] == end\_vertex and edge[1] == start\_vertex):

return edge[2]

def sort\_edges\_by\_weight(self):

self.edges\_list = sorted(self.edges\_list, key=lambda edge: edge[2])

def print\_info(self):

print("\nСписок вершин графа:", [x + 1 for x in self.vertices\_list])

print("Список ребер графа:")

for edge in self.edges\_list:

print((edge[0] + 1, edge[1] + 1, edge[2]))

print()

class MST:

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

self.num\_vertices = 0

self.edges\_list = []

self.vertices\_list = set()

def add\_edge\_and\_vertex(self, edge):

start\_vertex, end\_vertex, weight = edge

if start\_vertex not in self.vertices\_list:

self.vertices\_list.add(start\_vertex)

self.num\_vertices += 1

if end\_vertex not in self.vertices\_list:

self.vertices\_list.add(end\_vertex)

self.num\_vertices += 1

self.edges\_list.append((start\_vertex, end\_vertex, weight))

def print\_info(self):

print("Список вершин мінімального кістяка:", [x + 1 for x in self.vertices\_list])

print("Список ребер мінімального кістяка:")

for edge in self.edges\_list:

print((edge[0] + 1, edge[1] + 1, edge[2]))

print()

def main():

seed = 3216

n3 = 1

n4 = 6

N = 11

directed\_graph\_matrix = generateMatrix(n3, n4, N, seed)

undirected\_graph\_matrix = doUndir(directed\_graph\_matrix)

matrix\_b = getMatrixB(N, seed)

matrix\_c = getMatrixC(matrix\_b, undirected\_graph\_matrix)

matrix\_d = getMatrixD(matrix\_c)

matrix\_w = getMatrixW(matrix\_c, matrix\_d)

graph = createGraph(undirected\_graph\_matrix, matrix\_w)

mst = MST()

position = createPositions(undirected\_graph\_matrix)

drawVertices(position)

drawEdges(position, undirected\_graph\_matrix, graph)

graph.print\_info()

mst.print\_info()

findTree(graph, mst, position)

getWeight(mst)

print("\nПошук мінімального кістяка графа завершено!")

turtle.hideturtle()

turtle.done()

def generateMatrix(n3, n4, N, seed):

random.seed(seed)

adj\_matrix = [[random.random() \* 2.0 for \_ in range(N)] for \_ in range(N)]

k = 1.0 - n3 \* 0.01 - n4 \* 0.005 - 0.05

for i in range(N):

for j in range(N):

adj\_matrix[i][j] \*= k

adj\_matrix[i][j] = 0 if adj\_matrix[i][j] < 1.0 else 1

return adj\_matrix

def doUndir(adj\_matrix):

n = len(adj\_matrix)

undirected\_adj\_matrix = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(n):

if adj\_matrix[i][j] == 1:

undirected\_adj\_matrix[i][j] = 1

undirected\_adj\_matrix[j][i] = 1

print("\nМатриця суміжності ненапрямленого графа:")

for row in undirected\_adj\_matrix:

print(row)

return undirected\_adj\_matrix

def createPositions(undirected\_graph\_matrix):

vertices = []

center\_x = 0

center\_y = 0

radius = 300

n = len(undirected\_graph\_matrix)

for i in range(1, n):

angle = i \* (2 \* math.pi / (n-1))

x = center\_x + radius \* math.cos(angle)

y = center\_y + radius \* math.sin(angle)

vertex = Vertex(i, x, y)

vertices.append(vertex)

central\_vertex = Vertex(n, center\_x, center\_y)

vertices.append(central\_vertex)

return vertices

def drawVertices(vertices):

turtle.speed(0)

turtle.penup()

radius = 20

for vertex in vertices:

x, y = vertex.pos\_x, vertex.pos\_y

turtle.goto(x, y - radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.goto(x, y)

turtle.write(vertex.number, align="center")

def drawEdges(vertices, undirected\_graph\_matrix, graph):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

for i in range(n):

for j in range(i + 1, n):

if i != j and undirected\_graph\_matrix[i][j] == 1 and abs(vertices[i].number - vertices[j].number) != 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

turtle.penup()

turtle.goto(x1, y1)

turtle.pendown()

turtle.goto(x2, y2)

turtle.penup()

mid\_x = (x1 + x2) / 2

mid\_y = (y1 + y2) / 2

turtle.goto(mid\_x, mid\_y)

turtle.color("red")

turtle.write(graph.get\_weight(i, j), align="center")

turtle.color("black")

if i != j and undirected\_graph\_matrix[i][j] == 1 and abs(vertices[i].number - vertices[j].number) == 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

angle = math.degrees(math.atan2(y2 - y1, x2 - x1))

turtle.penup()

turtle.goto(x1, y1)

turtle.setheading(angle)

turtle.pendown()

distance = turtle.distance(x2, y2)

degrees = math.pi \* 2

b = distance / 2 / math.cos(math.radians(degrees))

turtle.right(degrees)

turtle.forward(b)

current\_position = turtle.position()

turtle.setheading(turtle.towards(x2, y2))

turtle.goto(x2, y2)

turtle.penup()

mid\_x, mid\_y = current\_position

turtle.goto(mid\_x, mid\_y)

turtle.color("red")

turtle.write(graph.get\_weight(i, j), align="center")

turtle.color("black")

for i in range(n):

if undirected\_graph\_matrix[i][i] == 1:

x, y = vertices[i].pos\_x, vertices[i].pos\_y

turtle.penup()

turtle.goto(x + math.pi / 2 \* 10, y + math.pi / 2 \* 10)

turtle.pendown()

turtle.circle(10)

turtle.penup()

turtle.goto(x + math.pi / 2 \* 10, y + math.pi / 2 \* 10 + 4)

turtle.color("red")

turtle.write(graph.get\_weight(i, i), align="center", font=('Arial', 7, 'normal'))

turtle.color("black")

def drawColoredEdges(vertices, i, j):

turtle.speed(0)

turtle.penup()

turtle.pensize(2)

turtle.color("blue")

radius = 20

if i != j and abs(vertices[i].number - vertices[j].number) != 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

turtle.penup()

turtle.goto(x1, y1)

turtle.pendown()

turtle.goto(x2, y2)

turtle.penup()

if i != j and abs(vertices[i].number - vertices[j].number) == 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

angle = math.degrees(math.atan2(y2 - y1, x2 - x1))

turtle.penup()

turtle.goto(x1, y1)

turtle.setheading(angle)

turtle.pendown()

distance = turtle.distance(x2, y2)

degrees = math.pi \* 2

b = distance / 2 / math.cos(math.radians(degrees))

turtle.right(degrees)

turtle.forward(b)

turtle.setheading(turtle.towards(x2, y2))

turtle.goto(x2, y2)

turtle.penup()

def drawColoredVertices(vertices, i):

turtle.speed(0)

turtle.penup()

turtle.pensize(3)

turtle.color("green")

radius = 20

x, y = vertices[i].pos\_x, vertices[i].pos\_y

turtle.goto(x+2, y - radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.goto(x, y)

def getMatrixB(N, seed):

random.seed(seed)

matrix\_b = [[random.random() \* 2.0 for \_ in range(N)] for \_ in range(N)]

return matrix\_b

def getMatrixC(matrix\_b, adj\_matrix):

n = len(matrix\_b)

matrix\_c = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(n):

matrix\_c[i][j] = matrix\_b[i][j] \* 100 \* adj\_matrix[i][j]

rounded\_matrix = [[math.ceil(x) for x in row] for row in matrix\_c]

return rounded\_matrix

def getMatrixD(matrix\_c):

n = len(matrix\_c)

matrix\_d = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(n):

if matrix\_c[i][j] > 0:

matrix\_d[i][j] = 1

return matrix\_d

def getMatrixW(matrix\_c, matrix\_d):

n = len(matrix\_c)

matrix\_w = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(i, n):

matrix\_w[i][j] = matrix\_w[j][i] = matrix\_d[i][j] \* matrix\_c[i][j]

print("\nМатриця ваг графа:")

for row in matrix\_w:

print("[{}]".format(" ".join("{: <3}".format(num) for num in row)))

return matrix\_w

def createGraph(adjacency\_matrix, weight\_matrix):

num\_vertices = len(adjacency\_matrix)

graph = Graph(num\_vertices)

for i in range(num\_vertices):

for j in range(i, num\_vertices):

if adjacency\_matrix[i][j] == 1:

graph.add\_edge(i, j, weight\_matrix[i][j])

return graph

def find(parent, vertex):

if parent[vertex] != vertex:

parent[vertex] = find(parent, parent[vertex])

return parent[vertex]

def union(parent, rank, vertex1, vertex2):

root1 = find(parent, vertex1)

root2 = find(parent, vertex2)

if root1 != root2:

if rank[root1] < rank[root2]:

parent[root1] = root2

elif rank[root1] > rank[root2]:

parent[root2] = root1

else:

parent[root2] = root1

rank[root1] += 1

def findTree(graph, mst, vertices):

num\_vertices = len(vertices)

counter = 0

graph.sort\_edges\_by\_weight()

parent = {}

rank = {}

for vertex in graph.vertices\_list:

parent[vertex] = vertex

rank[vertex] = 0

time.sleep(5)

print("\nВідсортовані за порядком зростання ваги ребра графа:")

for edge in graph.edges\_list:

print((edge[0] + 1, edge[1] + 1, edge[2]))

time.sleep(8)

while graph.edges\_list:

edge = graph.edges\_list[0]

start\_vertex, end\_vertex, weight = edge

print("\nДосліджуване ребро:", (edge[0] + 1, edge[1] + 1))

print("Вага ребра:", edge[2])

if find(parent, start\_vertex) != find(parent, end\_vertex):

if start\_vertex not in mst.vertices\_list:

drawColoredVertices(vertices, start\_vertex)

if end\_vertex not in mst.vertices\_list:

drawColoredVertices(vertices, end\_vertex)

if start\_vertex in graph.vertices\_list:

graph.remove\_vertex(start\_vertex)

if end\_vertex in graph.vertices\_list:

graph.remove\_vertex(end\_vertex)

drawColoredEdges(vertices, start\_vertex, end\_vertex)

mst.add\_edge\_and\_vertex(edge)

union(parent, rank, start\_vertex, end\_vertex)

counter += 1

print("Ребро не утворює цикл. Додаємо до мінімального кістяка")

print("Номер ребра:", counter)

else:

print("Ребро утворює цикл!")

graph.remove\_edge(edge)

mst.print\_info()

time.sleep(8)

if mst.num\_vertices == num\_vertices:

break

print("\nРезультати пошуку:")

mst.print\_info()

graph.print\_info()

def getWeight(mst):

weight = 0

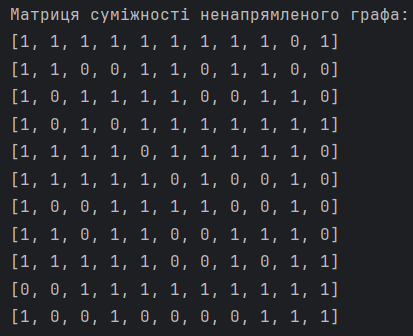
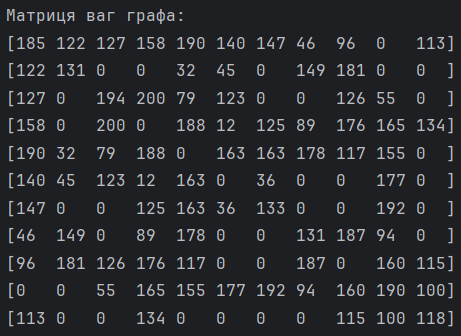
for edge in mst.edges\_list:

weight += edge[2]

print("\rСума ваг ребер мінімального кістяка:", weight)

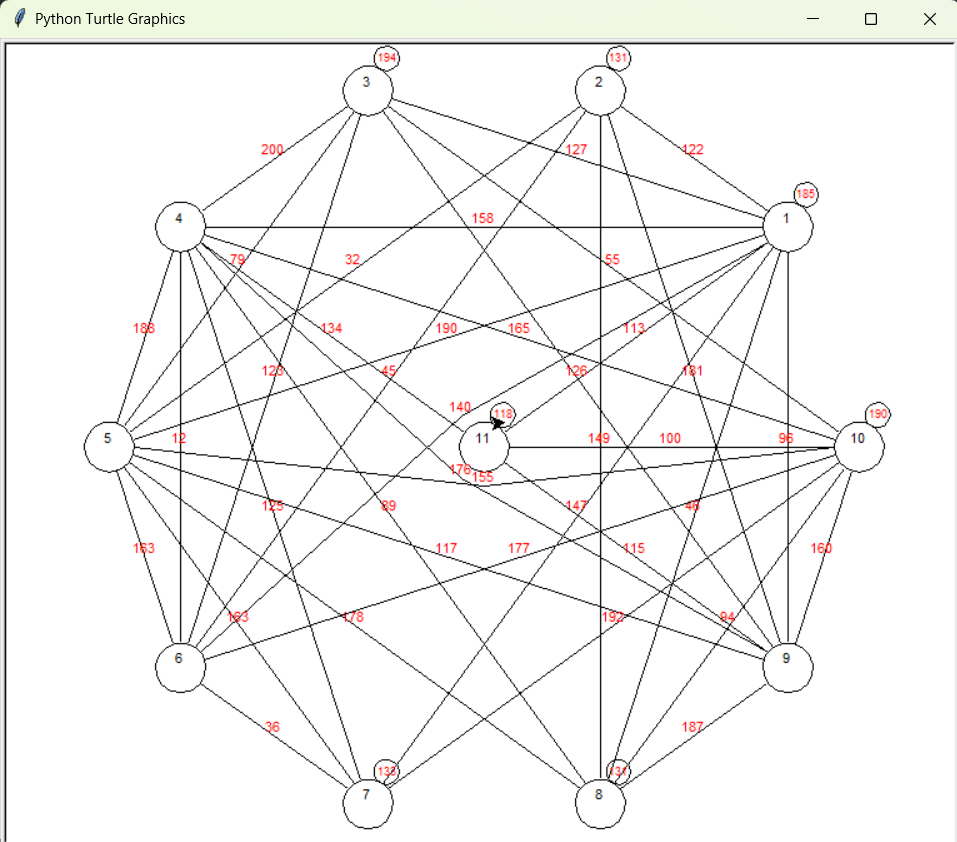
main()

**Матриця суміжності ненапрямленого графа та матриця ваг графа**

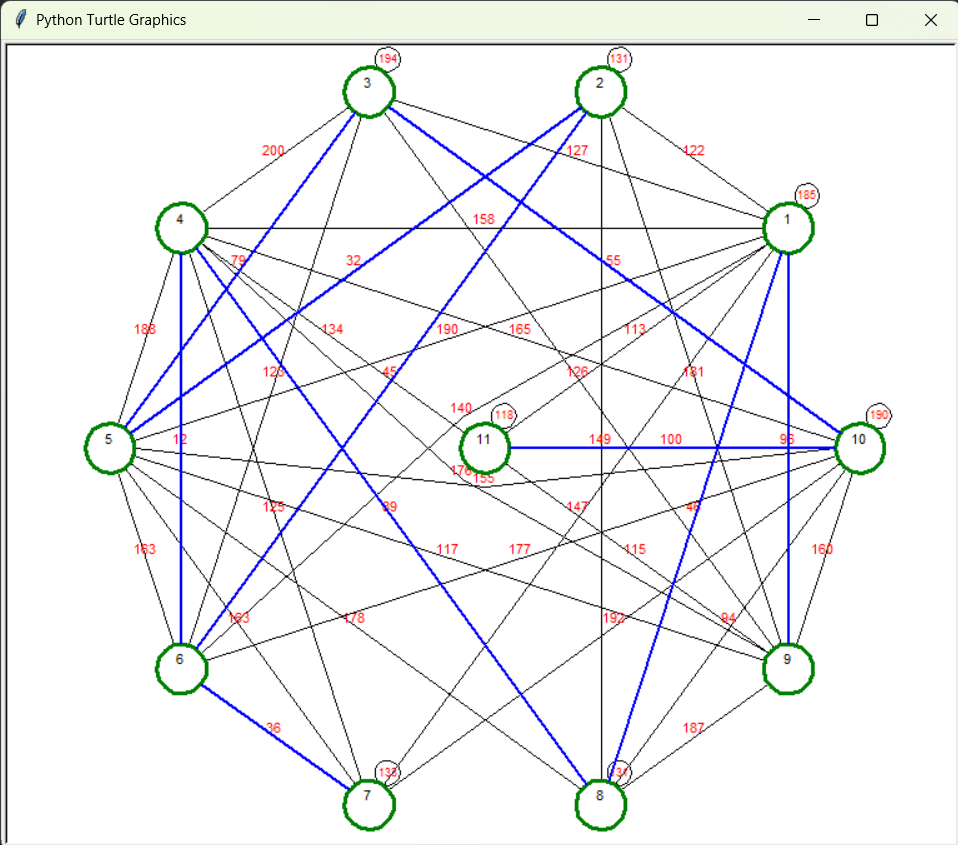
 

**Зображення графа та його мінімального кістяка**

*Граф*



*Мінімальний кістяк*



**Сума ваг ребер знайденого мінімального кістяка**



**Висновок:** під час виконання лабораторної роботи я навчився використовувати алгоритм Краскала для пошуку мінімального кістяка графа. Також для зберігання компонент зв’язності кістяка було використано union–find data structure. Ребра та вершини мінімального кістяка виділено кольором у графічному вікні, а також у консоль виведено протокол пошуку.