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»

Виконала:
студентка групи ІМ-41
Гармаш Євгенія Вадимівна
номер у списку групи: 4

Перевірив:
Сергієнко А. М.

Київ 2025

Завдання

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

Відмінність: коефіцієнт k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.15.

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;

4) елементи матриці округлюються: 0 елемент більший або дорівнює 1.0.

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

- обхід починати з вершини із найменшим номером, яка має щонайменше одну вихідну дугу;
- при обході враховувати порядок нумерації;
- у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.

3. Під час обходу графа побудувати дерево обходу. У програмі дерево обходу виводити покроково у процесі виконання обходу графа. Це можна виконати одним із двох способів:

- або виділяти іншим кольором ребра графа;
- або будувати дерево обходу поряд із графом.

4. Зміну статусів вершин у процесі обходу продемонструвати зміною кольорів вершин, графічними позначками тощо, або ж у процесі обходу виводити протокол обходу у графічне вікно або в консоль.

5. Якщо після обходу графа лишилися невідвідані вершини, продовжувати обхід з невідвіданої вершини з найменшим номером, но яка має щонайменше одну вихідну дугу.

Варіант 4:

n1n2n3n4 = 4104

Розміщення вершин: трикутником

Кількість вершин: 10 + n3 = 10

Текст програми

```
import math
import random
import tkinter as tk
from typing import List, Tuple

Coord = Tuple[float, float]
Matrix = List[List[int]]

VARIANT = 4104
PANEL_SIZE = 600
PANEL_GAP = 40
OUTER_RADIUS = 0.40 * PANEL_SIZE
NODE_RADIUS = 22
EDGE_WIDTH = 3

WHITE = "#ffffff"
GREEN = "#00cc44"
YELLOW = "#ffcc00"
RED = "#ff3333"
BLUE = "#3399ff"

class GraphState:
    def __init__(self, matrix: Matrix):
        self.matrix = matrix
        self.visited = [False] * len(matrix)
        self.queue = []
        self.stack = []
        self.tree_edges = set()
        self.current = None
        self.mode = None
        self.history = []
```

```
def reset(self):
    self.visited = [False] * len(self.matrix)
    self.queue = []
    self.stack = []
    self.tree_edges = set()
    self.current = None
    self.history = []

def next_step(self):
    if self.mode == 'BFS':
        if not self._bfs_step():
            self._print_bfs_result()
        return True
    elif self.mode == 'DFS':
        if not self._dfs_step():
            self._print_dfs_result()
        return True
    return False

def start_bfs(self):
    self.reset()
    self.mode = 'BFS'
    self._initialize()
    print("\nStarting BFS...")

def start_dfs(self):
    self.reset()
    self.mode = 'DFS'
    self._initialize()
    print("\nStarting DFS...")

def _initialize(self):
    for i, row in enumerate(self.matrix):
```

```
            if any(row):
                if self.mode == 'BFS':
                    self.queue.append(i)
                else:
                    self.stack.append(i)
                break

    def _bfs_step(self):
        while self.queue:
            v = self.queue.pop(0)
            if self.visited[v]:
                continue
            self.visited[v] = True
            self.current = v
            self.history.append(v)
            for u, connected in enumerate(self.matrix[v]):
                if connected and not self.visited[u]:
                    self.queue.append(u)
                    self.tree_edges.add((v, u))
            return True
        return False

    def _dfs_step(self):
        while self.stack:
            v = self.stack.pop()
            if self.visited[v]:
                continue
            self.visited[v] = True
            self.current = v
            self.history.append(v)
            for u in reversed(range(len(self.matrix[v]))):
                if self.matrix[v][u] and not self.visited[u]:
                    self.stack.append(u)
```

```
                self.tree_edges.add((v, u))
            return True
        return False

    def _print_matrix(self, matrix: Matrix, title: str):
        print(f"\n{title}")
        for row in matrix:
            print(" ".join(str(val) for val in row))

    def _build_tree_matrix(self) -> Matrix:
        size = len(self.matrix)
        tree = [[0] * size for _ in range(size)]
        for i, j in self.tree_edges:
            tree[i][j] = 1
        return tree

    def _print_bfs_result(self):
        print("\nBFS traversal order:")
        for i, v in enumerate(self.history, 1):
            print(f"{i} -> {v + 1}")
        self._print_matrix(self._build_tree_matrix(), "BFS tree matrix:")

    def _print_dfs_result(self):
        print("\nDFS traversal order:")
        for i, v in enumerate(self.history, 1):
            print(f"{i} -> {v + 1}")
        self._print_matrix(self._build_tree_matrix(), "DFS tree matrix:")

def generate_directed_matrix(size: int, seed: int, n3: int, n4: int) -> Matrix:
    random.seed(seed)
    k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.15
    return [[1 if random.uniform(0, 2) * k >= 1 else 0 for _ in range(size)] for _ in range(size)]
```

```
def node_positions_triangle(count: int, offset_x: int) -> List[Coord]:
    cx, cy, r = offset_x + PANEL_SIZE / 2, PANEL_SIZE / 2, OUTER_RADIUS
    angles = [math.pi/2 + i * 2*math.pi/3 for i in range(3)]
    verts = [(cx + r * math.cos(a), cy - r * math.sin(a)) for a in angles]
    base, extra = divmod(count, 3)

    def points_on_edge(p1, p2, n):
        return [(p1[0] + (p2[0]-p1[0]) * i / n, p1[1] + (p2[1]-p1[1]) * i / n) for i in range(n)]

    points = []
    for i in range(3):
        n_nodes = base + (1 if i < extra else 0)
        points += points_on_edge(verts[i], verts[(i+1)%3], n_nodes)
    return points[:count]

class GraphApp:
    def __init__(self, root, matrix: Matrix, positions: List[Coord]):
        self.root = root
        self.matrix = matrix
        self.positions = positions
        self.state = GraphState(matrix)
        self.canvas = tk.Canvas(root, width=PANEL_SIZE, height=PANEL_SIZE + 100,
bg="white")
        self.canvas.pack()

        btn_frame = tk.Frame(root)
        btn_frame.pack()
        tk.Button(btn_frame, text="Start BFS", command=self.start_bfs).pack(side=tk.LEFT)
        tk.Button(btn_frame, text="Start DFS", command=self.start_dfs).pack(side=tk.LEFT)
        tk.Button(btn_frame, text="Next Step", command=self.step).pack(side=tk.LEFT)

        self.draw_graph()

    def draw_graph(self):
```

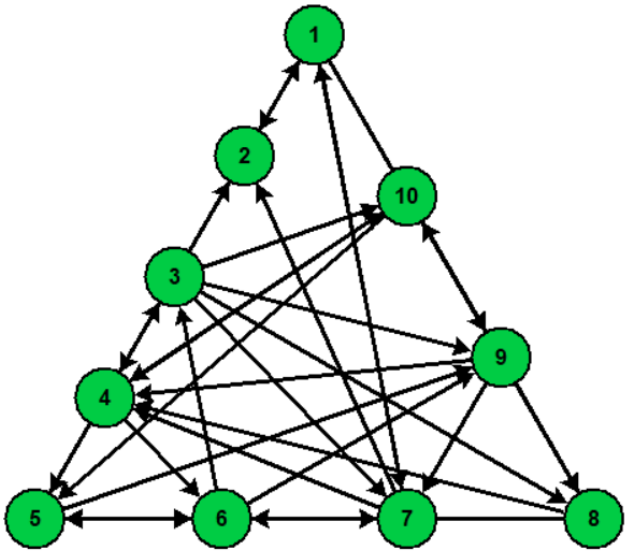
```
        self.canvas.delete("all")
        for i in range(len(self.matrix)):
            for j in range(len(self.matrix)):
                if self.matrix[i][j]:
                    color = BLUE if (i, j) in self.state.tree_edges else "black"
                    self.draw_edge(self.positions[i], self.positions[j], color=color)

        for idx, (x, y) in enumerate(self.positions):
            fill = GREEN
            if self.state.visited[idx]:
                fill = YELLOW
            if idx == self.state.current:
                fill = RED
            self.draw_node(x, y, str(idx + 1), fill)

    def draw_node(self, x, y, label, fill):
        self.canvas.create_oval(x - NODE_RADIUS, y - NODE_RADIUS, x +
NODE_RADIUS, y + NODE_RADIUS, fill=fill, outline="black", width=2)
        self.canvas.create_text(x, y, text=label, font=("Arial", 12, "bold"))

    def draw_edge(self, p1, p2, color):
        dx, dy = p2[0] - p1[0], p2[1] - p1[1]
        dist = math.hypot(dx, dy)
        if dist == 0:
            return
        start = (p1[0] + dx * NODE_RADIUS / dist, p1[1] + dy * NODE_RADIUS / dist)
        end = (p2[0] - dx * NODE_RADIUS / dist, p2[1] - dy * NODE_RADIUS / dist)
        self.canvas.create_line(*start, *end, width=EDGE_WIDTH, fill=color, arrow=tk.LAST,
arrowshape=(12, 14, 6))

    def start_bfs(self):
        self.state.start_bfs()
        self.draw_graph()
```

```
    def start_dfs(self):
        self.state.start_dfs()
        self.draw_graph()

    def step(self):
        if self.state.next_step():
            self.draw_graph()

if __name__ == "__main__":
    n1, n2, n3, n4 = map(int, str(VARIANT).zfill(4))
    vertex_count = 10 + n3
    matrix = generate_directed_matrix(vertex_count, VARIANT, n3, n4)

    print("Adjacency matrix:")
    for row in matrix:
        print(" ".join(str(x) for x in row))

    positions = node_positions_triangle(vertex_count, 0)

    root = tk.Tk()
    root.title("Graph Traversal BFS/DFS")
    app = GraphApp(root, matrix, positions)
    root.mainloop()
```

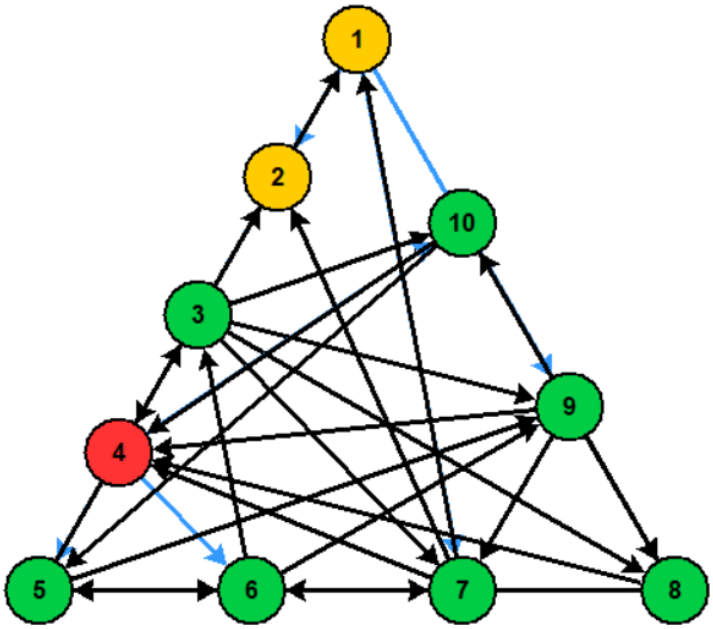
Результати тестування програми

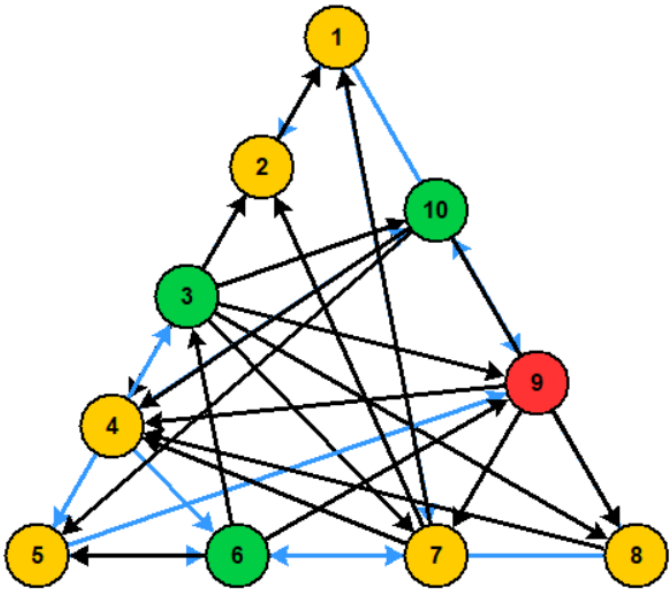
Adjacency matrix:

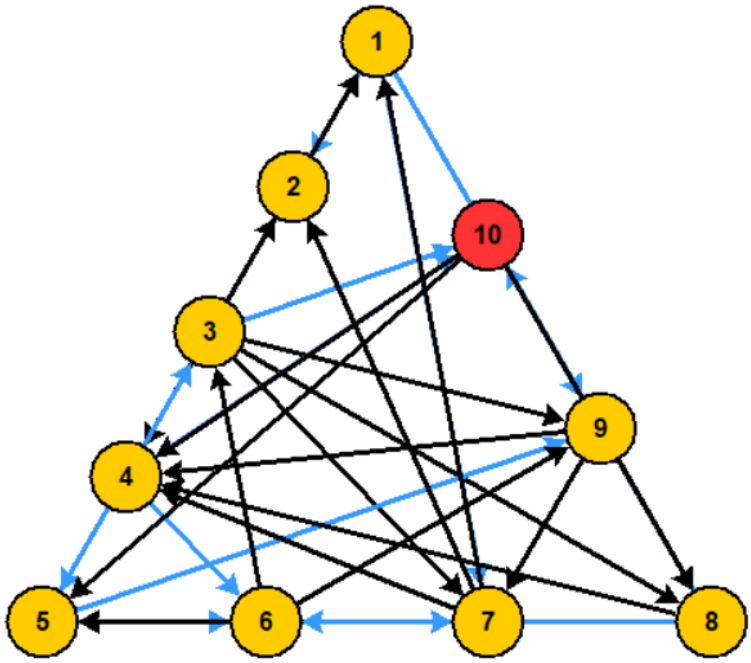
1 1 0 1 1 0 1 1 1 0
0 0 0 0 1 0 0 0 0 0
1 1 0 1 0 0 1 1 1 1
1 0 1 1 1 1 0 0 0 1
0 0 1 0 1 1 1 0 1 0
0 0 1 0 0 0 0 0 1 0
1 1 0 1 1 1 1 0 0 0
0 0 0 1 0 1 0 0 0 1
0 0 0 1 0 0 1 1 1 0
0 0 0 1 1 0 0 1 0 1



BFS







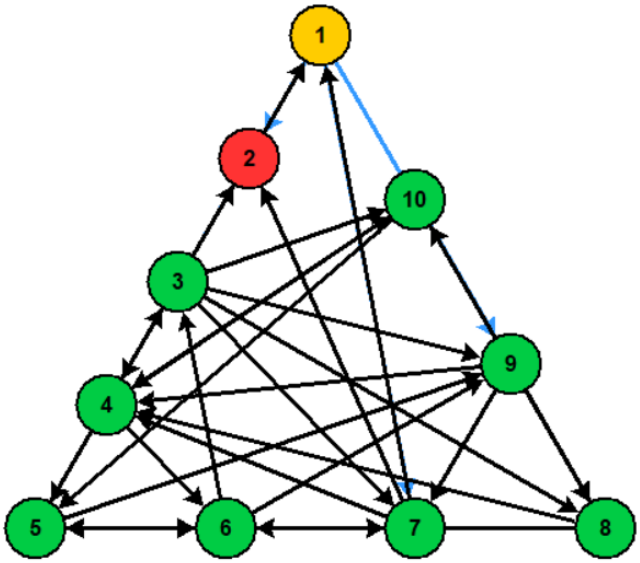
BFS traversal order:

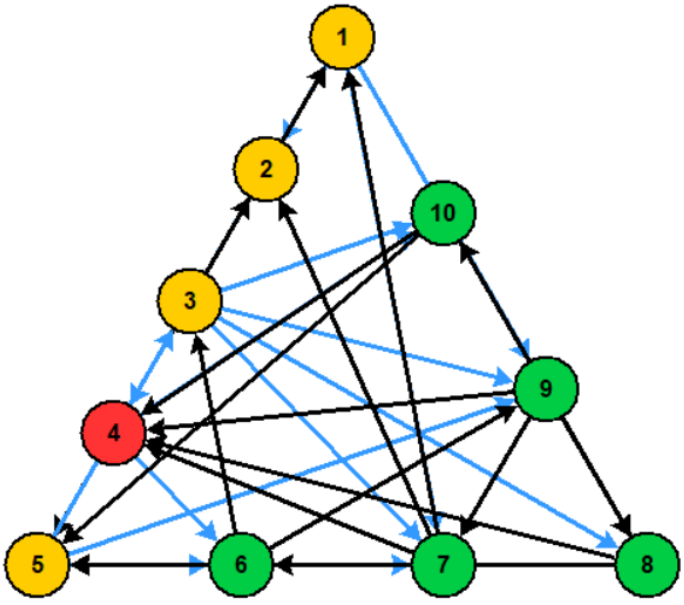
1 -> 1
2 -> 2
3 -> 4
4 -> 5
5 -> 7
6 -> 8
7 -> 9
8 -> 3
9 -> 6
10 -> 10

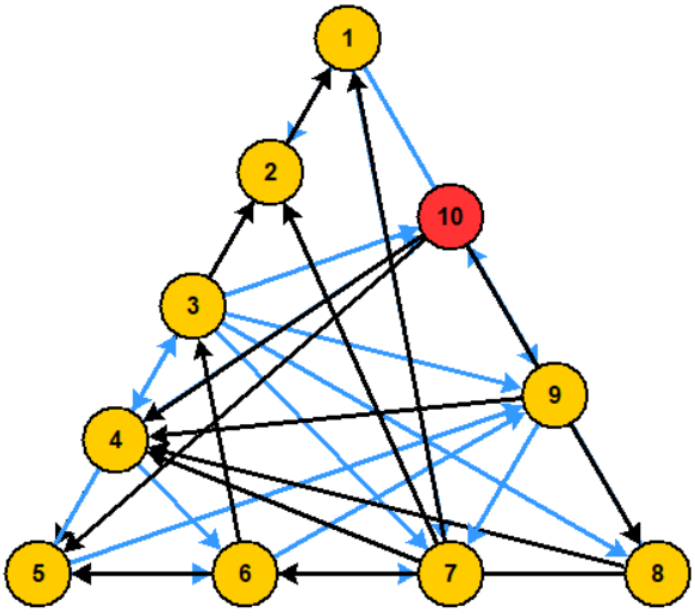
BFS tree matrix:

0 1 0 1 1 0 1 1 1 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 1 0 1 1 0 0 0 1
0 0 1 0 0 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

DFS


1
2
10
3
9
4
5
6
7
8


1
2
10
3
9
4
5
6
7
8


1
2
10
3
9
4
5
6
7
8

DFS traversal order:

1 -> 1
2 -> 2
3 -> 5
4 -> 3
5 -> 4
6 -> 6
7 -> 9
8 -> 7
9 -> 8
10 -> 10

DFS tree matrix:

0 1 0 1 1 0 1 1 1 0
0 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 1 1 1 1
0 0 0 0 0 1 0 0 0 1
0 0 1 0 0 1 1 0 1 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0

Висновок: лабораторна робота написана на мові Python. При виконанні лабораторної роботи було використано бібліотеку tkinter для гарного візуального спостереження. Код виводить матриці і всю іншу інформацію у консоль, графи у графічне вікно. Модифіковано програму лабораторної №3, щоб вона обходила граф за алгоритмами BFS та DFS, будувала дерева обходу та виводила новий порядок вершин для кожного з алгоритмів.