**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 №5**

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. Представити напрямлений граф iз заданими параметрами так само, як у лабораторнiй роботi №3.

Вiдмiннiсть: коефiцiєнт k = 1.0 - n3 \* 0.01 - n4 \* 0.005 - 0.15.

Отже, матриця сумiжностi Adir напрямленого графа за варiантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рiвне номеру варiанту n1n2n3n4;

2) матриця розмiром n \* n заповнюється згенерованими випадковими числами в дiапазонi [0, 2.0);

3) обчислюється коефiцiєнт k = 1.0 - n3 \* 0.01 - n4 \* 0.005 - 0.15, кожен елемент матрицi множиться на коефiцiєнт k;

4) елементи матрицi округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент бiльший або дорiвнює 1.0.

2. Створити програму, яка виконує обхiд напрямленого графа вшир (BFS) та вглиб (DFS).

• обхiд починати з вершини iз найменшим номером, яка має щонайменше одну вихiдну дугу;

• при обходi враховувати порядок нумерацiї;

• у програмi виконання обходу вiдображати покроково, черговий крок виконувати за натисканням кнопки у вiкнi або на клавiатурi.

3. Пiд час обходу графа побудувати дерево обходу. У програмi дерево обходу виводити покроково у процесi виконання обходу графа. Це можна виконати одним iз двох способiв:

• або видiляти iншим кольором ребра графа;

• або будувати дерево обходу поряд iз графом.

4. Змiну статусiв вершин у процесi обходу продемонструвати змiною кольорiв вершин, графiчними позначками тощо, або ж у процесi обходу виводити протокол обходу у графiчне вiкно або в консоль.

5. Якщо пiсля обходу графа лишилися невiдвiданi вершини, продовжувати обхiд з невiдвiданої вершини з найменшим номером, яка має щонайменше одну вихiдну дугу.

**Завдання за варіантом**

**Варіант 20**

n1n2n3n4 = 4220

Кількість вершин – 10 + 2 = 12

Розміщення вершин – колом, n4 = 0

**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 - 0.15

def calculate\_element(k):

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

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

for i in range(vertexes):

    for j in range(vertexes):

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

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) \* 200

def get\_y(i):

    return mid\_y - math.cos(i \* angle) \* 200

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):

    vertexes = len(matrix)

    width\_regular = 1

    width\_highlight = 3

    for i in range(vertexes):

        x = get\_x(i) - R

        y = get\_y(i) - R

        if(i == 0):

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

        else:

            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:

                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=width\_regular)

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

                    canvas.create\_line(p3[0], p3[1], p4[0], p4[1], width=width\_regular, arrow=tk.LAST)

                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=width\_regular, arrow=tk.LAST)

    visited\_vertices = [False] \* vertexes

    def highlight\_path(k):

        if k < len(paths):

            i = paths[k][0]

            j = paths[k][1]

            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=width\_highlight, arrow=tk.LAST, fill="magenta")

            canvas.update()

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

            visited\_vertices[i] = True

            x = get\_x(j) - R

            y = get\_y(j) - R

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

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

    highlight\_path(0)

    root.mainloop()

**Файл 2, utils.py:**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()

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

from utils import \*

from collections import deque

def bfs\_tree(matrix, start):

    print("\nBFS traversal order:")

    n = len(matrix)

    visited = set()

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

    queue = deque([start])

    visited.add(start)

    parent = [-1] \* n

    paths = []

    order = 1

    vertex\_order = {}

    while queue:

        node = queue.popleft()

        vertex\_order[node] = order

        print(f"{node + 1} -> {order}")

        order += 1

        for neighbor in range(n):

            if matrix[node][neighbor] == 1 and neighbor not in visited:

                visited.add(neighbor)

                queue.append(neighbor)

                tree[node][neighbor] = 1

                parent[neighbor] = node

                paths.append([node, neighbor])

    return [tree, paths, vertex\_order]

def dfs\_tree(matrix, start):

    print("\nDFS traversal order:")

    n = len(matrix)

    visited = set()

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

    stack = [(start, -1)]

    paths = []

    order = 1

    vertex\_order = {}

    while stack:

        node, parent = stack.pop()

        if node in visited:

            continue

        visited.add(node)

        vertex\_order[node] = order

        print(f"{node + 1} -> {order}")

        order += 1

        if parent != -1:

            tree[parent][node] = 1

            paths.append([parent, node])

        for neighbor in range(n - 1, -1, -1):

            if matrix[node][neighbor] == 1 and neighbor not in visited:

                stack.append((neighbor, node))

    return [tree, paths, vertex\_order]

print("\nDirected matrix:\n")

print\_matrix(matrix\_dir)

bfs\_tree\_result = bfs\_tree(matrix\_dir, 0)

bfs = bfs\_tree\_result[0]

bfs\_paths = bfs\_tree\_result[1]

bfs\_order = bfs\_tree\_result[2]

dfs\_tree\_result = dfs\_tree(matrix\_dir, 0)

dfs = dfs\_tree\_result[0]

dfs\_paths = dfs\_tree\_result[1]

dfs\_order = dfs\_tree\_result[2]

print("\nBFS tree:\n")

print\_matrix(bfs)

print("\nDFS tree:\n")

print\_matrix(dfs)

print("\nBFS vertex order:")

print(bfs\_order)

print("\nDFS vertex order:")

print(dfs\_order)

draw\_graph(matrix\_dir, bfs\_paths)

**Матриці суміжності**

Напрямлений граф:

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

Вміст, створений ШІ, може бути неправильним.

Дерево BFS:

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

Вміст, створений ШІ, може бути неправильним.

Дерево DFS:

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

Вміст, створений ШІ, може бути неправильним.

**Нові порядки вершин**

BFS:

Зображення, що містить текст, знімок екрана, Шрифт, графічний дизайн

Вміст, створений ШІ, може бути неправильним.

DFS:

Зображення, що містить текст, знімок екрана, Шрифт, дизайн

Вміст, створений ШІ, може бути неправильним.

**Зображення графа та дерева обходу**

1. Початок обходу (DFS та BFS)

Зображення, що містить коло, Симетрія, схема, ескіз

Вміст, створений ШІ, може бути неправильним.

1. Середина обходу

BFS:

Зображення, що містить схема, малюнок, коло, дизайн

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

Вміст, створений ШІ, може бути неправильним.

DFS:

Зображення, що містить схема, малюнок, дизайн, оригамі

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

Вміст, створений ШІ, може бути неправильним.

1. Кінець обходу

BFS:

Зображення, що містить малюнок, Симетрія, коло, дизайн

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

Вміст, створений ШІ, може бути неправильним.

DFS:

Зображення, що містить коло, мистецтво, Симетрія, дизайн

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

Вміст, створений ШІ, може бути неправильним.

**Висновок**

Модифікував програму лабораторної №3, щоб вона обходила граф за алгоритмами BFS та DFS, будувала дерева обходу та виводила новий порядок вершин для кожного з алгоритмів.