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 №1

з дисципліни
«Алгоритми і структури даних»

Виконала:
студентка групи ІМ - 43
Хоанг Чан Кам Лі
номер у списку групи: 29

Перевірив:
Сергієнко А. М.

Київ 2025

Завдання

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

Відмінність: коефіцієнт k = 1.0 - n3 * 0.01 - n4 * 0.01 - 0.3.

2. Обчислити:

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

Результати вивести у графічне вікно, консоль або файл.

3. ***Змінити матрицю*** A(dir), коефіцієнт k = 1.0 - n3*0.005 - n4*0.005 - 0.27.

4. Для нового орграфа обчислити:

- півстепені вершин;
- всі шляхи довжини 2 і 3;
- матрицю досяжності;
- матрицю сильної зв'язності;
- перелік компонент сильної зв'язності;
- граф конденсації.

Результати вивести у графічне вікно, в консоль або файл.

Варіант 29:
Номер варіанту: 4329

Кількість вершин n = 10 + 2 = 12

Розміщення вершин: квадратом (прямокутником) з вершиною в центрі, бо n4 = 9.

Текст програми

graph_generation_k1.py

```
import numpy as np


def generate_matrices(SEED=4329, NUM_NODES=12):

    np.random.seed(SEED)

    random_matrix = 2.0 * np.random.rand(NUM_NODES,
NUM_NODES)
```

```
    coeff_k = 1.0 - int(str(SEED)[2]) * 0.01 -
int(str(SEED)[3]) * 0.01 - 0.3
    adjacency_matrix = (random_matrix * coeff_k >=
1.0).astype(int)

    adjacency_matrix_undir = adjacency_matrix +
adjacency_matrix.T
    adjacency_matrix_undir[adjacency_matrix_undir > 1] =
1

    return adjacency_matrix, adjacency_matrix_undir

if __name__ == "__main__":
    adj_matrix_dir, adj_matrix_undir =
generate_matrices()
    print("Directed Adjacency Matrix:\n",
adj_matrix_dir)
    print("\nUndirected Adjacency Matrix:\n",
adj_matrix_undir)
```

graph_generation_k2.py

```
import numpy as np

def generate_matrices(SEED=4329, NUM_NODES=12):
    np.random.seed(SEED)
    random_matrix = 2.0 * np.random.rand(NUM_NODES,
NUM_NODES)
```

```
    coeff_k = 1.0 - int(str(SEED)[2]) * 0.005 -
int(str(SEED)[3]) * 0.005 - 0.27

    adjacency_matrix = (random_matrix * coeff_k >=
1.0).astype(int)


    return adjacency_matrix


if __name__ == "__main__":

    adj_matrix_dir = generate_matrices()

    print("Directed Adjacency Matrix:\n",
adj_matrix_dir)
```

draw.py

```
import numpy as np

import matplotlib.pyplot as plt

NUM_NODES = 12


def draw_graph(adj_matrix, directed):


# Parameters for visualization

    NODE_RADIUS = 0.05

    LINE_THICKNESS = 1.5


    NUM_NODES = adj_matrix.shape[0]


    center_node = np.array([[1, 1]])


    num_boundary_nodes = NUM_NODES - 1
```

```
    nodes_per_side = num_boundary_nodes // 4

    remainder_nodes = num_boundary_nodes % 4

    boundary_nodes = []

    sides = [0, 1, 2, 3]
    side_node_counts = [nodes_per_side] * 4

    for i in range(remainder_nodes):
      side_node_counts[i] += 1

    for side, count in zip(sides, side_node_counts):
      if count > 0:

        spacing = np.linspace(0.3, 1.7, count)

        if side == 0:
          boundary_nodes.extend([[x, 0] for x in
spacing])
        elif side == 1:
          boundary_nodes.extend([[2, y] for y in
spacing])
        elif side == 2:
```

```
          boundary_nodes.extend([[x, 2] for x in
spacing])
        elif side == 3:
          boundary_nodes.extend([[0, y] for y in
spacing])

    all_nodes = np.array(boundary_nodes)

    if NUM_NODES > 0:
      all_nodes = np.vstack((center_node, all_nodes))


    plt.figure(figsize=(8, 8))

# Draw edges first
    focus_point = np.array([1, 1])

    if directed:
      title = 'Directed Graph Representation'
      for i in range(NUM_NODES):
        for j in range(NUM_NODES):
          if adj_matrix[i, j] == 1:
            node1 = all_nodes[i]
            node2 = all_nodes[j]

            if i == j:
```

```
                x, y = node1
                r = NODE_RADIUS * 1.5
                theta = np.linspace(-np.pi/2, 3*np.pi/2,
100)
                circle_x = x + r * np.cos(theta)
                circle_y = y + r + r * np.sin(theta)
                plt.plot(circle_x, circle_y, 'r-',
linewidth=LINE_THICKNESS)

                arrow_point_idx = np.argmin(np.abs(theta
+ np.pi/2))
                arrow_point =
np.array([circle_x[arrow_point_idx],
circle_y[arrow_point_idx]])
                direction = np.array([
-np.sin(theta[arrow_point_idx]),
np.cos(theta[arrow_point_idx])])
                direction = direction /
np.linalg.norm(direction)

                arrow_base = arrow_point - direction *
NODE_RADIUS * 0.5
                plt.arrow(arrow_base[0], arrow_base[1],
direction[0] * 0.01, direction[1] * 0.01,
                          head_width=0.05,
head_length=0.08, fc='r', ec='r', linewidth=0,
length_includes_head=True)
            else:
              # Draw a quadratic Bezier curve with arrow
```

```
                midpoint = (node1 + node2) / 2
                control_point = midpoint * 0.3 +
focus_point * 0.7
                t = np.linspace(0, 1, 100)
                # Bezier curve formula: B(t) = (1-t)^2 *
P0 + 2*(1-t)*t * P1 + t**2 * P2
                curve_x = (1-t)**2 * node1[0] + 2*(1-t)*t
* control_point[0] + t**2 * node2[0]
                curve_y = (1-t)**2 * node1[1] + 2*(1-t)*t
* control_point[1] + t**2 * node2[1]

                plt.plot(curve_x, curve_y, 'r-',
linewidth=LINE_THICKNESS)

                end_point_idx = -2
                arrow_point =
np.array([curve_x[end_point_idx],
curve_y[end_point_idx]])
                direction = np.array([curve_x[-1] -
curve_x[end_point_idx], curve_y[-1] -
curve_y[end_point_idx]])
                direction = direction /
np.linalg.norm(direction)

                arrow_base = np.array([curve_x[-1],
curve_y[-1]]) - direction * NODE_RADIUS * 1.4

                plt.arrow(arrow_base[0], arrow_base[1],
direction[0] * 0.01, direction[1] * 0.01,
```

```
                        head_width=0.05,
head_length=0.08, fc='r', ec='r', linewidth=0,
length_includes_head=True)

    else: # Undirected graph
      title = 'Undirected Graph Representation'
      adj_matrix_undir = adj_matrix + adj_matrix.T
      adj_matrix_undir[adj_matrix_undir > 1] = 1

      for i in range(NUM_NODES):
        for j in range(i, NUM_NODES):
          if adj_matrix_undir[i, j] == 1:
            node1 = all_nodes[i]
            node2 = all_nodes[j]

            if i == j:
                x, y = node1
                r = NODE_RADIUS * 1.5

                theta = np.linspace(-np.pi/2, 3*np.pi/2,
100)
                circle_x = x + r * np.cos(theta)
                circle_y = y + r + r * np.sin(theta)
                plt.plot(circle_x, circle_y, 'r-',
linewidth=LINE_THICKNESS)
            else:
              midpoint = (node1 + node2) / 2
```

```
              control_point = midpoint * 0.3 +
focus_point * 0.7
              t = np.linspace(0, 1, 100)
              # Bezier curve formula: B(t) = (1-t)^2 *
P0 + 2*(1-t)*t * P1 + t**2 * P2
              curve_x = (1-t)**2 * node1[0] + 2*(1-t)*t
* control_point[0] + t**2 * node2[0]
              curve_y = (1-t)**2 * node1[1] + 2*(1-t)*t
* control_point[1] + t**2 * node2[1]
              plt.plot(curve_x, curve_y, 'r-',
linewidth=LINE_THICKNESS)

    for i, (x, y) in enumerate(all_nodes):
        circle = plt.Circle((x, y), NODE_RADIUS,
color='skyblue', zorder=5)
        plt.gca().add_patch(circle)
        plt.text(x, y, str(i), ha='center', va='center',
color='black', fontsize=8, zorder=6)

    plt.axis('off')

    plt.xlim(-0.2, 2.2)
    plt.ylim(-0.2, 2.2)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.title(title)

    plt.show()
```

analysis_k1.py

```python
import numpy as np
# Function to calculate degree, incoming, and outgoing edges for a directed graph
from graph_generation_k1 import generate_matrices
adjacency_matrix, adjacency_matrix_undir = generate_matrices()


def analyze_directed_graph(adj_matrix):
    num_nodes = adj_matrix.shape[0]
    degrees = {}
    in_degrees = {}
    out_degrees = {}
    leaf_nodes = []
    isolated_nodes = []


    print("\n--- Directed Graph Analysis ---")


    for i in range(num_nodes):
        out_degree = np.sum(adj_matrix[i, :])
        in_degree = np.sum(adj_matrix[:, i])
        degrees[i] = out_degree + in_degree
        in_degrees[i] = in_degree
        out_degrees[i] = out_degree


        print(f"Node {i}:")
        print(f"  Degree: {degrees[i]}")
```

```
        print(f"  Incoming Edges (In-degree): {in_degrees[i]}")
        print(f"  Outgoing Edges (Out-degree): {out_degrees[i]}")

        if out_degree == 0 and in_degree > 0:
            leaf_nodes.append(i)
            pass

        if degrees[i] == 0:
            isolated_nodes.append(i)

    # Перевірка на регулярність
    unique_degrees = set(degrees.values())
    if len(unique_degrees) == 1:
        regularity_degree = list(unique_degrees)[0]
        print(f"\nThe directed graph is regular with degree {regularity_degree}.")
    else:
        print("\nThe directed graph is not regular.")

    print("\nLeaf nodes (nodes with out-degree 0):", leaf_nodes)
    print("Isolated nodes (nodes with degree 0):", isolated_nodes)
    print("--------------------------------------")
```

```python
def analyze_undirected_graph(adj_matrix):
    num_nodes = adj_matrix.shape[0]
    degrees = {}
    leaf_nodes = []
    isolated_nodes = []

    print("\n--- Undirected Graph Analysis ---")

    adj_matrix_undir = adj_matrix + adj_matrix.T
    adj_matrix_undir[adj_matrix_undir > 1] = 1

    for i in range(num_nodes):
        degree = np.sum(adj_matrix_undir[i, :])
        degrees[i] = degree
        print(f"Node {i}: Degree: {degree}")

        if degree == 1:
            leaf_nodes.append(i)
        elif degree == 0:
            isolated_nodes.append(i)

    #Перевірка на регулярність
    unique_degrees = set(degrees.values())
    if len(unique_degrees) == 1:
        regularity_degree = list(unique_degrees)[0]
```

```
        print(f"\nThe undirected graph is regular with 
degree {regularity_degree}.")

    else:

        print("\nThe undirected graph is not regular.")


    print("\nLeaf nodes (nodes with degree 1):", 
leaf_nodes)

    print("Isolated nodes (nodes with degree 0):", 
isolated_nodes)

    print("---------------------------------------")


analyze_directed_graph(adjacency_matrix)

analyze_undirected_graph(adjacency_matrix_undir)
```

analysis_k2.py

```
import numpy as np

import matplotlib.pyplot as plt

from graph_generation_k2 import generate_matrices

from draw import draw_graph


adjacency_matrix = generate_matrices()

NUM_NODES = adjacency_matrix.shape[0]


print("\n--- Directed Graph Analysis ---")

for i in range(NUM_NODES):

    print(f"Node {i}:")

    print(f"  Degree: {np.sum(adjacency_matrix[i, :]) + 
np.sum(adjacency_matrix[:, i])}")
```

```
    print(f"  Incoming Edges (In-degree):
{np.sum(adjacency_matrix[:, i])}")
    print(f"  Outgoing Edges (Out-degree):
{np.sum(adjacency_matrix[i, :])}")

print("\n--- Directed Graph Paths ---")
adj_matrix_sq = np.linalg.matrix_power(adjacency_matrix,
2)
adj_matrix_cube =
np.linalg.matrix_power(adjacency_matrix, 3)

paths_2 = [f"{i}->{j}" for i in range(NUM_NODES) for j
in range(NUM_NODES) if adj_matrix_sq[i, j] > 0]
print("Paths of length 2:\n", ",".join(paths_2))

print("\nPaths of length 3:")
paths_3 = []
for i in range(NUM_NODES):
    for j in range(NUM_NODES):
        if adj_matrix_cube[i, j] > 0:
            for m in range(NUM_NODES):
                for n in range(NUM_NODES):
                    if adjacency_matrix[i, m] == 1 and
adjacency_matrix[m, n] == 1 and adjacency_matrix[n, j]
== 1:

paths_3.append(f"{i}->{m}->{n}->{j}")
```

```python
print(",".join(paths_3))
print("-----------------------------")

print("\n--- Reachability Matrix (Directed) ---")
reachability_matrix = np.copy(adjacency_matrix)
for i in range(NUM_NODES):
    reachability_matrix[i, i] = 1

for k in range(NUM_NODES):
    for i in range(NUM_NODES):
        for j in range(NUM_NODES):
            if reachability_matrix[i, k] == 1 and
reachability_matrix[k, j] == 1:
                reachability_matrix[i, j] = 1

print(reachability_matrix)

print("\n--- Strongly Connected Components (Directed)
---")
print("Strongly Connected Components:")

visited_scc1 = [False] * NUM_NODES
stack = []

def fill_order(node):
    visited_scc1[node] = True
```

```
    for neighbor in range(NUM_NODES):
        if adjacency_matrix[node, neighbor] == 1 and not visited_scc1[neighbor]:
            fill_order(neighbor)
    stack.append(node)

for i in range(NUM_NODES):
    if not visited_scc1[i]:
        fill_order(i)

transpose_matrix = adjacency_matrix.T
visited_scc2 = [False] * NUM_NODES
strongly_connected_components = []

def dfs_transpose(node, component):
    visited_scc2[node] = True
    component.append(node)
    for neighbor in range(NUM_NODES):
        if transpose_matrix[node, neighbor] == 1 and not visited_scc2[neighbor]:
            dfs_transpose(neighbor, component)

while stack:
    node = stack.pop()
    if not visited_scc2[node]:
        component = []
```

```
        dfs_transpose(node, component)

strongly_connected_components.append(sorted(component))


for i, scc in enumerate(strongly_connected_components,
start=1):
    print(f"SCC {i}: {scc}")


print("---------------------------------------------")


print("\n--- Condensed Graph ---")
num_sccs = len(strongly_connected_components)


if num_sccs == 1:
    print("Graph is fully strongly connected, no
condensation needed.")


    plt.figure(figsize=(6, 6))
    plt.text(0.5, 0.5, "No condensed graph",
fontsize=14, ha="center", va="center", color="red")
    plt.axis("off")
    plt.title("Condensed Graph (Empty)")
    plt.show()
else:
    condensed_adj_matrix = np.zeros((num_sccs,
num_sccs), dtype=int)
```

```
    scc_map = {node: i for i, scc in
enumerate(strongly_connected_components) for node in
scc}


    for i in range(NUM_NODES):
        for j in range(NUM_NODES):
            if adjacency_matrix[i, j] == 1:
                scc_i = scc_map[i]
                scc_j = scc_map[j]
                if scc_i != scc_j:
                    condensed_adj_matrix[scc_i, scc_j] =
1


    print("Condensed Graph Adjacency Matrix:")
    print(condensed_adj_matrix)
    print("----------------------------------")
    print("\n--- Drawing Condensed Directed Graph ---")
    draw_graph(condensed_adj_matrix, directed=True)
```
main.py

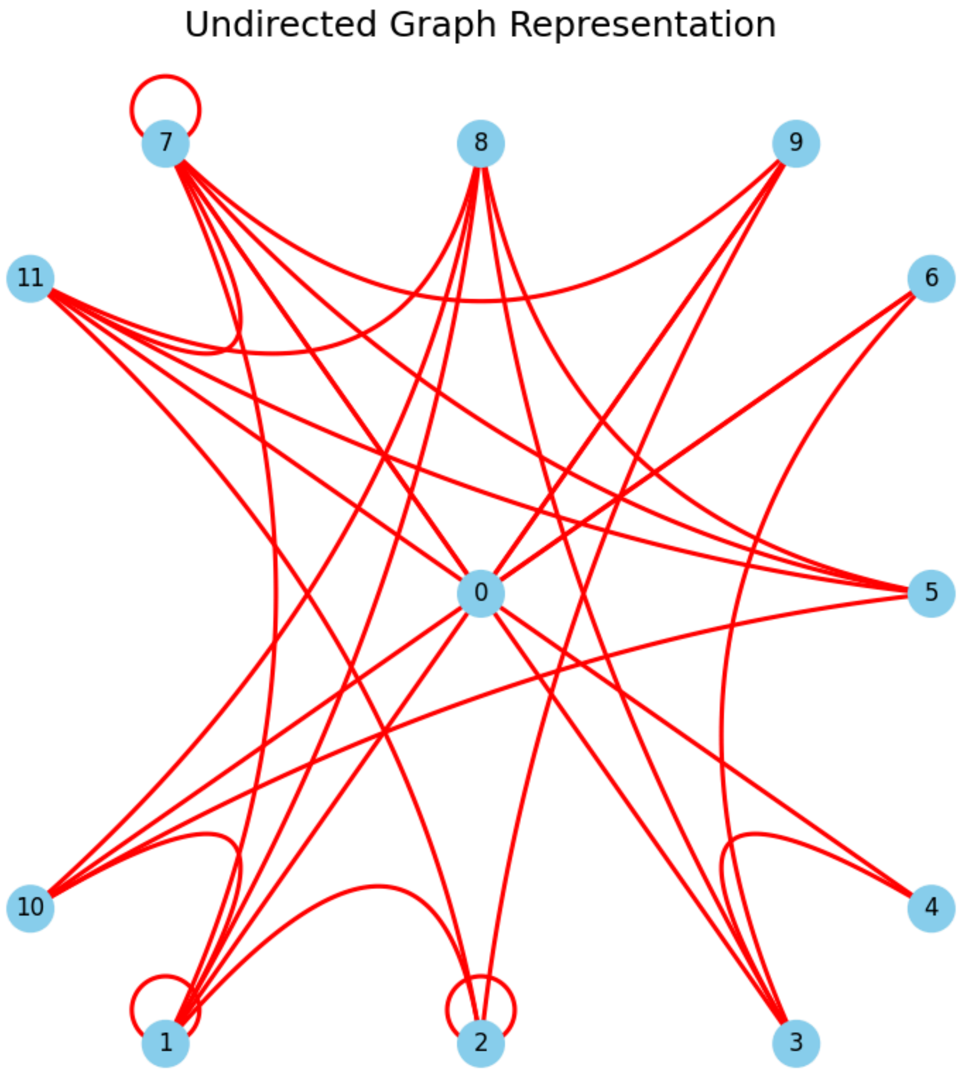
```
import graph_generation_k1
import draw


adj_matrix_dir, adj_matrix_undir =
graph_generation_k1.generate_matrices()


print("Directed Adjacency Matrix:\n", adj_matrix_dir)
```

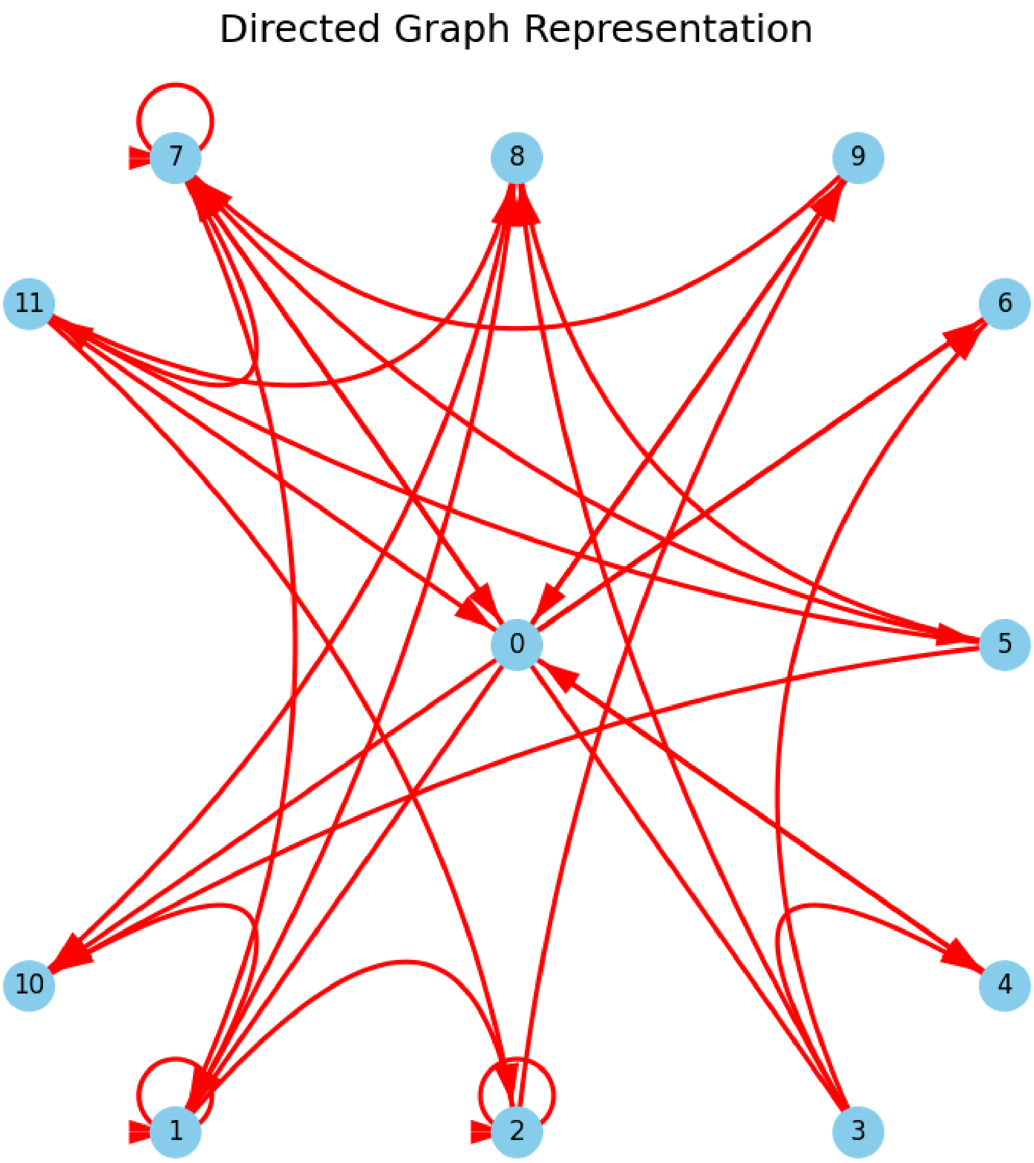
```
print("\nUndirected Adjacency Matrix:\n",
adj_matrix_undir)


draw.draw_graph(adj_matrix_undir, directed=False)


draw.draw_graph(adj_matrix_dir, directed=True)
```

Результати тестування програми та перевірки

За п. 1 завдання: згенеровані матриці суміжності напрямленого та ненапрямленого графів

```
Directed Adjacency Matrix:
 [[0 0 0 0 1 0 1 0 0 0 0 0]
 [0 1 1 0 0 0 0 0 1 1 0 0]
 [0 0 1 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 1 0 1 1 1 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 1]
 [0 0 0 0 0 0 0 0 0 0 1 0]
 [1 1 0 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 1 1 0 0 0]]

Undirected Adjacency Matrix:
 [[0 0 0 0 1 0 1 1 0 1 0 1]
 [0 1 1 0 0 0 0 1 1 1 1 0]
 [0 1 1 0 0 0 0 0 0 1 0 1]
 [0 0 0 0 1 0 1 1 1 0 0 0]
 [1 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 1 0 1 1]
 [1 0 0 1 0 0 0 0 0 0 1 0]
 [1 1 0 1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 0 0 0 1 1]
 [1 1 1 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 1 1 0 1 0 0 0]
 [1 0 1 0 0 1 0 1 1 0 0 0]]
```


Undirected Graph Representation
7
8
9
11
6
0
5
10
4
1
2
3



За п. 2: перелік степенів, півстепенів, результат перевірки на однорідність, переліки висячих та ізольованих верши

```
--- Directed Graph Analysis ---
Node 0:
  Degree: 6
  Incoming Edges (In-degree): 4
  Outgoing Edges (Out-degree): 2
Node 1:
  Degree: 7
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 4
Node 2:
  Degree: 5
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 2
Node 3:
  Degree: 4
  Incoming Edges (In-degree): 0
  Outgoing Edges (Out-degree): 4
Node 4:
  Degree: 3
  Incoming Edges (In-degree): 2
  Outgoing Edges (Out-degree): 1
Node 5:
  Degree: 4
  Incoming Edges (In-degree): 1
  Outgoing Edges (Out-degree): 3
Node 6:
  Degree: 3
  Incoming Edges (In-degree): 2
  Outgoing Edges (Out-degree): 1
Node 7:
  Degree: 8
  Incoming Edges (In-degree): 4
  Outgoing Edges (Out-degree): 4
Node 8:
  Degree: 5
  Incoming Edges (In-degree): 4
  Outgoing Edges (Out-degree): 1
```

```
Node 9:
  Degree: 4
  Incoming Edges (In-degree): 2
  Outgoing Edges (Out-degree): 2
Node 10:
  Degree: 4
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 1
Node 11:
  Degree: 5
  Incoming Edges (In-degree): 1
  Outgoing Edges (Out-degree): 4

The directed graph is not regular.

Leaf nodes (nodes with out-degree 0): []
Isolated nodes (nodes with degree 0): []
-------------------------------------


--- Undirected Graph Analysis ---
Node 0: Degree: 5
Node 1: Degree: 6
Node 2: Degree: 4
Node 3: Degree: 4
Node 4: Degree: 2
Node 5: Degree: 4
Node 6: Degree: 3
Node 7: Degree: 7
Node 8: Degree: 5
Node 9: Degree: 4
Node 10: Degree: 4
Node 11: Degree: 5

The undirected graph is not regular.

Leaf nodes (nodes with degree 1): []
Isolated nodes (nodes with degree 0): []
---------------------------------------
```

За п. 3: матриця другого орграфа

```
Directed Adjacency Matrix:
 [[0 0 0 0 1 0 1 0 0 0 0 0]
 [0 1 1 0 0 0 0 0 1 1 0 0]
 [0 1 1 0 0 0 0 0 0 1 0 1]
 [0 0 1 0 1 0 1 1 1 0 0 0]
 [1 0 0 0 0 0 1 0 0 0 1 0]
 [1 0 0 0 0 0 0 0 1 0 1 1]
 [0 0 0 0 0 0 0 0 0 0 1 0]
 [1 1 0 0 0 1 0 1 0 0 0 0]
 [1 0 1 1 0 0 0 0 0 0 1 0]
 [1 1 1 0 0 0 0 1 1 1 0 0]
 [0 1 0 0 0 1 0 0 0 0 0 0]
 [1 0 1 0 1 0 0 1 1 0 0 1]]
```

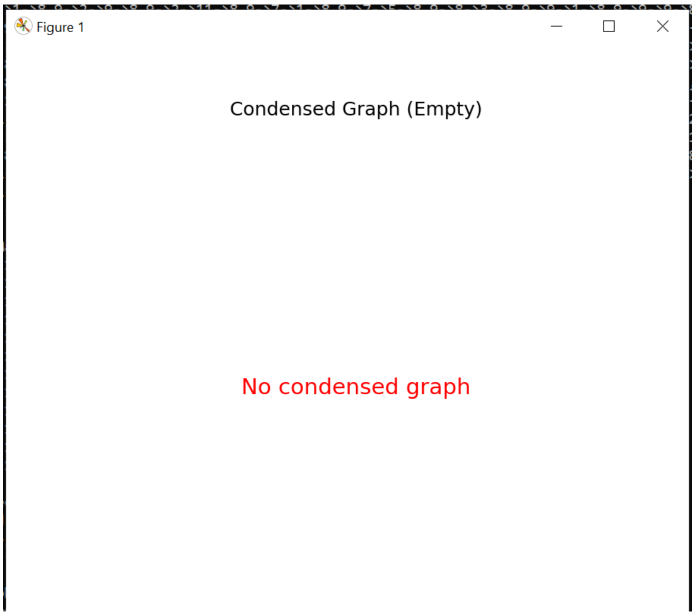
За п. 4: переліки півстепенів, шляхів, матриці досяжності та сильної зв’язності, перелік компонент сильної зв’язності, граф конденсації

```
--- Directed Graph Analysis ---
Node 0:
  Degree: 8
  Incoming Edges (In-degree): 6
  Outgoing Edges (Out-degree): 2
Node 1:
  Degree: 9
  Incoming Edges (In-degree): 5
  Outgoing Edges (Out-degree): 4
Node 2:
  Degree: 10
  Incoming Edges (In-degree): 6
  Outgoing Edges (Out-degree): 4
Node 3:
  Degree: 6
  Incoming Edges (In-degree): 1
  Outgoing Edges (Out-degree): 5
Node 4:
  Degree: 6
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 3
Node 5:
  Degree: 6
  Incoming Edges (In-degree): 2
  Outgoing Edges (Out-degree): 4
Node 6:
  Degree: 4
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 1
Node 7:
  Degree: 8
  Incoming Edges (In-degree): 4
  Outgoing Edges (Out-degree): 4
Node 8:
  Degree: 9
  Incoming Edges (In-degree): 5
  Outgoing Edges (Out-degree): 4
Node 9:
  Degree: 9
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 6
```

```
Node 9:
  Degree: 9
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 6
Node 10:
  Degree: 6
  Incoming Edges (In-degree): 4
  Outgoing Edges (Out-degree): 2
Node 11:
  Degree: 9
  Incoming Edges (In-degree): 3
  Outgoing Edges (Out-degree): 6

--- Directed Graph Paths ---
Paths of length 2:
 0->0,0->6,0->10,1->0,1->1,1->2,1->3,1->7,1->8,1->9,1->10,1->11,2->0,2->1,2->2,2->4,2->7,2->8,2->9,2->11,3->0,3->1,3->2,3->3,3->5,3->6,3->7,3->9,3->10,3->11,4->1,4->4,4->5,
4->6,4->10,5->0,5->1,5->2,5->3,5->4,5->5,5->6,5->7,5->8,5->10,5->11,6->1,6->5,7->0,7->1,7->2,7->4,7->5,7->6,7->7,7->8,7->9,7->10,7->11,8->1,8->2,8->4,8->5,8->6,8->7,8->8,8-
>9,8->11,9->0,9->1,9->2,9->3,9->4,9->5,9->6,9->7,9->8,9->9,9->10,9->11,10->0,10->1,10->2,10->8,10->9,10->10,10->11,11->0,11->1,11->2,11->3,11->4,11->5,11->6,11->7,11->8,11-
>9,11->10,11->11
```

```
Paths of length 3:
0->4->10->1,0->6->10->1,0->4->0->4,0->4->10->5,0->6->10->5,0->4->0->6,0->4->6->10,1->1->8->0,1->1->9->0,1->2->9->0,1->2->11->0,1->9->7->0,1->9->8->0,1->9->9->0,1->1->1->1,1
->1->2->1,1->1->9->1,1->2->1->1,1->2->2->1,1->2->9->1,1->8->2->1,1->8->10->1,1->9->1->1,1->9->2->1,1->9->7->1,1->9->9->1,1->1->1->2,1->1->2->2,1->1->8->2,1->1->9->2,1->2->1
->2,1->2->2->2,1->2->9->2,1->2->11->2,1->8->2->2,1->8->3->2,1->9->1->2,1->9->2->2,1->9->8->2,1->9->9->2,1->1->8->3,1->9->8->3,1->2->11->4,1->8->0->4,1->8->3->4,1->9->0->4,1
->8->10->5,1->9->7->5,1->8->0->6,1->8->3->6,1->9->0->6,1->1->9->7,1->2->9->7,1->2->11->7,1->8->3->7,1->9->7->7,1->9->9->7,1->1->1->8,1->1->9->8,1->2->1->8,1->2->9->8,1->2->
11->8,1->8->3->8,1->9->1->8,1->9->9->8,1->1->1->9,1->1->2->9,1->1->9->9,1->2->1->9,1->2->2->9,1->2->9->9,1->8->2->9,1->9->1->9,1->9->2->9,1->9->9->9,1->1->8->10,1->9->8->10
,1->1->2->11,1->2->2->11,1->2->11->11,1->8->2->11,1->9->2->11,2->1->8->0,2->1->9->0,2->2->9->0,2->2->11->0,2->9->7->0,2->9->8->0,2->9->9->0,2->11->4->0,2->11->7->0,2->11->8
->0,2->11->11->0,2->1->1->1,2->1->2->1,2->1->9->1,2->2->1->1,2->2->2->1,2->2->9->1,2->9->1->1,2->9->2->1,2->9->7->1,2->9->9->1,2->11->2->1,2->11->7->1,2->1->1->2,2->1->2->2
,2->1->8->2,2->1->9->2,2->2->1->2,2->2->2->2,2->2->9->2,2->2->11->2,2->9->1->2,2->9->2->2,2->9->8->2,2->9->9->2,2->11->2->2,2->11->8->2,2->11->11->2,2->1->8->3,2->9->8->3,2
->11->8->3,2->2->11->4,2->9->0->4,2->11->0->4,2->11->11->4,2->9->7->5,2->11->7->5,2->9->0->6,2->11->0->6,2->11->4->6,2->1->9->7,2->2->9->7,2->2->11->7,2->9->7->7,2->9->9->7
,2->11->7->7,2->11->11->7,2->1->1->8,2->1->9->8,2->2->1->8,2->2->9->8,2->2->11->8,2->9->1->8,2->9->9->8,2->11->11->8,2->1->1->9,2->1->2->9,2->1->9->9,2->2->1->9,2->2->2->9,
2->2->9->9,2->9->1->9,2->9->2->9,2->9->9->9,2->11->2->9,2->1->8->10,2->9->8->10,2->11->4->10,2->11->8->10,2->1->2->11,2->2->2->11,2->2->11->11,2->9->2->11,2->11->2->11,2->1
1->11->11,3->2->9->0,3->2->11->0,3->7->5->0,3->7->7->0,3->2->1->1,3->2->2->1,3->2->9->1,3->4->10->1,3->6->10->1,3->7->1->1,3->7->7->1,3->8->2->1,3->8->10->1,3->2->1->2,3->2
->2->2,3->2->9->2,3->2->11->2,3->7->1->2,3->8->2->2,3->8->3->2,3->2->11->4,3->4->0->4,3->7->0->4,3->8->0->4,3->8->3->4,3->4->10->5,3->6->10->5,3->7->7->5,3->8->10->5,3->4->
0->6,3->7->0->6,3->8->0->6,3->8->3->6,3->2->9->7,3->2->11->7,3->7->7->7,3->8->3->7,3->2->1->8,3->2->9->8,3->2->11->8,3->7->1->8,3->7->5->8,3->8->3->8,3->2->1->9,3->2->2->9,
3->2->9->9,3->7->1->9,3->8->2->9,3->4->6->10,3->7->5->10,3->2->2->11,3->2->11->11,3->7->5->11,3->8->2->11,4->0->4->0,4->10->5->0,4->6->10->1,4->10->1->1,4->10->1->2,4->6->1
0->5,4->0->4->6,4->10->1->8,4->10->5->8,4->10->1->9,4->0->4->10,4->0->6->10,4->10->5->10,4->10->5->11,5->0->4->0,5->10->5->0,5->11->4->0,5->11->7->0,5->11->8->0,5->11->11->
0,5->8->2->1,5->8->10->1,5->10->1->1,5->11->2->1,5->11->7->1,5->8->2->2,5->8->3->2,5->10->1->2,5->11->2->2,5->11->8->2,5->11->11->2,5->11->8->3,5->8->0->4,5->8->3->4,5->11-
>0->4,5->11->11->4,5->8->10->5,5->11->7->5,5->0->4->6,5->8->0->6,5->8->3->6,5->11->0->6,5->11->4->6,5->8->3->7,5->11->7->7,5->11->11->7,5->8->3->8,5->10->1->8,5->10->5->8,5
->11->11->8,5->8->2->9,5->10->1->9,5->11->2->9,5->0->4->10,5->0->6->10,5->10->5->10,5->11->4->10,5->11->8->10,5->8->2->11,5->10->5->11,5->11->2->11,5->11->11->11,6->10->5->
0,6->10->1->1,6->10->1->2,6->10->1->8,6->10->5->8,6->10->1->9,6->10->5->10,6->10->5->11,7->0->4->0,7->1->8->0,7->1->9->0,7->5->8->0,7->5->11->0,7->7->5->0,7->7->7->0,7->1->
1->1,7->1->2->1,7->1->9->1,7->5->10->1,7->7->1->1,7->7->7->1,7->1->1->2,7->1->2->2,7->1->8->2,7->1->9->2,7->5->8->2,7->5->11->2,7->7->1->2,7->1->8->3,7->5->8->3,7->5->0->4,
7->5->11->4,7->7->0->4,7->5->10->5,7->7->7->5,7->0->4->6,7->5->0->6,7->7->0->6,7->1->9->7,7->5->11->7,7->7->7->7,7->1->1->8,7->1->9->8,7->5->11->8,7->7->1->8,7->7->5->8,7->
1->1->9,7->1->2->9,7->1->9->9,7->7->1->9,7->0->4->10,7->0->6->10,7->1->8->10,7->5->8->10,7->7->5->10,7->1->2->11,7->5->11->11,7->7->5->11,8->0->4->0,8->2->9->0,8->2->11->0,
8->3->4->0,8->3->7->0,8->3->8->0,8->10->5->0,8->2->1->1,8->2->2->1,8->2->9->1,8->3->2->1,8->3->7->1,8->10->1->1,8->2->1->2,8->2->2->2,8->2->9->2,8->2->11->2,8->3->2->2,8->3
->8->2,8->10->1->2,8->3->8->3,8->2->11->4,8->3->7->5,8->0->4->6,8->3->4->6,8->2->9->7,8->2->11->7,8->3->7->7,8->2->1->8,8->2->9->8,8->2->11->8,8->10->1->8,8->10->5->8,8->2-
>1->9,8->2->2->9,8->2->9->9,8->3->2->9,8->10->1->9,8->0->4->10,8->0->6->10,8->3->4->10,8->3->6->10,8->3->8->10,8->10->5->10,8->2->2->11,8->2->11->11,8->3->2->11,8->10->5->1
1,9->0->4->0,9->1->8->0,9->1->9->0,9->2->9->0,9->2->11->0,9->7->5->0,9->7->7->0,9->9->7->0,9->9->8->0,9->9->9->0,9->1->1->1,9->1->2->1,9->1->9->1,9->2->1->1,9->2->2->1,9->2
->9->1,9->7->1->1,9->7->7->1,9->8->2->1,9->8->10->1,9->9->1->1,9->9->2->1,9->9->7->1,9->9->9->1,9->1->1->2,9->1->2->2,9->1->8->2,9->1->9->2,9->2->1->2,9->2->2->2,9->2->9->2
,9->2->11->2,9->7->1->2,9->8->2->2,9->8->3->2,9->9->1->2,9->9->2->2,9->9->8->2,9->9->9->2,9->1->8->3,9->9->8->3,9->2->11->4,9->7->0->4,9->8->0->4,9->8->3->4,9->9->0->4,9->7
->7->5,9->8->10->5,9->9->7->5,9->0->4->6,9->7->0->6,9->8->0->6,9->8->3->6,9->9->0->6,9->1->9->7,9->2->9->7,9->2->11->7,9->7->7->7,9->8->3->7,9->9->7->7,9->9->9->7,9->1->1->
8,9->1->9->8,9->2->1->8,9->2->9->8,9->2->11->8,9->7->1->8,9->7->5->8,9->8->3->8,9->9->1->8,9->9->9->8,9->1->1->9,9->1->2->9,9->1->9->9,9->2->1->9,9->2->2->9,9->2->9->9,9->7
->1->9,9->8->2->9,9->9->1->9,9->9->2->9,9->9->9->9,9->0->4->10,9->0->6->10,9->1->8->10,9->7->5->10,9->9->8->10,9->1->2->11,9->2->2->11,9->2->11->11,9->7->5->11,9->8->2->11,
9->9->2->11,10->1->8->0,10->1->9->0,10->5->8->0,10->5->11->0,10->1->1->1,10->1->2->1,10->1->9->1,10->5->10->1,10->1->1->2,10->1->2->2,10->1->8->2,10->1->9->2,10->5->8->2,10
->5->11->2,10->1->8->3,10->5->8->3,10->5->0->4,10->5->11->4,10->5->10->5,10->5->0->6,10->1->9->7,10->5->11->7,10->1->1->8,10->1->9->8,10->5->11->8,10->1->1->9,10->1->2->9,1
0->1->9->9,10->1->8->10,10->5->8->10,10->1->2->11,10->5->11->11,11->0->4->0,11->2->9->0,11->2->11->0,11->7->5->0,11->7->7->0,11->11->4->0,11->11->7->0,11->11->8->0,11->11->
11->0,11->2->1->1,11->2->2->1,11->2->9->1,11->4->10->1,11->7->1->1,11->7->7->1,11->8->2->1,11->8->10->1,11->11->2->1,11->11->7->1,11->2->1->2,11->2->2->2,11->2->9->2,11->2-
>11->2,11->7->1->2,11->8->2->2,11->8->3->2,11->11->2->2,11->11->8->2,11->11->11->2,11->11->8->3,11->2->11->4,11->4->0->4,11->7->0->4,11->8->0->4,11->8->3->4,11->11->0->4,11
->11->11->4,11->4->10->5,11->7->7->5,11->8->10->5,11->11->7->5,11->0->4->6,11->4->0->6,11->7->0->6,11->8->0->6,11->8->3->6,11->11->0->6,11->11->4->6,11->2->9->7,11->2->11->
7,11->7->7->7,11->8->3->7,11->11->7->7,11->11->11->7,11->2->1->8,11->2->9->8,11->2->11->8,11->7->1->8,11->7->5->8,11->8->3->8,11->11->11->8,11->2->1->9,11->2->2->9,11->2->9
->9,11->7->1->9,11->8->2->9,11->11->2->9,11->0->4->10,11->0->6->10,11->4->6->10,11->7->5->10,11->11->4->10,11->11->8->10,11->2->2->11,11->2->11->11,11->7->5->11,11->8->2->1
```

```
1,11->11->2->11,11->11->11->11
----------------------------


--- Reachability Matrix (Directed) ---
[[1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]]

--- Strongly Connected Components (Directed) ---
Strongly Connected Components:
SCC 1: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
---------------------------------------------


--- Condensed Graph ---
Graph is fully strongly connected, no condensation needed.
```



Висновки

У ході лабораторної роботи було досліджено орієнтовані графи, їх основні властивості та алгоритми аналізу. Орієнтовані графи є математичними структурами, що складаються з вершин та дуг (спрямованих ребер), які визначають односторонні зв'язки між об'єктами. Вони широко застосовуються у моделюванні транспортних систем, аналізі інформаційних потоків, соціальних мережах та комп'ютерних мережах.

Однією з ключових задач, розглянутих у роботі, було визначення матриці досяжності за допомогою алгоритму Флойда-Воршалла, який забезпечує побудову транзитивного замикання графа. Цей метод дозволяє встановити, чи існує шлях між довільними вершинами графа, що є важливим при аналізі зв'язності системи. Його перевагою є повнота отриманої інформації, проте недоліком виступає висока обчислювальна складність $O(N^3)$, що обмежує застосування для великих графів.

Також було проведено дослідження компонент сильної зв'язності (SCC), які складаються з підгруп вершин, де будь-яка вершина досяжна з будь-якої іншої у межах своєї компоненти. Виявлення SCC здійснювалось за допомогою алгоритму Косараджу. Цей алгоритм ефективно розділяє граф на незалежні підгрупи, що дозволяє краще зрозуміти його структуру. Основною перевагою є простота реалізації та ефективність, тоді як недоліками є потреба у двох проходах по графу та залежність від правильного вибору порядку обходу вершин.

Подальший етап роботи включав побудову графа конденсації, де кожна SCC представлена окремою вершиною, а ребра між ними визначають загальні зв'язки між компонентами. Цей метод дає змогу спростити аналіз графа, зменшити його розмір та покращити інтерпретацію складних структур. Проте граф конденсації має обмеження у втраті деталізації внутрішньої структури компонент.

Окрім цього, було проведено аналіз шляхів довжиною 2 і 3, що дозволяє визначати опосередковані зв'язки між вершинами. Використання піднесення матриці суміжності до відповідного степеня є ефективним методом, проте при високій щільності зв'язків може призводити до зайвої інформації, що ускладнює аналіз.