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
Костеніч Степан Станіславович
номер у списку групи: 17

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 $k = 1.0 - n_3 * 0.01 - n_4 * 0.01 - 0.3$.

   Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується 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 числами в діапазоні [0, 2.0);
   3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.01 - 0.3$, 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 напрямленогоі 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 $k = 1.0 - n_3 * 0.005 - n_4 * 0.005 - 0.27$.

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).

Завдання для конкретного варіанту

Група 43, варіант №17:

$n_1n_2n_3n_4 = 4317$

Кількість вершин n: 11

Розміщення вершин: колом з вершиною в центрі

Текст програми

```
package org.arcctg;

import javax.swing.SwingUtilities;

public class Main {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            GraphAnalyzer app = new GraphAnalyzer();
            app.setVisible(true);
        });
    }
}


package org.arcctg;

import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.util.List;

public class GraphAnalyzer extends JFrame {
    private static final int GROUP_NUMBER = 43;
    private static final int VARIANT_NUMBER = 17;
    private static final int VARIANT_CODE = GROUP_NUMBER * 100 +
VARIANT_NUMBER;

    private static final int N3 = 1;
    private static final int N4 = 7;

    private static final int VERTEX_COUNT = 10 + N3;

    private static final double K1 = 1.0 - N3 * 0.01 - N4 * 0.01 - 0.3;
    private static final double K2 = 1.0 - N3 * 0.005 - N4 * 0.005 -
0.27;

    private static final int WINDOW_WIDTH = 1200;
    private static final int WINDOW_HEIGHT = 800;

    private int[][] directedMatrix1;
    private int[][] undirectedMatrix1;

    private int[][] directedMatrix2;
```

```
    private int[][] undirectedMatrix2;

    private int[][] reachabilityMatrix;
    private int[][] strongConnectivityMatrix;
    private List<Set<Integer>> stronglyConnectedComponents;
    private int[][] condensationMatrix;

    public GraphAnalyzer() {
        initializeWindow();
        generateMatrices();
        analyzeGraphs();
        setupUI();
    }

    private void initializeWindow() {
        setTitle("Graph Analyzer - Lab 4");
        setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    private void setupUI() {
        JTabbedPane tabbedPane = new JTabbedPane();

        addPart1GraphsTab(tabbedPane);
        JTextArea textOutputArea = addPart1AnalysisTab(tabbedPane);
        addPart3GraphsTab(tabbedPane);
        addPart3AnalysisTab(tabbedPane);
        addCondensationGraphTab(tabbedPane);

        add(tabbedPane);

        textOutputArea.setText(getFirstAnalysisText());
    }

    private void addPart1GraphsTab(JTabbedPane tabbedPane) {
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(new GraphPanel(directedMatrix1, "Part 1
Visualization", false), BorderLayout.CENTER);
        tabbedPane.addTab("Part 1 Graphs", panel);
    }

    private JTextArea addPart1AnalysisTab(JTabbedPane tabbedPane) {
        JPanel panel = new JPanel(new BorderLayout());
        JTextArea textArea = new JTextArea(20, 40);
        textArea.setEditable(false);
```

```
        JScrollPane scrollPane = new JScrollPane(textArea);
        panel.add(scrollPane, BorderLayout.CENTER);
        tabbedPane.addTab("Part 1-2 Analysis", panel);
        return textArea;
    }

    private void addPart3GraphsTab(JTabbedPane tabbedPane) {
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(new GraphPanel(directedMatrix2, "Part 3
Visualization", false), BorderLayout.CENTER);
        tabbedPane.addTab("Part 3 Graphs", panel);
    }

    private void addPart3AnalysisTab(JTabbedPane tabbedPane) {
        JPanel panel = new JPanel(new BorderLayout());
        JTextArea textArea = new JTextArea(20, 40);
        textArea.setEditable(false);
        textArea.append(getSecondAnalysisText());
        JScrollPane scrollPane = new JScrollPane(textArea);
        panel.add(scrollPane, BorderLayout.CENTER);
        tabbedPane.addTab("Part 3-4 Analysis", panel);
    }

    private void addCondensationGraphTab(JTabbedPane tabbedPane) {
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(
            new CondensationGraphPanel(condensationMatrix,
stronglyConnectedComponents),
            BorderLayout.CENTER);
        tabbedPane.addTab("Condensation Graph", panel);
    }

    private void generateMatrices() {
        generatePart1and2Matrices();
        generatePart3and4Matrices();
        printDebugMatrices();
    }

    private void generatePart1and2Matrices() {
        directedMatrix1 = generateMatrix(K1);
        undirectedMatrix1 = generateUndirectedMatrix(directedMatrix1);
    }

    private void generatePart3and4Matrices() {
        directedMatrix2 = generateMatrix(K2);
        undirectedMatrix2 = generateUndirectedMatrix(directedMatrix2);
```

```
    }

    private void printDebugMatrices() {
        System.out.println("Part 1-2 Directed Graph Adjacency Matrix
(K=" + K1 + "):");
        printMatrix(directedMatrix1);

        System.out.println("\nPart 1-2 Undirected Graph Adjacency
Matrix:");
        printMatrix(undirectedMatrix1);

        System.out.println("\nPart 3-4 Directed Graph Adjacency Matrix
(K=" + K2 + "):");
        printMatrix(directedMatrix2);

        System.out.println("\nPart 3-4 Undirected Graph Adjacency
Matrix:");
        printMatrix(undirectedMatrix2);
    }

    private int[][] generateMatrix(double k) {
        Random random = new Random(VARIANT_CODE);
        int[][] resultMatrix = new int[VERTEX_COUNT][VERTEX_COUNT];

        for (int i = 0; i < VERTEX_COUNT; i++) {
            for (int j = 0; j < VERTEX_COUNT; j++) {
                double value = random.nextDouble(0.0, 2.0) * k;
                resultMatrix[i][j] = value >= 1.0 ? 1 : 0;
            }
        }

        return resultMatrix;
    }

    private int[][] generateUndirectedMatrix(int[][] directedMatrix) {
        int[][] resultMatrix = new int[VERTEX_COUNT][VERTEX_COUNT];

        for (int i = 0; i < VERTEX_COUNT; i++) {
            for (int j = 0; j < VERTEX_COUNT; j++) {
                if (directedMatrix[i][j] == 1) {
                    resultMatrix[i][j] = 1;
                    resultMatrix[j][i] = 1;
                }
            }
        }
```

```
        return resultMatrix;
    }

    private void analyzeGraphs() {
        calculateReachabilityMatrix();
        calculateStrongConnectivityMatrix();
        findStronglyConnectedComponents();
        buildCondensationGraph();
    }

    private void calculateReachabilityMatrix() {
        reachabilityMatrix = new int[VERTEX_COUNT][VERTEX_COUNT];

        initializeReachabilityMatrix();

        applyWarshallAlgorithm();
    }

    private void initializeReachabilityMatrix() {
        for (int i = 0; i < VERTEX_COUNT; i++) {
            System.arraycopy(directedMatrix2[i], 0,
reachabilityMatrix[i], 0, VERTEX_COUNT);
            reachabilityMatrix[i][i] = 1;
        }
    }

    private void applyWarshallAlgorithm() {
        for (int k = 0; k < VERTEX_COUNT; k++) {
            for (int i = 0; i < VERTEX_COUNT; i++) {
                for (int j = 0; j < VERTEX_COUNT; j++) {
                    if (reachabilityMatrix[i][k] == 1 &&
reachabilityMatrix[k][j] == 1) {
                        reachabilityMatrix[i][j] = 1;
                    }
                }
            }
        }
    }

    private void calculateStrongConnectivityMatrix() {
        strongConnectivityMatrix = new int[VERTEX_COUNT][VERTEX_COUNT];

        for (int i = 0; i < VERTEX_COUNT; i++) {
            for (int j = 0; j < VERTEX_COUNT; j++) {
                if (reachabilityMatrix[i][j] == 1 &&
reachabilityMatrix[j][i] == 1) {
```

```
                    strongConnectivityMatrix[i][j] = 1;
                }
            }
        }
    }

    private void findStronglyConnectedComponents() {
        stronglyConnectedComponents = new ArrayList<>();
        boolean[] visited = new boolean[VERTEX_COUNT];

        for (int i = 0; i < VERTEX_COUNT; i++) {
            if (!visited[i]) {
                Set<Integer> component = new HashSet<>();
                component.add(i);
                visited[i] = true;

                for (int j = 0; j < VERTEX_COUNT; j++) {
                    if (j != i && !visited[j] &&
strongConnectivityMatrix[i][j] == 1) {
                        component.add(j);
                        visited[j] = true;
                    }
                }

                stronglyConnectedComponents.add(component);
            }
        }
    }

    private void buildCondensationGraph() {
        int componentCount = stronglyConnectedComponents.size();
        condensationMatrix = new int[componentCount][componentCount];

        for (int i = 0; i < componentCount; i++) {
            for (int j = 0; j < componentCount; j++) {
                if (i != j) {
                    condensationMatrix[i][j] =
hasEdgeBetweenComponents(i, j);
                }
            }
        }
    }

    private int hasEdgeBetweenComponents(int fromComponentIndex, int
toComponentIndex) {
        Set<Integer> fromComponent =
```

```
stronglyConnectedComponents.get(fromComponentIndex);
        Set<Integer> toComponent =
stronglyConnectedComponents.get(toComponentIndex);

        for (int vertexFrom : fromComponent) {
            if (hasEdgeFromVertexToComponent(vertexFrom, toComponent)) {
                return 1;
            }
        }
        return 0;
    }

    private boolean hasEdgeFromVertexToComponent(int vertexFrom,
Set<Integer> toComponent) {
        for (int vertexTo : toComponent) {
            if (directedMatrix2[vertexFrom][vertexTo] == 1) {
                return true;
            }
        }
        return false;
    }

    private String getFirstAnalysisText() {
        StringBuilder sb = new StringBuilder();

        sb.append("PART 1-2 ANALYSIS (K = ").append(K1).append(")\n\n");

        int[] undirectedDegrees =
calculateOutDegrees(undirectedMatrix1);
        int[] inDegrees = calculateInDegrees(directedMatrix1);
        int[] outDegrees = calculateOutDegrees(directedMatrix1);

        appendUndirectedDegreeInfo(sb, undirectedDegrees);
        appendDirectedDegreeInfo(sb, inDegrees, outDegrees);

        appendRegularityInfo(sb, undirectedDegrees, inDegrees,
outDegrees);

        appendHangingVerticesInfo(sb, undirectedDegrees, "undirected
graph");
        appendIsolatedVerticesInfo(sb, undirectedDegrees, "undirected
graph");

        sb.append("\n");
        appendDirectedHangingVerticesInfo(sb, inDegrees, outDegrees);
        appendDirectedIsolatedVerticesInfo(sb, inDegrees, outDegrees);
```

```
        return sb.toString();
    }

    private void appendUndirectedDegreeInfo(StringBuilder sb, int[]
degrees) {
        sb.append("Undirected Graph Vertex Degrees:\n");
        for (int i = 0; i < VERTEX_COUNT; i++) {
            sb.append("Vertex ").append(i + 1).append(":
").append(degrees[i]).append("\n");
        }
        sb.append("\n");
    }

    private void appendDirectedDegreeInfo(StringBuilder sb, int[]
inDegrees, int[] outDegrees) {
        sb.append("Directed Graph In-Degrees and Out-Degrees:\n");
        for (int i = 0; i < VERTEX_COUNT; i++) {
            sb.append("Vertex ").append(i + 1).append(": in-degree =
").append(inDegrees[i])
                    .append(", out-degree =
").append(outDegrees[i]).append("\n");
        }
        sb.append("\n");
    }

    private void appendRegularityInfo(StringBuilder sb, int[]
undirectedDegrees, int[] inDegrees, int[] outDegrees) {
        sb.append("Regularity Check:\n");
        boolean isUndirectedRegular = isRegular(undirectedDegrees);
        boolean isDirectedRegular = isRegular(inDegrees) &&
isRegular(outDegrees);

        if (isUndirectedRegular) {
            sb.append("Undirected graph is regular with degree
").append(undirectedDegrees[0]).append("\n");
        } else {
            sb.append("Undirected graph is not regular\n");
        }

        if (isDirectedRegular) {
            sb.append("Directed graph is regular with in-degree
").append(inDegrees[0])
                    .append(" and out-degree
").append(outDegrees[0]).append("\n");
        } else {
```

```
            sb.append("Directed graph is not regular\n");
        }
        sb.append("\n");
    }

    private void appendVerticesList(StringBuilder sb, List<Integer>
vertices) {
        if (vertices.isEmpty()) {
            sb.append("none\n");
            return;
        }

        for (int vertex : vertices) {
            sb.append(vertex + 1).append(" ");
        }
        sb.append("\n");
    }

    private void appendHangingVerticesInfo(StringBuilder sb, int[]
degrees, String graphType) {
        sb.append("Hanging vertices in ").append(graphType).append(":
");
        List<Integer> hangingVertices = findHangingVertices(degrees);
        appendVerticesList(sb, hangingVertices);
    }

    private void appendIsolatedVerticesInfo(StringBuilder sb, int[]
degrees, String graphType) {
        sb.append("Isolated vertices in ").append(graphType).append(":
");
        List<Integer> isolatedVertices = findIsolatedVertices(degrees);
        appendVerticesList(sb, isolatedVertices);
    }

    private void appendDirectedHangingVerticesInfo(StringBuilder sb,
int[] inDegrees, int[] outDegrees) {
        sb.append("Hanging vertices in directed graph: ");
        List<Integer> directedHangingVertices =
findDirectedHangingVertices(inDegrees, outDegrees);
        appendVerticesList(sb, directedHangingVertices);
    }

    private void appendDirectedIsolatedVerticesInfo(StringBuilder sb,
int[] inDegrees, int[] outDegrees) {
        sb.append("Isolated vertices in directed graph: ");
        List<Integer> directedIsolatedVertices =
```

```
findDirectedIsolatedVertices(inDegrees, outDegrees);
        appendVerticesList(sb, directedIsolatedVertices);
    }

    private String getSecondAnalysisText() {
        StringBuilder sb = new StringBuilder();
        sb.append("PART 3-4 ANALYSIS (K = ").append(K2).append(")\n\n");

        appendDegreesInfo(sb);
        appendPathsInfo(sb);
        appendMatrixInfo(sb, "Reachability Matrix", reachabilityMatrix);
        appendMatrixInfo(sb, "Strong Connectivity Matrix",
strongConnectivityMatrix);
        appendComponentsInfo(sb);

        return sb.toString();
    }

    private void appendDegreesInfo(StringBuilder sb) {
        sb.append("Directed Graph In-Degrees and Out-Degrees:\n");
        int[] inDegrees = calculateInDegrees(directedMatrix2);
        int[] outDegrees = calculateOutDegrees(directedMatrix2);

        for (int i = 0; i < VERTEX_COUNT; i++) {
            sb.append("Vertex ").append(i + 1)
                .append(": in-degree = ").append(inDegrees[i])
                .append(", out-degree = ").append(outDegrees[i])
                .append("\n");
        }
        sb.append("\n");
    }

    private void appendPathsInfo(StringBuilder sb) {
        int[][] paths2 = multiplyMatrices(directedMatrix2,
directedMatrix2);
        appendPathsOfLength(sb, directedMatrix2, 2);

        int[][] paths3 = multiplyMatrices(paths2, directedMatrix2);
        appendPathsOfLength(sb, directedMatrix2, 3);
    }

    private void appendPathsOfLength(StringBuilder sb, int[][] matrix,
int length) {
        sb.append("Paths of length ").append(length).append(":\n");
        List<String> pathsList = findPaths(matrix, length);
```

```
        for (String path : pathsList) {
            sb.append(path).append("\n");
        }
        sb.append("\n");
    }

    private void appendMatrixInfo(StringBuilder sb, String title,
int[][] matrix) {
        sb.append(title).append(":\n");

        for (int i = 0; i < VERTEX_COUNT; i++) {
            for (int j = 0; j < VERTEX_COUNT; j++) {
                sb.append(matrix[i][j]).append(" ");
            }
            sb.append("\n");
        }
        sb.append("\n");
    }

    private void appendComponentsInfo(StringBuilder sb) {
        sb.append("Strongly Connected Components:\n");

        for (int i = 0; i < stronglyConnectedComponents.size(); i++) {
            sb.append("Component ").append(i + 1).append(": ");
            Set<Integer> component = stronglyConnectedComponents.get(i);

            for (int vertex : component) {
                sb.append(vertex + 1).append(" ");
            }
            sb.append("\n");
        }
    }

    private int[] calculateInDegrees(int[][] matrix) {
        int[] inDegrees = new int[VERTEX_COUNT];

        for (int j = 0; j < VERTEX_COUNT; j++) {
            int inDegree = 0;
            for (int i = 0; i < VERTEX_COUNT; i++) {
                inDegree += matrix[i][j];
            }
            inDegrees[j] = inDegree;
        }

        return inDegrees;
    }
```

```
private int[] calculateOutDegrees(int[][] matrix) {
    int[] outDegrees = new int[VERTEX_COUNT];

    for (int i = 0; i < VERTEX_COUNT; i++) {
        int outDegree = 0;
        for (int j = 0; j < VERTEX_COUNT; j++) {
            outDegree += matrix[i][j];
        }
        outDegrees[i] = outDegree;
    }

    return outDegrees;
}

private boolean isRegular(int[] degrees) {
    if (degrees.length == 0) return true;

    int firstDegree = degrees[0];
    for (int i = 1; i < degrees.length; i++) {
        if (degrees[i] != firstDegree) {
            return false;
        }
    }

    return true;
}

private List<Integer> findHangingVertices(int[] degrees) {
    List<Integer> hangingVertices = new ArrayList<>();

    for (int i = 0; i < degrees.length; i++) {
        if (degrees[i] == 1) {
            hangingVertices.add(i);
        }
    }

    return hangingVertices;
}

private List<Integer> findIsolatedVertices(int[] degrees) {
    List<Integer> isolatedVertices = new ArrayList<>();

    for (int i = 0; i < degrees.length; i++) {
        if (degrees[i] == 0) {
            isolatedVertices.add(i);
```

```
            }
        }

        return isolatedVertices;
    }

    private List<Integer> findDirectedHangingVertices(int[] inDegrees,
int[] outDegrees) {
        List<Integer> hangingVertices = new ArrayList<>();

        for (int i = 0; i < VERTEX_COUNT; i++) {
            if ((inDegrees[i] == 0 && outDegrees[i] == 1) ||
(inDegrees[i] == 1 && outDegrees[i] == 0)) {
                hangingVertices.add(i);
            }
        }

        return hangingVertices;
    }

    private List<Integer> findDirectedIsolatedVertices(int[] inDegrees,
int[] outDegrees) {
        List<Integer> isolatedVertices = new ArrayList<>();

        for (int i = 0; i < VERTEX_COUNT; i++) {
            if (inDegrees[i] == 0 && outDegrees[i] == 0) {
                isolatedVertices.add(i);
            }
        }

        return isolatedVertices;
    }

    private int[][] multiplyMatrices(int[][] a, int[][] b) {
        int size = a.length;
        int[][] result = new int[size][size];

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                for (int k = 0; k < size; k++) {
                    result[i][j] += a[i][k] * b[k][j];
                }
            }
        }

        return result;
```

```
    }

    private List<String> findPaths(int[][] matrix, int length) {
        if (length == 2) {
            return findPathsOfLength2(matrix);
        } else if (length == 3) {
            return findPathsOfLength3(matrix);
        }
        return new ArrayList<>();
    }

    private List<String> findPathsOfLength2(int[][] matrix) {
        List<String> paths = new ArrayList<>();

        for (int i = 0; i < VERTEX_COUNT; i++) {
            for (int j = 0; j < VERTEX_COUNT; j++) {
                findIntermediateVertices(matrix, i, j, paths);
            }
        }

        return paths;
    }

    private void findIntermediateVertices(int[][] matrix, int start, int
end, List<String> paths) {
        for (int intermediate = 0; intermediate < VERTEX_COUNT;
intermediate++) {
            if (matrix[start][intermediate] == 1 &&
matrix[intermediate][end] == 1) {
                paths.add(formatPath(start, intermediate, end));
            }
        }
    }

    private List<String> findPathsOfLength3(int[][] matrix) {
        List<String> paths = new ArrayList<>();

        for (int i = 0; i < VERTEX_COUNT; i++) {
            for (int j = 0; j < VERTEX_COUNT; j++) {
                findTwoIntermediateVertices(matrix, i, j, paths);
            }
        }

        return paths;
    }
```

```
    private void findTwoIntermediateVertices(int[][] matrix, int start,
int end, List<String> paths) {
        for (int first = 0; first < VERTEX_COUNT; first++) {
            for (int second = 0; second < VERTEX_COUNT; second++) {
                if (matrix[start][first] == 1 && matrix[first][second]
== 1 && matrix[second][end] == 1) {
                    paths.add(formatPath(start, first, second, end));
                }
            }
        }
    }

    private String formatPath(int... vertices) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < vertices.length; i++) {
            sb.append(vertices[i] + 1);
            if (i < vertices.length - 1) {
                sb.append(" - ");
            }
        }
        return sb.toString();
    }

    private void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int cell : row) {
                System.out.print(cell + " ");
            }
            System.out.println();
        }
    }
}


package org.arcctg;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.geom.Path2D;
import java.awt.geom.QuadCurve2D;
```

```
import javax.swing.JPanel;

public class TemplatePanel extends JPanel {

    protected static final int VERTEX_RADIUS = 20;
    protected static final double ARROW_SIZE = 10.0;
    protected static final int SELF_LOOP_OFFSET = 40;
    private static final int CURVE_CONTROL_OFFSET = 50;
    private int vertexCount;

    protected void drawGraph(Graphics2D g2d, int[][] matrix, Point
center, Dimension dimension, boolean isDirected, boolean isCondensation)
{
        vertexCount = matrix.length;
        int width = dimension.width;
        int height = dimension.height;

        int centerX = center.x + width / 2;
        int centerY = center.y + height / 2;

        int radius = Math.min(width, height) / 3;

        Point[] vertexPositions = calculateVertexPositions(centerX,
centerY, radius);

        drawEdges(g2d, matrix, vertexPositions, isDirected);
        drawVertices(g2d, vertexPositions, isCondensation);
    }

    private Point[] calculateVertexPositions(int centerX, int centerY,
int radius) {
        Point[] positions = new Point[vertexCount];

        positions[0] = new Point(centerX, centerY);

        double angleStep = 2 * Math.PI / (vertexCount - 1);
        for (int i = 1; i < vertexCount; i++) {
            int vx = centerX + (int) (radius * Math.cos((i - 1) *
angleStep));
            int vy = centerY + (int) (radius * Math.sin((i - 1) *
angleStep));
            positions[i] = new Point(vx, vy);
        }

        return positions;
    }
```

```
    private void drawEdges(Graphics2D g2d, int[][] matrix, Point[]
vertexPositions, boolean isDirected) {
        boolean[][] bidirectionalEdges = findBidirectionalEdges(matrix,
isDirected);

        for (int i = 0; i < vertexCount; i++) {
            for (int j = 0; j < vertexCount; j++) {
                if (matrix[i][j] == 1) {
                    boolean isBidirectional = bidirectionalEdges[i][j];
                    drawEdge(g2d, vertexPositions[i],
vertexPositions[j], i, j, vertexPositions[0],
                        isDirected,
                        isBidirectional);
                }
            }
        }
    }

    private boolean[][] findBidirectionalEdges(int[][] matrix, boolean
isDirected) {
        boolean[][] bidirectionalEdges = new
boolean[vertexCount][vertexCount];

        if (isDirected) {
            for (int i = 0; i < vertexCount; i++) {
                for (int j = 0; j < vertexCount; j++) {
                    if (matrix[i][j] == 1 && matrix[j][i] == 1 && i !=
j) {
                        bidirectionalEdges[i][j] = true;
                        bidirectionalEdges[j][i] = true;
                    }
                }
            }
        }

        return bidirectionalEdges;
    }

    private void drawVertices(Graphics2D g2d, Point[] vertexPositions,
boolean isCondensation) {
        for (int i = 0; i < vertexCount; i++) {
            if (isCondensation) {
                drawComponentVertex(g2d, vertexPositions[i], i + 1);
            } else {
                drawVertex(g2d, vertexPositions[i], i + 1);
```

```
            }
        }
    }

    private void drawVertex(Graphics2D g2d, Point position, int index) {
        g2d.setColor(Color.WHITE);
        Ellipse2D.Double circle = new Ellipse2D.Double(
            position.x - VERTEX_RADIUS,
            position.y - VERTEX_RADIUS,
            2.0 * VERTEX_RADIUS,
            2.0 * VERTEX_RADIUS
        );
        g2d.fill(circle);

        g2d.setColor(Color.BLACK);
        g2d.draw(circle);

        drawVertexLabel(g2d, position, index);
    }

    private void drawVertexLabel(Graphics2D g2d, Point position, int
index) {
        g2d.setFont(new Font("Arial", Font.BOLD, 14));
        String label = String.valueOf(index);
        FontMetrics metrics = g2d.getFontMetrics();
        int labelWidth = metrics.stringWidth(label);
        int labelHeight = metrics.getHeight();

        g2d.drawString(
            label,
            position.x - labelWidth / 2,
            position.y + labelHeight / 4
        );
    }

    private void drawEdge(Graphics2D g2d, Point from, Point to, int
fromIndex, int toIndex,
        Point centerPoint, boolean isDirected, boolean isBidirectional)
{
        boolean throughCenter = fromIndex != 0 && toIndex != 0 &&
linePassesThroughCenter(from, to, centerPoint);

        if (fromIndex == toIndex) {
            drawSelfLoop(g2d, from, fromIndex);
        } else if (isBidirectional || throughCenter) {
            drawCurvedEdge(g2d, from, to, centerPoint, isDirected,
```

```
isBidirectional);
        } else {
            drawStraightEdge(g2d, from, to, isDirected);
        }
    }

    private boolean linePassesThroughCenter(Point from, Point to, Point
center) {
        double distance = distanceFromPointToLine(center, from, to);
        return distance < 2 * VERTEX_RADIUS;
    }

    private double distanceFromPointToLine(Point point, Point lineStart,
Point lineEnd) {
        double numerator = Math.abs(
            (lineEnd.y - lineStart.y) * point.x -
                (lineEnd.x - lineStart.x) * point.y +
                lineEnd.x * lineStart.y -
                lineEnd.y * lineStart.x
        );

        double denominator = Math.sqrt(
            Math.pow(lineEnd.y - lineStart.y, 2) +
                Math.pow(lineEnd.x - lineStart.x, 2)
        );

        return numerator / denominator;
    }

    private void drawCurvedEdge(Graphics2D g2d, Point from, Point to,
Point center, boolean isDirected,
        boolean isBidirectional) {
        double dx = to.x - from.x;
        double dy = to.y - from.y;
        double length = Math.sqrt(dx * dx + dy * dy);

        double perpX = -dy / length;
        double perpY = dx / length;

        double dotProduct = (center.x - from.x) * perpX + (center.y -
from.y) * perpY;
        double sign = (dotProduct > 0) ? -1 : 1;
        int curveOffset = isBidirectional ? 12 : CURVE_CONTROL_OFFSET;

        double midX = (from.x + to.x) / 2.0;
        double midY = (from.y + to.y) / 2.0;
```

```
        int controlX = (int) (midX + sign * perpX * curveOffset);
        int controlY = (int) (midY + sign * perpY * curveOffset);

        double startAngle = Math.atan2(controlY - from.y, controlX -
from.x);
        int startX = from.x + (int) (VERTEX_RADIUS *
Math.cos(startAngle));
        int startY = from.y + (int) (VERTEX_RADIUS *
Math.sin(startAngle));

        double endAngle = Math.atan2(controlY - to.y, controlX - to.x);
        int endX = to.x + (int) (VERTEX_RADIUS * Math.cos(endAngle));
        int endY = to.y + (int) (VERTEX_RADIUS * Math.sin(endAngle));

        g2d.draw(new QuadCurve2D.Double(startX, startY, controlX,
controlY, endX, endY));

        if (isDirected) {
            double t = 0.95;
            double curvePointX = (1-t)*(1-t)*startX + 2*(1-t)*t*controlX
+ t*t*endX;
            double curvePointY = (1-t)*(1-t)*startY + 2*(1-t)*t*controlY
+ t*t*endY;

            drawArrow(g2d, (int)curvePointX, (int)curvePointY, endX,
endY);
        }
    }

    private void drawSelfLoop(Graphics2D g2d, Point vertex, int
vertexIndex) {
        double angleOffset = (vertexIndex == 0) ? Math.PI / 4 :
getAngleForVertex(vertexIndex);

        int loopSize = VERTEX_RADIUS * 3;
        int offsetX = (int) (SELF_LOOP_OFFSET * Math.cos(angleOffset));
        int offsetY = (int) (SELF_LOOP_OFFSET * Math.sin(angleOffset));

        int loopX = vertex.x + offsetX - loopSize / 2;
        int loopY = vertex.y + offsetY - loopSize / 2;

        g2d.drawOval(loopX, loopY, loopSize, loopSize);
    }

    private double getAngleForVertex(int vertexIndex) {
        return vertexIndex == 0 ?
```

```
            Math.PI / 4 :
            2 * Math.PI * (vertexIndex - 1) / (vertexCount - 1);
    }

    protected void drawComponentVertex(Graphics2D g2d, Point position,
int componentIndex) {
        g2d.setColor(new Color(173, 216, 230)); // Light blue
        Ellipse2D.Double circle = new Ellipse2D.Double(
            position.x - VERTEX_RADIUS * 1.0,
            position.y - VERTEX_RADIUS * 1.0,
            2.0 * VERTEX_RADIUS,
            2.0 * VERTEX_RADIUS
        );
        g2d.fill(circle);

        g2d.setColor(Color.BLACK);
        g2d.draw(circle);

        g2d.setFont(new Font("Arial", Font.BOLD, 14));
        String label = "C" + componentIndex;
        FontMetrics metrics = g2d.getFontMetrics();
        int labelWidth = metrics.stringWidth(label);
        int labelHeight = metrics.getHeight();

        g2d.drawString(
            label,
            position.x - labelWidth / 2,
            position.y + labelHeight / 4
        );
    }

    protected void drawStraightEdge(Graphics2D g2d, Point from, Point
to, boolean isDirected) {
        double dx = to.x - from.x;
        double dy = to.y - from.y;
        double length = Math.sqrt(dx * dx + dy * dy);

        double nx = dx / length;
        double ny = dy / length;

        int startX = from.x + (int) (nx * VERTEX_RADIUS);
        int startY = from.y + (int) (ny * VERTEX_RADIUS);
        int endX = to.x - (int) (nx * VERTEX_RADIUS);
        int endY = to.y - (int) (ny * VERTEX_RADIUS);

        g2d.setColor(Color.BLACK);
```

```
        g2d.draw(new Line2D.Double(startX, startY, endX, endY));

        if (isDirected) {
            drawArrow(g2d, startX, startY, endX, endY);
        }
    }

    protected void drawArrow(Graphics2D g2d, int x1, int y1, int x2, int
y2) {
        double dx = x2 - x1;
        double dy = y2 - y1;
        double angle = Math.atan2(dy, dx);

        Path2D.Double path = new Path2D.Double();
        path.moveTo(x2, y2);
        path.lineTo(x2 - ARROW_SIZE * Math.cos(angle - Math.PI/6),
            y2 - ARROW_SIZE * Math.sin(angle - Math.PI/6));
        path.lineTo(x2 - ARROW_SIZE * Math.cos(angle + Math.PI/6),
            y2 - ARROW_SIZE * Math.sin(angle + Math.PI/6));
        path.closePath();

        g2d.fill(path);
    }
}


package org.arcctg;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;

public class GraphPanel extends TemplatePanel {

    private final int[][] directed;
    private final String title;
    private final boolean isCondensation;

    public GraphPanel(int[][] directed, String title, boolean
isCondensation) {
        this.directed = directed;
```

```
        this.title = title;
        this.isCondensation = isCondensation;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        configureGraphics(g2d);

        int panelWidth = getWidth();
        Dimension dimension = new Dimension(panelWidth / 2,
getHeight());

        drawGraph(g2d, directed, new Point(0, 0), dimension, true,
isCondensation);
        drawGraph(g2d, directed, new Point(panelWidth / 2, 0),
dimension, false, isCondensation);

        drawTitles(g2d, panelWidth);
    }

    private void configureGraphics(Graphics2D g2d) {
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
        g2d.setStroke(new BasicStroke(1.5f));
    }

    private void drawTitles(Graphics2D g2d, int panelWidth) {
        g2d.setColor(Color.BLACK);
        g2d.setFont(new Font("Arial", Font.BOLD, 20));
        g2d.drawString(title + " - Directed Graph", 100, 30);
        g2d.drawString(title + " - Undirected Graph", panelWidth / 2 +
100, 30);
    }

}



package org.arcctg;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
```

```
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;
import java.util.List;
import java.util.Set;

public class CondensationGraphPanel extends TemplatePanel {

    private final int[][] condensationMatrix;
    private final List<Set<Integer>> stronglyConnectedComponents;

    public CondensationGraphPanel(int[][] condensationMatrix,
        List<Set<Integer>> stronglyConnectedComponents) {
        this.condensationMatrix = condensationMatrix;
        this.stronglyConnectedComponents = stronglyConnectedComponents;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = configureGraphics(g);

        drawTitle(g2d);
        drawCondensationGraph(g2d);
        drawComponentContents(g2d);
    }

    private Graphics2D configureGraphics(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
        g2d.setStroke(new BasicStroke(1.5f));
        return g2d;
    }

    private void drawTitle(Graphics2D g2d) {
        g2d.setColor(Color.BLACK);
        g2d.setFont(new Font("Arial", Font.BOLD, 20));
        g2d.drawString("Condensation Graph", getWidth() / 2 - 100, 30);
    }

    private void drawCondensationGraph(Graphics2D g2d) {
```

```
        Dimension graphSize = new Dimension(getWidth() - 100,
getHeight() - 70);
        Point graphPosition = new Point(0, 50);
        drawGraph(g2d, condensationMatrix, graphPosition, graphSize,
true, true);
    }

    private void drawComponentContents(Graphics2D g2d) {
        g2d.setFont(new Font("Arial", Font.PLAIN, 14));
        int y = getHeight() - 150;

        g2d.drawString("Component Contents:", 50, y);
        y += 20;

        for (int i = 0; i < stronglyConnectedComponents.size(); i++) {
            StringBuilder sb = new StringBuilder();
            sb.append("C").append(i + 1).append(": ");

            Set<Integer> component = stronglyConnectedComponents.get(i);
            for (int vertex : component) {
                sb.append(vertex + 1).append(" ");
            }

            g2d.drawString(sb.toString(), 50, y);
            y += 20;
        }
    }
}
```

За п. 1 завдання:

Матриця суміжності напрамленого графа k1:

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

Матриця суміжності ненапрамленого графа k1:

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

За п. 2 завдання:

Undirected Graph Vertex Degrees:
Vertex 1: 5
Vertex 2: 4
Vertex 3: 5
Vertex 4: 6
Vertex 5: 5
Vertex 6: 7
Vertex 7: 3
Vertex 8: 4
Vertex 9: 3
Vertex 10: 6
Vertex 11: 3

Directed Graph In-Degrees and Out-Degrees:
Vertex 1: in-degree = 5, out-degree = 1
Vertex 2: in-degree = 2, out-degree = 2
Vertex 3: in-degree = 0, out-degree = 5
Vertex 4: in-degree = 6, out-degree = 1
Vertex 5: in-degree = 1, out-degree = 4
Vertex 6: in-degree = 4, out-degree = 3
Vertex 7: in-degree = 1, out-degree = 2
Vertex 8: in-degree = 2, out-degree = 2
Vertex 9: in-degree = 3, out-degree = 2
Vertex 10: in-degree = 2, out-degree = 4
Vertex 11: in-degree = 2, out-degree = 2

Regularity Check:
Undirected graph is not regular
Directed graph is not regular

Hanging vertices in undirected graph: none
Isolated vertices in undirected graph: none

Hanging vertices in directed graph: none
Isolated vertices in directed graph: none

За п. 3 завдання:

Матриця суміжності напрамленого другого орграфа k2:

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

Матриця суміжності ненапрамленого другого орграфа k2:

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

За п. 4 завдання:

Directed Graph In-Degrees and Out-Degrees:
Vertex 1: in-degree = 5, out-degree = 2
Vertex 2: in-degree = 2, out-degree = 3
Vertex 3: in-degree = 0, out-degree = 6
Vertex 4: in-degree = 6, out-degree = 2
Vertex 5: in-degree = 3, out-degree = 5
Vertex 6: in-degree = 4, out-degree = 3
Vertex 7: in-degree = 1, out-degree = 3
Vertex 8: in-degree = 3, out-degree = 3
Vertex 9: in-degree = 6, out-degree = 2
Vertex 10: in-degree = 2, out-degree = 4
Vertex 11: in-degree = 4, out-degree = 3

Paths of length 2:
1 – 4 – 1
1 – 9 – 9
1 – 4 – 11
1 – 9 – 11
2 – 4 – 1
2 – 6 – 4
2 – 6 – 9
2 – 9 – 9
2 – 6 – 10
2 – 4 – 11
2 – 9 – 11
3 – 4 – 1
3 – 5 – 1
3 – 10 – 1
3 – 11 – 1
3 – 8 – 2
3 – 10 – 2
3 – 6 – 4
3 – 8 – 4
3 – 10 – 4
3 – 10 – 5
3 – 11 – 5
3 – 5 – 6
3 – 5 – 7
3 – 5 – 8
3 – 6 – 9
3 – 8 – 9
3 – 11 – 9
3 – 6 – 10
3 – 4 – 11
3 – 5 – 11

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

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

Paths of length 3:
1 – 4 – 11 – 1
1 – 9 – 11 – 1
1 – 4 – 1 – 4
1 – 4 – 11 – 5
1 – 9 – 11 – 5
1 – 4 – 1 – 9
1 – 4 – 11 – 9
1 – 9 – 9 – 9
1 – 9 – 11 – 9
1 – 9 – 9 – 11
2 – 4 – 11 – 1
2 – 6 – 4 – 1
2 – 6 – 10 – 1
2 – 9 – 11 – 1
2 – 6 – 10 – 2
2 – 4 – 1 – 4
2 – 6 – 10 – 4
2 – 4 – 11 – 5
2 – 6 – 10 – 5
2 – 9 – 11 – 5
2 – 4 – 1 – 9
2 – 4 – 11 – 9
2 – 6 – 9 – 9
2 – 9 – 9 – 9

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

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

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

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

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

Reachability Matrix:

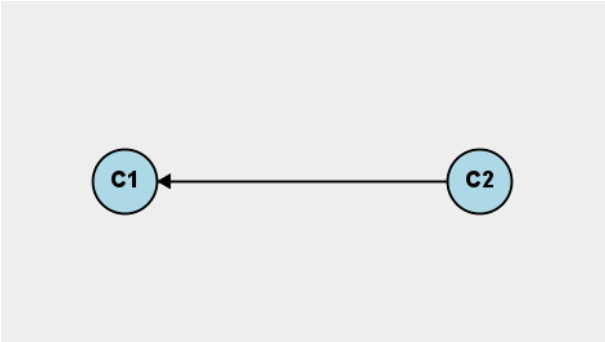
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1

Strong Connectivity Matrix:

1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
0 0 1 0 0 0 0 0 0 0 0
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1

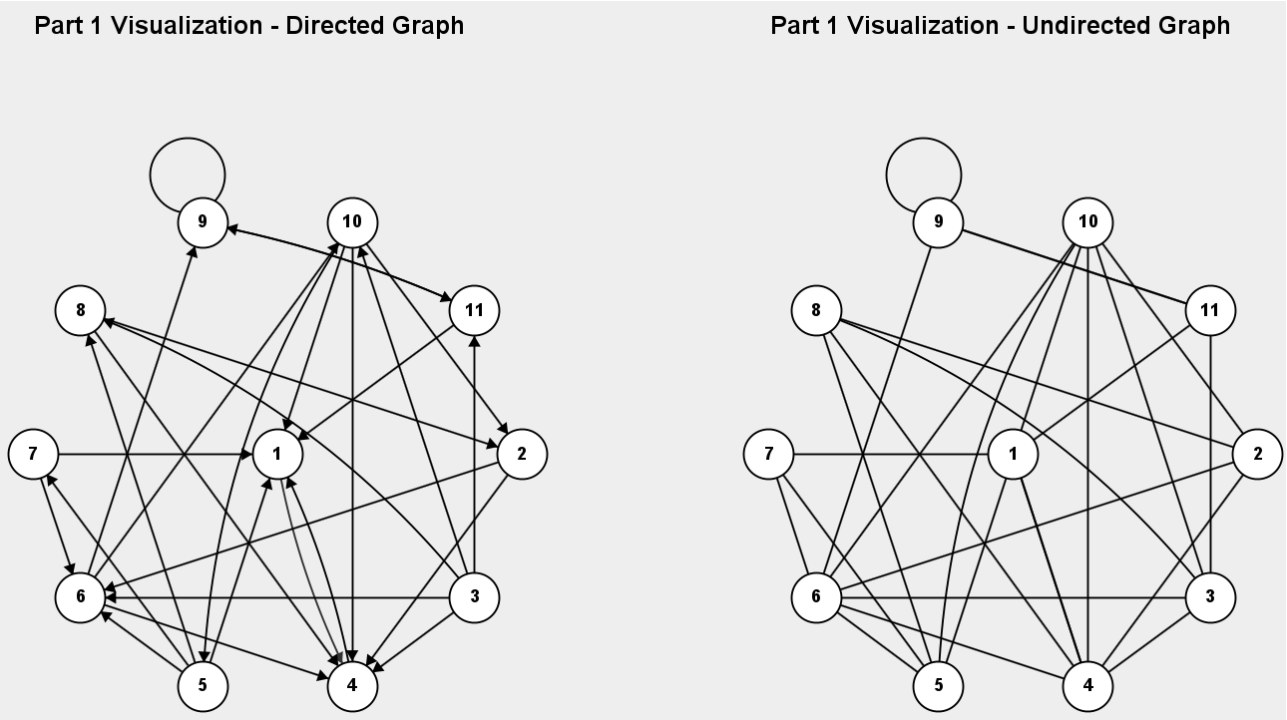
Strongly Connected Components:
Component 1: 1 2 4 5 6 7 8 9 10 11
Component 2: 3

Condensation Graph

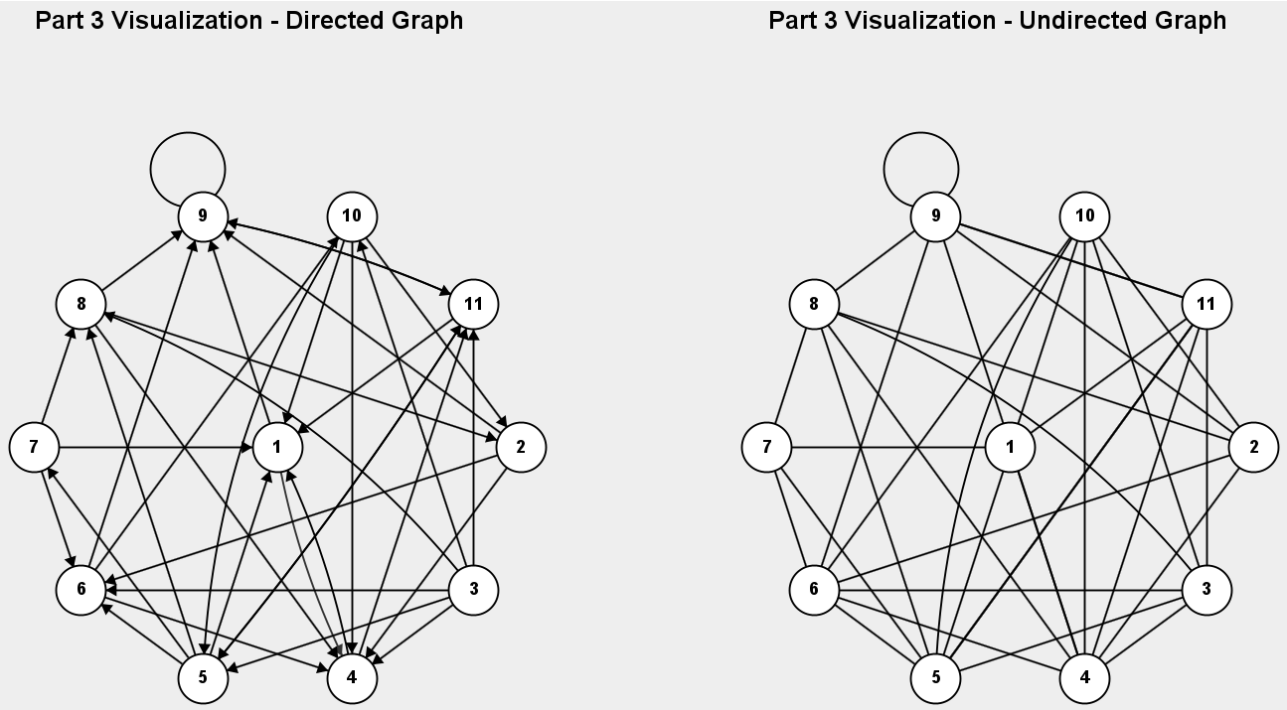


Скриншоти графів

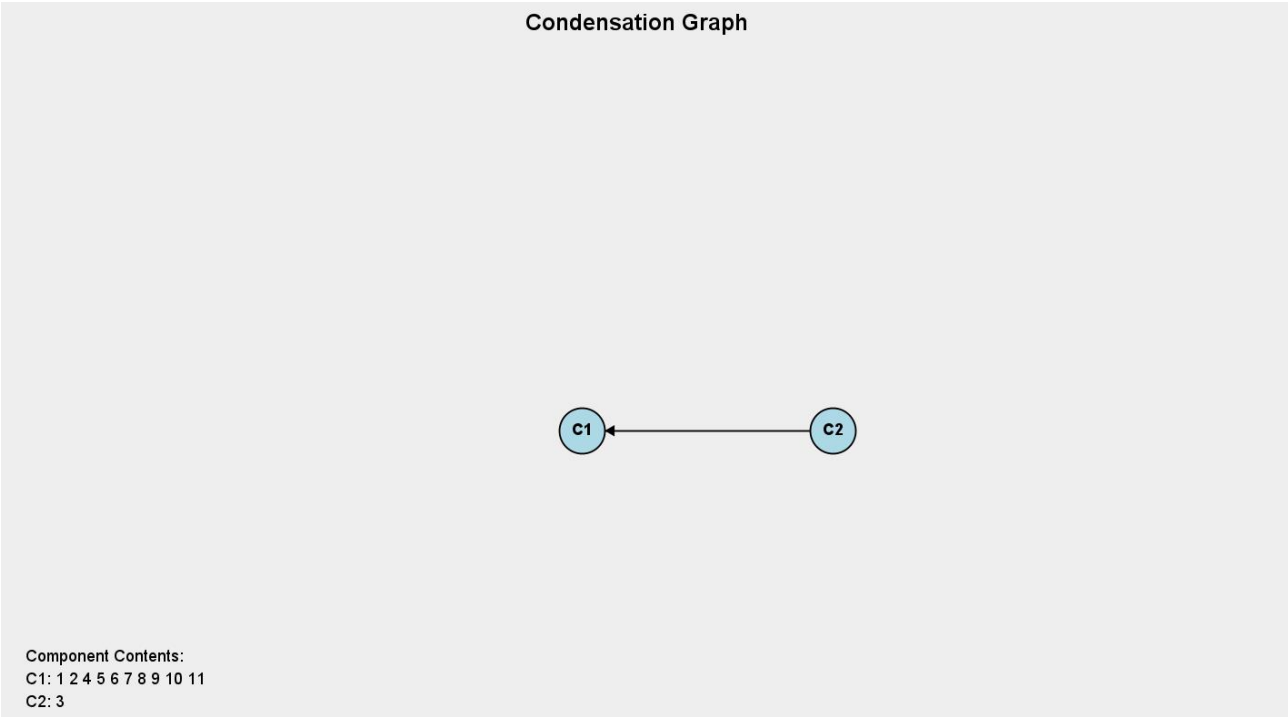
Орієнтований і неорієнтований графи k1



Орієнтований і неорієнтований модифіковані графи k2



Граф конденсації



Висновок

Під час виконання лабораторної роботи № 4 я засвоїв теоретичний матеріал та набув навичок аналізу характеристик та зв'язності графів. Я поглибив свої знання щодо обчислення степенів та напівстепенів вершин, визначення однорідності графа, пошуку шляхів заданої довжини, аналізу досяжності та сильної зв'язності.

Я набув практичного досвіду реалізації алгоритмів для обчислення різноманітних метрик графів як для напрямлених, так і для ненапрямлених випадків, засвоївши методи аналізу матриць суміжності та їх степенів для знаходження шляхів, а також застосування транзитивного замикання для визначення матриці досяжності та компонент сильної зв'язності.

Під час роботи я реалізував обчислення характеристик графів на основі згенерованих матриць суміжності з попередньої роботи. За допомогою мови Java та стандартної бібліотеки Swing я навчився не тільки візуалізувати вихідні графи, але й будувати та відображати граф конденсації, що включало програмне визначення компонентів сильної зв'язності та зв'язків між ними.

Виконання цього завдання допомогло мені краще зрозуміти важливість аналізу зв'язності для розуміння структури графа та практичне застосування таких концепцій, як матриця досяжності та сильна зв'язність. Отриманий практичний досвід роботи з алгоритмами аналізу графів та їх візуалізацією безпосередньо сприятиме удосконаленню моїх навичок алгоритмічного мислення та програмування.

Отже, виконання лабораторної роботи № 4 було корисним, дозволило закріпити теоретичні знання про характеристики та методи аналізу зв'язності графів та набути практичних навичок в їх програмній реалізації та інтерпретації результатів мовою Java.