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 №3

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 у програмі напрямлений і ненапрямлений графи з заданими параметрами:
   - кількість вершин n;
   - розміщення вершин;
   - матриця суміжності A.
2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

Згадані вище параметри графа задаються на основі чотиризначного номера варіанту $n_1n_2n_3n_4$, де n_1n_2 це десяткові цифри номера групи, а n_3n_4 — десяткові цифри номера варіанту, який був у студента для двох попередніх робіт (див. таблицю з поточними оцінками з АСД, надану викладачем на початку поточного семестру).

Кількість вершин n дорівнює $10 + n_3$.

Розміщення вершин:

- колом при $n_4 = 0, 1$;
- квадратом (прямокутником) при $n_4 = 2, 3$;
- трикутником при $n_4 = 4, 5$;
- колом з вершиною в центрі при $n_4 = 6, 7$;
- квадратом (прямокутником) з вершиною в центрі при $n_4 = 8, 9$.

Наприклад, при $n_4 = 9$ розміщення вершин прямокутником з вершиною в центрі повинно виглядати так, як на прикладі графа на рис. 8.

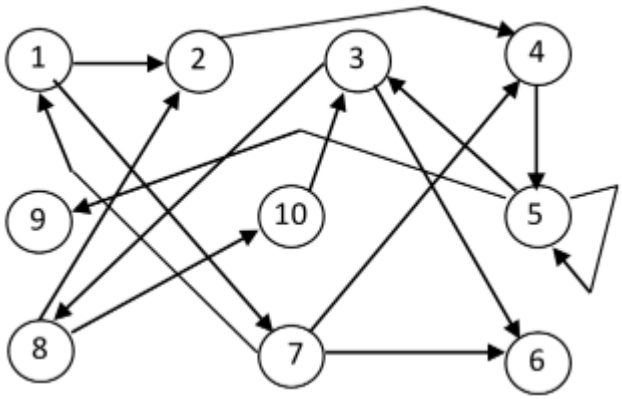


Рис. 8. Приклад зображення графа

Матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту $n_1n_2n_3n_4$ — детальніше див. с. 12;
2) матриця розміром n*n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);
3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.02 - n_4 * 0.005 - 0.25$;
4) кожен елемент матриці множиться на коефіцієнт k;
5) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

Матриця суміжності A_{undir} ненапрямленого графа одержується з матриці A_{dir}:

$$a_{dir(i,j)} = 1 \Rightarrow a_{undir(i,j)} = 1,\ a_{undir(j,i)} = 1.$$

Наприклад, якщо запрограмувати додаткові функції randm та mulmr, у програмі мовою С генерація матриці A_{dir} напрямленого графа може виглядати так:

```
srand(n1n2n3n4);
T = randm(n);
k = 1.0 - n3*0.02 - n4*0.005 - 0.25;
A = mulmr(T, k);
```

Тут randm(n) — розроблена функція, яка формує матрицю розміром nn, що складається з випадкових чисел у діапазоні [0, 2.0);

mulmr(T, k) — розроблена функція множення матриці на коефіцієнт та округлення результату до 0 чи 1.

При проєктуванні програм **слід врахувати наступне**:

1) мова програмування обирається студентом самостійно;
2) графічне зображення графа має формуватися на основі графічних примітивів з графічної бібліотеки (таких як еліпс, пряма, дуга, текст тощо);
3) використання готових бібліотек для роботи з графами не дозволяється;
4) вивід графа має бути реалізований універсальним чином: вершини і ребра мають виводитися в циклі, а не окремими командами для кожного графічного елемента;
5) типи та структури даних для внутрішнього представлення графа у програмі слід вибрати само;
6) матриці суміжності графів можна виводити в графічне вікно або консоль — на розсуд студента;
7) матриці суміжності мають виводитися як матриці: в квадратному вигляді, з 1 та 0.

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 java.awt.geom.Line2D;
import java.awt.geom.QuadCurve2D;
import javax.swing.*;
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Path2D;
import java.util.Random;

public class GraphVisualizer 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 K = 1.0 - N3 * 0.02 - N4 * 0.005 - 0.25;

    private static final int WINDOW_WIDTH = 1200;
    private static final int WINDOW_HEIGHT = 800;
    private static final int VERTEX_RADIUS = 20;
    private static final int ARROW_SIZE = 10;
    private static final int SELF_LOOP_OFFSET = 40;
    private static final int CURVE_CONTROL_OFFSET = 50;

    private int[][] directedMatrix;
    private int[][] undirectedMatrix;

    public GraphVisualizer() {
        initializeWindow();
        generateMatrices();
        printMatrices();
        add(new GraphPanel());
    }

    private void initializeWindow() {
        setTitle("Graph Visualizer");
        setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
```

```
}

private void generateMatrices() {
    directedMatrix = generateDirectedMatrix();
    undirectedMatrix = generateUndirectedMatrix(directedMatrix);
}

private void printMatrices() {
    System.out.println("Directed Graph Adjacency Matrix:");
    printMatrix(directedMatrix);
    System.out.println("\nUndirected Graph Adjacency Matrix:");
    printMatrix(undirectedMatrix);
}

private int[][] generateDirectedMatrix() {
    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 printMatrix(int[][] matrix) {
    for (int[] row : matrix) {
        for (int cell : row) {
```

```
                System.out.print(cell + " ");
            }
            System.out.println();
        }
    }

    class GraphPanel extends JPanel {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            Graphics2D g2d = (Graphics2D) g;
            setupGraphics(g2d);

            int panelWidth = getWidth();
            int panelHeight = getHeight();

            drawGraph(g2d, directedMatrix, 0, 0, panelWidth / 2,
panelHeight, true);

            drawGraph(g2d, directedMatrix, panelWidth / 2, 0, panelWidth
/ 2, panelHeight, false);

            drawTitles(g2d, panelWidth);
        }

        private void setupGraphics(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("Directed Graph", 100, 30);
            g2d.drawString("Undirected Graph", panelWidth / 2 + 100,
30);
        }

        private void drawGraph(Graphics2D g2d, int[][] matrix, int x,
int y, int width, int height, boolean isDirected) {
            int centerX = x + width / 2;
            int centerY = y + height / 2;
```

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

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

            drawEdges(g2d, matrix, vertexPositions, isDirected);

            drawVertices(g2d, vertexPositions);
        }

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

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

            double angleStep = 2 * Math.PI / (VERTEX_COUNT - 1);
            for (int i = 1; i < VERTEX_COUNT; 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 < VERTEX_COUNT; i++) {
                for (int j = 0; j < VERTEX_COUNT; 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[VERTEX_COUNT][VERTEX_COUNT];

            if (isDirected) {
                for (int i = 0; i < VERTEX_COUNT; i++) {
                    for (int j = 0; j < VERTEX_COUNT; 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) {
            for (int i = 0; i < VERTEX_COUNT; i++) {
                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 * VERTEX_RADIUS,
                2 * 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 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);
            }
        }

        private 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);
        }

        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) / (VERTEX_COUNT - 1);
        }
    }

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

Згенеровані за варіантом матриці суміжності

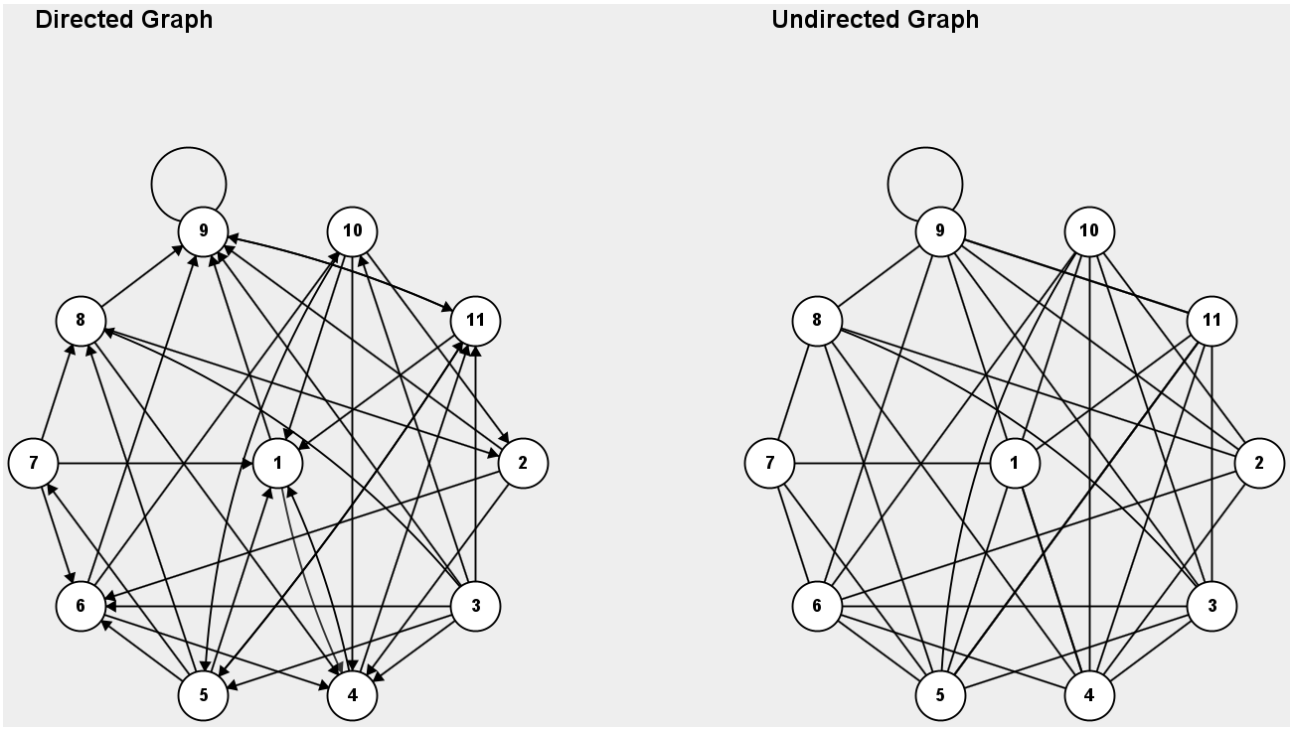
Матриця суміжності напрамленого графа:

```
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 1 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 графа:

```
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 1 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 1 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
```

Скриншоти графів



Висновок

Під час виконання лабораторної роботи № 3 я засвоїв теоретичний матеріал та набув навичок представлення графів у комп’ютері та ознайомлення з принципами роботи ОС.

Я поглибив свої знання щодо представлення графів у комп’ютері та роботи з графічними інтерфейсами. Я набув практичного досвіду реалізації алгоритмів для побудови як напрямлених, так і ненапрямлених графів, засвоївши методи генерування та обробки матриць суміжності за заданими параметрами.

Під час роботи я реалізував генерацію випадкових значень для заповнення матриць графів з використанням визначеного параметру seed, що дозволило отримати відтворюваний результат. За допомогою мови Java та стандартної бібліотеки Swing я навчився використовувати графічні примітиви для створення зображення графів, що включає побудову вершин, ребер, а також спеціальні механізми для відображення з’єднання вершини графа із собою та уникання перетину ребер з центром графа.

Виконання цього завдання допомогло мені краще зрозуміти важливість правильного вибору структур даних та алгоритмів для вирішення конкретних задач. Отриманий практичний досвід роботи з графічними компонентами та алгоритмами побудови графів безпосередньо сприятиме удосконаленню моїх навичок програмування та розумінню принципів роботи операційних систем.

Отже, виконання лабораторної роботи № 3 було корисним, дозволило закріпити теоретичні знання та набути практичних навичок в області програмування мовою Java.