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:
студент групи ІМ-43
Олексійчук Станіслав Юрійович
номер у списку групи: 22

Перевірив:
Сергієнко А. М.

Київ 2025

Постановка задачі

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

   Відмінність: коефіцієнт $k = 1.0 - n_3 \cdot 0.01 - n_4 \cdot 0.01 - 0.3$;

   Отже, матриця суміжності A_{dir} напрямленого графа за варіантом фор- мується 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
   3) числами в діапазоні [0, 2.0);
   4) обчислюється коефіцієнт $k = 1.0 - n_3 \cdot 0.01 - n_4 \cdot 0.01 - 0.3$, кожен елемент матриці множиться на коефіцієнт k;
   5) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

2. Обчислити:
   1) степені вершин напрямленого і ненапрямленого графів;
   2) напівстепені виходу та заходу напрямленого графа;
   3) чи є граф однорідним (регулярним), і якщо так, вказати степінь однорідності графа;
   4) перелік висячих та ізольованих вершин.

   Результати вивести у графічне вікно, консоль або файл.

3. Змінити матрицю A_{dir}, коефіцієнт $k = 1.0 - n_3 \cdot 0.005 - n_4 \cdot 0.005 - 0.27$.

4. Для нового орграфа обчислити:
   1) півстепені вершин;
   2) всі шляхи довжини 2 і 3;
   3) матрицю досяжності;
   4) матрицю сильної зв’язності;
   5) перелік компонент сильної зв’язності;

6) граф конденсації.

Результати вивести у графічне вікно, в консоль або файл.

Шляхи довжиною 2 і 3 слід шукати за матрицями A^2 і A^3, відповідно. Як результат вивести перелік шляхів, включно з усіма проміжними вершинами, через які проходить шлях.

Матрицю досяжності та компоненти сильної зв'язності слід шукати за допомогою операції транзитивного замикання. У переліку компонент слід вказати, які вершини належать до кожної компоненти.

Граф конденсації вивести у графічне вікно.

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
```

2) positions.py – створення масиву позицій вершин (4x4)

```
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
```

3) 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
```

4) 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"
        )
```

5) 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()
```

6) 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:

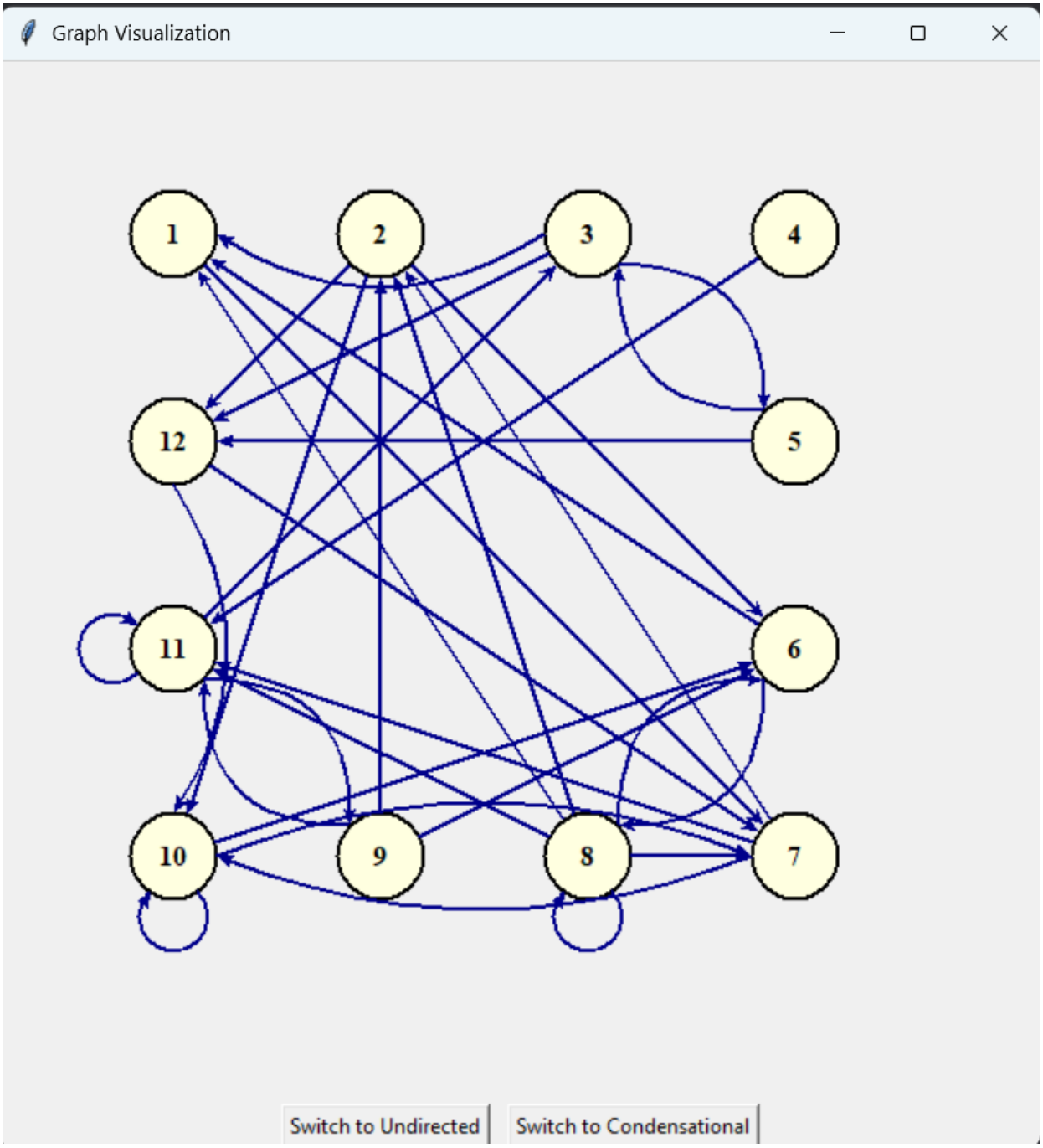
```
Current k: 0.6599999999999999

Current n: 12
```

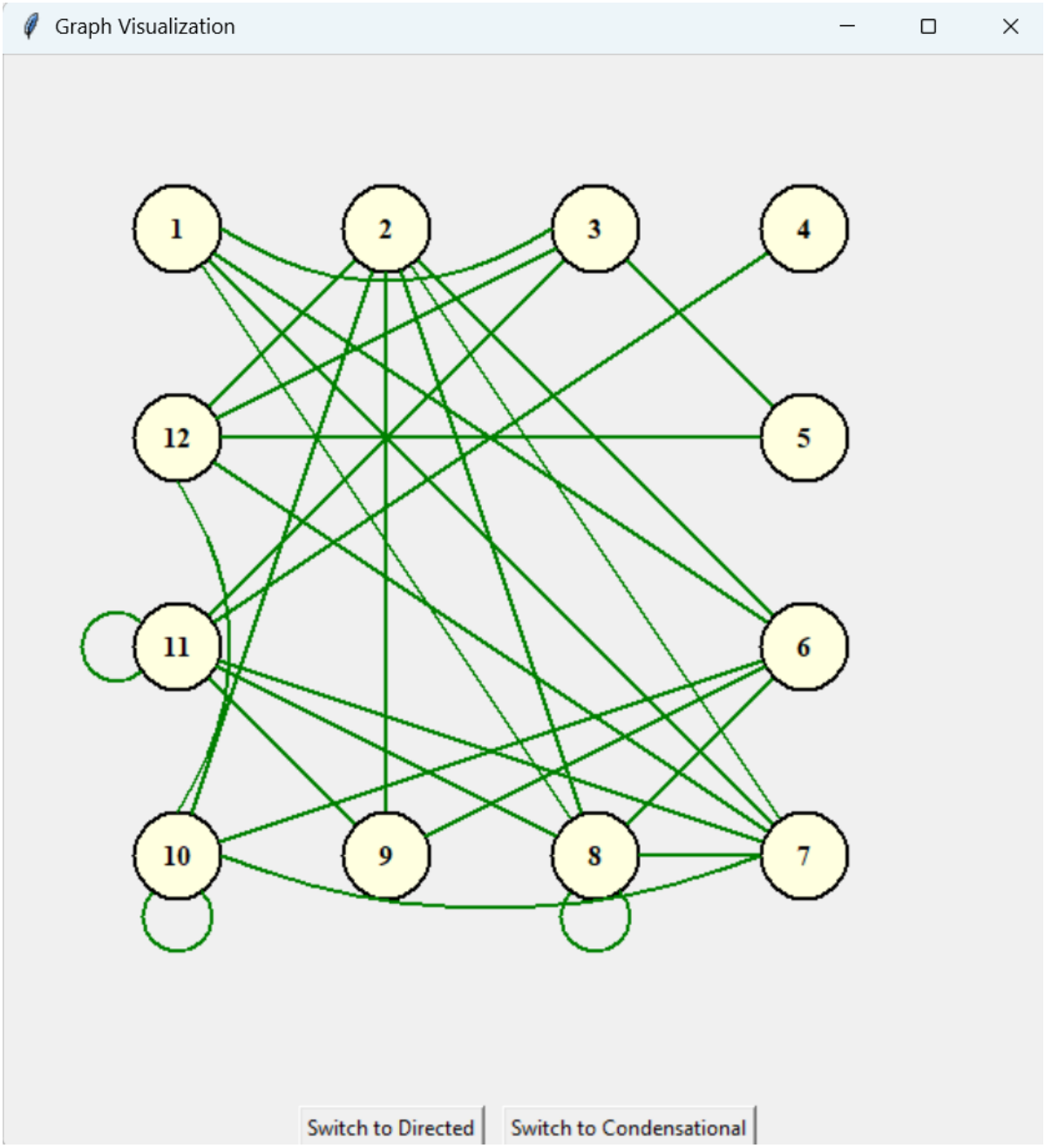
Directed Graph Adjancency Matrix

	1	2	3	4	5	6	7	8	9	A	B	C
1	0	0	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	1	0	1
3	1	0	0	0	1	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	0	0	0	1	0
5	0	0	1	0	0	0	0	0	0	0	0	1
6	1	0	0	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0	1	1	0
8	1	1	0	0	0	1	1	1	0	0	1	0
9	0	1	0	0	0	1	0	0	0	0	1	0
A	0	0	0	0	0	1	1	0	0	1	0	0
B	0	0	1	0	0	0	0	0	1	0	1	0
C	0	0	0	0	0	0	1	0	0	1	0	0

Undirected Graph Adjancency Matrix

	1	2	3	4	5	6	7	8	9	A	B	C
1	0	0	1	0	0	1	1	1	0	0	0	0
2	0	0	0	0	0	1	1	1	1	1	0	1
3	1	0	0	0	1	0	0	0	0	0	1	1
4	0	0	0	0	0	0	0	0	0	0	1	0
5	0	0	1	0	0	0	0	0	0	0	0	1
6	1	1	0	0	0	0	0	1	1	1	0	0
7	1	1	0	0	0	0	0	1	0	1	1	1
8	1	1	0	0	0	1	1	1	0	0	1	0
9	0	1	0	0	0	1	0	0	0	0	1	0
A	0	1	0	0	0	1	1	0	0	1	0	1
B	0	0	1	1	0	0	1	1	1	0	1	0
C	0	1	1	0	1	0	1	0	0	1	0	0


Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Switch to Undirected
Switch to Condensational


Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Switch to Directed
Switch to Condensational

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

```
Vertex in the directed graph:
(1): in = 3, out = 1
(2): in = 3, out = 3
(3): in = 2, out = 3
(4): in = 0, out = 1
(5): in = 1, out = 2
(6): in = 4, out = 2
(7): in = 4, out = 3
(8): in = 2, out = 6
(9): in = 1, out = 3
(10): in = 4, out = 3
(11): in = 5, out = 3
(12): in = 3, out = 2
```

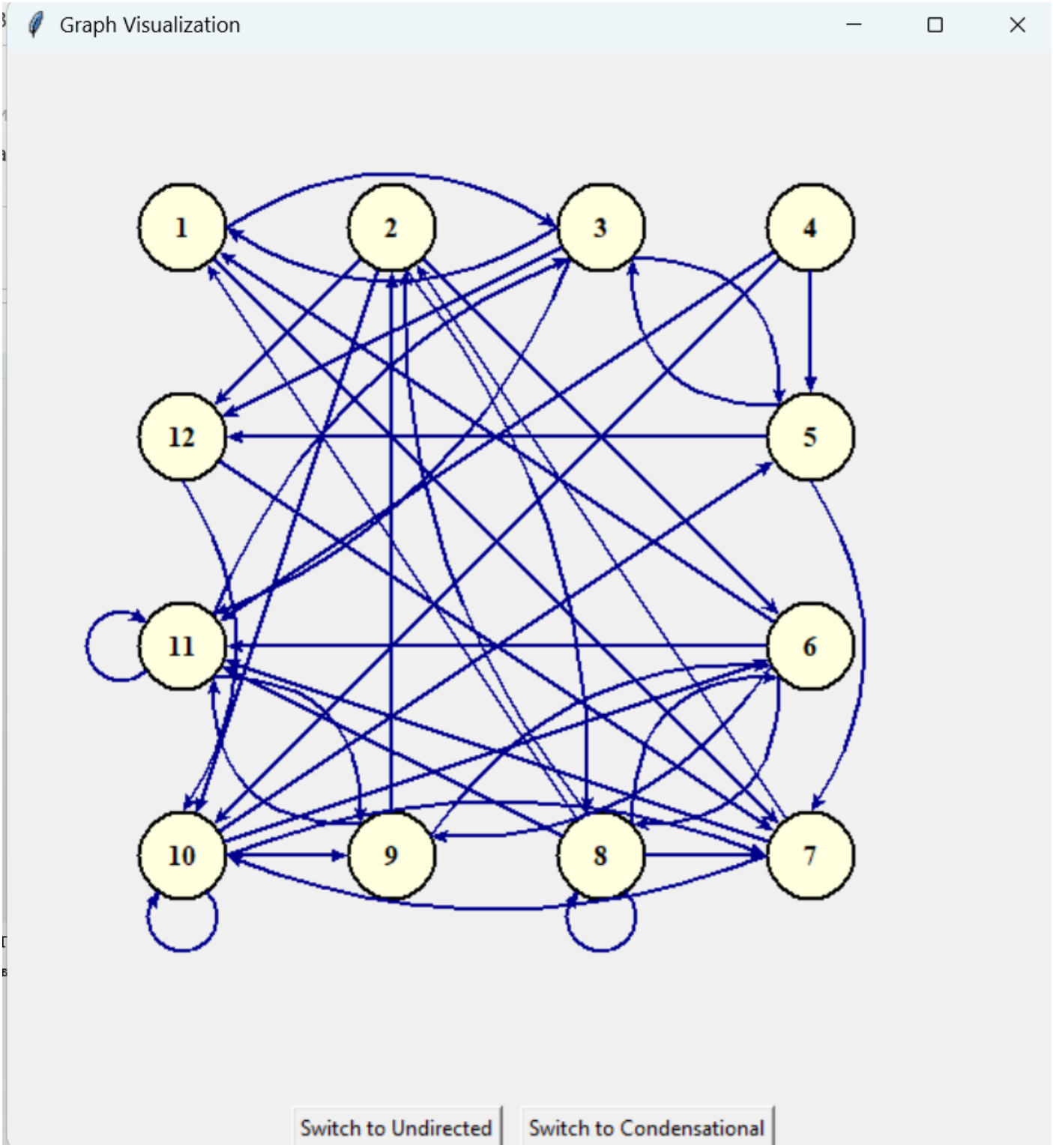
```
Vertex in the undirected graph:
(1): 4
(2): 6
(3): 4
(4): 1
(5): 2
(6): 5
(7): 6
(8): 6
(9): 3
(10): 5
(11): 6
(12): 5
```

```
Isolated vertices
- None found

Hanging vertices
(4)
```

3. Матриця суміжності другого орграфа, відображення графа.

```
Current k: 0.71

Current n: 12
```

```
Directed Graph Adjancency Matrix

   | 1 2 3 4 5 6 7 8 9 A B C
   |_______________________
1  | 0 0 1 0 0 0 1 0 0 0 0 0
2  | 0 0 0 0 0 1 0 1 0 1 0 1
3  | 1 0 0 0 1 0 0 0 0 0 1 1
4  | 0 0 0 0 1 0 0 0 0 1 1 0
5  | 0 0 1 0 0 0 1 0 0 0 0 1
6  | 1 0 0 0 0 0 0 1 1 0 1 0
7  | 0 1 0 0 0 0 0 0 0 1 1 0
8  | 1 1 0 0 0 1 1 1 0 0 1 0
9  | 0 1 0 0 0 1 0 0 0 0 1 0
A  | 0 0 0 0 1 1 1 0 1 1 0 0
B  | 0 0 1 0 0 0 0 0 1 0 1 0
C  | 0 0 0 0 0 0 1 0 0 1 0 0
```



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

```
Vertex in the directed graph:
(1): in = 3, out = 2
(2): in = 3, out = 4
(3): in = 3, out = 4
(4): in = 0, out = 3
(5): in = 3, out = 3
(6): in = 4, out = 4
(7): in = 5, out = 3
(8): in = 3, out = 6
(9): in = 3, out = 3
(10): in = 5, out = 5
(11): in = 7, out = 3
(12): in = 3, out = 2
```

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 -> 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

Reachability Matrix

	1	2	3	4	5	6	7	8	9	A	B	C
1	1	1	1	0	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1	1	1
4	1	1	1	0	1	1	1	1	1	1	1	1
5	1	1	1	0	1	1	1	1	1	1	1	1
6	1	1	1	0	1	1	1	1	1	1	1	1
7	1	1	1	0	1	1	1	1	1	1	1	1
8	1	1	1	0	1	1	1	1	1	1	1	1
9	1	1	1	0	1	1	1	1	1	1	1	1
A	1	1	1	0	1	1	1	1	1	1	1	1
B	1	1	1	0	1	1	1	1	1	1	1	1
C	1	1	1	0	1	1	1	1	1	1	1	1

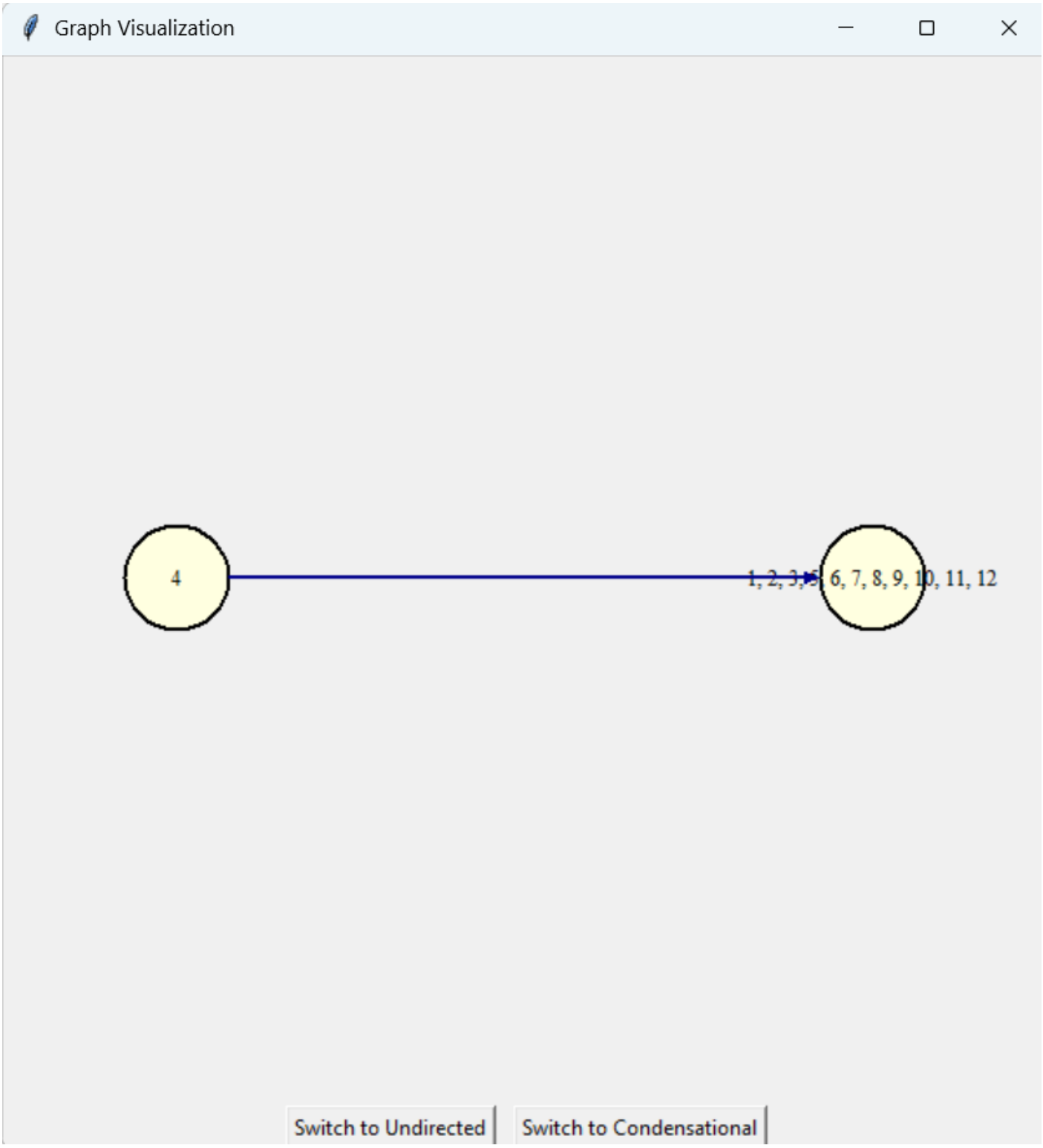
Strong Connectivity Matrix

	1	2	3	4	5	6	7	8	9	A	B	C
1	1	1	1	0	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	0	1	1	1	1	1	1	1	1
6	1	1	1	0	1	1	1	1	1	1	1	1
7	1	1	1	0	1	1	1	1	1	1	1	1
8	1	1	1	0	1	1	1	1	1	1	1	1
9	1	1	1	0	1	1	1	1	1	1	1	1
A	1	1	1	0	1	1	1	1	1	1	1	1
B	1	1	1	0	1	1	1	1	1	1	1	1
C	1	1	1	0	1	1	1	1	1	1	1	1

Strongly Connected Components:
Component 1: (1), (2), (3), (5), (6), (7), (8), (9), (10), (11), (12)
Component 2: (4)

Condensation Matrix

	C1	C2
C1	0	0
C2	1	0


Graph Visualization
4
1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12
Switch to Undirected
Switch to Condensational

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.