**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 IM-43 Сергієнко А. М.

Олексійчук Станіслав Юрійович

номер у списку групи: 22

Київ 2025

***Постановка задачі***

1. Представити напрямлений та ненапрямлений графи iз заданими пара- метрами так само, як у лабораторнiй роботi №3.

Вiдмiннiсть: коефiцiєнт ;

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

1. встановлюється параметр (seed) генератора випадкових чисел, рiв- не номеру варiанту n1n2n3n4;
2. матриця розмiром nn заповнюється згенерованими випадковими числами в дiапазонi [0, 2.0);
3. обчислюється коефiцiєнт k, кожен елемент матрицi множиться на коефiцiєнт k;

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

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

* обхiд починати з вершини iз найменшим номером, яка має щонай- менше 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.

1. Пiд час обходу графа побудувати дерево обходу. У програмi дерево об- ходу 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.

1. Змiну статусiв вершин у процесi обходу продемонструвати змiною ко- льорiв вершин, графiчними позначками тощо, або ж у процесi обходу виводити протокол обходу у графiчне вiкно або в консоль.
2. Якщо пiсля обходу графа лишилися невiдвiданi вершини, продовжува- ти обхiд з невiдвiданої вершини з найменшим номером, яка має щонай- менше одну вихiдну дугу.

***Варіант 22:***

Номер групи: 43

Номер варіанту: 22

Seed: 4322

Кількість вершин: 12

Формат графа: прямокутник (квадрат)

***Текст програм***

Це завдання було написане на мові програмування Python із використанням графічної бібліотеки tkinter; рішення розділено на певні модулі для логічності та зручності читання:

1. matrix\_print.py – функції створення матриць та їхнього виводу

import random

n1 = 4

n2 = 3

n3 = 2

n4 = 2

n = 10 + n3

random.seed(4322)

k = 1.0 - n3 \* 0.01 - n4 \* 0.005 - 0.15

directed\_matrix = [

[1 if random.uniform(0, 2.0) \* k >= 1.0 else 0 for \_ in range(n)] for \_ in range(n)

]

undirected\_matrix = [

[max(directed\_matrix[i][j], directed\_matrix[j][i]) for j in range(n)]

for i in range(n)

]

labels = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C"]

def matrix\_print(matrix, vertex, param, title):

print(f"\n {title}:\n")

print(" ", " ".join(param[:vertex]))

print(" " + "\_" \* (vertex \* 2 + 1))

for row in range(vertex):

print(

f"{param[row]} |",

" ".join(str(matrix[row][column]) for column in range(vertex)),

)

1. positions.py – створення масиву позицій вершин (4х4)

def generate\_positions():

positions = []

spacing = 120

offset = 100

for i in range(3):

positions.append((offset + i \* spacing, offset))

for i in range(3):

positions.append((offset + 3 \* spacing, offset + i \* spacing))

for i in range(3, 0, -1):

positions.append((offset + i \* spacing, offset + 3 \* spacing))

for i in range(3, 0, -1):

positions.append((offset, offset + i \* spacing))

return positions

1. graph\_traversal.py – клас для створення функцій обходу графів методами BFS/DFS, скасування (reset), виведення матриці суміжності дерева обходу та списку (вектора) відповідності номерів вершин і їх нової нумерації.

from collections import deque

class GraphTraversal:

def \_\_init\_\_(self, matrix):

self.matrix = matrix

self.n = len(matrix)

self.reset()

def reset(self):

self.visited = [False] \* self.n

self.parent = [-1] \* self.n

self.queue = deque()

self.stack = []

self.current\_vertex = -1

self.traversal\_type = None

self.traversal\_order = []

self.traversal\_tree = []

self.started = False

self.completed = False

def find\_start\_vertex(self):

for i in range(self.n):

if any(self.matrix[i]) and not self.visited[i]:

return i

return -1

def find\_unvisited\_vertex(self):

for i in range(self.n):

if not self.visited[i] and any(self.matrix[i]):

return i

return -1

def init\_bfs(self):

self.reset()

self.traversal\_type = "BFS"

start\_vertex = self.find\_start\_vertex()

if start\_vertex != -1:

self.queue.append(start\_vertex)

self.visited[start\_vertex] = True

self.current\_vertex = start\_vertex

self.traversal\_order.append(start\_vertex)

self.started = True

self.completed = False

return start\_vertex

return -1

def init\_dfs(self):

self.reset()

self.traversal\_type = "DFS"

start\_vertex = self.find\_start\_vertex()

if start\_vertex != -1:

self.stack.append(start\_vertex)

self.visited[start\_vertex] = True

self.current\_vertex = start\_vertex

self.traversal\_order.append(start\_vertex)

self.started = True

self.completed = False

return start\_vertex

return -1

def get\_unvisited\_neighbors(self, vertex):

return [

i for i in range(self.n) if self.matrix[vertex][i] and not self.visited[i]

]

def next\_step(self):

if not self.started or self.completed:

return None

if self.traversal\_type == "BFS":

return self.bfs\_step()

else:

return self.dfs\_step()

def bfs\_step(self):

if not self.queue:

start\_vertex = self.find\_unvisited\_vertex()

if start\_vertex == -1:

self.completed = True

if self.completed:

self.print\_traversal\_tree\_matrix()

self.print\_renumbering()

return None

self.queue.append(start\_vertex)

self.visited[start\_vertex] = True

self.traversal\_order.append(start\_vertex)

self.current\_vertex = start\_vertex

return start\_vertex

current = self.queue[0]

self.current\_vertex = current

neighbors = self.get\_unvisited\_neighbors(current)

if neighbors:

neighbor = neighbors[0]

self.visited[neighbor] = True

self.parent[neighbor] = current

self.traversal\_tree.append((current, neighbor))

self.queue.append(neighbor)

self.traversal\_order.append(neighbor)

self.current\_vertex = neighbor

return neighbor

else:

self.queue.popleft()

return current

def dfs\_step(self):

if not self.stack:

start\_vertex = self.find\_unvisited\_vertex()

if start\_vertex == -1:

self.completed = True

if self.completed:

self.print\_traversal\_tree\_matrix()

self.print\_renumbering()

return None

self.stack.append(start\_vertex)

self.visited[start\_vertex] = True

self.traversal\_order.append(start\_vertex)

self.current\_vertex = start\_vertex

return start\_vertex

current = self.stack[-1]

self.current\_vertex = current

neighbors = self.get\_unvisited\_neighbors(current)

if neighbors:

neighbor = neighbors[0]

self.visited[neighbor] = True

self.parent[neighbor] = current

self.traversal\_tree.append((current, neighbor))

self.stack.append(neighbor)

self.traversal\_order.append(neighbor)

self.current\_vertex = neighbor

return neighbor

else:

self.stack.pop()

return current

def get\_vertex\_color(self, vertex):

if vertex == self.current\_vertex:

return "red"

if self.visited[vertex]:

return "lightgreen"

return "lightyellow"

def print\_traversal\_tree\_matrix(self):

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

for child in range(self.n):

parent = self.parent[child]

if parent != -1:

tree\_matrix[parent][child] = 1

print("\n=== Traversal Tree Adjacency Matrix ===")

print(" " + " ".join(f"{i+1:2d}" for i in range(self.n)))

print(" " + "-" \* (self.n \* 3 + 1))

for i in range(self.n):

if self.visited[i] or any(tree\_matrix[i]):

row\_str = " ".join(f"{val:2d}" for val in tree\_matrix[i])

print(f"{i+1:2d}|{row\_str}")

def print\_renumbering(self):

print("\n=== Vertex Renumbering ===")

print("Format: [original number] -> [new number in traversal order]")

renumbering = {}

for new\_num, original\_num in enumerate(self.traversal\_order, 1):

renumbering[original\_num] = new\_num

for i in range(self.n):

if i in renumbering:

print(f"{i+1:2d} -> {renumbering[i]:2d}")

else:

print(f"{i+1:2d} -> Not reached in traversal")

1. draw\_utils.py – функції малювання орієнтованих ребер, ребер, використовуючи різні фігури для належного та зручного відтворення в подальшому графа у вікні.

import math

import tkinter as tk

def draw\_arrow(canvas, x1, y1, x2, y2, radius=25, color="darkblue", width=2):

dx, dy = x2 - x1, y2 - y1

dist = math.hypot(dx, dy)

if dist == 0:

return

dx, dy = dx / dist, dy / dist

start\_x, start\_y = x1 + dx \* radius, y1 + dy \* radius

end\_x, end\_y = x2 - dx \* radius, y2 - dy \* radius

canvas.create\_line(

start\_x, start\_y, end\_x, end\_y, arrow=tk.LAST, width=width, fill=color

)

def draw\_line(canvas, x1, y1, x2, y2, radius=25, color="green", width=2):

dx, dy = x2 - x1, y2 - y1

dist = math.hypot(dx, dy)

dx, dy = dx / dist, dy / dist

start\_x, start\_y = x1 + dx \* radius, y1 + dy \* radius

end\_x, end\_y = x2 - dx \* radius, y2 - dy \* radius

canvas.create\_line(start\_x, start\_y, end\_x, end\_y, width=width, fill=color)

def draw\_arc(

canvas, x1, y1, x2, y2, radius=25, directed=True, color="darkblue", width=2

):

dx, dy = x2 - x1, y2 - y1

dist = math.hypot(dx, dy)

if dist == 0:

return

dx, dy = dx / dist, dy / dist

start\_x, start\_y = x1 + dx \* radius, y1 + dy \* radius

end\_x, end\_y = x2 - dx \* radius, y2 - dy \* radius

mx, my = dy, -dx

norm = math.hypot(mx, my)

mx, my = mx / norm, my / norm

control\_x = (start\_x + end\_x) / 2 + mx \* 60

control\_y = (start\_y + end\_y) / 2 + my \* 60

canvas.create\_line(

start\_x,

start\_y,

control\_x,

control\_y,

end\_x,

end\_y,

smooth=True,

width=width,

fill=color,

arrow=tk.LAST if directed else None,

)

def draw\_self\_loop(canvas, x, y, directed=True, color="darkblue", width=2):

loop\_radius = 20

canvas\_width, canvas\_height = int(canvas["width"]), int(canvas["height"])

center\_x, center\_y = canvas\_width // 2, canvas\_height // 2

margin = 100

if abs(y - center\_y) < margin:

if x < center\_x:

bbox, arrow\_start, arrow\_end, angle = (

(x - 55, y - loop\_radius, x - 15, y + loop\_radius),

(x - 23, y - loop\_radius + 5),

(x - 20, y - loop\_radius + 7),

45,

)

else:

bbox, arrow\_start, arrow\_end, angle = (

(x + 15, y - loop\_radius, x + 55, y + loop\_radius),

(x + 23, y + loop\_radius - 35),

(x + 20, y + loop\_radius - 33),

225,

)

elif y < center\_y:

bbox, arrow\_start, arrow\_end, angle = (

(x - loop\_radius, y - 55, x + loop\_radius, y - 15),

(x - 16, y - loop\_radius - 5),

(x - 13, y - loop\_radius),

-45,

)

else:

bbox, arrow\_start, arrow\_end, angle = (

(x - loop\_radius, y + 15, x + loop\_radius, y + 55),

(x - loop\_radius + 4, y + 25),

(x - loop\_radius + 7, y + 20),

135,

)

canvas.create\_arc(

bbox,

start=angle,

extent=270,

style=tk.ARC,

width=width,

outline=color,

)

if directed:

canvas.create\_line(

\*arrow\_start, \*arrow\_end, width=width, fill=color, arrow=tk.LAST

)

def is\_crossing\_vertex(x1, y1, x2, y2, positions, skip\_indices, radius=25):

for i, (cx, cy) in enumerate(positions):

if i in skip\_indices:

continue

num = abs((y2 - y1) \* cx - (x2 - x1) \* cy + x2 \* y1 - y2 \* x1)

den = math.hypot(y2 - y1, x2 - x1)

if den == 0:

continue

dist = num / den

if dist < radius:

dot1 = (cx - x1) \* (x2 - x1) + (cy - y1) \* (y2 - y1)

dot2 = (cx - x2) \* (x1 - x2) + (cy - y2) \* (y1 - y2)

if dot1 > 0 and dot2 > 0:

return True

return False

1. graph\_draw.py – створення графа, використання кнопок для перемикання між напрямленим, ненапрямленим графом, методами обходу BFS, DFS, перемикання між кроками, скасування (reset) та показу матриці суміжності дерева обходу й списку (вектора) відповідності номерів вершин і їх нової нумерації, набутої в процесі обходу на той момент.

from draw\_utils import \*

from positions import generate\_positions

from matrix\_print import directed\_matrix, n

from tkinter import Button, Tk, Canvas, Frame, messagebox, Label

from graph\_traversal import GraphTraversal

threshold = 1

positions = generate\_positions()

def create\_graph\_window():

graph\_window = Tk()

graph\_window.title("Graph Visualization")

canvas = Canvas(graph\_window, width=600, height=600)

canvas.pack()

info\_frame = Frame(graph\_window)

info\_frame.pack(pady=5)

status\_label = Label(

info\_frame, text="Status: Ready for traversal", font=("Arial", 12)

)

status\_label.pack()

traversal\_info = Label(info\_frame, text="", font=("Arial", 10))

traversal\_info.pack()

button\_frame = Frame(graph\_window)

button\_frame.pack()

traversal = GraphTraversal(directed\_matrix)

def draw\_all(directed=True):

canvas.delete("all")

for i in range(n):

for j in range(n):

if directed\_matrix[i][j] == threshold:

if not directed and directed\_matrix[j][i] == threshold and j < i:

continue

x1, y1 = positions[i]

x2, y2 = positions[j]

edge\_color = "gray"

width = 1

if (i, j) in traversal.traversal\_tree:

edge\_color = "red"

width = 3

if i == j:

draw\_self\_loop(

canvas,

x1,

y1,

directed=directed,

color=edge\_color,

width=width,

)

elif is\_crossing\_vertex(

x1, y1, x2, y2, positions, skip\_indices={i, j}

):

draw\_arc(

canvas,

x1,

y1,

x2,

y2,

directed=directed,

color=edge\_color,

width=width,

)

elif directed:

if directed\_matrix[j][i] == threshold:

draw\_arc(

canvas,

x1,

y1,

x2,

y2,

directed=directed,

color=edge\_color,

width=width,

)

else:

draw\_arrow(

canvas, x1, y1, x2, y2, color=edge\_color, width=width

)

else:

draw\_line(canvas, x1, y1, x2, y2, color=edge\_color, width=width)

radius = 25

for i, (x, y) in enumerate(positions):

color = traversal.get\_vertex\_color(i)

canvas.create\_oval(

x - radius,

y - radius,

x + radius,

y + radius,

fill=color,

outline="black",

width=2,

)

canvas.create\_text(

x,

y,

text=str(i + 1),

font=("Times New Roman", 12, "bold"),

fill="black",

)

status\_text = "Status: "

if not traversal.started:

status\_text += "Ready for traversal"

traversal\_info.config(text="")

elif traversal.completed:

status\_text += "Traversal completed"

order\_text = "Traversal order: " + " → ".join(

[str(v + 1) for v in traversal.traversal\_order]

)

traversal\_info.config(text=order\_text)

status\_label.config(text=status\_text)

messagebox.showinfo(

"Traversal Complete",

"Graph traversal completed!\nTraversal tree adjacency matrix and vertex renumbering are displayed in the console.",

)

else:

if traversal.traversal\_type == "BFS":

status\_text += (

f"BFS traversal - current vertex: {traversal.current\_vertex + 1}"

)

else:

status\_text += (

f"DFS traversal - current vertex: {traversal.current\_vertex + 1}"

)

if traversal.traversal\_order:

order\_text = "Traversal order: " + " → ".join(

[str(v + 1) for v in traversal.traversal\_order]

)

traversal\_info.config(text=order\_text)

status\_label.config(text=status\_text)

def toggle\_graph():

if button\_mode.cget("text") == "Switch to Undirected":

button\_mode.config(text="Switch to Directed")

draw\_all(directed=False)

else:

button\_mode.config(text="Switch to Undirected")

draw\_all(directed=True)

def start\_bfs():

if traversal.init\_bfs() == -1:

messagebox.showwarning(

"No Start Vertex", "No vertex with outgoing edges found!"

)

draw\_all()

def start\_dfs():

if traversal.init\_dfs() == -1:

messagebox.showwarning(

"No Start Vertex", "No vertex with outgoing edges found!"

)

draw\_all()

def next\_step():

if not traversal.started:

messagebox.showwarning("Not Started", "Please start BFS or DFS first!")

return

traversal.next\_step()

draw\_all()

def reset\_traversal():

traversal.reset()

traversal\_info.config(text="")

draw\_all()

button\_mode = Button(

button\_frame, text="Switch to Undirected", command=toggle\_graph

)

button\_mode.pack(side="left", padx=5)

Button(button\_frame, text="BFS", command=start\_bfs).pack(side="left", padx=5)

Button(button\_frame, text="DFS", command=start\_dfs).pack(side="left", padx=5)

Button(button\_frame, text="Next Step", command=next\_step).pack(side="left", padx=5)

Button(button\_frame, text="Reset", command=reset\_traversal).pack(

side="left", padx=5

)

def show\_matrix\_and\_renumbering():

if not traversal.completed and traversal.started:

traversal.print\_traversal\_tree\_matrix()

traversal.print\_renumbering()

elif not traversal.started:

messagebox.showinfo("Info", "First start traversal (BFS or DFS).")

else:

traversal.print\_traversal\_tree\_matrix()

traversal.print\_renumbering()

Button(

button\_frame,

text="Show Matrix & Renumbering",

command=show\_matrix\_and\_renumbering,

).pack(side="left", padx=5)

draw\_all(directed=True)

graph\_window.mainloop()

1. main.py – вивід матриць суміжності та виконання головної графічної функції

from graph\_draw import create\_graph\_window

from matrix\_print import matrix\_print, directed\_matrix, undirected\_matrix, labels, n

matrix\_print(directed\_matrix, n, labels, "Directed graph")

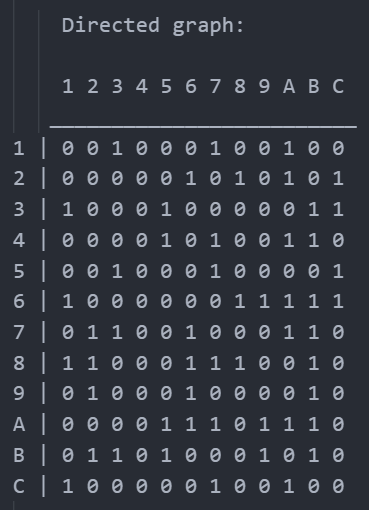
print("\n ----------------------\n")

matrix\_print(undirected\_matrix, n, labels, "Undirected graph")

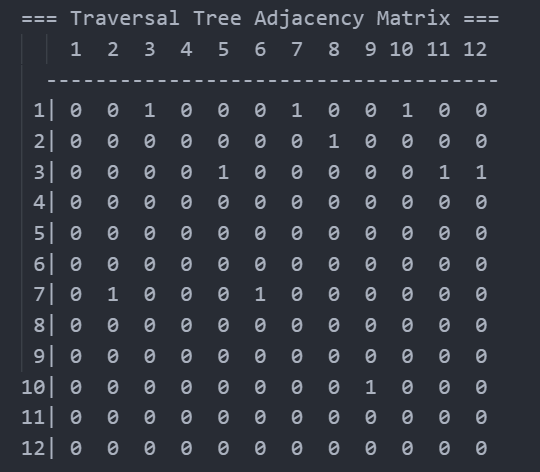
create\_graph\_window()

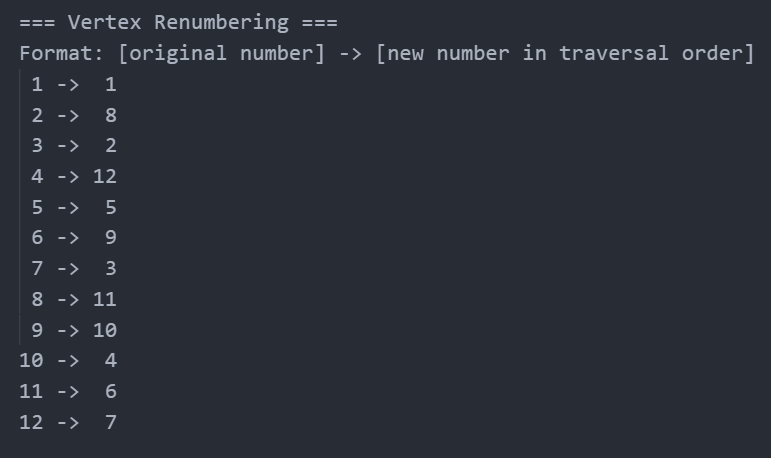
***Результати тестування програми***

1. Матриця суміжності напрямленого графа

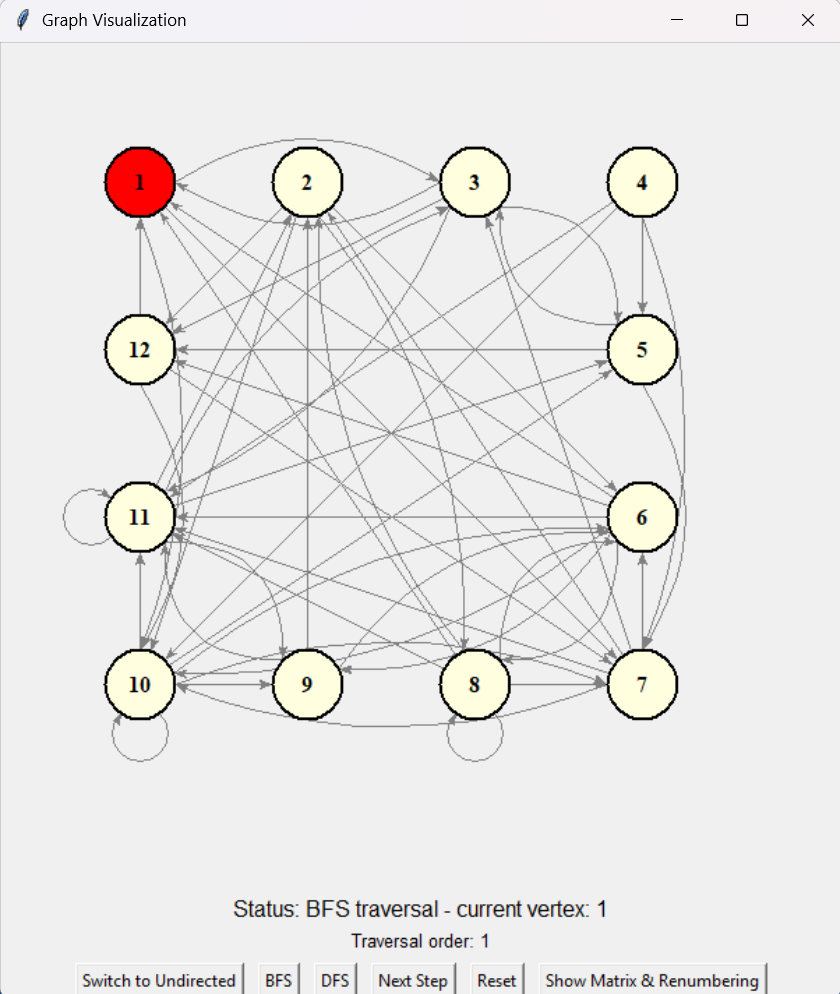


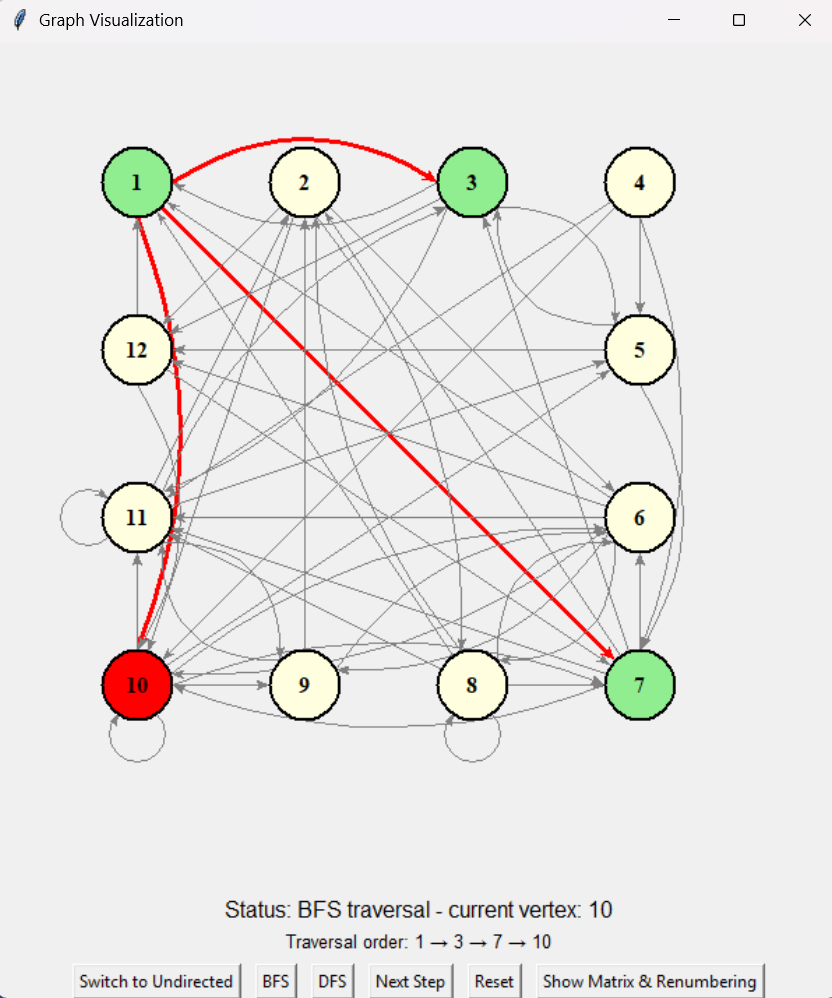
1. BFS

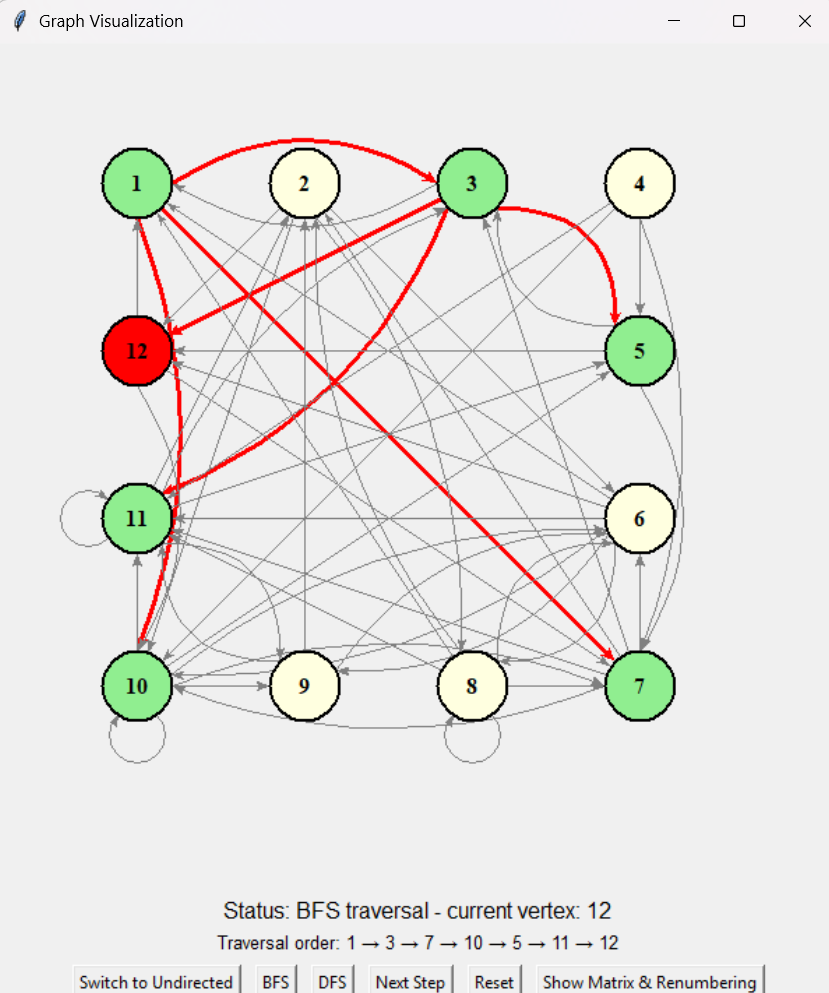
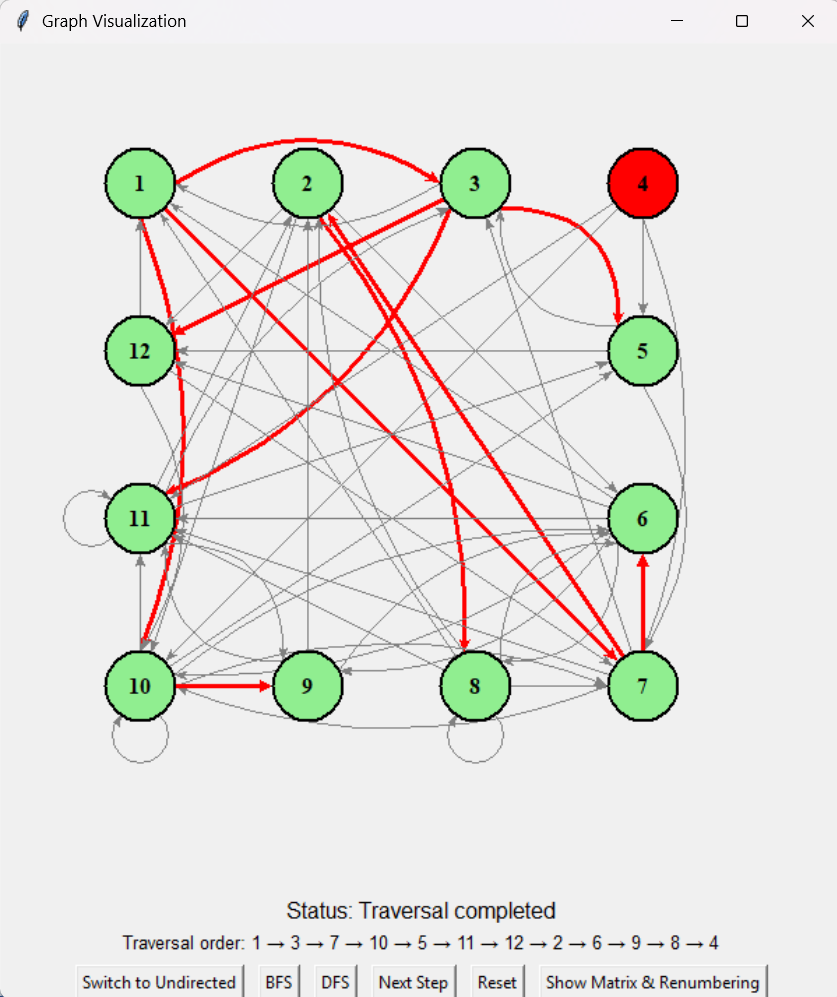




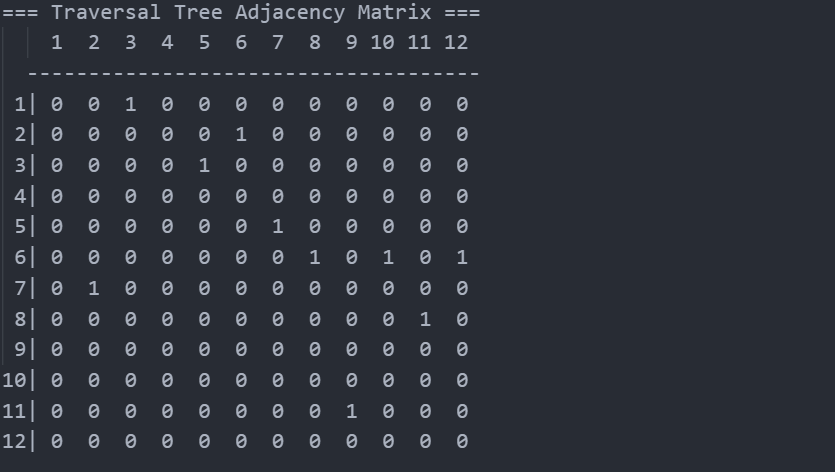
Етапи обходу графа:

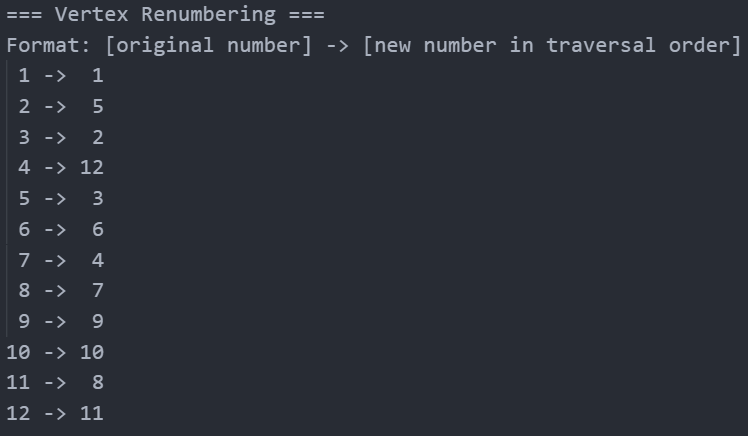




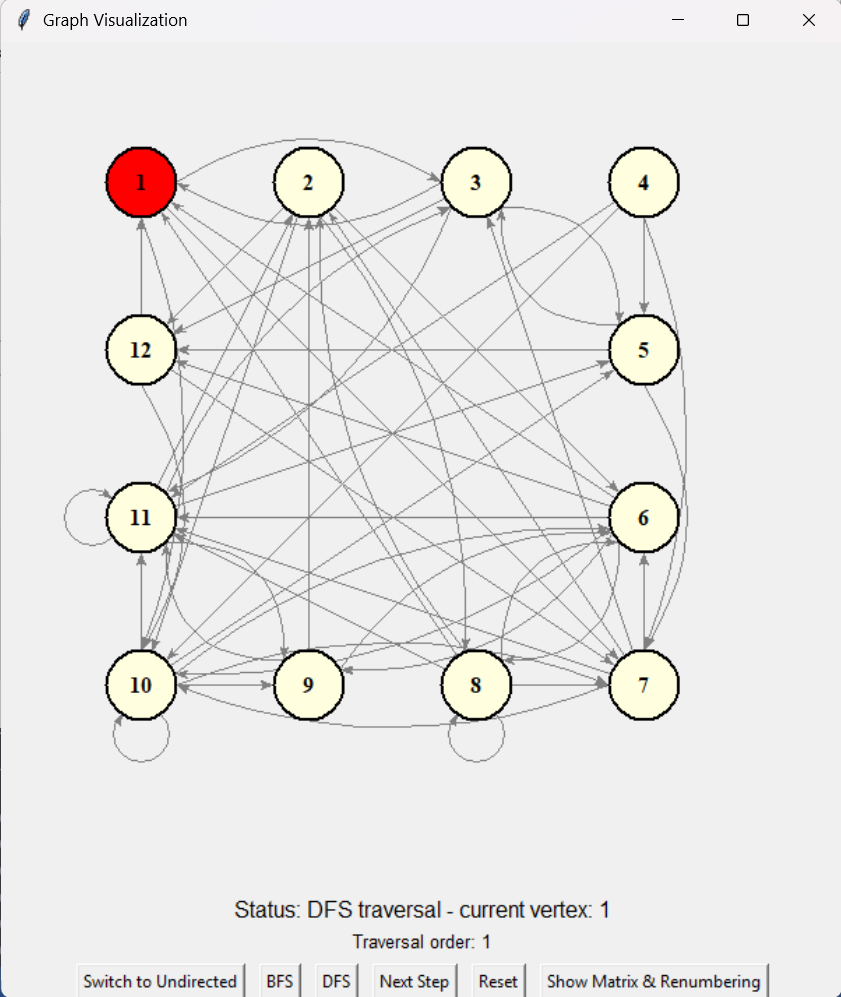
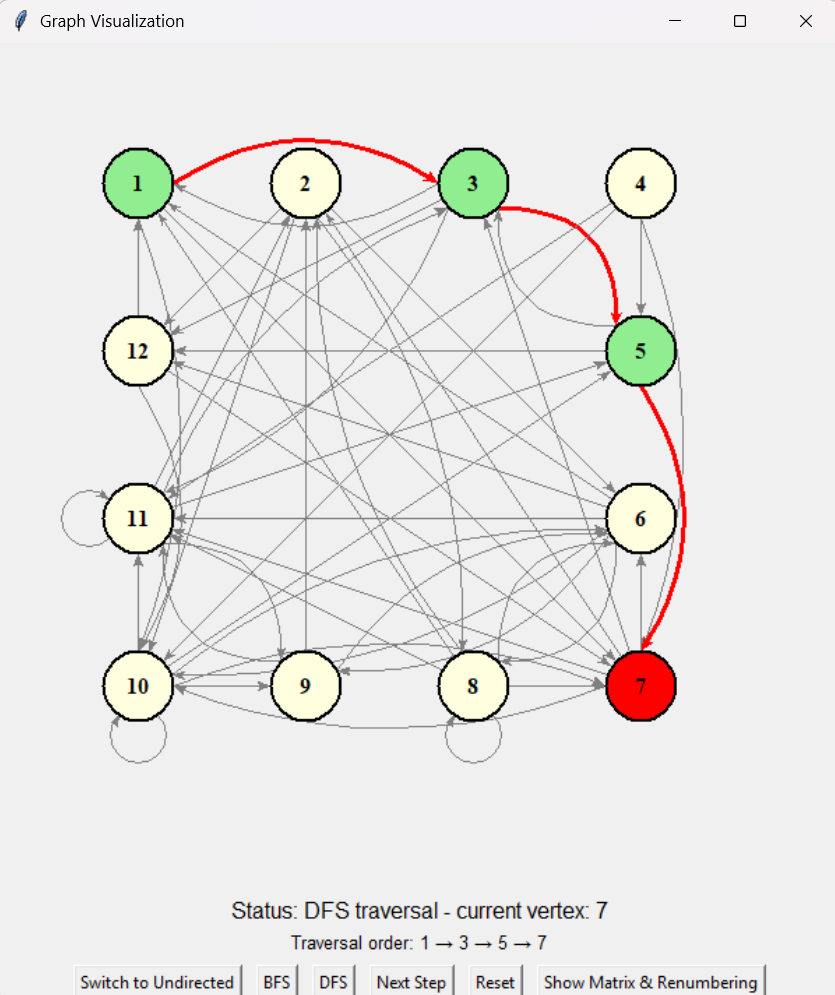
 

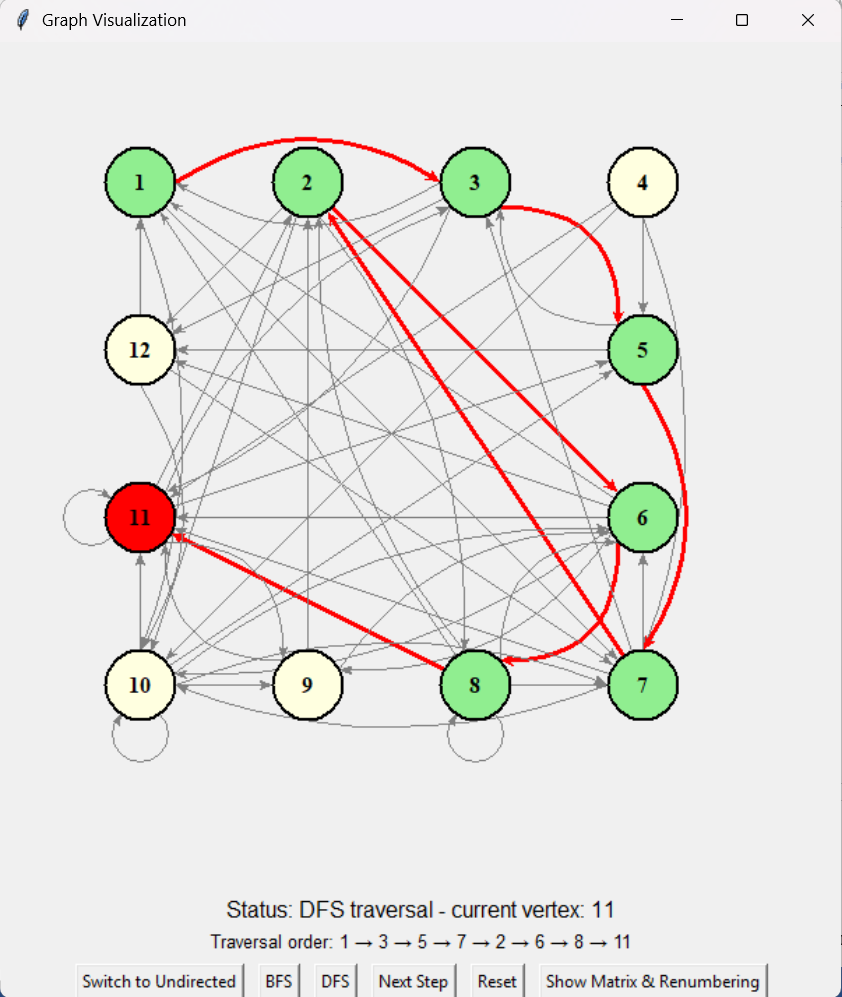
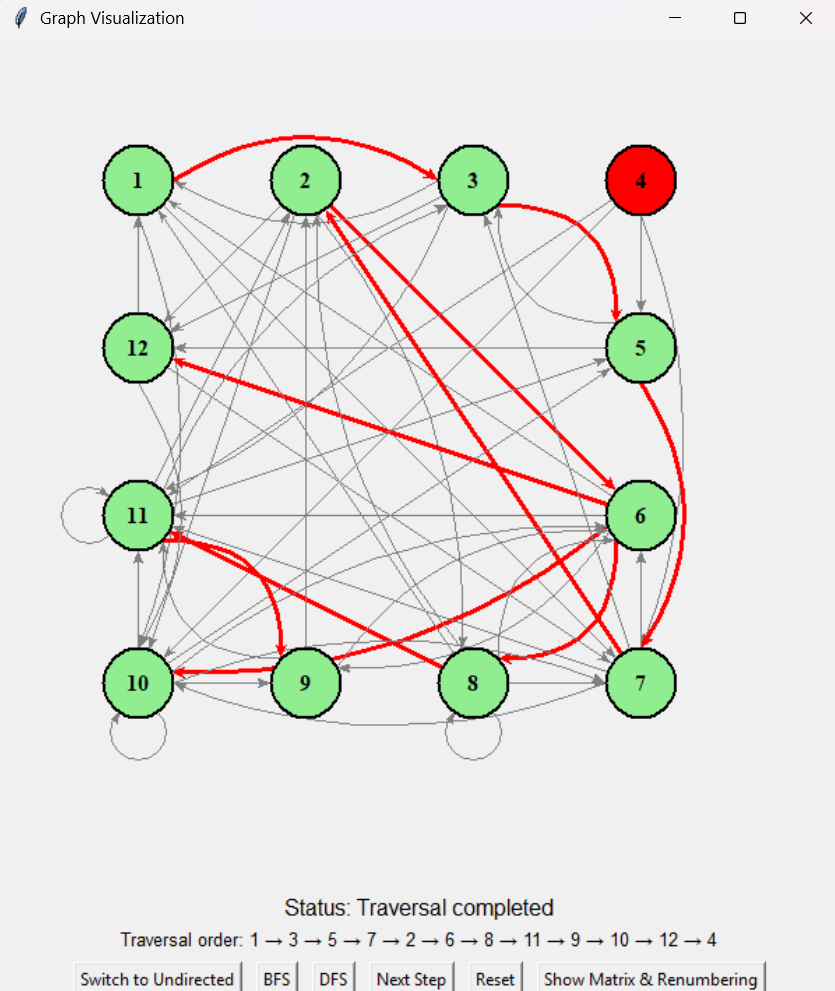
1. DFS





Етапи обходу графа:

***Висновки***

У ході виконання роботи я закріпив навички роботи з напрямленими графами, зокрема генерування графа з заданими параметрами за допомогою генератора випадкових чисел з фіксованим seed та коефіцієнтом k. Успішно реалізував алгоритми BFS та DFS з автоматичним вибором стартової вершини (найменший номер з вихідними дугами) та покроковим виконанням. Опанував методику візуалізації обходу через виділення ребер дерева обходу кольором, динамічну зміну кольорів вершин (поточна, відвідана, невідвідана) та виведення матриці суміжності дерева обходу. Отримані результати підтвердили коректність реалізації алгоритмів та їх ефективність для аналізу різних типів графів. Ця робота значно поглибила моє розуміння алгоритмів обходу графів та методів їх візуалізації, що є важливим для подальшого вивчення складніших алгоритмів.