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-42
Федоренко Іван Русланович
номер у списку групи: 29

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 - n3 *0.01 - n4 * 0.01 - 0.3.

Отже, матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту n1n2n3n4;

2) матриця розміром n n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);

3) обчислюється коефіцієнт k = 1.0 - n3 *0.01 - n4 * 0.01 - 0.3., кожен елемент матриці множиться на коефіцієнт k;

4) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

2. 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 Adir, коефіцієнт k = 1.0 - n3* 0.005 - n4*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.

При проєктуванні програми слід врахувати наступне:

1) мова програмування обирається студентом самостійно;

2) графічне зображення усіх графів має формуватися програмою з тими ж вимогами, як у ЛР №3;

3) всі графи, включно із графом конденсації, обов’язково зображувати у графічному вікні;

4) типи та структури даних для внутрішнього представлення всіх даних у програмі слід вибрати самостійно;

5) обчислення перелічених у завданні результатів має виконуватися розробленою програмою (не вручну і не сторонніми засобами);

6) матриці, переліки степенів та маршрутів тощо можна виводити в графічне вікно або консоль — на розсуд студента;

7) у переліку знайдених шляхів треба вказувати не лише початок та кінець шляху, але й усі проміжні вершини, через які він проходить (наприклад, 1 – 5 – 3 – 2).

Код програми:

Програма складається з наступних файлів:

Main.py, drawing_methods.py, matrix_methods.py, graph_analysis.py.

Main.py

```
import shutil
import matplotlib.pyplot as plt
from matrix_methods import (
    generate_adjacency_matrix, get_undirected_matrix,
    calculate_reachability_matrix, calculate_strong_connectivity_matrix,
    print_matrix, calculate_degrees
)
from drawing_methods import (
    draw_graph, draw_condensation_graph,
    get_vertex_positions
)
from graph_analysis import (
    is_regular_graph, find_special_vertices,
    find_paths_of_length, find_strongly_connected_components,
    create_condensation_graph, format_path
)

def main():
    variant_number = 4229
    n3 = 2
    n4 = 9
    n = 10 + n3

    print(f"Variant number: {variant_number}")
    print(f"Number of vertices n = 10 + {n3} = {n}")

    print("\n=== PART 1: Original Graph Analysis ===")
    directed_matrix = generate_adjacency_matrix(n, variant_number, k_formula=1)
    undirected_matrix = get_undirected_matrix(directed_matrix)

    print_matrix(directed_matrix, f"Directed Graph Adjacency Matrix ({n}x{n})")
    print_matrix(undirected_matrix, f"Undirected Graph Adjacency Matrix ({n}x{n})")

    in_degrees, out_degrees = calculate_degrees(directed_matrix, is_directed=True)
    undirected_degrees = calculate_degrees(undirected_matrix, is_directed=False)

    print("\nDirected Graph:")
    print("Vertex | In-degree | Out-degree")
    for i in range(n):
        print(f"{i+1:6d} | {in_degrees[i]:9d} | {out_degrees[i]:10d}")
```

```
    print("\nUndirected Graph:")
    print("Vertex | Degree")
    for i in range(n):
        print(f"{i+1:6d} | {undirected_degrees[i]:6d}")

    if is_regular_graph(undirected_degrees):
        print(f"\nUndirected graph is regular with degree {undirected_degrees[0]}")
    else:
        print("\nUndirected graph is not regular")

    if is_regular_graph(in_degrees) and is_regular_graph(out_degrees) and
in_degrees[0] == out_degrees[0]:
        print(f"Directed graph is regular with in-degree = out-degree =
{in_degrees[0]}")
    else:
        print("Directed graph is not regular")

    dir_hanging, dir_isolated = find_special_vertices(directed_matrix,
is_directed=True)
    undir_hanging, undir_isolated = find_special_vertices(undirected_matrix,
is_directed=False)

    print("\nDirected Graph:")
    print(f"Hanging vertices: {dir_hanging}")
    print(f"Isolated vertices: {dir_isolated}")

    print("\nUndirected Graph:")
    print(f"Hanging vertices: {undir_hanging}")
    print(f"Isolated vertices: {undir_isolated}")

    # Draw graphs
    positions = get_vertex_positions(n, n4)
    fig1 = draw_graph(directed_matrix, positions, is_directed=True, title="Original")
    fig1.savefig('directed_graph_original.png')

    fig2 = draw_graph(directed_matrix, positions, is_directed=False, title="Original")
    fig2.savefig('undirected_graph_original.png')

    print("\n\n=== PART 2: Modified Graph Analysis ===")
    new_directed_matrix = generate_adjacency_matrix(n, variant_number, k_formula=2)
    print_matrix(new_directed_matrix, f"Modified Directed Graph Adjacency Matrix
({n}x{n})")

    new_in_degrees, new_out_degrees = calculate_degrees(new_directed_matrix,
is_directed=True)

    print("\nModified Directed Graph:")
    print("Vertex | In-degree | Out-degree")
```

```
    for i in range(n):
        print(f"{i+1:6d} | {new_in_degrees[i]:9d} | {new_out_degrees[i]:10d}")

    def print_paths_optimized(paths, paths_length_title):
        print(f"\n{paths_length_title} (count: {len(paths)}):")

        if not paths:
            print("Немає шляхів для відображення")
            return

        formatted_paths = [format_path(path) for path in paths]
        max_path_length = max(len(path) for path in formatted_paths) + 2

        terminal_width = shutil.get_terminal_size().columns

        max_columns = max(1, terminal_width // (max_path_length + 1))

        for i in range(0, len(paths), max_columns):
            row_paths = formatted_paths[i:i+max_columns]
            row_str = ""
            for j, path in enumerate(row_paths):
                if j > 0:
                    row_str += "| "
                row_str += f"{path:<{max_path_length-2}}"
            print(row_str)

    paths_length_2 = find_paths_of_length(new_directed_matrix, 2)
    print_paths_optimized(paths_length_2, "Paths of length 2")

    paths_length_3 = find_paths_of_length(new_directed_matrix, 3)
    print_paths_optimized(paths_length_3, "Paths of length 3")

    reachability_matrix = calculate_reachability_matrix(new_directed_matrix)
    print_matrix(reachability_matrix, "Reachability Matrix")

    strong_connectivity_matrix =
calculate_strong_connectivity_matrix(reachability_matrix)
    print_matrix(strong_connectivity_matrix, "Strong Connectivity Matrix")

    components = find_strongly_connected_components(strong_connectivity_matrix)
    print("\nStrongly Connected Components:")
    for i, component in enumerate(components):
        print(f"Component {i+1}: {component}")

    condensation_matrix = create_condensation_graph(new_directed_matrix, components)
    print_matrix(condensation_matrix, "Condensation Graph Adjacency Matrix")

    fig3 = draw_graph(new_directed_matrix, positions, is_directed=True,
title="Modified")
```

```
    fig3.savefig('directed_graph_modified.png')

    fig4 = draw_condensation_graph(condensation_matrix, components)
    fig4.savefig('condensation_graph.png')

    plt.show()

if __name__ == "__main__":
    main()
```

Graph_analysis.py:

```
import numpy as np
from matrix_methods import calculate_degrees

def is_regular_graph(degrees):
    """Check if graph is regular (all vertices have the same degree)"""
    return np.all(degrees == degrees[0])

def find_special_vertices(matrix, is_directed=True):
    """Find hanging (leaf) and isolated vertices"""
    n = matrix.shape[0]

    if is_directed:
        in_degrees, out_degrees = calculate_degrees(matrix, is_directed=True)
        total_degrees = in_degrees + out_degrees

        isolated = [i+1 for i in range(n) if in_degrees[i] == 0 and out_degrees[i] ==
0]
        hanging = [i+1 for i in range(n) if total_degrees[i] == 1]
    else:
        degrees = calculate_degrees(matrix, is_directed=False)

        isolated = [i+1 for i in range(n) if degrees[i] == 0]
        hanging = [i+1 for i in range(n) if degrees[i] == 1]

    return hanging, isolated

def find_paths_of_length(matrix, length):
    """Find all paths of specific length using matrix powers"""
```

```
    n = matrix.shape[0]

    power_matrix = np.linalg.matrix_power(matrix, length)

    paths = []
    for i in range(n):
        for j in range(n):
            if power_matrix[i, j] > 0:
                intermediate_paths = find_all_paths(matrix, i, j, length)
                paths.extend(intermediate_paths)

    return paths

def find_all_paths(matrix, start, end, length, current_path=None, current_length=0):
    """Recursively find all paths of a specific length from start to end"""
    n = matrix.shape[0]

    if current_path is None:
        current_path = [start]

    if current_length == length:
        if current_path[-1] == end:
            return [current_path]
        return []

    paths = []
    for next_vertex in range(n):
        if matrix[current_path[-1], next_vertex] == 1:
            if current_length == length - 1 and next_vertex == end:
                paths.append(current_path + [next_vertex])
            elif current_length < length - 1:
                new_paths = find_all_paths(
                    matrix, start, end, length,
                    current_path + [next_vertex],
                    current_length + 1
                )
                paths.extend(new_paths)

    return paths

def find_strongly_connected_components(strong_connectivity):
    """Find strongly connected components from strong connectivity matrix"""
    n = strong_connectivity.shape[0]
    visited = [False] * n
    components = []

    for vertex in range(n):
        if not visited[vertex]:
            component = []
```

```
            dfs_component(vertex, strong_connectivity, visited, component)
            components.append([v+1 for v in component])  # Convert to 1-indexed

    return components

def dfs_component(vertex, matrix, visited, component):
    """DFS to find connected components"""
    visited[vertex] = True
    component.append(vertex)

    for next_vertex in range(matrix.shape[0]):
        if matrix[vertex, next_vertex] == 1 and not visited[next_vertex]:
            dfs_component(next_vertex, matrix, visited, component)

def create_condensation_graph(matrix, components):
    """Create condensation graph from strongly connected components"""
    n_components = len(components)
    condensation_matrix = np.zeros((n_components, n_components), dtype=int)

    # Check for edges between components
    for i in range(n_components):
        for j in range(n_components):
            if i != j:
                # Check if there's an edge from any vertex in component i
                # to any vertex in component j
                for v1 in [v-1 for v in components[i]]:  # Convert to 0-indexed
                    for v2 in [v-1 for v in components[j]]:  # Convert to 0-indexed
                        if matrix[v1, v2] == 1:
                            condensation_matrix[i, j] = 1
                            break
                    if condensation_matrix[i, j] == 1:
                        break

    return condensation_matrix

def format_path(path):
    """Format a path for printing"""
    return " - ".join(str(v+1) for v in path)  # Convert to 1-indexed
```

Matrix_methods.py:

```
import numpy as np

def calculate_degrees(matrix, is_directed=True):
    """Calculate vertex degrees
```

```
    For directed graphs, returns in-degrees and out-degrees
    For undirected graphs, returns degrees
    """
    n = matrix.shape[0]

    if is_directed:
        in_degrees = np.sum(matrix, axis=0)
        out_degrees = np.sum(matrix, axis=1)
        return in_degrees, out_degrees
    else:
        degrees = np.sum(matrix, axis=1)
        return degrees

def generate_adjacency_matrix(n, variant_number, k_formula=1):
    """Generate directed adjacency matrix based on variant number and k_formula
    k_formula=1: k = 1.0 - n3 * 0.01 - n4 * 0.01 - 0.3
    k_formula=2: k = 1.0 - n3 * 0.005 - n4 * 0.005 - 0.27
    """
    np.random.seed(variant_number)
    T = np.random.random((n, n)) * 2.0

    n3 = 2  # From variant
    n4 = 9  # From variant

    if k_formula == 1:
        k = 1.0 - n3 * 0.01 - n4 * 0.01 - 0.3
    else:
        k = 1.0 - n3 * 0.005 - n4 * 0.005 - 0.27

    print(f"Using k coefficient: {k}")

    A = np.zeros((n, n), dtype=int)
    for i in range(n):
        for j in range(n):
            A[i, j] = 1 if T[i, j] * k >= 1.0 else 0

    return A

def get_undirected_matrix(directed_matrix):
    """Convert directed adjacency matrix to undirected"""
    n = directed_matrix.shape[0]
    undirected_matrix = np.zeros((n, n), dtype=int)

    for i in range(n):
        for j in range(n):
            if directed_matrix[i, j] == 1 or directed_matrix[j, i] == 1:
                undirected_matrix[i, j] = 1
                undirected_matrix[j, i] = 1
```

```
    return undirected_matrix

def calculate_strong_connectivity_matrix(reachability):
    """Calculate strong connectivity matrix from reachability matrix"""
    n = reachability.shape[0]
    strong_connectivity = np.zeros((n, n), dtype=int)

    for i in range(n):
        for j in range(n):
            if reachability[i, j] == 1 and reachability[j, i] == 1:
                strong_connectivity[i, j] = 1

    return strong_connectivity

def print_matrix(matrix, title):
    """Print the adjacency matrix in a readable format"""
    print(f"\n{title}:")
    for row in matrix:
        print(" ".join(map(str, row)))

def calculate_reachability_matrix(matrix):
    """Calculate reachability matrix using transitive closure"""
    n = matrix.shape[0]

    # Initialize reachability with the adjacency matrix
    reachability = matrix.copy()

    # Add self-loops
    for i in range(n):
        reachability[i, i] = 1

    # Warshall's algorithm for transitive closure
    for k in range(n):
        for i in range(n):
            for j in range(n):
                reachability[i, j] = reachability[i, j] or (reachability[i, k] and
reachability[k, j])

    return reachability
```

Drawing_methods.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math
```

```python
def rotate_around_center(x, y, cx, cy, angle):
    """Обертання точки (x, y) навколо центру (cx, cy) на кут angle (в радіанах)"""
    dx = x - cx
    dy = y - cy
    cos_a = math.cos(angle)
    sin_a = math.sin(angle)
    x_new = cx + dx * cos_a - dy * sin_a
    y_new = cy + dx * sin_a + dy * cos_a
    return (x_new, y_new)

def draw_self_loop(ax, pos, color='blue', linewidth=1.5, is_directed=True):
    """Draw a self-loop using polygonal lines under the vertex (with rotation)"""
    cx, cy = pos
    R = 0.5
    index_angle = 0
    theta = index_angle

    cx += R * math.sin(theta)
    cy -= R * math.cos(theta)

    dx = 3 * R / 4
    dy = R * (1 - math.sqrt(7)) / 4

    p1 = (cx - dx, cy - dy)
    p2 = (cx - 3 * dx / 2, cy - R / 2)
    p3 = (cx + 3 * dx / 2, cy - R / 2)
    p4 = (cx + dx, cy - dy)

    p1 = rotate_around_center(p1[0], p1[1], cx, cy, theta)
    p2 = rotate_around_center(p2[0], p2[1], cx, cy, theta)
    p3 = rotate_around_center(p3[0], p3[1], cx, cy, theta)
    p4 = rotate_around_center(p4[0], p4[1], cx, cy, theta)

    ax.plot([p1[0], p2[0]], [p1[1], p2[1]], color=color, linewidth=linewidth)
    ax.plot([p2[0], p3[0]], [p2[1], p3[1]], color=color, linewidth=linewidth)

    if is_directed:
        dx_arrow = p4[0] - p3[0]
        dy_arrow = p4[1] - p3[1]
        ax.arrow(p3[0], p3[1], dx_arrow, dy_arrow,
                 head_width=0.15, head_length=0.15,
                 fc=color, ec=color, linewidth=linewidth,
                 length_includes_head=True)
    else:
        ax.plot([p3[0], p4[0]], [p3[1], p4[1]], color=color, linewidth=linewidth)

def draw_edge(ax, start, end, is_directed=True, color='blue', linewidth=1.5):
    """Draw an edge between two vertices"""
    dx = end[0] - start[0]
```

```
    dy = end[1] - start[1]

    dist = np.sqrt(dx**2 + dy**2)
    if dist < 0.001:
        return

    vertex_radius = 0.5
    ratio = vertex_radius / dist
    start_x = start[0] + dx * ratio
    start_y = start[1] + dy * ratio
    end_x = end[0] - dx * ratio
    end_y = end[1] - dy * ratio

    rad = 0.2

    if is_directed:
        arrow = patches.FancyArrowPatch(
            (start_x, start_y), (end_x, end_y),
            arrowstyle='->',
            color=color,
            linewidth=linewidth,
            connectionstyle=f'arc3,rad={rad}',
            mutation_scale=15
        )
    else:
        arrow = patches.FancyArrowPatch(
            (start_x, start_y), (end_x, end_y),
            arrowstyle='-',
            color=color,
            linewidth=linewidth,
            connectionstyle=f'arc3,rad={rad}'
        )

    ax.add_patch(arrow)

def draw_graph(adjacency_matrix, positions, is_directed=True, title="Graph"):
    """Draw a graph based on adjacency matrix and vertex positions"""
    n = adjacency_matrix.shape[0]

    fig, ax = plt.subplots(figsize=(12, 10))

    rect = patches.Rectangle((-6, -4), 12, 8, linewidth=1, edgecolor='gray',
                             facecolor='none', linestyle='--')
    ax.add_patch(rect)

    for i in range(n):
        for j in range(n):
            if adjacency_matrix[i, j] == 1:
                if i == j:
```

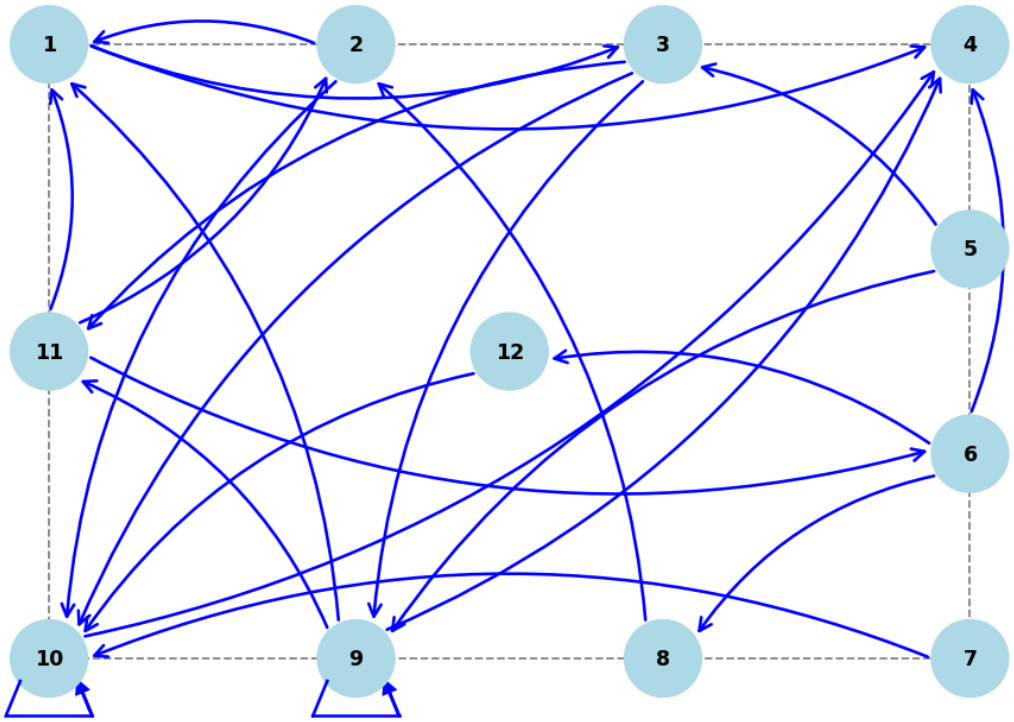
```
                    draw_self_loop(ax, positions[i], color='blue', linewidth=1.5,
is_directed=is_directed)
                else:
                    draw_edge(ax, positions[i], positions[j], is_directed,
color='blue')

    for i, pos in enumerate(positions):
        circle = plt.Circle(pos, 0.5, fill=True, color='lightblue', edgecolor='blue')
        ax.add_patch(circle)

        ax.text(pos[0], pos[1], str(i+1), horizontalalignment='center',
                verticalalignment='center', fontsize=10, color='black',
fontweight='bold')

    ax.set_aspect('equal')
    margin = 2
    ax.set_xlim(min(positions[:, 0])-margin, max(positions[:, 0])+margin)
    ax.set_ylim(min(positions[:, 1])-margin, max(positions[:, 1])+margin)

    graph_type = "Directed" if is_directed else "Undirected"
    plt.title(f"{title} - {graph_type} Graph - {n} vertices")

    plt.axis('off')
    return fig

def get_component_positions(components):
    """Create positions for components in condensation graph"""
    n_components = len(components)
    radius = 5

    positions = []
    for i in range(n_components):
        angle = 2 * np.pi * i / n_components
        x = radius * np.cos(angle)
        y = radius * np.sin(angle)
        positions.append([x, y])

    return np.array(positions)

def draw_condensation_graph(condensation_matrix, components, positions=None):
    """Draw the condensation graph"""
    n_components = len(components)

    if positions is None:
        positions = get_component_positions(components)

    fig, ax = plt.subplots(figsize=(12, 10))

    for i in range(n_components):
```

```
        for j in range(n_components):
            if condensation_matrix[i, j] == 1:
                draw_edge(ax, positions[i], positions[j], is_directed=True,
color='red')

    for i, pos in enumerate(positions):
        radius = 0.8
        circle = plt.Circle(pos, radius, fill=True, color='lightgreen',
edgecolor='green')
        ax.add_patch(circle)

        component_label = f"C{i+1}: {components[i]}"
        ax.text(pos[0], pos[1], component_label, horizontalalignment='center',
                verticalalignment='center', fontsize=9, color='black')

    ax.set_aspect('equal')
    margin = 3
    ax.set_xlim(min(positions[:, 0])-margin, max(positions[:, 0])+margin)
    ax.set_ylim(min(positions[:, 1])-margin, max(positions[:, 1])+margin)

    plt.title(f"Condensation Graph - {n_components} components")
    plt.axis('off')
    return fig

def get_vertex_positions(n, n4):
    """Get vertex positions based on n4 value"""
    positions = np.zeros((n, 2))

    if n4 in [8, 9]:
        width, height = 12, 8
        positions[n-1] = [0, 0]

        perimeter_vertices = n - 1
        sides = [0, 0, 0, 0]
        remaining = perimeter_vertices - 4

        for i in range(remaining):
            sides[i % 4] += 1

        vertex_index = 0

        positions[vertex_index] = [-width/2, height/2]
        vertex_index += 1

        for i in range(sides[0]):
            x = -width/2 + (i+1) * width / (sides[0]+1)
            positions[vertex_index] = [x, height/2]
            vertex_index += 1
```

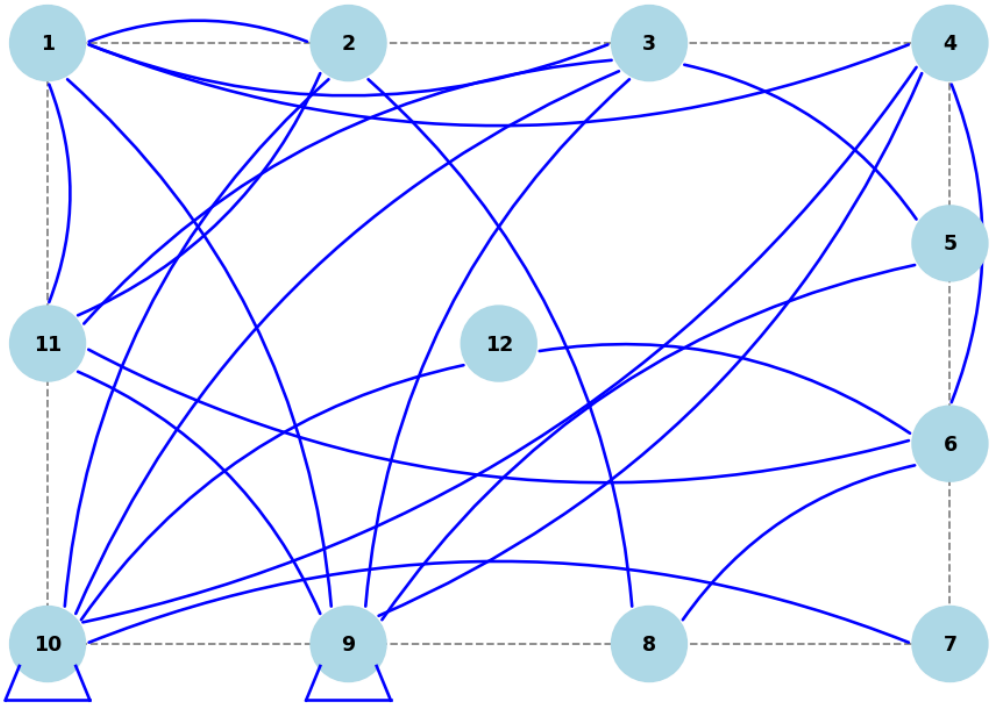
```
        positions[vertex_index] = [width/2, height/2]
        vertex_index += 1

        for i in range(sides[1]):
            y = height/2 - (i+1) * height / (sides[1]+1)
            positions[vertex_index] = [width/2, y]
            vertex_index += 1

        positions[vertex_index] = [width/2, -height/2]
        vertex_index += 1

        for i in range(sides[2]):
            x = width/2 - (i+1) * width / (sides[2]+1)
            positions[vertex_index] = [x, -height/2]
            vertex_index += 1

        positions[vertex_index] = [-width/2, -height/2]
        vertex_index += 1

        for i in range(sides[3]):
            y = -height/2 + (i+1) * height / (sides[3]+1)
            positions[vertex_index] = [-width/2, y]
            vertex_index += 1

    return positions
```

Вивід програми:

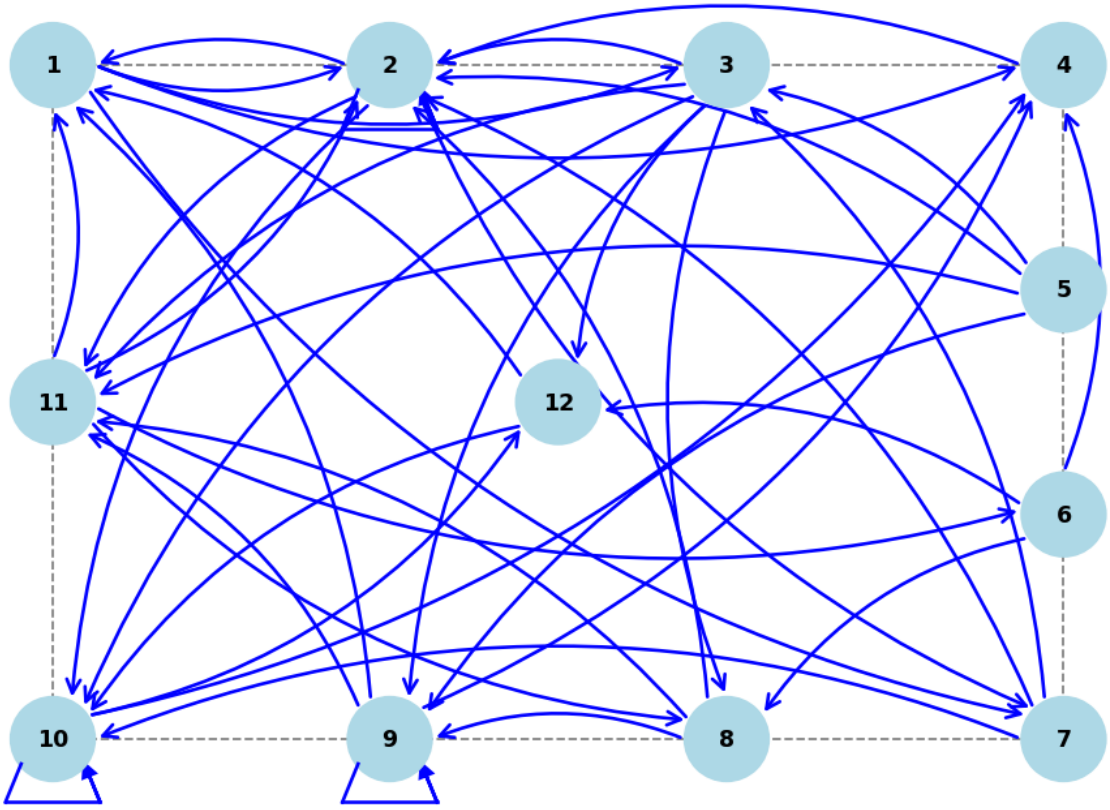
Original - Directed Graph - 12 vertices



Original - Undirected Graph - 12 vertices



Modified - Directed Graph - 12 vertices



Condensation Graph - 2 components

```
=== PART 2: Modified Graph Analysis ===
Using k coefficient: 0.6749999999999999

Modified Directed Graph Adjacency Matrix (12x12):
0 1 1 1 0 0 1 0 0 0 0 0
1 1 0 0 0 0 1 0 0 1 1 0
0 1 0 0 0 0 0 1 1 1 1 1
0 1 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 1 0 1 0
0 0 0 1 0 0 0 1 0 0 0 1
0 1 1 0 0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 1 0 1 0
1 0 0 1 0 0 0 0 1 0 1 0
0 0 0 1 0 0 0 0 0 1 0 1
1 1 0 0 0 1 0 1 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0

Modified Directed Graph:
Vertex | In-degree | Out-degree
     1 |         4 |          4
     2 |         8 |          5
     3 |         3 |          6
     4 |         4 |          1
     5 |         0 |          4
     6 |         1 |          3
     7 |         2 |          3
     8 |         3 |          3
     9 |         4 |          4
    10 |         5 |          3
    11 |         5 |          4
    12 |         3 |          2
```

Reachability Matrix:

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

Strong Connectivity Matrix:

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

Шляхи:

Paths of length 2 (count: 153):

1 – 2 – 1	1 – 2 – 2	1 – 3 – 2	1 – 4 – 2	1 – 7 – 2	1 – 7 – 3	1 – 2 – 7	1 – 3 – 8	1 – 3 – 9	1 – 2 – 10
1 – 3 – 10	1 – 7 – 10	1 – 2 – 11	1 – 3 – 11	1 – 3 – 12	2 – 2 – 1	2 – 11 – 1	2 – 1 – 2	2 – 2 – 2	2 – 7 – 2
2 – 11 – 2	2 – 1 – 3	2 – 7 – 3	2 – 1 – 4	2 – 10 – 4	2 – 11 – 6	2 – 1 – 7	2 – 2 – 7	2 – 11 – 8	2 – 2 – 10
2 – 7 – 10	2 – 10 – 10	2 – 2 – 11	2 – 10 – 12	3 – 2 – 1	3 – 9 – 1	3 – 11 – 1	3 – 12 – 1	3 – 2 – 2	3 – 8 – 2
3 – 11 – 2	3 – 9 – 4	3 – 10 – 4	3 – 11 – 6	3 – 2 – 7	3 – 11 – 8	3 – 8 – 9	3 – 9 – 9	3 – 2 – 10	3 – 10 – 10
3 – 12 – 10	3 – 2 – 11	3 – 8 – 11	3 – 9 – 11	3 – 10 – 12	4 – 2 – 1	4 – 2 – 2	4 – 2 – 7	4 – 2 – 10	4 – 2 – 11
5 – 2 – 1	5 – 9 – 1	5 – 11 – 1	5 – 2 – 2	5 – 3 – 2	5 – 11 – 2	5 – 9 – 4	5 – 11 – 6	5 – 2 – 7	5 – 3 – 8
5 – 11 – 8	5 – 3 – 9	5 – 9 – 9	5 – 2 – 10	5 – 3 – 10	5 – 2 – 11	5 – 3 – 11	5 – 9 – 11	5 – 3 – 12	6 – 12 – 1
6 – 4 – 2	6 – 8 – 2	6 – 8 – 9	6 – 12 – 10	6 – 8 – 11	7 – 2 – 1	7 – 2 – 2	7 – 3 – 2	7 – 10 – 4	7 – 2 – 7
7 – 3 – 8	7 – 3 – 9	7 – 2 – 10	7 – 3 – 10	7 – 10 – 10	7 – 2 – 11	7 – 3 – 11	7 – 3 – 12	7 – 10 – 12	8 – 2 – 1
8 – 9 – 1	8 – 11 – 1	8 – 2 – 2	8 – 11 – 2	8 – 9 – 4	8 – 11 – 6	8 – 2 – 7	8 – 11 – 8	8 – 9 – 9	8 – 2 – 10
8 – 2 – 11	8 – 9 – 11	9 – 9 – 1	9 – 11 – 1	9 – 1 – 2	9 – 4 – 2	9 – 11 – 2	9 – 1 – 3	9 – 1 – 4	9 – 9 – 4
9 – 11 – 6	9 – 1 – 7	9 – 11 – 8	9 – 9 – 9	9 – 9 – 11	10 – 12 – 1	10 – 4 – 2	10 – 10 – 4	10 – 10 – 10	10 – 12 – 10
10 – 10 – 12	11 – 2 – 1	11 – 1 – 2	11 – 2 – 2	11 – 8 – 2	11 – 1 – 3	11 – 1 – 4	11 – 6 – 4	11 – 1 – 7	11 – 2 – 7
11 – 6 – 8	11 – 8 – 9	11 – 2 – 10	11 – 2 – 11	11 – 8 – 11	11 – 6 – 12	12 – 1 – 2	12 – 1 – 3	12 – 1 – 4	12 – 10 – 4
12 – 1 – 7	12 – 10 – 10	12 – 10 – 12							

Paths of length 3 (count: 546):

1 – 2 – 2 – 1	1 – 2 – 11 – 1	1 – 3 – 2 – 1	1 – 3 – 9 – 1	1 – 3 – 11 – 1	1 – 3 – 12 – 1	1 – 4 – 2 – 1
1 – 7 – 2 – 1	1 – 2 – 1 – 2	1 – 2 – 2 – 2	1 – 2 – 7 – 2	1 – 2 – 11 – 2	1 – 3 – 2 – 2	1 – 3 – 8 – 2
1 – 3 – 11 – 2	1 – 4 – 2 – 2	1 – 7 – 2 – 2	1 – 7 – 3 – 2	1 – 2 – 1 – 3	1 – 2 – 7 – 3	1 – 2 – 1 – 4
1 – 2 – 10 – 4	1 – 3 – 9 – 4	1 – 3 – 10 – 4	1 – 7 – 10 – 4	1 – 2 – 11 – 6	1 – 3 – 11 – 6	1 – 2 – 1 – 7
1 – 2 – 2 – 7	1 – 3 – 2 – 7	1 – 4 – 2 – 7	1 – 7 – 2 – 7	1 – 2 – 11 – 8	1 – 3 – 11 – 8	1 – 7 – 3 – 8
1 – 3 – 8 – 9	1 – 3 – 9 – 9	1 – 7 – 3 – 9	1 – 2 – 2 – 10	1 – 2 – 7 – 10	1 – 2 – 10 – 10	1 – 3 – 2 – 10
1 – 3 – 10 – 10	1 – 3 – 12 – 10	1 – 4 – 2 – 10	1 – 7 – 2 – 10	1 – 7 – 3 – 10	1 – 7 – 10 – 10	1 – 2 – 2 – 11
1 – 3 – 2 – 11	1 – 3 – 8 – 11	1 – 3 – 9 – 11	1 – 4 – 2 – 11	1 – 7 – 2 – 11	1 – 7 – 3 – 11	1 – 2 – 10 – 12
1 – 3 – 10 – 12	1 – 7 – 3 – 12	1 – 7 – 10 – 12	2 – 1 – 2 – 1	2 – 2 – 2 – 1	2 – 2 – 11 – 1	2 – 7 – 2 – 1
2 – 10 – 12 – 1	2 – 11 – 2 – 1	2 – 1 – 2 – 2	2 – 1 – 3 – 2	2 – 1 – 4 – 2	2 – 1 – 7 – 2	2 – 2 – 1 – 2
2 – 2 – 2 – 2	2 – 2 – 7 – 2	2 – 2 – 11 – 2	2 – 7 – 2 – 2	2 – 7 – 3 – 2	2 – 10 – 4 – 2	2 – 11 – 1 – 2
2 – 11 – 2 – 2	2 – 11 – 8 – 2	2 – 1 – 7 – 3	2 – 2 – 1 – 3	2 – 2 – 7 – 3	2 – 11 – 1 – 3	2 – 2 – 1 – 4
2 – 2 – 10 – 4	2 – 7 – 10 – 4	2 – 10 – 10 – 4	2 – 11 – 1 – 4	2 – 11 – 6 – 4	2 – 2 – 11 – 6	2 – 1 – 2 – 7
2 – 2 – 1 – 7	2 – 2 – 2 – 7	2 – 7 – 2 – 7	2 – 11 – 1 – 7	2 – 11 – 2 – 7	2 – 1 – 3 – 8	2 – 2 – 11 – 8
2 – 7 – 3 – 8	2 – 11 – 6 – 8	2 – 1 – 3 – 9	2 – 7 – 3 – 9	2 – 11 – 8 – 9	2 – 1 – 2 – 10	2 – 1 – 3 – 10
2 – 1 – 7 – 10	2 – 2 – 2 – 10	2 – 2 – 7 – 10	2 – 2 – 10 – 10	2 – 7 – 2 – 10	2 – 7 – 3 – 10	2 – 7 – 10 – 10
2 – 10 – 10 – 10	2 – 10 – 12 – 10	2 – 11 – 2 – 10	2 – 1 – 2 – 11	2 – 1 – 3 – 11	2 – 2 – 2 – 11	2 – 7 – 2 – 11
2 – 7 – 3 – 11	2 – 11 – 2 – 11	2 – 11 – 8 – 11	2 – 1 – 3 – 12	2 – 2 – 10 – 12	2 – 7 – 3 – 12	2 – 7 – 10 – 12
2 – 10 – 10 – 12	2 – 11 – 6 – 12	3 – 2 – 2 – 1	3 – 2 – 11 – 1	3 – 8 – 2 – 1	3 – 8 – 9 – 1	3 – 8 – 11 – 1
3 – 9 – 9 – 1	3 – 9 – 11 – 1	3 – 10 – 12 – 1	3 – 11 – 2 – 1	3 – 2 – 1 – 2	3 – 2 – 2 – 2	3 – 2 – 7 – 2
3 – 2 – 11 – 2	3 – 8 – 2 – 2	3 – 8 – 11 – 2	3 – 9 – 1 – 2	3 – 9 – 4 – 2	3 – 9 – 11 – 2	3 – 10 – 4 – 2
3 – 11 – 1 – 2	3 – 11 – 2 – 2	3 – 11 – 8 – 2	3 – 12 – 1 – 2	3 – 2 – 1 – 3	3 – 2 – 7 – 3	3 – 9 – 1 – 3
3 – 11 – 1 – 3	3 – 12 – 1 – 3	3 – 2 – 1 – 4	3 – 2 – 10 – 4	3 – 8 – 9 – 4	3 – 9 – 1 – 4	3 – 9 – 9 – 4
3 – 10 – 10 – 4	3 – 11 – 1 – 4	3 – 11 – 6 – 4	3 – 12 – 1 – 4	3 – 12 – 10 – 4	3 – 2 – 11 – 6	3 – 8 – 11 – 6
3 – 9 – 11 – 6	3 – 2 – 1 – 7	3 – 2 – 2 – 7	3 – 8 – 2 – 7	3 – 9 – 1 – 7	3 – 11 – 1 – 7	3 – 11 – 2 – 7
3 – 12 – 1 – 7	3 – 2 – 11 – 8	3 – 8 – 11 – 8	3 – 9 – 11 – 8	3 – 11 – 6 – 8	3 – 8 – 9 – 9	3 – 9 – 9 – 9
3 – 11 – 8 – 9	3 – 2 – 2 – 10	3 – 2 – 7 – 10	3 – 2 – 10 – 10	3 – 8 – 2 – 10	3 – 10 – 10 – 10	3 – 10 – 12 – 10
3 – 11 – 2 – 10	3 – 12 – 10 – 10	3 – 2 – 2 – 11	3 – 8 – 2 – 11	3 – 8 – 9 – 11	3 – 9 – 9 – 11	3 – 11 – 2 – 11
3 – 11 – 8 – 11	3 – 2 – 10 – 12	3 – 10 – 10 – 12	3 – 11 – 6 – 12	3 – 12 – 10 – 12	4 – 2 – 2 – 1	4 – 2 – 11 – 1
4 – 2 – 1 – 2	4 – 2 – 2 – 2	4 – 2 – 7 – 2	4 – 2 – 11 – 2	4 – 2 – 1 – 3	4 – 2 – 7 – 3	4 – 2 – 1 – 4
4 – 2 – 10 – 4	4 – 2 – 11 – 6	4 – 2 – 1 – 7	4 – 2 – 2 – 7	4 – 2 – 11 – 8	4 – 2 – 2 – 10	4 – 2 – 7 – 10
4 – 2 – 10 – 10	4 – 2 – 2 – 11	4 – 2 – 10 – 12	5 – 2 – 2 – 1	5 – 2 – 11 – 1	5 – 3 – 2 – 1	5 – 3 – 9 – 1
5 – 3 – 11 – 1	5 – 3 – 12 – 1	5 – 9 – 9 – 1	5 – 9 – 11 – 1	5 – 11 – 2 – 1	5 – 2 – 1 – 2	5 – 2 – 2 – 2
5 – 2 – 7 – 2	5 – 2 – 11 – 2	5 – 3 – 2 – 2	5 – 3 – 8 – 2	5 – 3 – 11 – 2	5 – 9 – 1 – 2	5 – 9 – 4 – 2
5 – 9 – 11 – 2	5 – 11 – 1 – 2	5 – 11 – 2 – 2	5 – 11 – 8 – 2	5 – 2 – 1 – 3	5 – 2 – 7 – 3	5 – 9 – 1 – 3
5 – 11 – 1 – 3	5 – 2 – 1 – 4	5 – 2 – 10 – 4	5 – 3 – 9 – 4	5 – 3 – 10 – 4	5 – 9 – 1 – 4	5 – 9 – 9 – 4
5 – 11 – 1 – 4	5 – 11 – 6 – 4	5 – 2 – 11 – 6	5 – 3 – 11 – 6	5 – 9 – 11 – 6	5 – 2 – 1 – 7	5 – 2 – 2 – 7
5 – 3 – 2 – 7	5 – 9 – 1 – 7	5 – 11 – 1 – 7	5 – 11 – 2 – 7	5 – 2 – 11 – 8	5 – 3 – 11 – 8	5 – 9 – 11 – 8
5 – 11 – 6 – 8	5 – 3 – 8 – 9	5 – 3 – 9 – 9	5 – 9 – 9 – 9	5 – 11 – 8 – 9	5 – 2 – 2 – 10	5 – 2 – 7 – 10
5 – 2 – 10 – 10	5 – 3 – 2 – 10	5 – 3 – 10 – 10	5 – 3 – 12 – 10	5 – 11 – 2 – 10	5 – 2 – 2 – 11	5 – 3 – 2 – 11
5 – 3 – 8 – 11	5 – 3 – 9 – 11	5 – 9 – 9 – 11	5 – 11 – 2 – 11	5 – 11 – 8 – 11	5 – 2 – 10 – 12	5 – 3 – 10 – 12

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

Висновки:

В ході виконання лабораторної роботи було досліджено алгоритми пошуку та представлення шляхів у направлених графах. Був реалізований алгоритм для знаходження всіх можливих шляхів заданої довжини в направленому графі. Було виявлено, що кількість шляхів зростає зі збільшенням довжини шляху (від 2 до 3), що відповідає теоретичним очікуванням для більшості графів. Лабораторна робота дозволила поглибити розуміння алгоритмів на графах та отримати практичні навички роботи з матрицями суміжності і файловим вводом-виводом у Python.