**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 №4**

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 фор- мується таким чином:

1. встановлюється параметр (seed) генератора випадкових чисел, рiвне номеру варiанту n1n2n3n4;
2. матриця розмiром n n заповнюється згенерованими випадковими
3. числами в дiапазонi [0, 2.0);
4. обчислюється коефiцiєнт , кожен елемент матрицi множиться на коефiцiєнт k;
5. елементи матрицi округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент бiльший або дорiвнює 1.0.

2. Обчислити:

1. степенi вершин напрямленого i ненапрямленого графiв;
2. напiвстепенi виходу та заходу напрямленого графа;
3. чи є граф однорiдним (регулярним), i якщо так, вказати степiнь однорiдностi графа;
4. перелiк висячих та iзольованих вершин.

Результати вивести у графiчне вiкно, консоль або файл.

3. Змiнити матрицю Adir, коефiцiєнт .

4. Для нового орграфа обчислити:

1. пiвстепенi вершин;
2. всi шляхи довжини 2 i 3;
3. матрицю досяжностi;
4. матрицю сильної зв’язностi;
5. перелiк компонент сильної зв’язностi;
6. граф конденсацiї.

Результати вивести у графiчне вiкно, в консоль або файл.

Шляхи довжиною 2 i 3 слiд шукати за матрицями A2 i A3, вiдповiдно. Як результат вивести перелiк шляхiв, включно з усiма промiжними вершинами, через якi проходить шлях.

Матрицю досяжностi та компоненти сильної зв’язностi слiд шукати за допомогою операцiї транзитивного замикання. У перелiку компонент слiд вказати, якi вершини належать до кожної компоненти.

Граф конденсацiї вивести у графiчне вiкно.

***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. shared\_data.py – базові дані для подальшої роботи

n1, n2, n3, n4 = 4, 3, 2, 2

n = 10 + n3

k1 = 1.0 - n3 \* 0.01 - n4 \* 0.01 - 0.3

k2 = 1.0 - n3 \* 0.005 - n4 \* 0.005 - 0.27

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

numeric\_labels = [str(i + 1) for i in range(n)]

directed\_matrix = []

undirected\_matrix = []

k = k1

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. condensational\_matrix.py – функції для знаходження матриці сильної зв’язності, пошуку компонентів сильної зв’язності та їхнього використання для побудови матриці конденсації.

def transitive\_closure(matrix):

size = len(matrix)

closure = [row[:] for row in matrix]

for k in range(size):

for i in range(size):

for j in range(size):

closure[i][j] = closure[i][j] or (closure[i][k] and closure[k][j])

return closure

def strong\_connectivity\_matrix(matrix):

n = len(matrix)

R = transitive\_closure(matrix)

S = [[int(R[i][j] and R[j][i]) for j in range(n)] for i in range(n)]

return S

def find\_strongly\_connected\_components(matrix):

size = len(matrix)

S = strong\_connectivity\_matrix(matrix)

components = []

visited = [False] \* size

def dfs(v, component):

visited[v] = True

component.append(v)

for i in range(size):

if not visited[i] and S[v][i]:

dfs(i, component)

for v in range(size):

if not visited[v]:

component = []

dfs(v, component)

components.append(component)

return components

def build\_condensation\_matrix(adj\_matrix, components):

n = len(components)

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

for i in range(n):

for j in range(n):

if i == j:

continue

if any(adj\_matrix[u][v] for u in components[i] for v in components[j]):

cond\_matrix[i][j] = 1

return cond\_matrix

def get\_condensation\_params(adj\_matrix, components):

cond\_matrix = build\_condensation\_matrix(adj\_matrix, components)

vertex = len(components)

param = [f"C{i+1}" for i in range(vertex)]

title = "Condensation Matrix"

return cond\_matrix, vertex, param, title

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):

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=2, fill="darkblue"

)

def draw\_line(canvas, x1, y1, x2, y2, radius=25):

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=2, fill="green")

def draw\_arc(canvas, x1, y1, x2, y2, radius=25, directed=True):

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=2,

fill="darkblue" if directed else "green",

arrow=tk.LAST if directed else None,

)

def draw\_self\_loop(canvas, x, y, directed=True):

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=2,

outline="darkblue" if directed else "green",

)

if directed:

canvas.create\_line(

\*arrow\_start, \*arrow\_end, width=2, fill="darkblue", 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

def draw\_graph(canvas, positions, radius=25):

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

canvas.create\_oval(

x - radius,

y - radius,

x + radius,

y + radius,

fill="lightyellow",

outline="black",

width=2,

)

canvas.create\_text(

x, y, text=str(i + 1), font=("Times New Roman", 12, "bold"), fill="black"

)

1. graph\_draw.py – створення графа, використання кнопок для перемикання між напрямленим, ненапрямленим графом та графом конденсації.

from tkinter import Tk, Canvas, Button, Frame

from draw\_utils import \*

from positions import generate\_positions

from shared\_data import n

import shared\_data, math

from condensational\_matrix import find\_strongly\_connected\_components

threshold = 1

positions = generate\_positions()

def draw\_condensation\_graph(canvas, directed=True):

canvas.delete("all")

components = find\_strongly\_connected\_components(shared\_data.directed\_matrix)

component\_positions = []

radius = 30

spacing = 100

for i, component in enumerate(components):

angle = 2 \* 3.1415 \* i / len(components)

x = 300 + 200 \* math.cos(angle)

y = 300 + 200 \* math.sin(angle)

component\_positions.append((x, y))

for i, (x, y) in enumerate(component\_positions):

canvas.create\_oval(

x - radius,

y - radius,

x + radius,

y + radius,

fill="lightyellow",

outline="black",

width=2,

)

component\_label = ", ".join(

shared\_data.numeric\_labels[v] for v in components[i]

)

canvas.create\_text(x, y, text=component\_label, font=("Times New Roman", 10))

for i in range(len(components)):

for j in range(len(components)):

if i == j:

continue

has\_edge = any(

shared\_data.directed\_matrix[u][v] == threshold

for u in components[i]

for v in components[j]

)

if has\_edge:

x1, y1 = component\_positions[i]

x2, y2 = component\_positions[j]

draw\_arrow(canvas, x1, y1, x2, y2, radius=radius)

def create\_graph\_window():

graph\_window = Tk()

graph\_window.title("Graph Visualization")

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

canvas.pack()

button\_frame = Frame(graph\_window)

button\_frame.pack()

def draw\_all(directed=True):

canvas.delete("all")

draw\_graph(canvas, positions)

for i in range(n):

for j in range(n):

if shared\_data.directed\_matrix[i][j] == threshold:

if (

not directed

and shared\_data.directed\_matrix[j][i] == threshold

and j < i

):

continue

x1, y1 = positions[i]

x2, y2 = positions[j]

if i == j:

draw\_self\_loop(canvas, x1, y1, directed=directed)

elif is\_crossing\_vertex(

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

):

draw\_arc(canvas, x1, y1, x2, y2, directed=directed)

elif directed:

if shared\_data.directed\_matrix[j][i] == threshold:

draw\_arc(canvas, x1, y1, x2, y2, directed=directed)

else:

draw\_arrow(canvas, x1, y1, x2, y2)

else:

draw\_line(canvas, x1, y1, x2, y2)

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 show\_condensation():

draw\_condensation\_graph(canvas)

button\_mode = Button(

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

)

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

button\_condensation = Button(

button\_frame, text="Switch to Condensational", command=show\_condensation

)

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

draw\_all(directed=True)

graph\_window.mainloop()

1. main.py – вивід результатів усіх пунктів лабораторної роботи в консоль та створення логіки перемикання між двома заданими коефіцієнтами k

import random, math, tkinter as tk

from shared\_data import n, k1, k2, labels

import shared\_data

from graph\_draw import create\_graph\_window

from condensational\_matrix import \*

random.seed(4322)

def tkinter\_print():

create\_graph()

root.destroy()

create\_graph\_window()

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

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

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

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

for row in range(vertex):

print(

f"{param[row]} |", " ".join(str(matrix[row][col]) for col in range(vertex))

)

def set\_k1():

shared\_data.k = k1

k\_label.config(text=f"Selected k1: {shared\_data.k}")

def set\_k2():

shared\_data.k = k2

k\_label.config(text=f"Selected k2: {shared\_data.k}")

def get\_degrees\_undirected(matrix):

return [sum(row) for row in matrix]

def get\_degrees\_directed(matrix):

in\_degree = [sum(row[col] for row in matrix) for col in range(len(matrix))]

out\_degree = [sum(row) for row in matrix]

return in\_degree, out\_degree

def get\_regular\_degree(degrees):

if isinstance(degrees, tuple) and len(degrees) == 2:

in\_deg, out\_deg = degrees

if all(deg == in\_deg[0] for deg in in\_deg) and all(

deg == out\_deg[0] for deg in out\_deg

):

return (in\_deg[0], out\_deg[0])

elif isinstance(degrees, list):

if all(deg == degrees[0] for deg in degrees):

return degrees[0]

return None

def find\_isolated\_vertices(matrix):

return [i for i, row in enumerate(matrix) if sum(row) == 0]

def find\_hanging\_vertices(matrix):

n = len(matrix)

hanging = []

for i in range(n):

out\_degree = sum(matrix[i])

in\_degree = sum(row[i] for row in matrix)

if in\_degree + out\_degree == 1:

hanging.append(i)

return hanging

def format\_vertices(indices):

return ", ".join(f"({shared\_data.numeric\_labels[i]})" for i in indices)

def multiply\_matrices(A, B):

size = len(A)

result = [[0] \* size for \_ in range(size)]

for i in range(size):

for j in range(size):

for k in range(size):

result[i][j] += A[i][k] \* B[k][j]

return result

def create\_graph():

shared\_data.directed\_matrix = [

[math.floor(random.uniform(0, 2.0) \* shared\_data.k) for \_ in range(n)]

for \_ in range(n)

]

shared\_data.undirected\_matrix = [

[

max(shared\_data.directed\_matrix[i][j], shared\_data.directed\_matrix[j][i])

for j in range(n)

]

for i in range(n)

]

print(f"\nCurrent k: {shared\_data.k}")

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

matrix\_print(

shared\_data.directed\_matrix, n, labels, "Directed Graph Adjancency Matrix"

)

matrix\_print(

shared\_data.undirected\_matrix, n, labels, "Undirected Graph Adjancency Matrix"

)

in\_deg, out\_deg = get\_degrees\_directed(shared\_data.directed\_matrix)

print("\nVertex in the directed graph:")

for i in range(n):

print(f"({i + 1}): in = {in\_deg[i]}, out = {out\_deg[i]}")

degrees\_undirected = get\_degrees\_undirected(shared\_data.undirected\_matrix)

print("\nVertex in the undirected graph:")

for i, degree in enumerate(degrees\_undirected):

print(f"({i + 1}): {degree}")

print("\nIs regular (undirected):")

reg\_deg = get\_regular\_degree(degrees\_undirected)

print(

f"The graph is regular. Degree of regularity: {reg\_deg}"

if reg\_deg

else "The graph is not regular."

)

print("\nIs regular (directed):")

reg\_deg\_directed = get\_regular\_degree((in\_deg, out\_deg))

if reg\_deg\_directed:

in\_reg, out\_reg = reg\_deg\_directed

print(f"The graph is regular. In-degree: {in\_reg}, Out-degree: {out\_reg}")

else:

print("The graph is not regular.")

print("\nIsolated vertices:")

isolated = find\_isolated\_vertices(shared\_data.undirected\_matrix)

print(format\_vertices(isolated) if isolated else "- None found")

print("\nHanging vertices:")

hanging = find\_hanging\_vertices(shared\_data.directed\_matrix)

print(format\_vertices(hanging) if hanging else "- None found")

print("\nPaths of length 2:")

A2 = multiply\_matrices(shared\_data.directed\_matrix, shared\_data.directed\_matrix)

for i in range(n):

for j in range(n):

if A2[i][j] > 0:

for k in range(n):

if (

shared\_data.directed\_matrix[i][k] > 0

and shared\_data.directed\_matrix[k][j] > 0

):

print(f"{i + 1} -> {k + 1} -> {j + 1}")

print("\nPaths of length 3:")

A3 = multiply\_matrices(A2, shared\_data.directed\_matrix)

for i in range(n):

for j in range(n):

if A3[i][j] > 0:

for k in range(n):

if (

shared\_data.directed\_matrix[i][k] > 0

and shared\_data.directed\_matrix[k][j] > 0

):

for l in range(n):

if shared\_data.directed\_matrix[j][l] > 0:

print(f"{i + 1} -> {k + 1} -> {j + 1} -> {l + 1}")

reachability\_matrix = transitive\_closure(shared\_data.directed\_matrix)

matrix\_print(reachability\_matrix, n, labels, "Reachability Matrix")

strong\_matrix = strong\_connectivity\_matrix(shared\_data.directed\_matrix)

matrix\_print(strong\_matrix, n, labels, "Strong Connectivity Matrix")

print("\nStrongly Connected Components:")

components = find\_strongly\_connected\_components(shared\_data.directed\_matrix)

for i, component in enumerate(components):

print(f"Component {i+1}:", format\_vertices(component))

if components:

cond\_matrix, vertex, param, title = get\_condensation\_params(

shared\_data.directed\_matrix, components

)

matrix\_print(cond\_matrix, vertex, param, title)

root = tk.Tk()

root.title("Select k")

canvas = tk.Canvas(root, width=250, height=20)

canvas.pack()

button\_k1 = tk.Button(root, text="Use k1", command=set\_k1)

button\_k1.pack(pady=5)

button\_k2 = tk.Button(root, text="Use k2", command=set\_k2)

button\_k2.pack(pady=5)

create\_button = tk.Button(root, text="Create Graph", command=tkinter\_print)

create\_button.pack(pady=10)

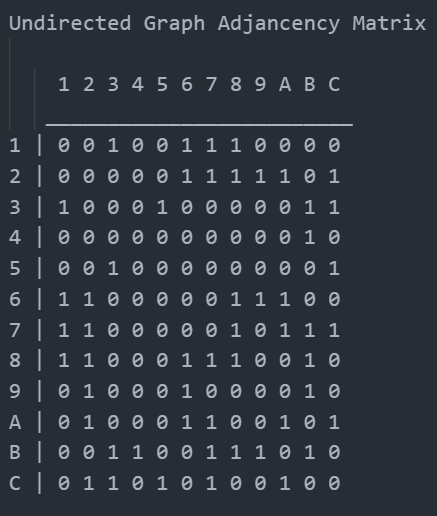
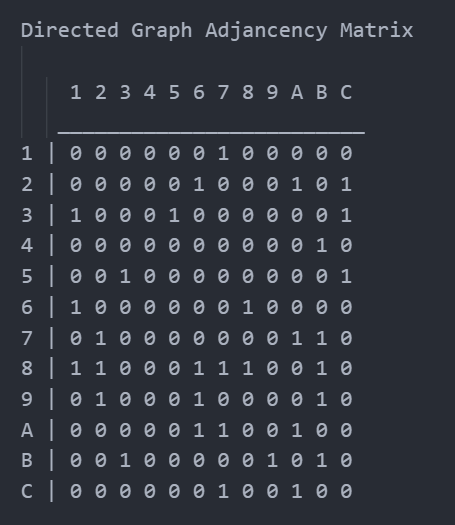
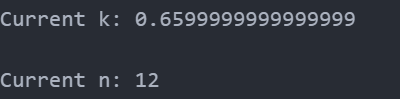
k\_label = tk.Label(root, text=f"Current k: {shared\_data.k}")

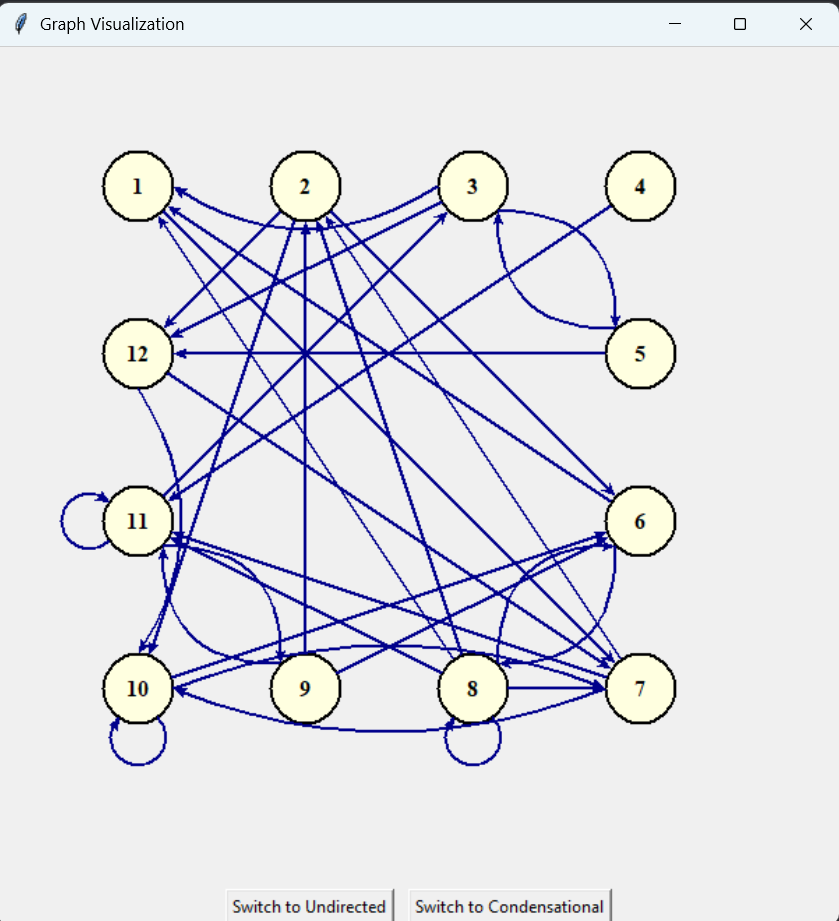
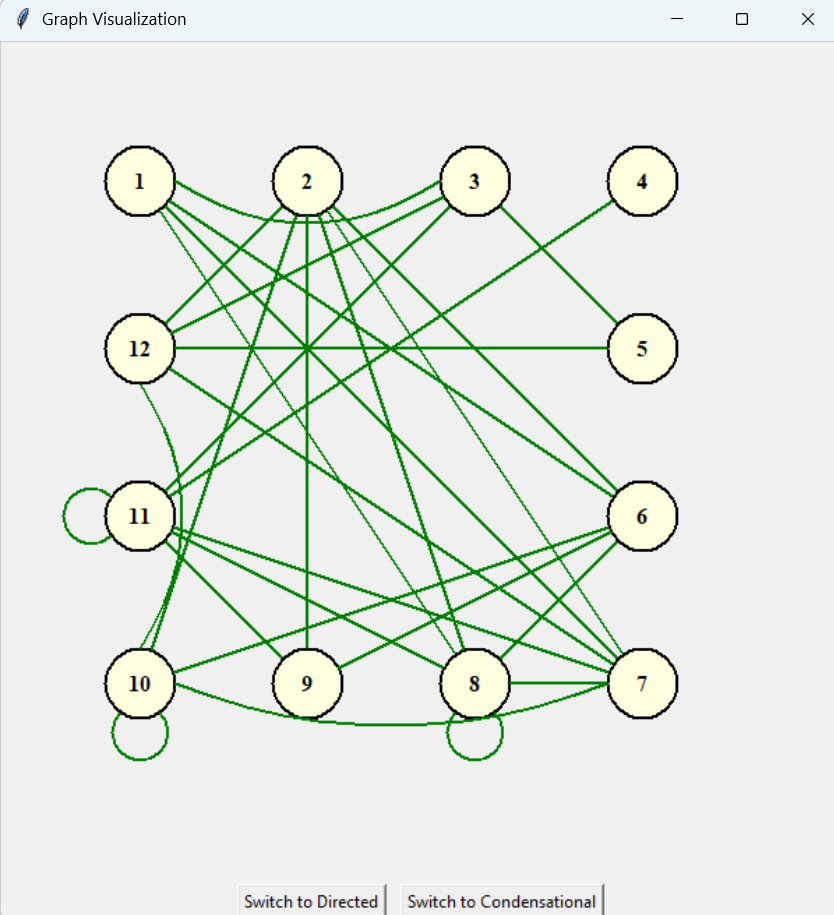
k\_label.pack()

root.mainloop()

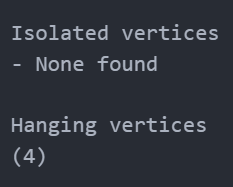
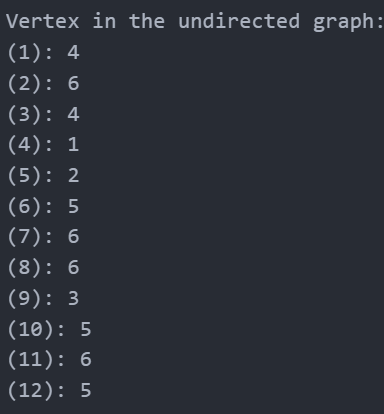
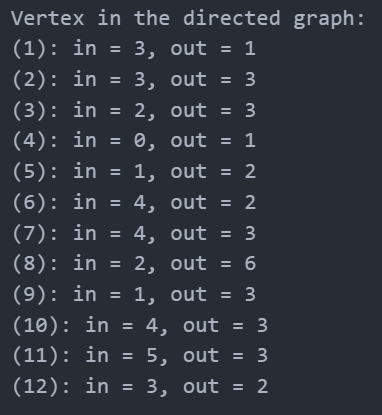
***Результати тестування програми***

1. Матриці суміжності напрямленого та ненапрямленого графів для першого k, їхні графи:

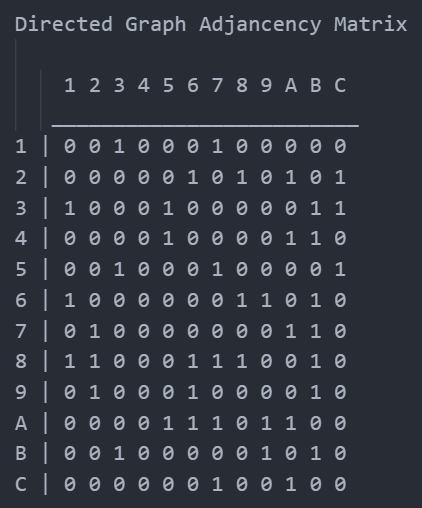
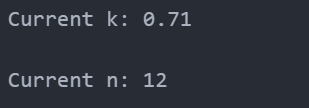


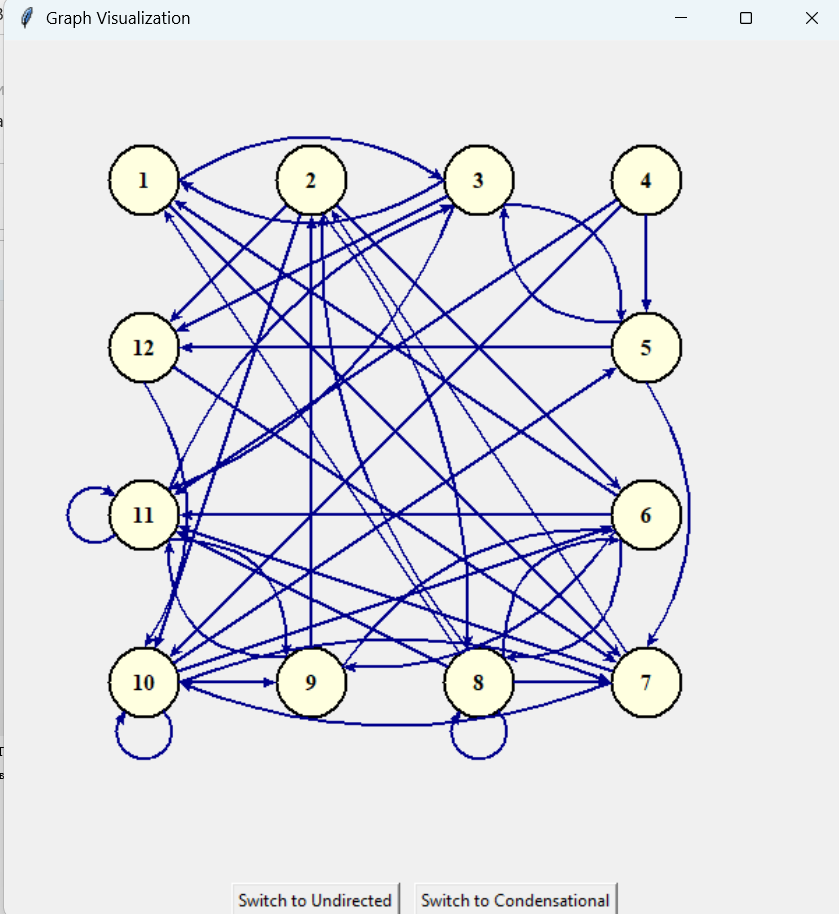
 

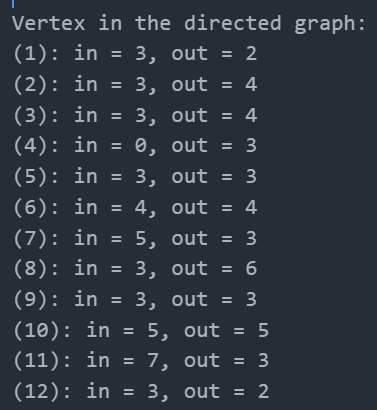
1. Перелік степенів, півстепенів, результат перевірки на однорідність, переліки висячих та ізольованих вершин



1. Матриця суміжності другого орграфа, відображення графа.



4. Переліки півстепенів, шляхів, матриці досяжності та сильної зв’язності, перелік компонентів сильної зв’язності, матриці конденсації та графа конденсації



Paths of length 2 (скопійовано з консолі):

1 -> 3 -> 1

1 -> 7 -> 2

1 -> 3 -> 5

1 -> 7 -> 10

1 -> 3 -> 11

1 -> 7 -> 11

1 -> 3 -> 12

2 -> 6 -> 1

2 -> 8 -> 1

2 -> 8 -> 2

2 -> 10 -> 5

2 -> 8 -> 6

2 -> 10 -> 6

2 -> 8 -> 7

2 -> 10 -> 7

2 -> 12 -> 7

2 -> 6 -> 8

2 -> 8 -> 8

2 -> 6 -> 9

2 -> 10 -> 9

2 -> 10 -> 10

2 -> 12 -> 10

2 -> 6 -> 11

2 -> 8 -> 11

3 -> 1 -> 3

3 -> 5 -> 3

3 -> 11 -> 3

3 -> 1 -> 7

3 -> 5 -> 7

3 -> 12 -> 7

3 -> 11 -> 9

3 -> 12 -> 10

3 -> 11 -> 11

3 -> 5 -> 12

4 -> 5 -> 3

4 -> 11 -> 3

4 -> 10 -> 5

4 -> 10 -> 6

4 -> 5 -> 7

4 -> 10 -> 7

4 -> 10 -> 9

4 -> 11 -> 9

4 -> 10 -> 10

4 -> 11 -> 11

4 -> 5 -> 12

5 -> 3 -> 1

5 -> 7 -> 2

5 -> 3 -> 5

5 -> 12 -> 7

5 -> 7 -> 10

5 -> 12 -> 10

5 -> 3 -> 11

5 -> 7 -> 11

5 -> 3 -> 12

6 -> 8 -> 1

6 -> 8 -> 2

6 -> 9 -> 2

6 -> 1 -> 3

6 -> 11 -> 3

6 -> 8 -> 6

6 -> 9 -> 6

6 -> 1 -> 7

6 -> 8 -> 7

6 -> 8 -> 8

6 -> 11 -> 9

6 -> 8 -> 11

6 -> 9 -> 11

6 -> 11 -> 11

7 -> 11 -> 3

7 -> 10 -> 5

7 -> 2 -> 6

7 -> 10 -> 6

7 -> 10 -> 7

7 -> 2 -> 8

7 -> 10 -> 9

7 -> 11 -> 9

7 -> 2 -> 10

7 -> 10 -> 10

7 -> 11 -> 11

7 -> 2 -> 12

8 -> 6 -> 1

8 -> 8 -> 1

8 -> 7 -> 2

8 -> 8 -> 2

8 -> 1 -> 3

8 -> 11 -> 3

8 -> 2 -> 6

8 -> 8 -> 6

8 -> 1 -> 7

8 -> 8 -> 7

8 -> 2 -> 8

8 -> 6 -> 8

8 -> 8 -> 8

8 -> 6 -> 9

8 -> 11 -> 9

8 -> 2 -> 10

8 -> 7 -> 10

8 -> 6 -> 11

8 -> 7 -> 11

8 -> 8 -> 11

8 -> 11 -> 11

8 -> 2 -> 12

9 -> 6 -> 1

9 -> 11 -> 3

9 -> 2 -> 6

9 -> 2 -> 8

9 -> 6 -> 8

9 -> 6 -> 9

9 -> 11 -> 9

9 -> 2 -> 10

9 -> 6 -> 11

9 -> 11 -> 11

9 -> 2 -> 12

10 -> 6 -> 1

10 -> 7 -> 2

10 -> 9 -> 2

10 -> 5 -> 3

10 -> 10 -> 5

10 -> 9 -> 6

10 -> 10 -> 6

10 -> 5 -> 7

10 -> 10 -> 7

10 -> 6 -> 8

10 -> 6 -> 9

10 -> 10 -> 9

10 -> 7 -> 10

10 -> 10 -> 10

10 -> 6 -> 11

10 -> 7 -> 11

10 -> 9 -> 11

10 -> 5 -> 12

11 -> 3 -> 1

11 -> 9 -> 2

11 -> 11 -> 3

11 -> 3 -> 5

11 -> 9 -> 6

11 -> 11 -> 9

11 -> 3 -> 11

11 -> 9 -> 11

11 -> 11 -> 11

11 -> 3 -> 12

12 -> 7 -> 2

12 -> 10 -> 5

12 -> 10 -> 6

12 -> 10 -> 7

12 -> 10 -> 9

12 -> 7 -> 10

12 -> 10 -> 10

12 -> 7 -> 11

Paths of length 3 (скопійовано з консолі):

1 -> 3 -> 5 -> 3

1 -> 3 -> 5 -> 7

1 -> 3 -> 5 -> 12

1 -> 7 -> 10 -> 5

1 -> 7 -> 10 -> 6

1 -> 7 -> 10 -> 7

1 -> 7 -> 10 -> 9

1 -> 7 -> 10 -> 10

1 -> 3 -> 11 -> 3

1 -> 3 -> 11 -> 9

1 -> 3 -> 11 -> 11

1 -> 7 -> 11 -> 3

1 -> 7 -> 11 -> 9

1 -> 7 -> 11 -> 11

1 -> 3 -> 12 -> 7

1 -> 3 -> 12 -> 10

2 -> 6 -> 1 -> 3

2 -> 6 -> 1 -> 7

2 -> 8 -> 1 -> 3

2 -> 8 -> 1 -> 7

2 -> 8 -> 2 -> 6

2 -> 8 -> 2 -> 8

2 -> 8 -> 2 -> 10

2 -> 8 -> 2 -> 12

2 -> 10 -> 5 -> 3

2 -> 10 -> 5 -> 7

2 -> 10 -> 5 -> 12

2 -> 8 -> 6 -> 1

2 -> 8 -> 6 -> 8

2 -> 8 -> 6 -> 9

2 -> 8 -> 6 -> 11

2 -> 10 -> 6 -> 1

2 -> 10 -> 6 -> 8

2 -> 10 -> 6 -> 9

2 -> 10 -> 6 -> 11

2 -> 8 -> 7 -> 2

2 -> 8 -> 7 -> 10

2 -> 8 -> 7 -> 11

2 -> 10 -> 7 -> 2

2 -> 10 -> 7 -> 10

2 -> 10 -> 7 -> 11

2 -> 12 -> 7 -> 2

2 -> 12 -> 7 -> 10

2 -> 12 -> 7 -> 11

2 -> 6 -> 8 -> 1

2 -> 6 -> 8 -> 2

2 -> 6 -> 8 -> 6

2 -> 6 -> 8 -> 7

2 -> 6 -> 8 -> 8

2 -> 6 -> 8 -> 11

2 -> 8 -> 8 -> 1

2 -> 8 -> 8 -> 2

2 -> 8 -> 8 -> 6

2 -> 8 -> 8 -> 7

2 -> 8 -> 8 -> 8

2 -> 8 -> 8 -> 11

2 -> 6 -> 9 -> 2

2 -> 6 -> 9 -> 6

2 -> 6 -> 9 -> 11

2 -> 10 -> 9 -> 2

2 -> 10 -> 9 -> 6

2 -> 10 -> 9 -> 11

2 -> 10 -> 10 -> 5

2 -> 10 -> 10 -> 6

2 -> 10 -> 10 -> 7

2 -> 10 -> 10 -> 9

2 -> 10 -> 10 -> 10

2 -> 12 -> 10 -> 5

2 -> 12 -> 10 -> 6

2 -> 12 -> 10 -> 7

2 -> 12 -> 10 -> 9

2 -> 12 -> 10 -> 10

2 -> 6 -> 11 -> 3

2 -> 6 -> 11 -> 9

2 -> 6 -> 11 -> 11

2 -> 8 -> 11 -> 3

2 -> 8 -> 11 -> 9

2 -> 8 -> 11 -> 11

3 -> 1 -> 3 -> 1

3 -> 1 -> 3 -> 5

3 -> 1 -> 3 -> 11

3 -> 1 -> 3 -> 12

3 -> 5 -> 3 -> 1

3 -> 5 -> 3 -> 5

3 -> 5 -> 3 -> 11

3 -> 5 -> 3 -> 12

3 -> 11 -> 3 -> 1

3 -> 11 -> 3 -> 5

3 -> 11 -> 3 -> 11

3 -> 11 -> 3 -> 12

3 -> 1 -> 7 -> 2

3 -> 1 -> 7 -> 10

3 -> 1 -> 7 -> 11

3 -> 5 -> 7 -> 2

3 -> 5 -> 7 -> 10

3 -> 5 -> 7 -> 11

3 -> 12 -> 7 -> 2

3 -> 12 -> 7 -> 10

3 -> 12 -> 7 -> 11

3 -> 11 -> 9 -> 2

3 -> 11 -> 9 -> 6

3 -> 11 -> 9 -> 11

3 -> 12 -> 10 -> 5

3 -> 12 -> 10 -> 6

3 -> 12 -> 10 -> 7

3 -> 12 -> 10 -> 9

3 -> 12 -> 10 -> 10

3 -> 11 -> 11 -> 3

3 -> 11 -> 11 -> 9

3 -> 11 -> 11 -> 11

3 -> 5 -> 12 -> 7

3 -> 5 -> 12 -> 10

4 -> 5 -> 3 -> 1

4 -> 5 -> 3 -> 5

4 -> 5 -> 3 -> 11

4 -> 5 -> 3 -> 12

4 -> 11 -> 3 -> 1

4 -> 11 -> 3 -> 5

4 -> 11 -> 3 -> 11

4 -> 11 -> 3 -> 12

4 -> 10 -> 5 -> 3

4 -> 10 -> 5 -> 7

4 -> 10 -> 5 -> 12

4 -> 10 -> 6 -> 1

4 -> 10 -> 6 -> 8

4 -> 10 -> 6 -> 9

4 -> 10 -> 6 -> 11

4 -> 5 -> 7 -> 2

4 -> 5 -> 7 -> 10

4 -> 5 -> 7 -> 11

4 -> 10 -> 7 -> 2

4 -> 10 -> 7 -> 10

4 -> 10 -> 7 -> 11

4 -> 10 -> 9 -> 2

4 -> 10 -> 9 -> 6

4 -> 10 -> 9 -> 11

4 -> 11 -> 9 -> 2

4 -> 11 -> 9 -> 6

4 -> 11 -> 9 -> 11

4 -> 10 -> 10 -> 5

4 -> 10 -> 10 -> 6

4 -> 10 -> 10 -> 7

4 -> 10 -> 10 -> 9

4 -> 10 -> 10 -> 10

4 -> 11 -> 11 -> 3

4 -> 11 -> 11 -> 9

4 -> 11 -> 11 -> 11

4 -> 5 -> 12 -> 7

4 -> 5 -> 12 -> 10

5 -> 7 -> 2 -> 6

5 -> 7 -> 2 -> 8

5 -> 7 -> 2 -> 10

5 -> 7 -> 2 -> 12

5 -> 3 -> 5 -> 3

5 -> 3 -> 5 -> 7

5 -> 3 -> 5 -> 12

5 -> 12 -> 7 -> 2

5 -> 12 -> 7 -> 10

5 -> 12 -> 7 -> 11

5 -> 7 -> 10 -> 5

5 -> 7 -> 10 -> 6

5 -> 7 -> 10 -> 7

5 -> 7 -> 10 -> 9

5 -> 7 -> 10 -> 10

5 -> 12 -> 10 -> 5

5 -> 12 -> 10 -> 6

5 -> 12 -> 10 -> 7

5 -> 12 -> 10 -> 9

5 -> 12 -> 10 -> 10

5 -> 3 -> 11 -> 3

5 -> 3 -> 11 -> 9

5 -> 3 -> 11 -> 11

5 -> 7 -> 11 -> 3

5 -> 7 -> 11 -> 9

5 -> 7 -> 11 -> 11

5 -> 3 -> 12 -> 7

5 -> 3 -> 12 -> 10

6 -> 8 -> 1 -> 3

6 -> 8 -> 1 -> 7

6 -> 8 -> 2 -> 6

6 -> 8 -> 2 -> 8

6 -> 8 -> 2 -> 10

6 -> 8 -> 2 -> 12

6 -> 9 -> 2 -> 6

6 -> 9 -> 2 -> 8

6 -> 9 -> 2 -> 10

6 -> 9 -> 2 -> 12

6 -> 1 -> 3 -> 1

6 -> 1 -> 3 -> 5

6 -> 1 -> 3 -> 11

6 -> 1 -> 3 -> 12

6 -> 11 -> 3 -> 1

6 -> 11 -> 3 -> 5

6 -> 11 -> 3 -> 11

6 -> 11 -> 3 -> 12

6 -> 8 -> 6 -> 1

6 -> 8 -> 6 -> 8

6 -> 8 -> 6 -> 9

6 -> 8 -> 6 -> 11

6 -> 9 -> 6 -> 1

6 -> 9 -> 6 -> 8

6 -> 9 -> 6 -> 9

6 -> 9 -> 6 -> 11

6 -> 1 -> 7 -> 2

6 -> 1 -> 7 -> 10

6 -> 1 -> 7 -> 11

6 -> 8 -> 7 -> 2

6 -> 8 -> 7 -> 10

6 -> 8 -> 7 -> 11

6 -> 8 -> 8 -> 1

6 -> 8 -> 8 -> 2

6 -> 8 -> 8 -> 6

6 -> 8 -> 8 -> 7

6 -> 8 -> 8 -> 8

6 -> 8 -> 8 -> 11

6 -> 11 -> 9 -> 2

6 -> 11 -> 9 -> 6

6 -> 11 -> 9 -> 11

6 -> 8 -> 11 -> 3

6 -> 8 -> 11 -> 9

6 -> 8 -> 11 -> 11

6 -> 9 -> 11 -> 3

6 -> 9 -> 11 -> 9

6 -> 9 -> 11 -> 11

6 -> 11 -> 11 -> 3

6 -> 11 -> 11 -> 9

6 -> 11 -> 11 -> 11

7 -> 11 -> 3 -> 1

7 -> 11 -> 3 -> 5

7 -> 11 -> 3 -> 11

7 -> 11 -> 3 -> 12

7 -> 10 -> 5 -> 3

7 -> 10 -> 5 -> 7

7 -> 10 -> 5 -> 12

7 -> 2 -> 6 -> 1

7 -> 2 -> 6 -> 8

7 -> 2 -> 6 -> 9

7 -> 2 -> 6 -> 11

7 -> 10 -> 6 -> 1

7 -> 10 -> 6 -> 8

7 -> 10 -> 6 -> 9

7 -> 10 -> 6 -> 11

7 -> 10 -> 7 -> 2

7 -> 10 -> 7 -> 10

7 -> 10 -> 7 -> 11

7 -> 2 -> 8 -> 1

7 -> 2 -> 8 -> 2

7 -> 2 -> 8 -> 6

7 -> 2 -> 8 -> 7

7 -> 2 -> 8 -> 8

7 -> 2 -> 8 -> 11

7 -> 10 -> 9 -> 2

7 -> 10 -> 9 -> 6

7 -> 10 -> 9 -> 11

7 -> 11 -> 9 -> 2

7 -> 11 -> 9 -> 6

7 -> 11 -> 9 -> 11

7 -> 2 -> 10 -> 5

7 -> 2 -> 10 -> 6

7 -> 2 -> 10 -> 7

7 -> 2 -> 10 -> 9

7 -> 2 -> 10 -> 10

7 -> 10 -> 10 -> 5

7 -> 10 -> 10 -> 6

7 -> 10 -> 10 -> 7

7 -> 10 -> 10 -> 9

7 -> 10 -> 10 -> 10

7 -> 11 -> 11 -> 3

7 -> 11 -> 11 -> 9

7 -> 11 -> 11 -> 11

7 -> 2 -> 12 -> 7

7 -> 2 -> 12 -> 10

8 -> 6 -> 1 -> 3

8 -> 6 -> 1 -> 7

8 -> 8 -> 1 -> 3

8 -> 8 -> 1 -> 7

8 -> 7 -> 2 -> 6

8 -> 7 -> 2 -> 8

8 -> 7 -> 2 -> 10

8 -> 7 -> 2 -> 12

8 -> 8 -> 2 -> 6

8 -> 8 -> 2 -> 8

8 -> 8 -> 2 -> 10

8 -> 8 -> 2 -> 12

8 -> 1 -> 3 -> 1

8 -> 1 -> 3 -> 5

8 -> 1 -> 3 -> 11

8 -> 1 -> 3 -> 12

8 -> 11 -> 3 -> 1

8 -> 11 -> 3 -> 5

8 -> 11 -> 3 -> 11

8 -> 11 -> 3 -> 12

8 -> 2 -> 6 -> 1

8 -> 2 -> 6 -> 8

8 -> 2 -> 6 -> 9

8 -> 2 -> 6 -> 11

8 -> 8 -> 6 -> 1

8 -> 8 -> 6 -> 8

8 -> 8 -> 6 -> 9

8 -> 8 -> 6 -> 11

8 -> 1 -> 7 -> 2

8 -> 1 -> 7 -> 10

8 -> 1 -> 7 -> 11

8 -> 8 -> 7 -> 2

8 -> 8 -> 7 -> 10

8 -> 8 -> 7 -> 11

8 -> 2 -> 8 -> 1

8 -> 2 -> 8 -> 2

8 -> 2 -> 8 -> 6

8 -> 2 -> 8 -> 7

8 -> 2 -> 8 -> 8

8 -> 2 -> 8 -> 11

8 -> 6 -> 8 -> 1

8 -> 6 -> 8 -> 2

8 -> 6 -> 8 -> 6

8 -> 6 -> 8 -> 7

8 -> 6 -> 8 -> 8

8 -> 6 -> 8 -> 11

8 -> 8 -> 8 -> 1

8 -> 8 -> 8 -> 2

8 -> 8 -> 8 -> 6

8 -> 8 -> 8 -> 7

8 -> 8 -> 8 -> 8

8 -> 8 -> 8 -> 11

8 -> 6 -> 9 -> 2

8 -> 6 -> 9 -> 6

8 -> 6 -> 9 -> 11

8 -> 11 -> 9 -> 2

8 -> 11 -> 9 -> 6

8 -> 11 -> 9 -> 11

8 -> 2 -> 10 -> 5

8 -> 2 -> 10 -> 6

8 -> 2 -> 10 -> 7

8 -> 2 -> 10 -> 9

8 -> 2 -> 10 -> 10

8 -> 7 -> 10 -> 5

8 -> 7 -> 10 -> 6

8 -> 7 -> 10 -> 7

8 -> 7 -> 10 -> 9

8 -> 7 -> 10 -> 10

8 -> 6 -> 11 -> 3

8 -> 6 -> 11 -> 9

8 -> 6 -> 11 -> 11

8 -> 7 -> 11 -> 3

8 -> 7 -> 11 -> 9

8 -> 7 -> 11 -> 11

8 -> 8 -> 11 -> 3

8 -> 8 -> 11 -> 9

8 -> 8 -> 11 -> 11

8 -> 11 -> 11 -> 3

8 -> 11 -> 11 -> 9

8 -> 11 -> 11 -> 11

8 -> 2 -> 12 -> 7

8 -> 2 -> 12 -> 10

9 -> 6 -> 1 -> 3

9 -> 6 -> 1 -> 7

9 -> 11 -> 3 -> 1

9 -> 11 -> 3 -> 5

9 -> 11 -> 3 -> 11

9 -> 11 -> 3 -> 12

9 -> 2 -> 6 -> 1

9 -> 2 -> 6 -> 8

9 -> 2 -> 6 -> 9

9 -> 2 -> 6 -> 11

9 -> 2 -> 8 -> 1

9 -> 2 -> 8 -> 2

9 -> 2 -> 8 -> 6

9 -> 2 -> 8 -> 7

9 -> 2 -> 8 -> 8

9 -> 2 -> 8 -> 11

9 -> 6 -> 8 -> 1

9 -> 6 -> 8 -> 2

9 -> 6 -> 8 -> 6

9 -> 6 -> 8 -> 7

9 -> 6 -> 8 -> 8

9 -> 6 -> 8 -> 11

9 -> 6 -> 9 -> 2

9 -> 6 -> 9 -> 6

9 -> 6 -> 9 -> 11

9 -> 11 -> 9 -> 2

9 -> 11 -> 9 -> 6

9 -> 11 -> 9 -> 11

9 -> 2 -> 10 -> 5

9 -> 2 -> 10 -> 6

9 -> 2 -> 10 -> 7

9 -> 2 -> 10 -> 9

9 -> 2 -> 10 -> 10

9 -> 6 -> 11 -> 3

9 -> 6 -> 11 -> 9

9 -> 6 -> 11 -> 11

9 -> 11 -> 11 -> 3

9 -> 11 -> 11 -> 9

9 -> 11 -> 11 -> 11

9 -> 2 -> 12 -> 7

9 -> 2 -> 12 -> 10

10 -> 6 -> 1 -> 3

10 -> 6 -> 1 -> 7

10 -> 7 -> 2 -> 6

10 -> 7 -> 2 -> 8

10 -> 7 -> 2 -> 10

10 -> 7 -> 2 -> 12

10 -> 9 -> 2 -> 6

10 -> 9 -> 2 -> 8

10 -> 9 -> 2 -> 10

10 -> 9 -> 2 -> 12

10 -> 5 -> 3 -> 1

10 -> 5 -> 3 -> 5

10 -> 5 -> 3 -> 11

10 -> 5 -> 3 -> 12

10 -> 10 -> 5 -> 3

10 -> 10 -> 5 -> 7

10 -> 10 -> 5 -> 12

10 -> 9 -> 6 -> 1

10 -> 9 -> 6 -> 8

10 -> 9 -> 6 -> 9

10 -> 9 -> 6 -> 11

10 -> 10 -> 6 -> 1

10 -> 10 -> 6 -> 8

10 -> 10 -> 6 -> 9

10 -> 10 -> 6 -> 11

10 -> 5 -> 7 -> 2

10 -> 5 -> 7 -> 10

10 -> 5 -> 7 -> 11

10 -> 10 -> 7 -> 2

10 -> 10 -> 7 -> 10

10 -> 10 -> 7 -> 11

10 -> 6 -> 8 -> 1

10 -> 6 -> 8 -> 2

10 -> 6 -> 8 -> 6

10 -> 6 -> 8 -> 7

10 -> 6 -> 8 -> 8

10 -> 6 -> 8 -> 11

10 -> 6 -> 9 -> 2

10 -> 6 -> 9 -> 6

10 -> 6 -> 9 -> 11

10 -> 10 -> 9 -> 2

10 -> 10 -> 9 -> 6

10 -> 10 -> 9 -> 11

10 -> 7 -> 10 -> 5

10 -> 7 -> 10 -> 6

10 -> 7 -> 10 -> 7

10 -> 7 -> 10 -> 9

10 -> 7 -> 10 -> 10

10 -> 10 -> 10 -> 5

10 -> 10 -> 10 -> 6

10 -> 10 -> 10 -> 7

10 -> 10 -> 10 -> 9

10 -> 10 -> 10 -> 10

10 -> 6 -> 11 -> 3

10 -> 6 -> 11 -> 9

10 -> 6 -> 11 -> 11

10 -> 7 -> 11 -> 3

10 -> 7 -> 11 -> 9

10 -> 7 -> 11 -> 11

10 -> 9 -> 11 -> 3

10 -> 9 -> 11 -> 9

10 -> 9 -> 11 -> 11

10 -> 5 -> 12 -> 7

10 -> 5 -> 12 -> 10

11 -> 3 -> 1 -> 3

11 -> 3 -> 1 -> 7

11 -> 9 -> 2 -> 6

11 -> 9 -> 2 -> 8

11 -> 9 -> 2 -> 10

11 -> 9 -> 2 -> 12

11 -> 11 -> 3 -> 1

11 -> 11 -> 3 -> 5

11 -> 11 -> 3 -> 11

11 -> 11 -> 3 -> 12

11 -> 3 -> 5 -> 3

11 -> 3 -> 5 -> 7

11 -> 3 -> 5 -> 12

11 -> 9 -> 6 -> 1

11 -> 9 -> 6 -> 8

11 -> 9 -> 6 -> 9

11 -> 9 -> 6 -> 11

11 -> 11 -> 9 -> 2

11 -> 11 -> 9 -> 6

11 -> 11 -> 9 -> 11

11 -> 3 -> 11 -> 3

11 -> 3 -> 11 -> 9

11 -> 3 -> 11 -> 11

11 -> 9 -> 11 -> 3

11 -> 9 -> 11 -> 9

11 -> 9 -> 11 -> 11

11 -> 11 -> 11 -> 3

11 -> 11 -> 11 -> 9

11 -> 11 -> 11 -> 11

11 -> 3 -> 12 -> 7

11 -> 3 -> 12 -> 10

12 -> 7 -> 2 -> 6

12 -> 7 -> 2 -> 8

12 -> 7 -> 2 -> 10

12 -> 7 -> 2 -> 12

12 -> 10 -> 5 -> 3

12 -> 10 -> 5 -> 7

12 -> 10 -> 5 -> 12

12 -> 10 -> 6 -> 1

12 -> 10 -> 6 -> 8

12 -> 10 -> 6 -> 9

12 -> 10 -> 6 -> 11

12 -> 10 -> 7 -> 2

12 -> 10 -> 7 -> 10

12 -> 10 -> 7 -> 11

12 -> 10 -> 9 -> 2

12 -> 10 -> 9 -> 6

12 -> 10 -> 9 -> 11

12 -> 7 -> 10 -> 5

12 -> 7 -> 10 -> 6

12 -> 7 -> 10 -> 7

12 -> 7 -> 10 -> 9

12 -> 7 -> 10 -> 10

12 -> 10 -> 10 -> 5

12 -> 10 -> 10 -> 6

12 -> 10 -> 10 -> 7

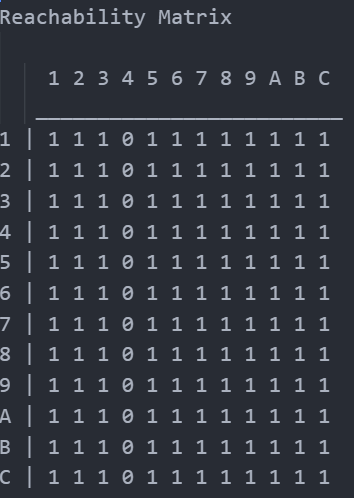
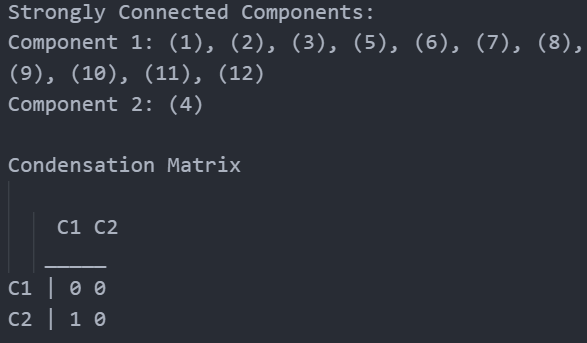
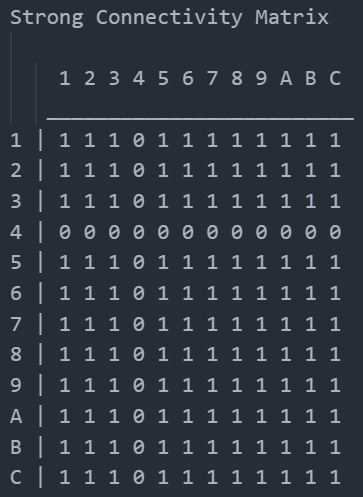
12 -> 10 -> 10 -> 9

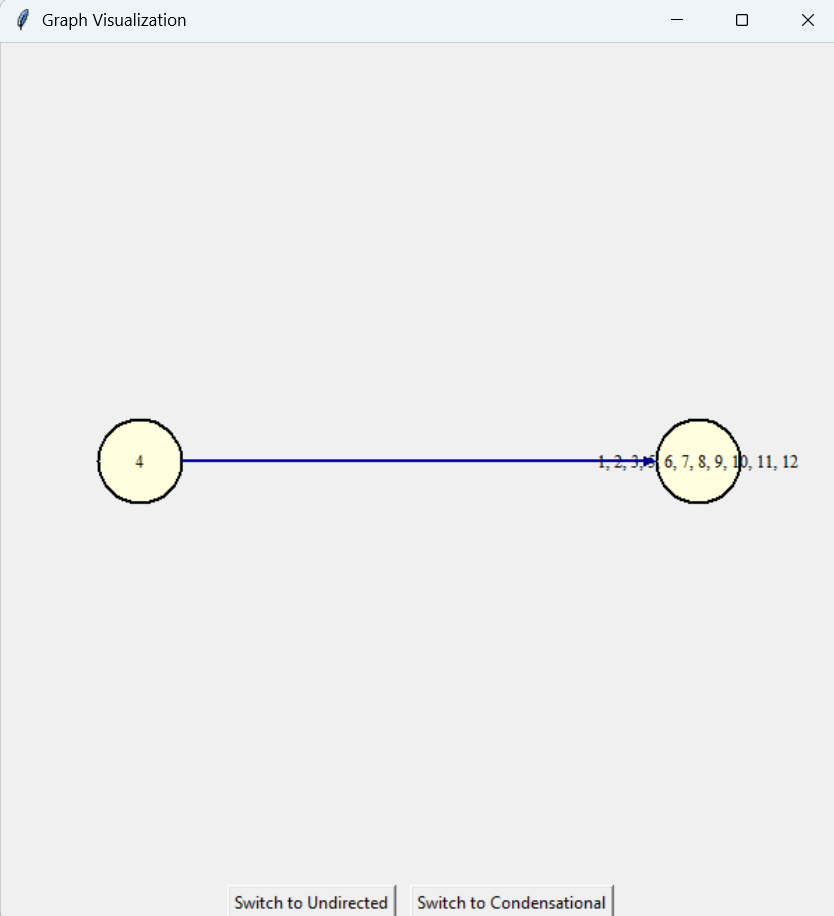
12 -> 10 -> 10 -> 10

12 -> 7 -> 11 -> 3

12 -> 7 -> 11 -> 9

12 -> 7 -> 11 -> 11



***Висновки***

У результаті виконання лабораторної роботи я поглибив розуміння структури та властивостей напрямлених і ненапрямлених графів. Повторив, як формувати матриці суміжності, обчислювати степені та півстепені вершин, перевіряти граф на однорідність, знаходити ізольовані та висячі вершини. Також засвоїв способи побудови матриці досяжності, визначення компонент сильної зв’язності та побудови графа конденсації. Додатково, я покращив навички роботи з бібліотекою tkinter — повторив знання зі створення графічного вікна, організовував перемикання між візуалізаціями графів і зручно, у залежності від коефіцієнта, вивів результати трьох графів: напрямленого, ненапрямленого та графа конденсації для другого коефіцієнта. Це дозволило поєднати алгоритмічну частину з візуальною, що сприяло моєму кращому розумінню логіки побудови графів та їх властивостей.