

**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:

Студент групи ІМ-42

Лобань Михайло Юрійович

номер у списку групи: 20

Перевірив:

Сергієнко А. М.

Київ 2025

Загальна постановка завдання

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

Відмінність: коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.15$.

Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

- 1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту $n_1n_2n_3n_4$;
- 2) матриця розміром $n * n$ заповнюється згенерованими випадковими числами в діапазоні $[0, 2.0)$;
- 3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.15$, 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.

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, яка має щонайменше одну вихідну дугу.

Завдання за варіантом

Варіант 20

$$n_1 n_2 n_3 n_4 = 4220$$

$$\text{Кількість вершин} - 10 + 2 = 12$$

$$\text{Розміщення вершин} - \text{колом}, n_4 = 0$$

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:

Directed matrix:

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

Дерево BFS:

BFS tree:

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

Дерево DFS:

DFS tree:

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

Нові порядки вершин

BFS:

BFS traversal order:

```
1 -> 1
4 -> 2
6 -> 3
7 -> 4
10 -> 5
11 -> 6
8 -> 7
2 -> 8
3 -> 9
9 -> 10
12 -> 11
5 -> 12
```

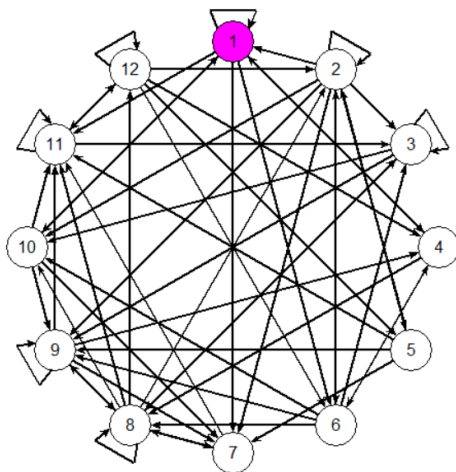
DFS:

DFS traversal order:

```
1 -> 1
4 -> 2
6 -> 3
2 -> 4
3 -> 5
8 -> 6
7 -> 7
11 -> 8
12 -> 9
5 -> 10
9 -> 11
10 -> 12
```

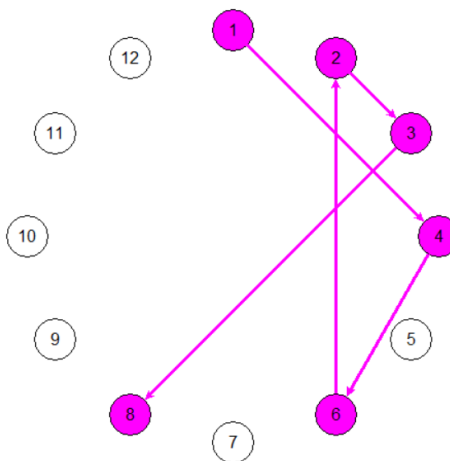
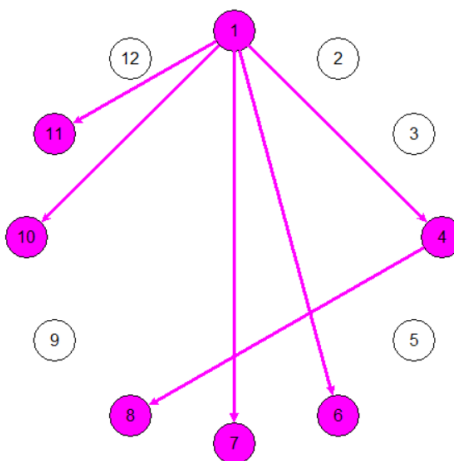
Зображення графа та дерева обходу

1) Початок обходу (DFS та BFS)

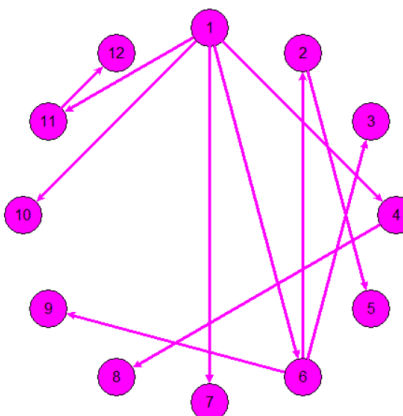
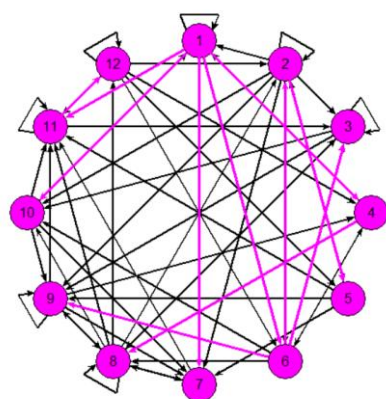


2) Середина обходу

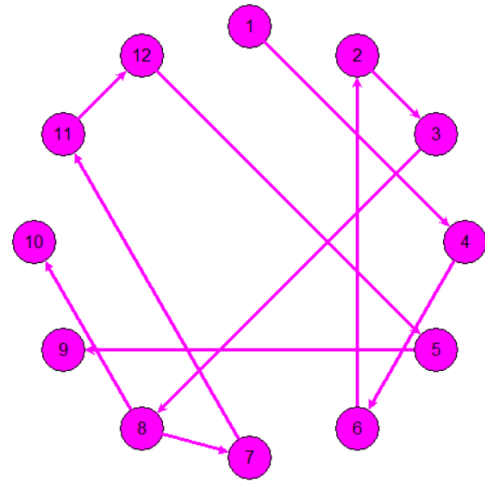
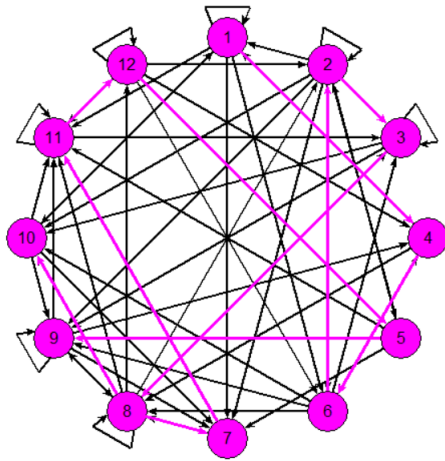
BFS:



BFS:



DFS:



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.