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:

Студент групи ІМ-42

Лобань Михайло Юрійович

номер у списку групи: 20

Перевірив:

Сергієнко А. М.

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

$$k = 1.0 - n_3 \times 0.01 - n_4 \times 0.005 - 0.05$$

Матриця суміжності Adir напрямленого графа за варіантом формується наступним чином:

1. Встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту $n_1n_2n_3n_4$.
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 — $n_1n_2n_3n_4$).
2. Одержується матриця C:
    - $c_{ij} = ceil(b_{ij} \times 100 \times aundir_{ij})$
    - де ceil — це функція округлення до більшого цілого числа.
3. Одержується матриця D, у якій:
    - $d_{ij} = 0$, якщо $c_{ij} = 0$
    - $d_{ij} = 1$, якщо $c_{ij} > 0$
4. Одержується матриця H, у якій:
    - $h_{ij} = 1$, якщо $d_{ij} = d_{ji}$

- $h_{ij} = 0$ — інакше

5. Tr — верхня трикутна матриця з одиниць ($tr_{ij} = 1$, якщо $i < j$)
6. Матриця ваг W симетрична, її елементи обчислюються за формулою:
   $w_{ij} = w_{ji} = d_{ij} \times h_{ij} \times tr_{ij} \times c_{ij}$

2. Побудова мінімального кістяка

Створити програму для знаходження мінімального кістяка:

- Алгоритм Краскала — якщо n_4 парне.
- Алгоритм Пріма — якщо n_4 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:

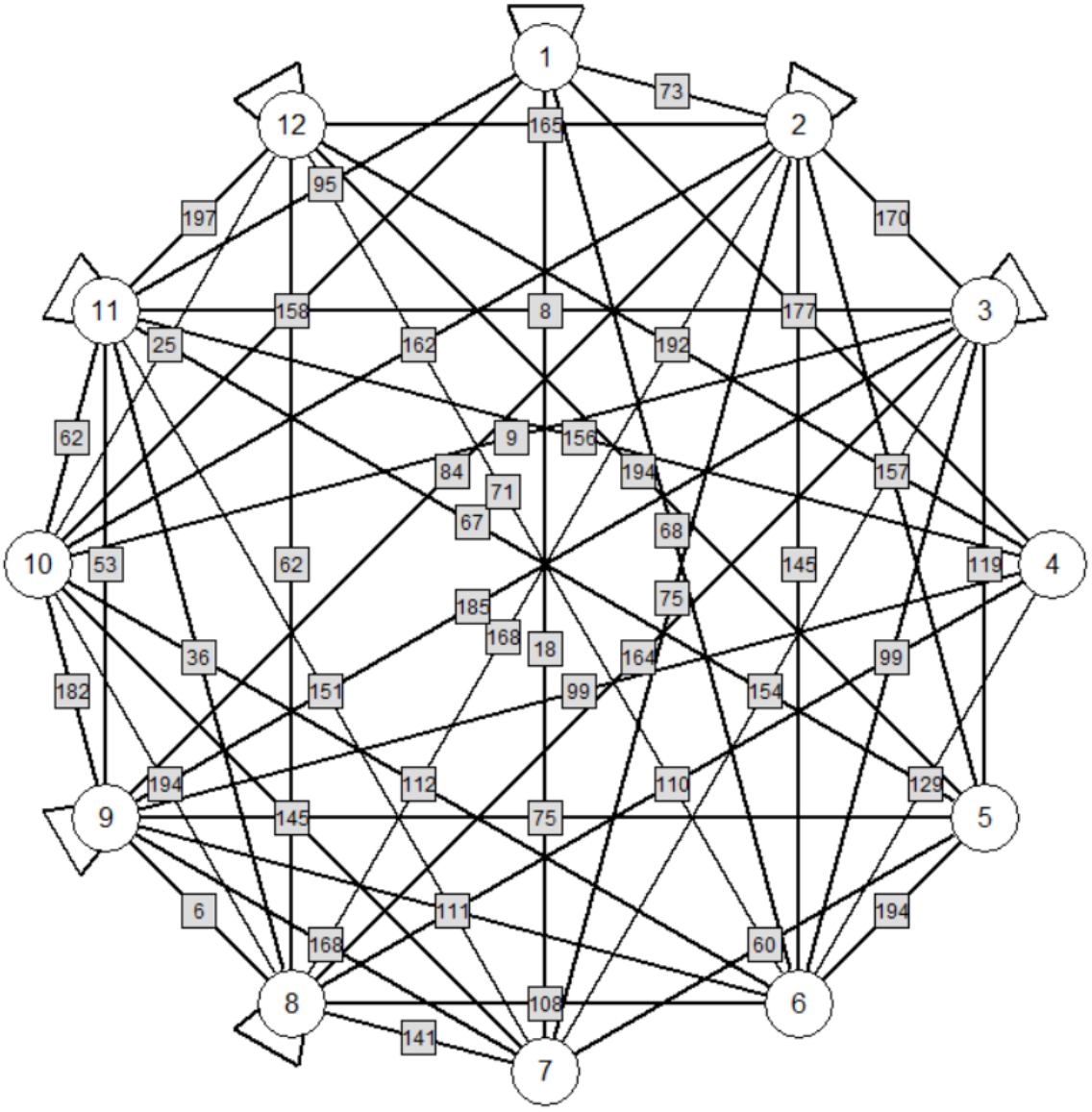
```
Undirected matrix:

1 1 0 1 0 1 1 0 0 1 1 0
1 1 1 0 1 1 1 1 1 1 0 1
0 1 1 0 1 1 1 1 1 1 1 0
1 0 0 0 0 1 0 1 1 0 1 1
0 1 1 0 0 1 1 0 1 0 1 1
1 1 1 1 1 0 0 1 1 1 0 1
1 1 1 0 1 0 0 1 1 1 1 0
0 1 1 1 0 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 0
1 1 1 0 0 1 1 1 1 0 1 1
1 0 1 1 1 0 1 1 1 1 1 1
0 1 0 1 1 1 0 1 0 1 1 1
```

Згенерована матриця ваг

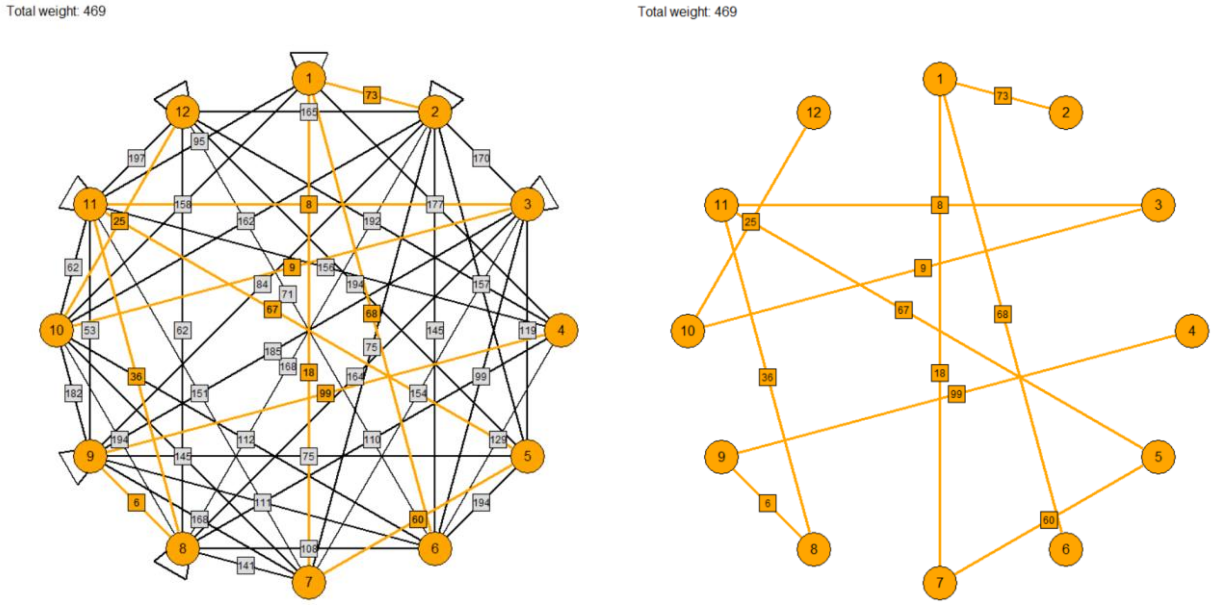
```
W:

  0   73  inf  177  inf   68   18  inf  inf  158   95  inf
 73    0  170  inf  157  145   75  168   84  162  inf  165
inf  170    0  inf  119   99  154  164  185    9    8  inf
177  inf  inf    0  inf  129  inf  110   99  inf  156  192
inf  157  119  inf    0  194   60  inf   75  inf   67  194
 68  145   99  129  194    0  inf  108  111  112  inf   71
 18   75  154  inf   60  inf    0  141  168  145  151  inf
inf  168  164  110  inf  108  141    0    6  194   36   62
inf   84  185   99   75  111  168    6    0  182   53  inf
158  162    9  inf  inf  112  145  194  182    0   62   25
 95  inf    8  156   67  inf  151   36   53   62    0  197
inf  165  inf  192  194   71  inf   62  inf   25  197    0
```

Зображення графа та кістяка

Граф:



Кістяк графа:



Сума ваг ребер знайденого мінімального кістяка: 469

Висновок

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