**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-42 Сергієнко А. М.

Лобань Михайло Юрійович

номер у списку групи: 20

Київ 2025

**Загальна постановка завдання**

1. Побудова графа

Представити зважений 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

Матриця суміжності Adir напрямленого графа за варіантом формується наступним чином:

1. Встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту n₁n₂n₃n₄.
2. Матриця розміром n × n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0).
3. Обчислюється коефіцієнт k, кожен елемент матриці множиться на 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 наступним чином:

1. Матриця B розміром n × n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0) (параметр генератора той самий — n₁n₂n₃n₄).
2. Одержується матриця C:
   * cᵢⱼ = ceil(bᵢⱼ × 100 × aundirᵢⱼ)
   * де ceil — це функція округлення до більшого цілого числа.
3. Одержується матриця D, у якій:
   * dᵢⱼ = 0, якщо cᵢⱼ = 0
   * dᵢⱼ = 1, якщо cᵢⱼ > 0
4. Одержується матриця H, у якій:
   * hᵢⱼ = 1, якщо dᵢⱼ = dⱼᵢ
   * hᵢⱼ = 0 — інакше
5. Tr — верхня трикутна матриця з одиниць (trᵢⱼ = 1, якщо i < j)
6. Матриця ваг W симетрична, її елементи обчислюються за формулою:  
       wᵢⱼ = wⱼᵢ = dᵢⱼ × hᵢⱼ × trᵢⱼ × cᵢⱼ

2. Побудова мінімального кістяка

Створити програму для знаходження мінімального кістяка:

* Алгоритм Краскала — якщо n₄ парне.
* Алгоритм Пріма — якщо n₄ непарне.

У програмі:

* Графи представляти у вигляді динамічних списків.
* Обхід графа, додавання та видалення вершин/ребер реалізовувати як окремі функції.
* Обхід графа відображати покроково — черговий крок виконується за натисканням кнопки у вікні або на клавіатурі.

3. Побудова дерева кістяка

Під час обходу графа побудувати дерево його кістяка.

У програмі дерево кістяка виводити покроково у процесі виконання алгоритму. Це можна реалізувати одним із двох способів:

* або виділяти іншим кольором ребра графа;
* або будувати кістяк поряд із графом.

4. Виведення ваг

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

**Завдання за варіантом**

**Варіант 20**

n1n2n3n4 = 4220

Кількість вершин – 10 + 2 = 12

Розміщення вершин – колом, n4 = 0

Алгоритм Краскала, n4 - парне

**Текст програм**

**Файл 1, graph.py:**import math

import tkinter as tk

import random

n3 = 2

n4 = 0

vertexes = n3 + 10

variant = 4220

random.seed(variant)

k = 1 - n3 \* 0.01 - n4 \* 0.005 - 0.05

def calculate\_element(k):

    return math.floor(random.random() \* 2 \* k)

matrix\_dir = [[0] \* vertexes for \_ in range(vertexes)]

matrix\_undir = [[0] \* vertexes for \_ in range(vertexes)]

for i in range(vertexes):

    for j in range(vertexes):

        matrix\_dir[i][j] = calculate\_element(k)

for i in range(vertexes):

    for j in range(vertexes):

        matrix\_undir[i][j] = matrix\_dir[i][j] or matrix\_dir[j][i]

root = tk.Tk()

root.title("Graph")

canvas = tk.Canvas(root, width=800, height=800, bg="white")

canvas.pack()

mid\_x = mid\_y = 400

angle = math.pi \* 2 / vertexes

R = 20

def get\_x(i):

    return mid\_x + math.sin(i \* angle) \* 300

def get\_y(i):

    return mid\_y - math.cos(i \* angle) \* 300

def rotate\_around\_center(x, y, cx, cy, theta):

    x -= cx

    y -= cy

    new\_x = x \* math.cos(theta) - y \* math.sin(theta) + cx

    new\_y = x \* math.sin(theta) + y \* math.cos(theta) + cy

    return new\_x, new\_y

def draw\_graph(matrix, paths, weights):

    vertexes = len(matrix)

    for i in range(vertexes):

        x = get\_x(i) - R

        y = get\_y(i) - R

        canvas.create\_oval(x, y, x + 2 \* R, y + 2 \* R, fill="white")

        canvas.create\_text(x + R, y + R, text=str(i + 1), font=("Montserrat", 12))

    for i in range(vertexes):

        for j in range(vertexes):

            if matrix[i][j] == 1 and i <= j:

                if i == j:

                    cx, cy = get\_x(i), get\_y(i)

                    theta = i \* angle

                    cx += R \* math.sin(theta)

                    cy -= R \* math.cos(theta)

                    dx = 3 \* R / 4

                    dy = R \* (1 - math.sqrt(7)) / 4

                    p1 = (cx - dx, cy - dy)

                    p2 = (cx - 3 \* dx / 2, cy - R / 2)

                    p3 = (cx + 3 \* dx / 2, cy - R / 2)

                    p4 = (cx + dx, cy - dy)

                    p1 = rotate\_around\_center(p1[0], p1[1], cx, cy, theta)

                    p2 = rotate\_around\_center(p2[0], p2[1], cx, cy, theta)

                    p3 = rotate\_around\_center(p3[0], p3[1], cx, cy, theta)

                    p4 = rotate\_around\_center(p4[0], p4[1], cx, cy, theta)

                    canvas.create\_line(p1[0], p1[1], p2[0], p2[1], width=2)

                    canvas.create\_line(p2[0], p2[1], p3[0], p3[1], width=2)

                    canvas.create\_line(p3[0], p3[1], p4[0], p4[1], width=2)

                else:

                    x1, y1 = get\_x(i), get\_y(i)

                    x2, y2 = get\_x(j), get\_y(j)

                    dx, dy = x2 - x1, y2 - y1

                    length = math.sqrt(dx \*\* 2 + dy \*\* 2)

                    dx /= length

                    dy /= length

                    x1 += dx \* R

                    y1 += dy \* R

                    x2 -= dx \* R

                    y2 -= dy \* R

                    canvas.create\_line(x1, y1, x2, y2, width=2, tags="arr")

                    if weights[i][j] != 0:

                        if abs(i - j) == vertexes / 2:

                            mx = (x1 + x2) / 2

                            my = (y1 + y2) / 2

                            dx = x2 - x1

                            dy = y2 - y1

                            length = math.sqrt(dx \*\* 2 + dy \*\* 2)

                            dx /= length

                            dy /= length

                            offset = 5 \* R / 2

                            mx += dx \* offset

                            my += dy \* offset

                        else:

                            mx = (x1 + x2) / 2

                            my = (y1 + y2) / 2

                        size = 9

                        canvas.create\_text(mx, my, text=str(weights[i][j]), font=("Montserrat", size), fill="black", tags="bgt")

                        canvas.create\_rectangle(mx - size - 1, my - size - 1, mx + size + 1, my + size + 1, fill="#DDDDDD", outline="black", tags="bg")

                        canvas.tag\_raise("bgt", "bg")

                        canvas.tag\_lower("arr")

    visited\_vertices = [False] \* vertexes

    def highlight\_path(k, total, total\_text\_id):

        if k < len(paths):

            i = paths[k][0]

            j = paths[k][1]

            total += weights[i][j]

            canvas.delete(total\_text\_id)

            total\_text\_id = canvas.create\_text(5\*R, R, text=f"Total weight: {total}", font=("Montserrat", 12), fill="black")

            x1, y1 = get\_x(i), get\_y(i)

            x2, y2 = get\_x(j), get\_y(j)

            dx, dy = x2 - x1, y2 - y1

            length = math.sqrt(dx \*\* 2 + dy \*\* 2)

            dx /= length

            dy /= length

            x1 += dx \* R

            y1 += dy \* R

            x2 -= dx \* R

            y2 -= dy \* R

            canvas.create\_line(x1, y1, x2, y2, width=2.5, fill="orange", tags="arr")

            visited\_vertices[i] = True

            x = get\_x(i) - R

            y = get\_y(i) - R

            canvas.create\_oval(x, y, x + 2 \* R, y + 2 \* R, fill="orange")

            canvas.create\_text(x + R, y + R, text=str(i + 1), font=("Montserrat", 12))

            if abs(i - j) == vertexes / 2:

                mx = (x1 + x2) / 2

                my = (y1 + y2) / 2

                dx = x2 - x1

                dy = y2 - y1

                length = math.sqrt(dx \*\* 2 + dy \*\* 2)

                dx /= length

                dy /= length

                offset = 5 \* R / 2

                mx += dx \* offset

                my += dy \* offset

            else:

                mx = (x1 + x2) / 2

                my = (y1 + y2) / 2

            size = 9

            canvas.create\_rectangle(mx - size - 1, my - size - 1, mx + size + 1, my + size + 1, fill="orange", outline="black", tags="bg")

            canvas.create\_text(mx, my, text=str(weights[i][j]), font=("Montserrat", size), fill="black", tags="bgt")

            canvas.tag\_raise("bgt", "bg")

            canvas.tag\_lower("arr")

            visited\_vertices[j] = True

            x = get\_x(j) - R

            y = get\_y(j) - R

            canvas.create\_oval(x, y, x + 2 \* R, y + 2 \* R, fill="orange")

            canvas.create\_text(x + R, y + R, text=str(j + 1), font=("Montserrat", 12))

            canvas.update()

            canvas.after(1000, highlight\_path, k + 1, total, total\_text\_id)

        return total\_text\_id

    total = 0

    total\_text\_id = canvas.create\_text(5\*R, R, text=f"Total weight: {total}", font=("Montserrat", 12), fill="black")

    highlight\_path(0, total, total\_text\_id)

    root.mainloop()

**Файл 2, utils.py:**import math

import random

def print\_array(arr, text, separator):

    print(text, end = " ")

    if(len(arr) == 0):

        print("no such vertexes", end=" ")

        print()

    else:

        for i in range(len(arr)):

            print(arr[i], end=separator)

        print()

def print\_matrix(matrix):

    for row in matrix:

        for element in row:

            print(element, end=" ")

        print()

def matrix\_multiply(A, B):

    n = len(A)

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

    for i in range(n):

        for j in range(n):

            for k in range(n):

                result[i][j] += A[i][k] \* B[k][j]

    return result

def matrix\_add(A, B):

    n = len(A)

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

    for i in range(n):

        for j in range(n):

            result[i][j] = A[i][j] + B[i][j]

    return result

def get\_weights(matrix\_undir):

    vertexes = len(matrix\_undir)

    B = [[0] \* vertexes for \_ in range(vertexes)]

    C = [[0] \* vertexes for \_ in range(vertexes)]

    D = [[0] \* vertexes for \_ in range(vertexes)]

    H = [[0] \* vertexes for \_ in range(vertexes)]

    W = [[0] \* vertexes for \_ in range(vertexes)]

    for i in range(vertexes):

        for j in range(vertexes):

            B[i][j] = random.random() \* 2

    for i in range(vertexes):

        for j in range(vertexes):

            C[i][j] = math.ceil(100 \* B[i][j] \* matrix\_undir[i][j])

    for i in range(vertexes):

        for j in range(vertexes):

            if(C[i][j] > 0):

                D[i][j] = 1

    for i in range(vertexes):

        for j in range(vertexes):

            if(D[i][j] != D[j][i]):

                H[i][j] = 1

    for i in range(vertexes):

        for j in range(vertexes):

            val = C[i][j] \* (D[i][j] + H[i][j] \* (i < j))

            if(i == j):

                W[i][j] = W[j][i] = 0

            elif(val == 0):

                W[i][j] = W[j][i] = math.inf

            else:

                W[i][j] = W[j][i] = val

    return W

**Файл 3, main.py:**from graph import \*

from utils import \*

print("\nUndirected matrix:\n")

print\_matrix(matrix\_undir)

W = get\_weights(matrix\_undir)

print("\nW:\n")

for row in W:

    print(" ".join(f"{num:4}" for num in row))

print()

def get\_edges(matrix, weights):

    edges = []

    size = len(matrix)

    for i in range(size):

        for j in range(i + 1, size):

            if matrix[i][j] == 1:

                edges.append((i, j, weights[i][j]))

    return edges

class UnionFind:

    def \_\_init\_\_(self, n):

        self.parent = list(range(n))

    def find(self, u):

        if self.parent[u] != u:

            self.parent[u] = self.find(self.parent[u])

        return self.parent[u]

    def union(self, u, v):

        root\_u = self.find(u)

        root\_v = self.find(v)

        if root\_u != root\_v:

            self.parent[root\_v] = root\_u

            return True

        return False

def kruskal(matrix, weights):

    edges = get\_edges(matrix, weights)

    edges.sort(key=lambda x: x[2])

    uf = UnionFind(len(matrix))

    mst\_edges = []

    for u, v, weight in edges:

        if uf.union(u, v):

            mst\_edges.append((u, v))

    return mst\_edges

edges = get\_edges(matrix\_undir, W)

mst = kruskal(matrix\_undir, W)

print\_array(edges, "Edges: ", ", ")

print\_array(mst, "MST: ", ", ")

draw\_graph(matrix\_undir, mst, W)

**Матриці суміжності**

Ненапрямлений граф:

Зображення, що містить текст, знімок екрана, Шрифт, чорно-білий

Вміст, створений ШІ, може бути неправильним.

**Згенерована матриця ваг**

Зображення, що містить текст, знімок екрана, чорно-білий, чорний

Вміст, створений ШІ, може бути неправильним.

**Зображення графа та кістяка**

Граф:

Зображення, що містить ескіз, оригамі

Вміст, створений ШІ, може бути неправильним.

Кістяк графа:

Зображення, що містить коло, Симетрія, схема

Вміст, створений ШІ, може бути неправильним.Зображення, що містить ряд, схема, коло

Вміст, створений ШІ, може бути неправильним.

**Сума ваг ребер знайденого мiнiмального кiстяка: 469**

**Висновок**

Модифікував програму лабораторної №3, щоб вона будувала мінімальний кістяк графа за алгоритмом Краскала.