МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Построение минимального остовного дерева при помощи алгоритма Прима.

Студент гр. 2384 _____ Березин Д.А.

Студент гр. 2384 _____ Исмаилов М.В.

Студент гр. 2384 _____ Гребенников Д.А.

Руководитель _____ Шестопалов Р.П.

Санкт-Петербург
2024

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Березин Д.А. группы 2384

Студент Гребенников Д.А. группы 2384

Студент Исмаилов М.В. группы 2384

Тема практики: построение минимального остовного дерева при помощи алгоритма Прима.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: Прима.

Сроки прохождения практики: 26.06.2024 – 09.07.2024

Дата сдачи отчета: 08.07.2024

Дата защиты отчета: 08.07.2024

Студент гр. 2384		Березин Д.А.
Студент гр. 2384		Исмаилов М.В.
Студент гр. 2384		Гребенников Д.А.
Руководитель		Шестопалов Р.П.

АННОТАЦИЯ

Целью проекта является создание приложения с графическим интерфейсом, которое реализует алгоритм Прима для построения минимального остовного дерева. Разработка ведется итеративным методом – согласовывается спецификация, в которой описано поведение программы и ее основные функции, затем создаются несколько версий проекта, каждая из которых расширяет возможности предыдущей.

SUMMARY

The goal of the project is to create an application with a graphical interface that implements the Prim algorithm for building a minimal spanning tree. Development is carried out iteratively – a specification is agreed upon, which describes the behavior of the program and its main functions, then several versions of the project are created, each of which expands the capabilities of the previous one.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью практики является визуализация алгоритма на языке программирования Java. Для выполнения практики необходимо выполнить следующие задачи:

1) Изучить реализуемый алгоритм
2) Составить и согласовать спецификацию
3) Составить план разработки приложения
4) Распределить роли
5) Создать нулевую версию приложения
6) Расширять возможности приложения, в соответствии с планом
7) Предоставит проект руководителю

Алгоритм Прима - алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость. Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа. Результатом работы алгоритма является остовное дерево минимальной стоимости.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Программы изначально должна запускать графический интерфейс, ожидающий входных данных от пользователя, которые затем передаются в класс, отвечающий за алгоритм Прима. После его завершения пользователю будет показан результат алгоритма, так же будет реализована возможность выполнить алгоритм пошагово.

1.1.1 Требования к визуализации работы алгоритма

Необходимо реализовать визуализацию построения графа и работы алгоритма, в начале пользователь строит граф или загружает уже готовый из файла, затем выполняется алгоритм, каждый шаг которого сохраняется и визуализируется.

1.1.2 Требования к визуализации пользовательского интерфейса

Интерфейс состоит окна редактирования графа, в котором находится поле для построения графа, а также панель с кнопками (добавление вершины, добавление ребра, удаление вершины/ребра, перемещение вершины), окна, в котором показано выполнение алгоритма по шагам, а также панели управления, в которой располагаются кнопки выбора алгоритма (для последующего расширения возможностей программы), сохранения/загрузки графа, а также выполнения алгоритма.

1.1.3 Требования к входным данным

Входными данным для приложения является граф, построенный в редакторе, получить такой граф возможно 2 способами: загрузить из файла, либо построить в редакторе.

1.2. Уточнение требований после сдачи прототипа

Добавить кнопку, которая сразу показывает результат выполнения алгоритма

1.3. Уточнение требований после сдачи 1-ой версии

Добавить обработку случая, когда в графе несколько компонент связности

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Дата	Этап проекта	Реализованные возможности	Выполнено
27.06	Согласование спецификации	Определены основные возможности приложения, реализована UML диаграмма, создан прототип графического окна	Определены основные возможности приложения, реализована UML диаграмма, создан прототип графического окна
27.06	Сдача прототипа	Реализация графического интерфейса	Реализован графический интерфейс
03.07	Сдача версии 1	Реализация алгоритма, возможности создания графа в приложении с использованием графического интерфейса	Реализован алгоритм, добавлена возможность создавать граф в приложении
05.07	Сдача версии 2	Добавление алгоритма к интерфейсу, реализация возможности выполнения алгоритма по шагам, реализация сохранения графа в файл, загрузки графа из файла	Алгоритм был добавлен в графический интерфейс, реализована возможность выполнения по шагам, реализована

			возможность сохранения графа в файл, загрузки графа из файла
05.07	Сдача версии 3	Доработка приложения в соответствии с замечаниями	Доработано приложение
08.07	Сдача отчёта		
08.07	Защита отчёта		

2.2. Распределение ролей в бригаде

1) Исмаилов Максим – реализация графического интерфейса, возможности сохранения графа в файл и загрузки графа из файла

2) Гребенников Дмитрий – разработка алгоритма, реализация возможности выполнить алгоритм пошагово

3) Березин Дмитрий – подключение алгоритма к графическому интерфейсу, тестирование приложения.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Основные методы и классы

3.1.1 Класс Edge

Это класс для хранения ребра графа

Поля класса:

1) Vertex start – исходящая вершина
2) Vertex end – конечная вершина
3) String colorE – цвет ребра
4) Weight – вес ребра
5) Int screenWidth, int screenHeight – высота и ширина экрана соответственно
6) Private Boolean isInMst – поле, показывающее, находится ли ребро в МОД

Методы классы:

1) void updateAbsoluteCoordinates(int screenWidth, int screenHeight) – метод, использующийся для адаптивной визуализации положения ребер при изменении размера окна.
2) Point getStart, Point getEnd – геттеры для получения начальной и конечной вершины
3) Vertex getStartVertex(), Vertex getEndVertex(), int getWeight(), String getColorE() – геттеры для получения полей класса
4) void setColorE(String colorE), public void setInMst(boolean isInMst), public boolean isInMst() – сеттеры для измененя полей класса

3.1.2 Класс Vertex

Это класс для хранения вершины графа

Поля класса:

1) int number – номер вершины
2) String colorV – цвет вершины

3) Double relativeX, doubleY – координаты вершины
4) private static final int VERTEX_RADIUS – радиус вершины при визуализации

Методы класса:

1) void setLocation(Point point, int screenWidth, int screenHeight) – устанавливает координаты вершины
2) Point getAbsoluteLocation(int screenWidth, int screenHeight) - метод, использующийся для адаптивной визуализации положения вершины при изменении размера окна.
3) boolean contains(Point p, int screenWidth, int screenHeight) – проверяет, находится ли вершина в определенных координатах
4) void updateAbsoluteCoordinates(int screenWidth, int screenHeight) - метод, использующийся для адаптивной визуализации положения вершины при изменении размера окна.
5) int getNumber(), String getColorV(), void setColorV(String colorV) – сеттеры для изменения полей класса

3.1.3 Класс Graph

Это класс для хранения графа

Поля класса:

1) private ArrayList<Vertex> vertices – контейнер для хранения вершин
2) private ArrayList<Edge> edges – контейнер для хранения ребер

Методы класса:

1) public List<Vertex> getVertices(), List<Edge> getEdges() – геттеры для получения списка вершин и ребер соответственно
2) public ArrayList<Edge> runPrimsAlgorithm() – метод отвечающий за запуск алгоритма прима

3.1.4 Класс MainWindow

Это класс, отвечающий за использование и вызов всех классов и компонент, отвечающих за визуализацию

Методы класса:

1) public static void run() – метод, отрисовывающий основное окно, в котором происходит работа с графом

3.1.5 Класс CalculatorNavigation

Это класс, реализующий работу кнопок

Поля класса:

1) private BufferedImage handImage, private BufferedImage plusImage, private BufferedImage urnImage, private BufferedImage arrowImage – поля, хранящие в себе иконки для кнопок
2) private BufferedImage handImageHovered, private BufferedImage plusImageHovered, private BufferedImage urnImageHovered – поля, хранящие в себе иконки для кнопок при наведении на них
3) public JButton handButton, public JButton plusButton, public JButton urnButton, public JButton arrowButton – поля, отвечающие за отслеживаний действий с кнопками
4) private boolean handButtonPressed, private Boolean plusButtonPressed, private boolean urnButtonPressed, private boolean arrowButtonPressed, private boolean handButtonHovered, private boolean plusButtonHovered, private boolean urnButtonHovered, private boolean arrowButtonHovered – поля, отвечающие за состояние кнопки

Вложенные классы:

1) private class ButtonActionListener implements ActionListener – отслеживает полученное событие, если это событие является и проверяет, является ли оно нажатием какой-либо кнопки
2) private class ButtonHoverListener extends MouseAdapter – отслеживает наведение на иконки

3) public void mouseExited(MouseEvent e) – отслеживает навдение на иконки

Методы класса:

1) protected void paintComponent(Graphics g) – отрисовывает все действия с учетом адаптивности визуализации
2) public boolean isHandButtonPressed(), public boolean isPlusButtonPressed(), public boolean isUrnButtonPressed(), public boolean isArrowButtonPressed() – методы, показывающие, нажата ли соответствующая кнопка

3.1.6 Класс DelimiterLineCalculator

Это класс отвечающий за отрисовку линии в калькуляторе

Методы класса:

1) protected void paintComponent(Graphics g) – отрисовывает линию в калькуляторе

3.1.7 Класс DelimiterLineSteps

Это класс, отвечающий за отрисовку линии в области шагов алгоритма

Методы класса:

1) protected void paintComponent(Graphics g) – отрисовывает линию в области шагов алгоритма

3.1.8 Класс DelimiterLineToolBar

Это класс отвечающий за отрисовку линии в области панели инструментов

Методы класса:

1) protected void paintComponent(Graphics g) – отрисовывает линию в области панели инструментов

3.1.9 Класс RoundedRectangleCalculatorComponent

Это класс, реализующий визуализацию и работу с графами, включая взаимодействие с вершинами, рёбрами и алгоритмами графов.

Поля класса:

1) private ArrayList<Vertex> vertices – список, хранящий вершины графа.
2) private ArrayList<Edge> edges – список, хранящий рёбра графа.
3) private ArrayList<String> steps – список шагов выполнения алгоритма, представленных в виде строк.
4) public Graph graph – объект графа, который используется для выполнения операций.
5) public ArrayList<Edge> mst – список рёбер, составляющих минимальное остовное дерево (МОТ).
6) private Vertex hoveredVertex – вершина, на которую наведена мышь.
7) private Vertex selectedVertex – выбранная вершина.
8) private Edge selectedEdge – выбранное ребро.
9) private Edge hoveredEdge – ребро, на которое наведена мышь.
10) private CalculatorNavigation navigation – объект класса CalculatorNavigation, отвечающий за навигацию.
11) private ToolBarItems toolBarItems – объект класса ToolBarItems, отвечающий за элементы панели инструментов.
12) private int vertexCounter – счетчик, используемый для подсчета добавленных вершин.
13) private static final int VERTEX_RADIUS – радиус вершины.
14) private static final double EDGE_ADJUSTMENT_FACTOR – коэффициент корректировки рёбер для масштабирования или других операций.
15) public boolean actionMenu – состояние, указывающее, открыто ли меню действий.
16) public boolean actionRun – состояние, указывающее, запущено ли выполнение алгоритма.

17) public boolean actionSteps – состояние, указывающее, отображаются ли шаги выполнения алгоритма.
18) public boolean actionDownload – состояние, указывающее, запущена ли загрузка данных.
19) public boolean actionSave – состояние, указывающее, запущено ли сохранение данных.
20) public boolean actionPreviousStep – состояние, указывающее, выбрана ли команда перехода к предыдущему шагу.
21) public boolean actionNextStep – состояние, указывающее, выбрана ли команда перехода к следующему шагу.
22) public int num – номер текущего шага выполнения алгоритма.
23) private boolean algPrimaSelected – состояние, указывающее, выбран ли алгоритм Прима для построения минимального остовного дерева.

Методы класса:

1) private void handleMousePressed(MouseEvent e) – проверяет, нажата ли какая-либо из кнопок навигации, если да, то наведение на область калькулятора и использование лкм позволяет проводить выбранную операцию
2) private void handleMouseReleased(MouseEvent e) – сбрасывает операцию соединения 2 вершин ребром при нажатии на область, в которой нет вершины
3) private void handleMouseDragged(MouseEvent e) – при выбранной операции перемещения вершин, позволяет перемещать вершину
4) private void handleMouseMoved(MouseEvent e) – при движении мыши, проверяет, наведен ли курсор на вершину/ребро
5) private void handleResize() – обновляет координат, применяется для адаптивности изображения
6) protected void paintComponent(Graphics g) – отрисовывает все компоненты в области калькулятора

7) public void showAlgorithmSelectionDialog() – создает новое окно для выбора алгоритма
8) public void saveGraphToFile(String filename) – открывает окно для сохранения графа
9) public void loadGraphFromFile(String filename) – открывает окно для загрузки графа из файла
10) private Vertex findVertexByNumber(int number) – находит вершину по номеру
11) void setMst(ArrayList<Edge> edges) – устанавливает ребра и вершины в контейнер МОД

3.1.10 Класс RoundedRectangleStepComponent

Это класс, отрисовывающий окно пошаговавого выполнения алгоритма

Поля класса:

1) private static final int PANEL_WIDTH_PERCENTAGE, private static final int PANEL_HEIGHT_PERCENTAGE – отвечают за процентное отношение окна к экрану

Методы класса:

1) protected void paintComponent(Graphics g) – отвечает за отрисовку окно пошаговавого выполнения алгоритма

3.1.11 Класс RoundedRectangleTollBarComponent

Это класс, отвечающий за отрисовку панели инструментов

Методы класса:

2) protected void paintComponent(Graphics g) – отрисовывает панель инструментов

3.1.12 Класс StepsText

Это класс, отвечающий за вывод пошагового выполнения алгоритма

Методы класса:

1) protected void paintComponent(Graphics g) – выводит шаги алгоритма

3.1.13 Класс ToolBarItems

Это класс, реализующий работу панели инструментов с кнопками для управления различными действиями.

Поля класса:

1) private BufferedImage menuImage, private BufferedImage runImage, private BufferedImage stepsImage, private BufferedImage downloadImage, private BufferedImage uploadImage, private BufferedImage rightArrowImage, private BufferedImage leftArrowImage – изображения для кнопок меню, запуска, шагов, загрузки, выгрузки, стрелки вправо и стрелки влево.
2) private BufferedImage menuImageHovered, private BufferedImage runImageHovered, private BufferedImage stepsImageHovered, private BufferedImage downloadImageHovered, private BufferedImage uploadImageHovered, private BufferedImage rightArrowImageHovered, private BufferedImage leftArrowImageHovered – изображения для кнопок при наведении на них.
3) private BufferedImage menuImagePressed, private BufferedImage runImagePressed, private BufferedImage stepsImagePressed, private BufferedImage downloadImagePressed, private BufferedImage uploadImagePressed, private BufferedImage rightArrowImagePressed, private BufferedImage leftArrowImagePressed – изображения для кнопок при нажатии на них.
4) private JButton menuButton, private JButton runButton, private JButton stepsButton, private JButton downloadButton, private JButton uploadButton, private JButton rightArrowButton, private JButton leftArrowButton – кнопки для меню, запуска, шагов, загрузки, выгрузки, стрелки вправо и стрелки влево.

5) private boolean menuButtonPressed, private boolean runButtonPressed, private boolean stepsButtonPressed, private boolean downloadButtonPressed, private boolean uploadButtonPressed, private boolean rightArrowButtonPressed, private boolean leftArrowButtonPressed – логические переменные, указывающие, нажата ли соответствующая кнопка.
6) private boolean menuButtonHovered, private boolean runButtonHovered, private boolean stepsButtonHovered, private boolean downloadButtonHovered, private boolean uploadButtonHovered, private boolean rightArrowButtonHovered, private boolean leftArrowButtonHovered – логические переменные, указывающие, наведена ли мышь на соответствующую кнопку.
7) private int currentIteration – текущая итерация выполнения.
8) private ArrayList<Edge> result – список рёбер, представляющих результат выполнения алгоритма.
9) private boolean stepsClicked – логическая переменная, указывающая, была ли нажата кнопка шагов.
10) private final RoundedRectangleCalculatorComponent roundedRectangleCalculatorComponent – объект класса
11) RoundedRectangleCalculatorComponent, используемый для взаимодействия с компонентом графа.

Вложенные классы:

1) ButtonActionListener - отслеживает полученное событие, если это событие является и проверяет, является ли оно нажатием какой-либо кнопки
2) ButtonHoverListener – отслеживает наведение на иконки

Методы класса:

1) protected void paintComponent(Graphics g) – отрисовывает панель инструментов

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Представлено в приложении А

4.2. Тестирование визуализации алгоритма

Представлено в приложении А

ЗАКЛЮЧЕНИЕ

В ходе практики был изучен язык программирования Java. Был изучен алгоритм Прима для нахождения минимального остовного дерева. Была изучена библиотека Swing для визуализации алгоритма. Результатом практики является приложение, которое визуализирует алгоритм Прима для построения МОД, оно позволяет создать граф через приложение и сохранить его в файл, выполнить алгоритм пошагово.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://docs.oracle.com/en/java/

2. https://www.javatpoint.com/java-swing

3.https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9F%D1%80%D0%B8%D0%BC%D0%B0

4. https://www.geeksforgeeks.org/introduction-to-java-swing/

ПРИЛОЖЕНИЕ А

РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ ПРОГРАММЫ

Рисунок 1- Открытие окна

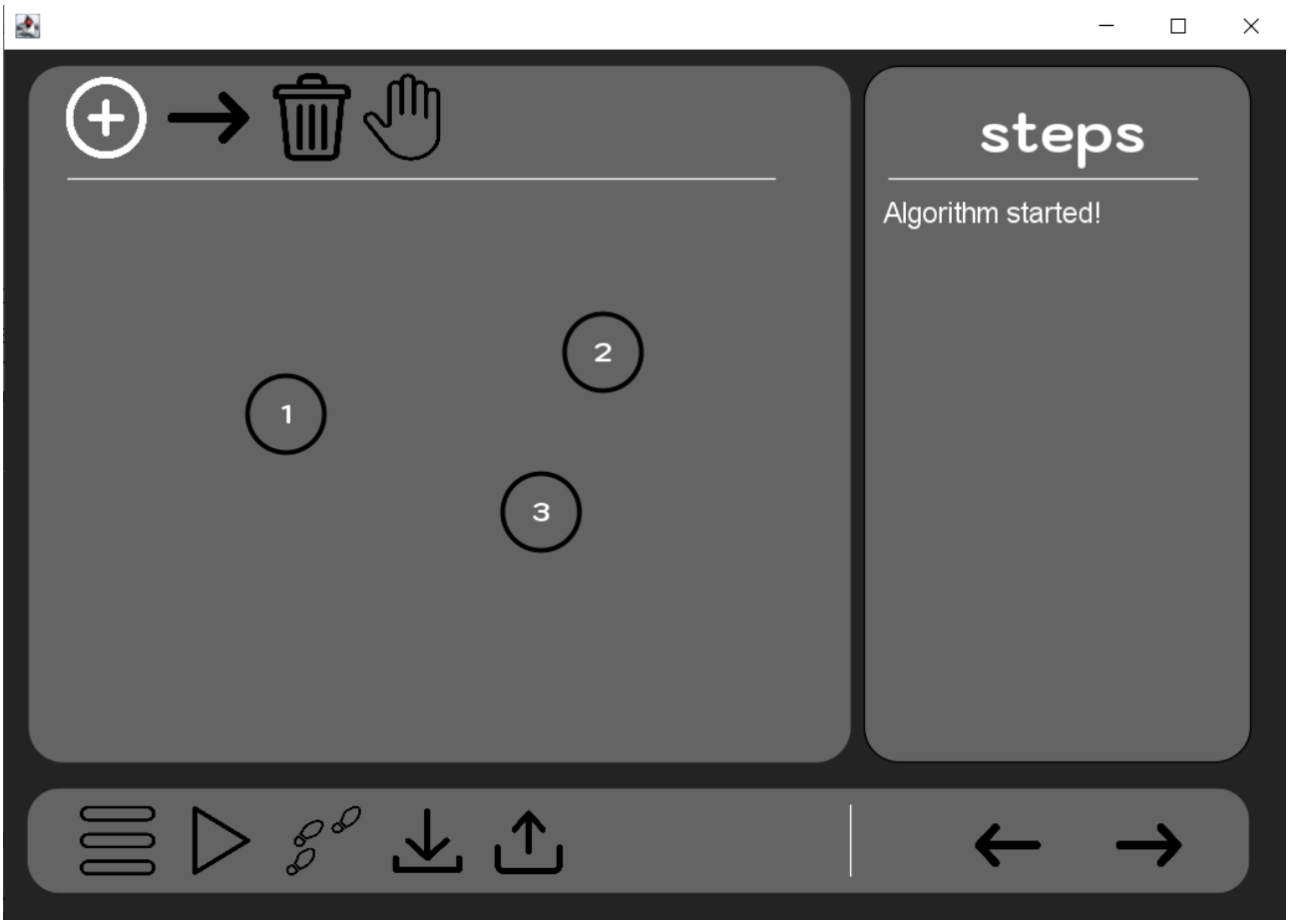

Рисунок 2 - Добавление вершин

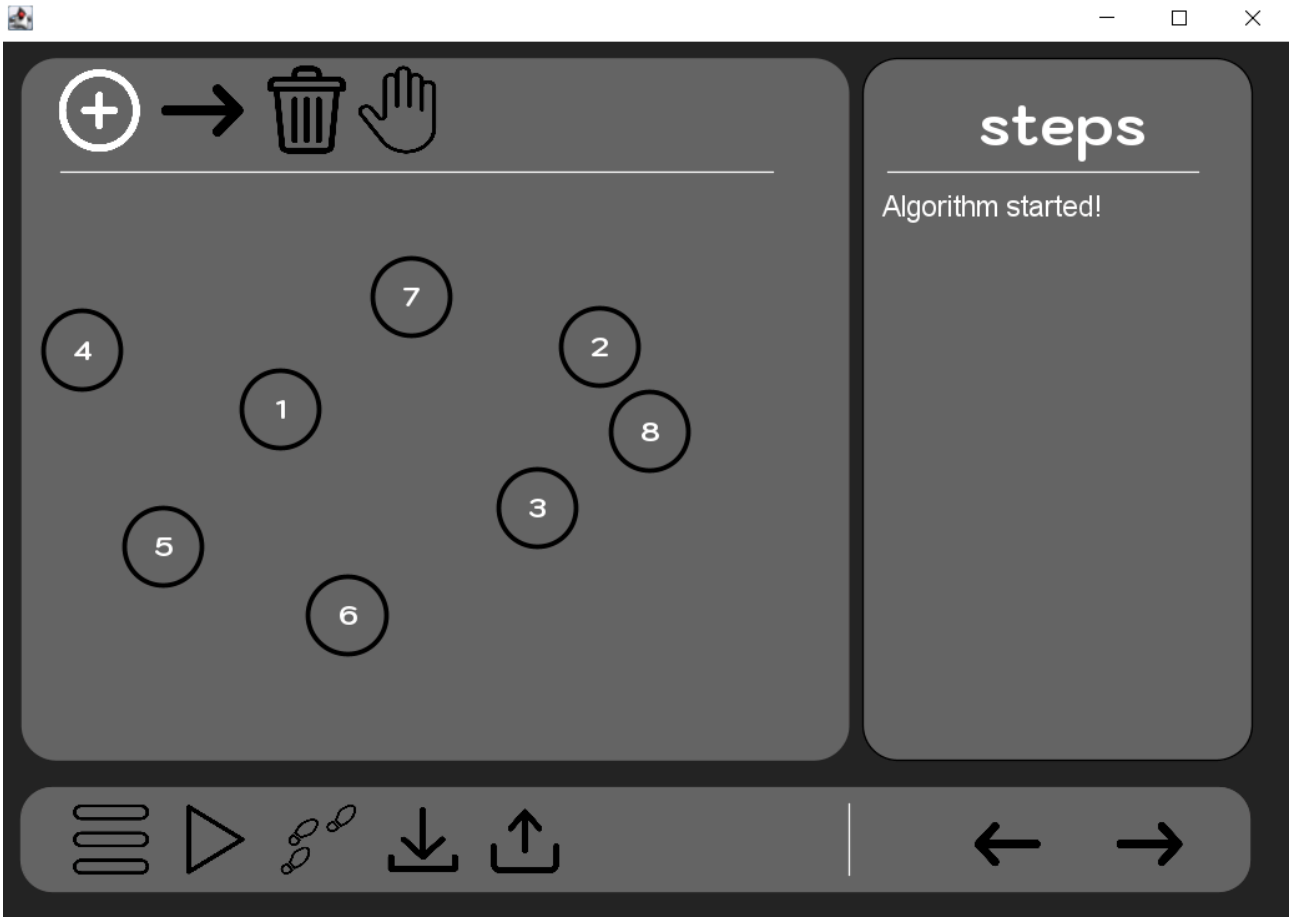

Рисунок 3 - Нумерация вершин

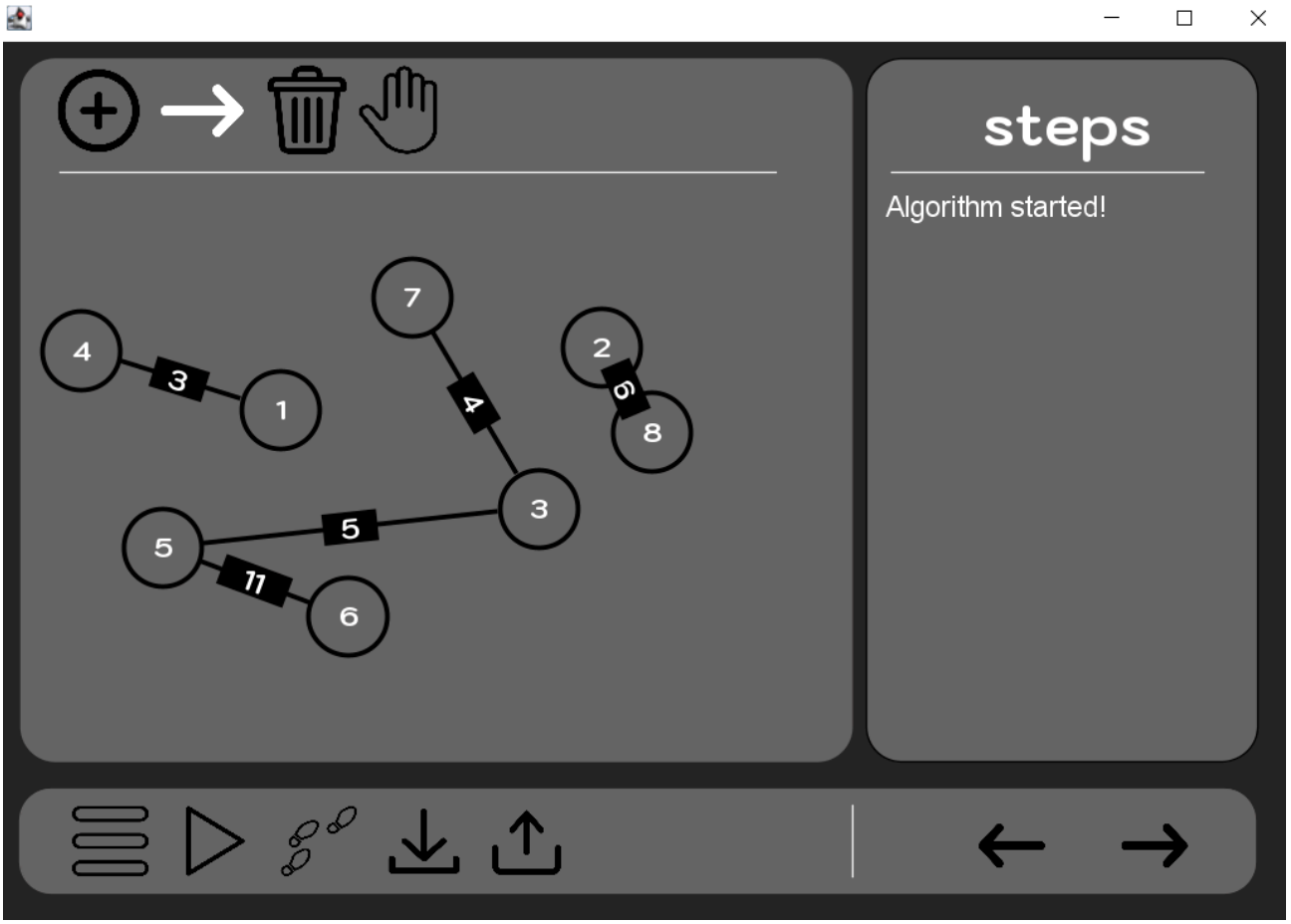

Рисунок 4 - Добавление ребер

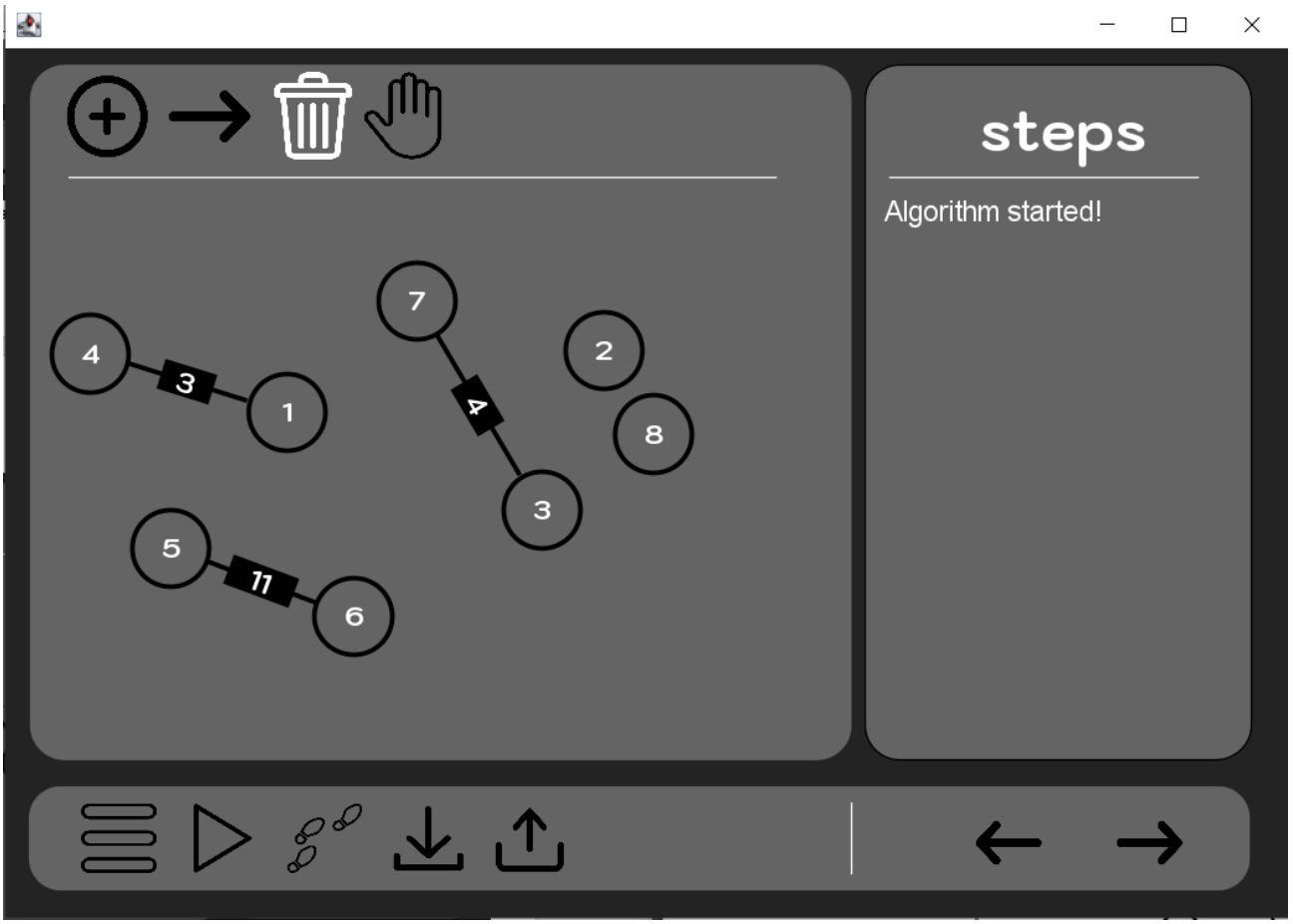

Рисунок 5 - Удаление ребер

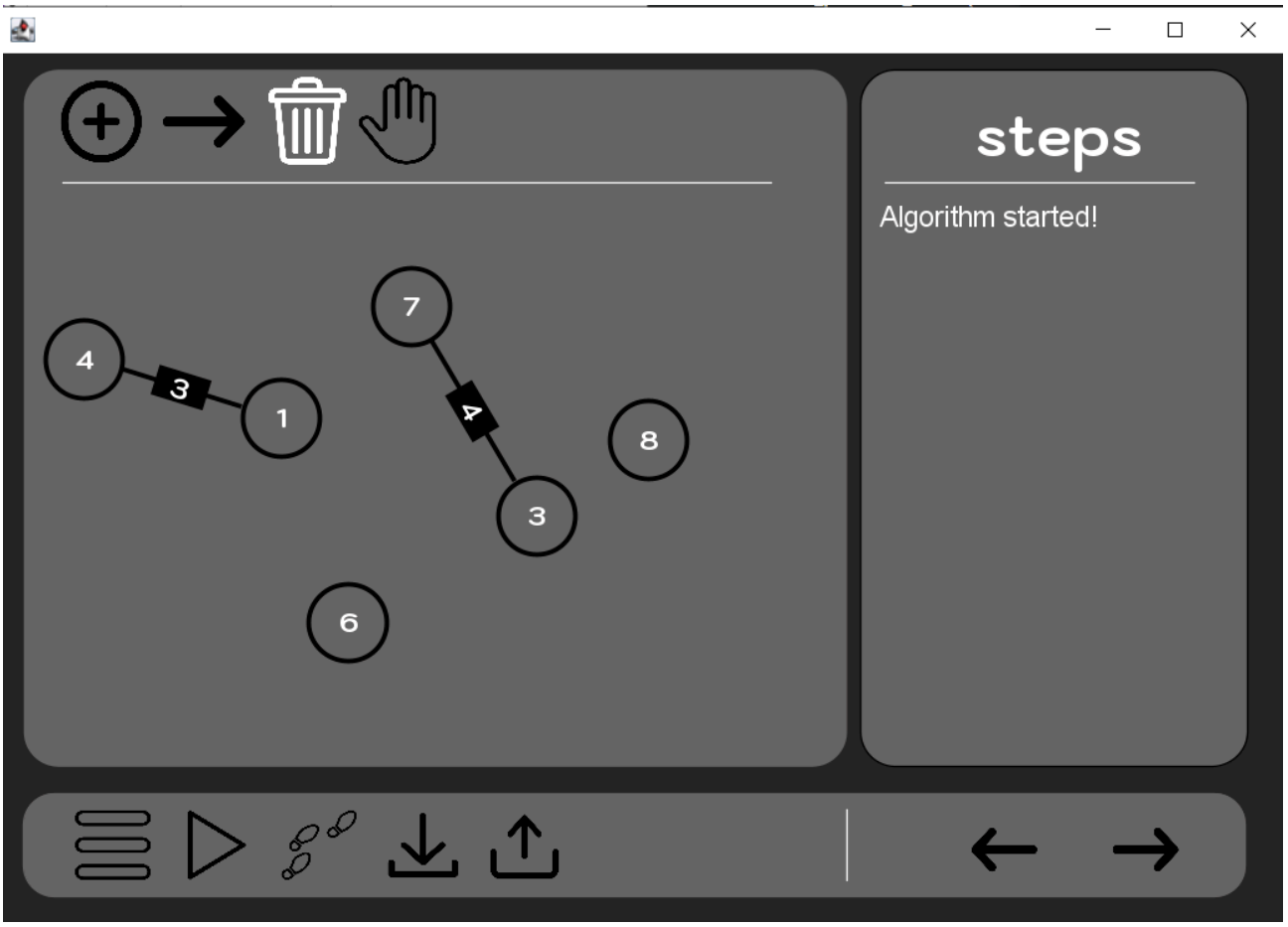

Рисунок 6 - Удаление вершин

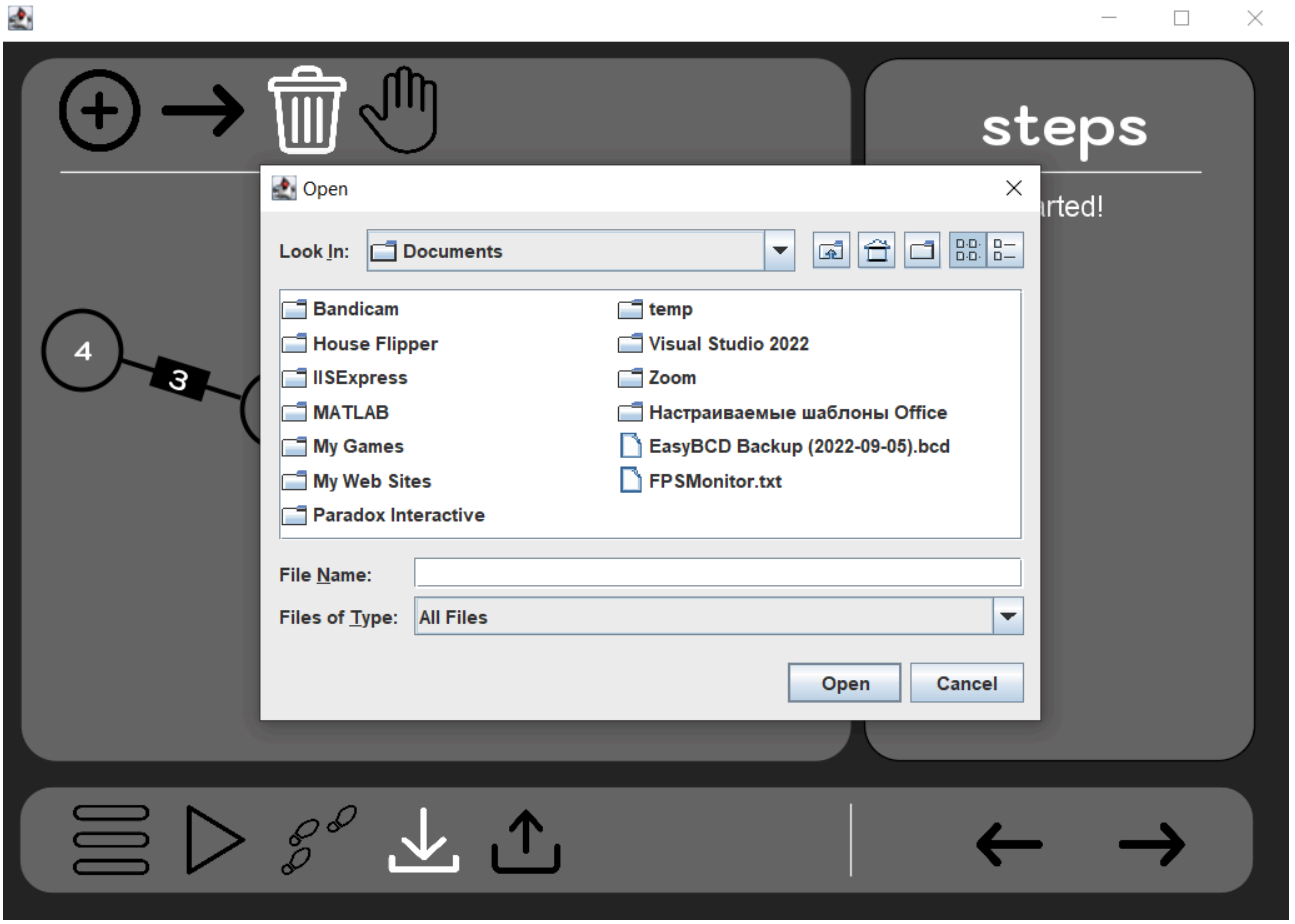

Рисунок 7 - Открытие меню для загрузки графа

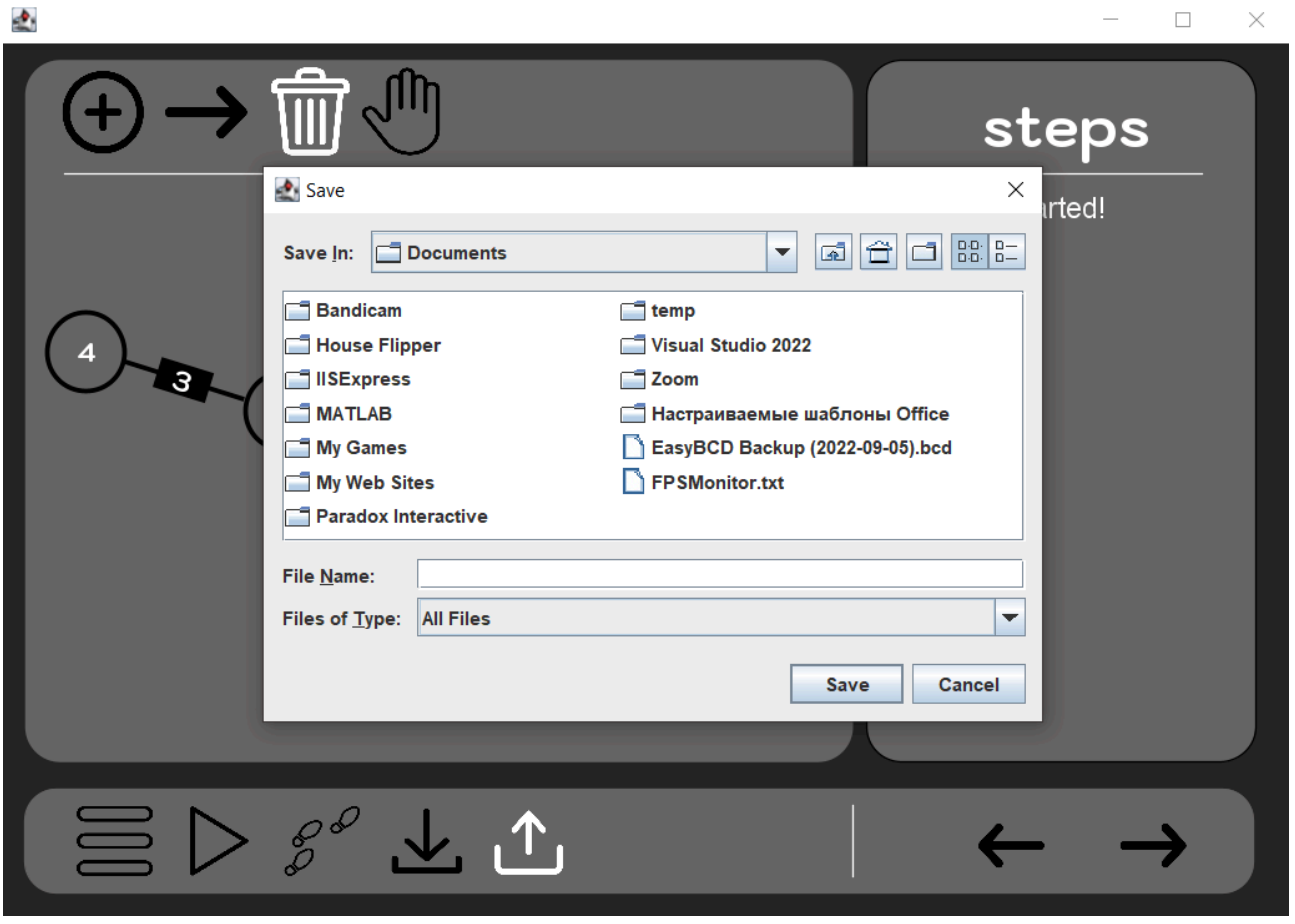

Рисунок 8 - Открытие меню для сохранения графа

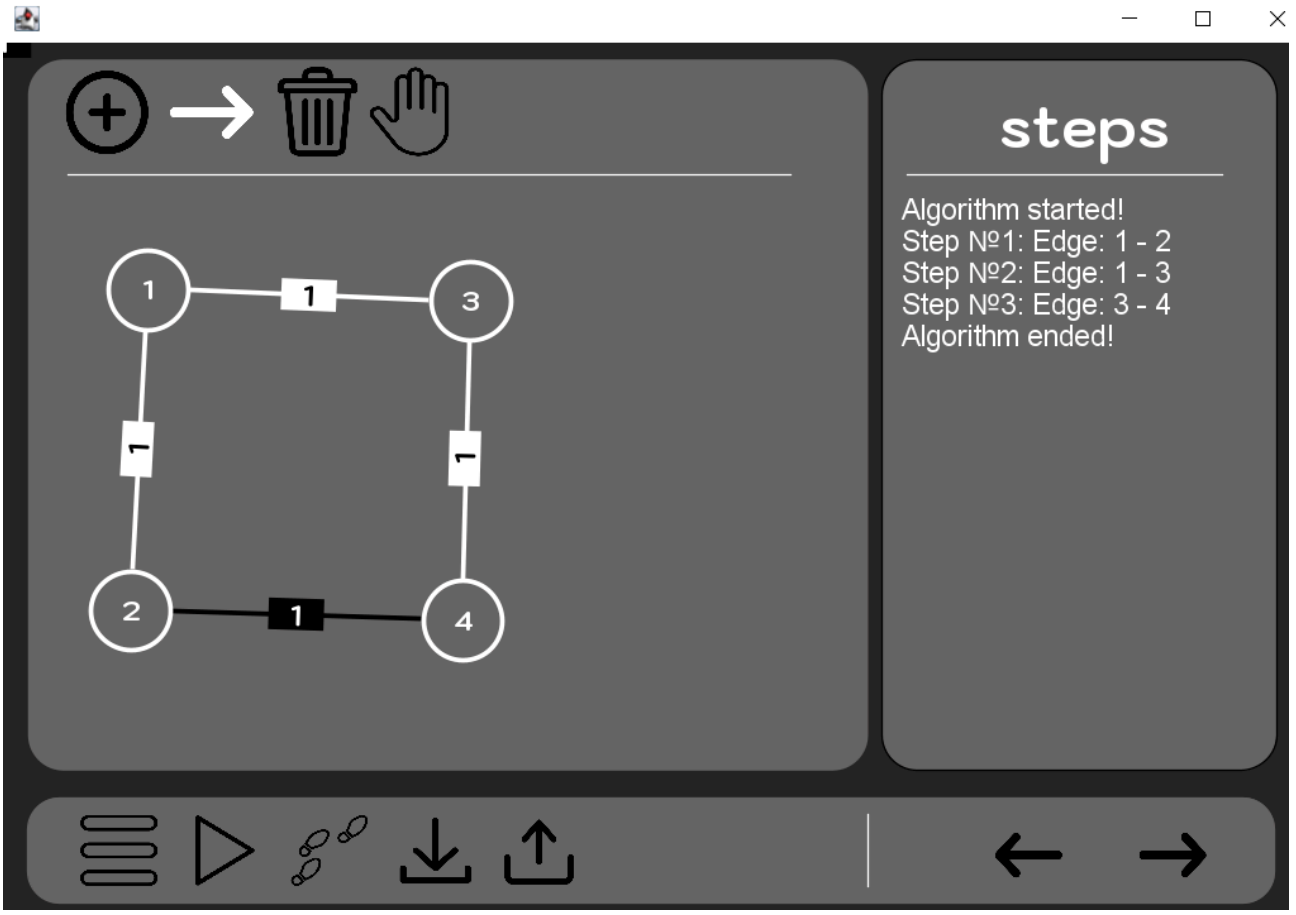

Рисунок 9 - Открытие графа из файла

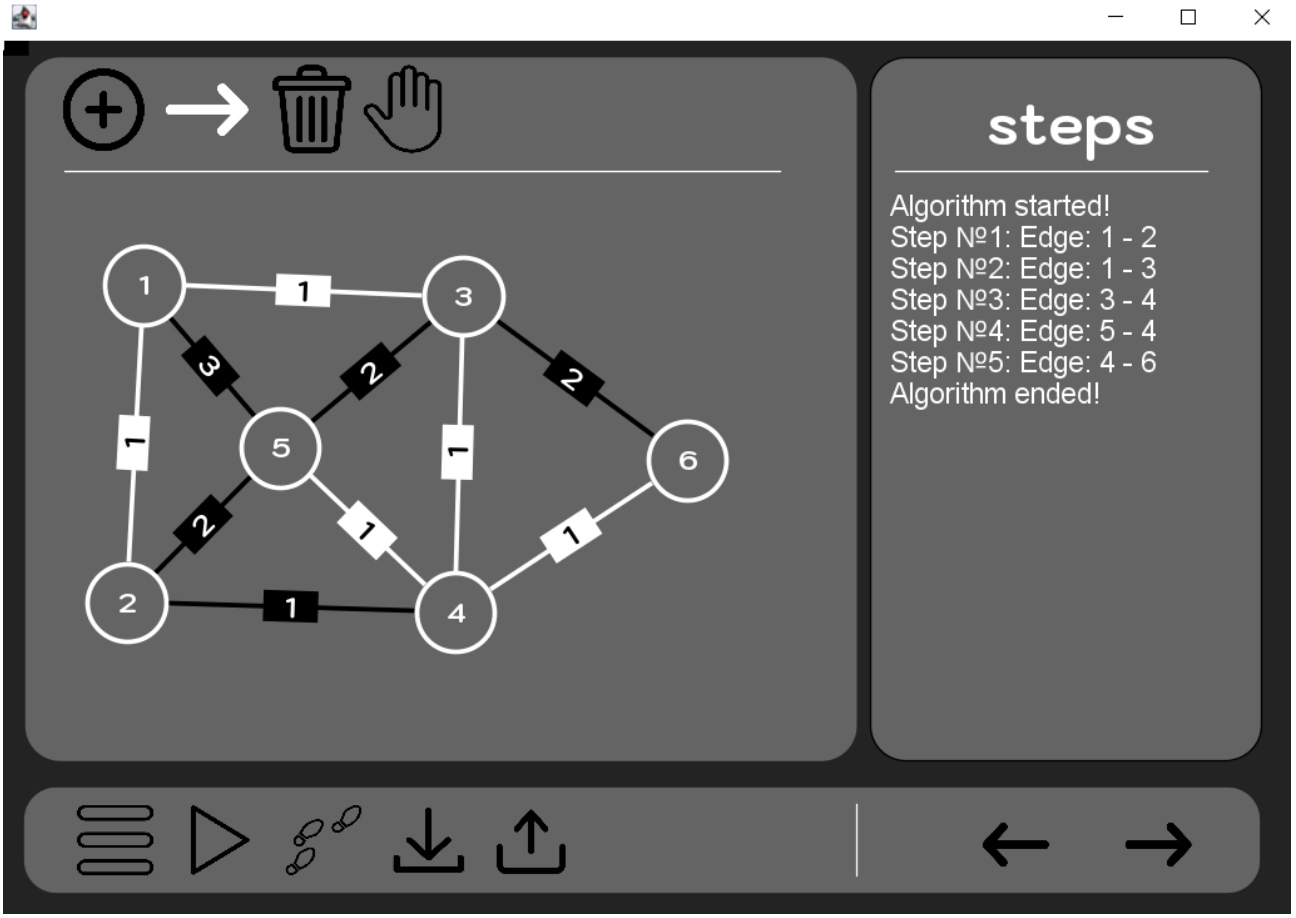

Рисунок 10 - выполнение алгоритма и сохранение шагов

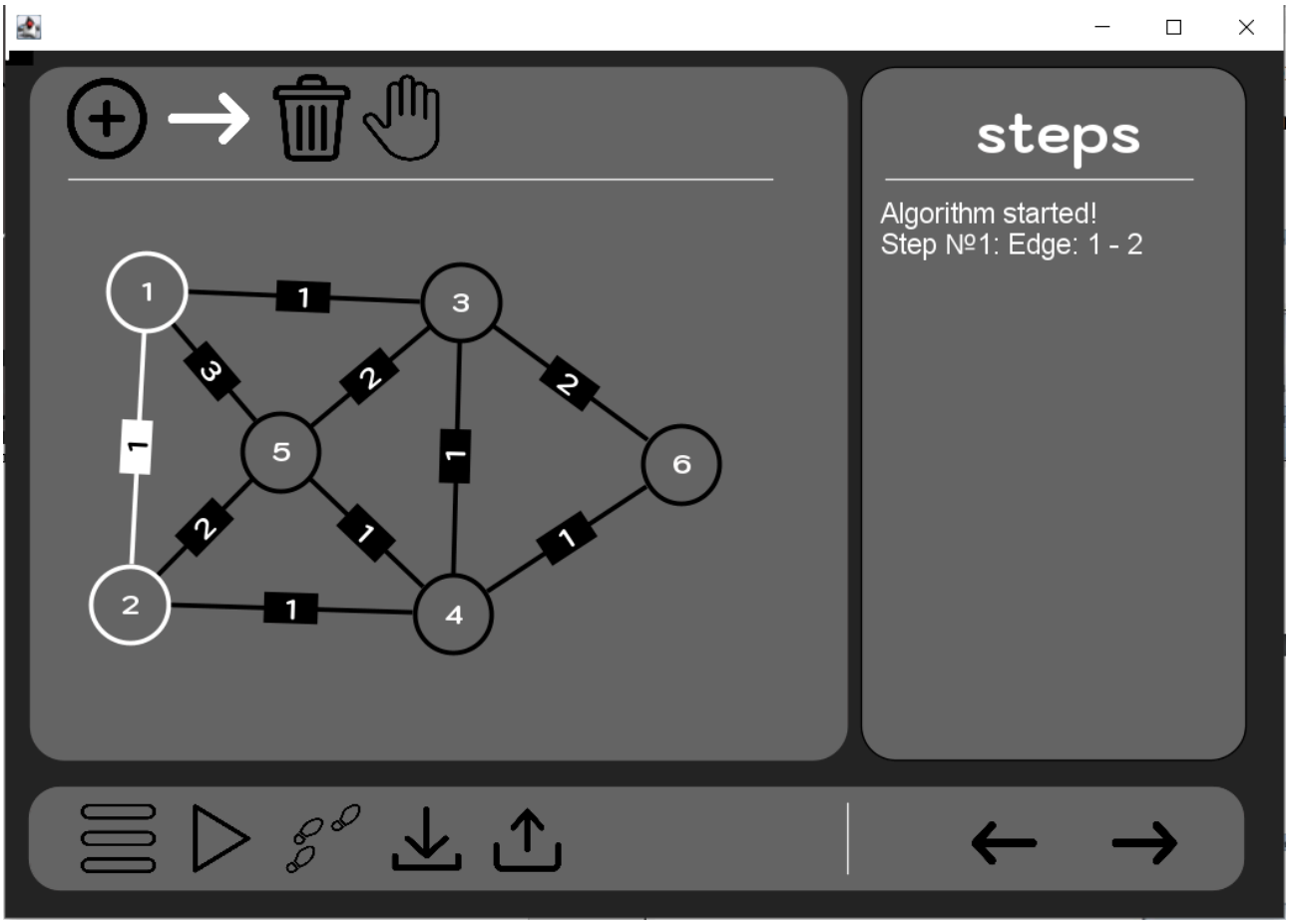

Рисунок 11 - выполнение алгоритма по шагам

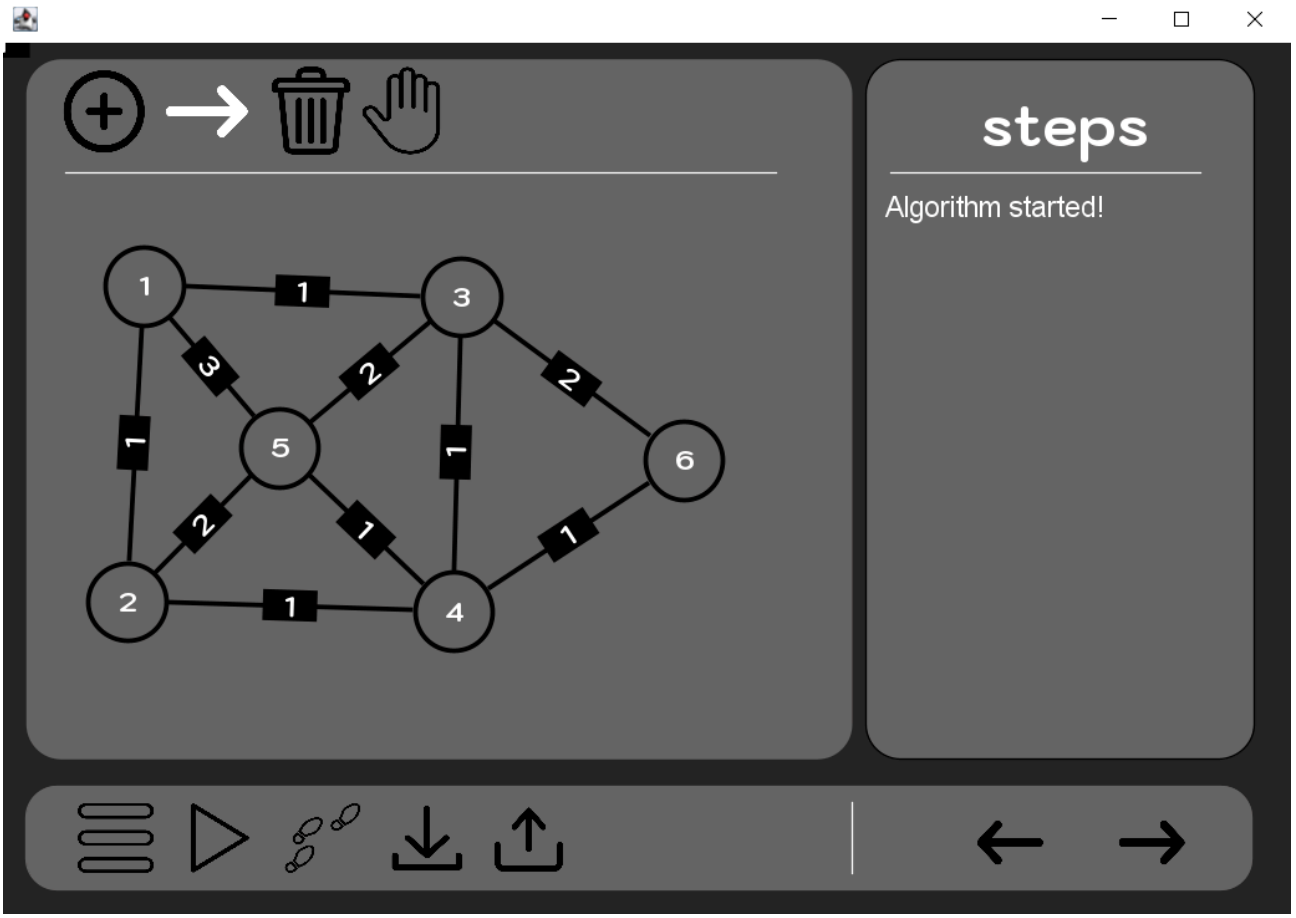

Рисунок 12 - возвращение к предыдущему шагу

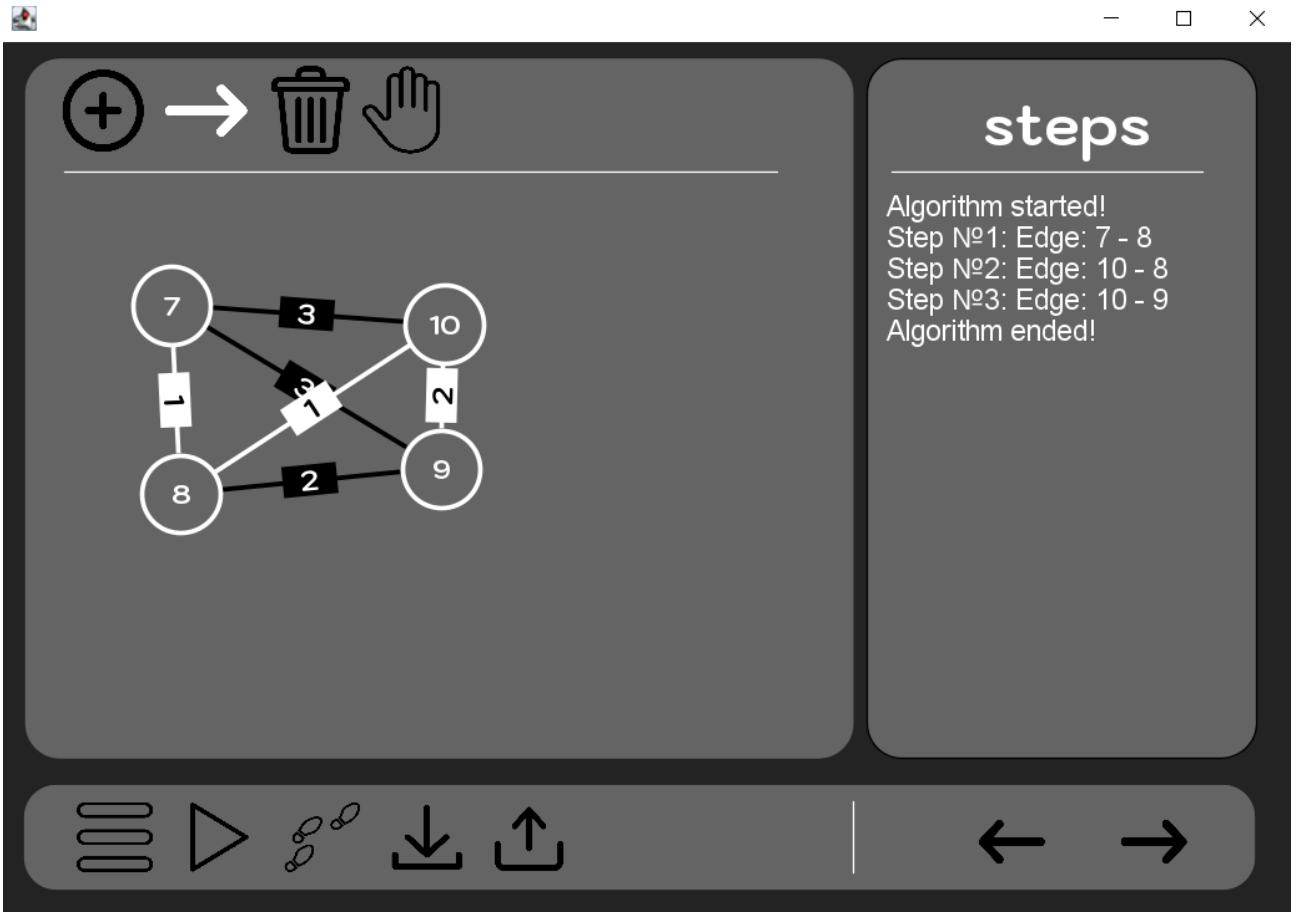

Рисунок 13 - выполнение алгоритма на полном графе

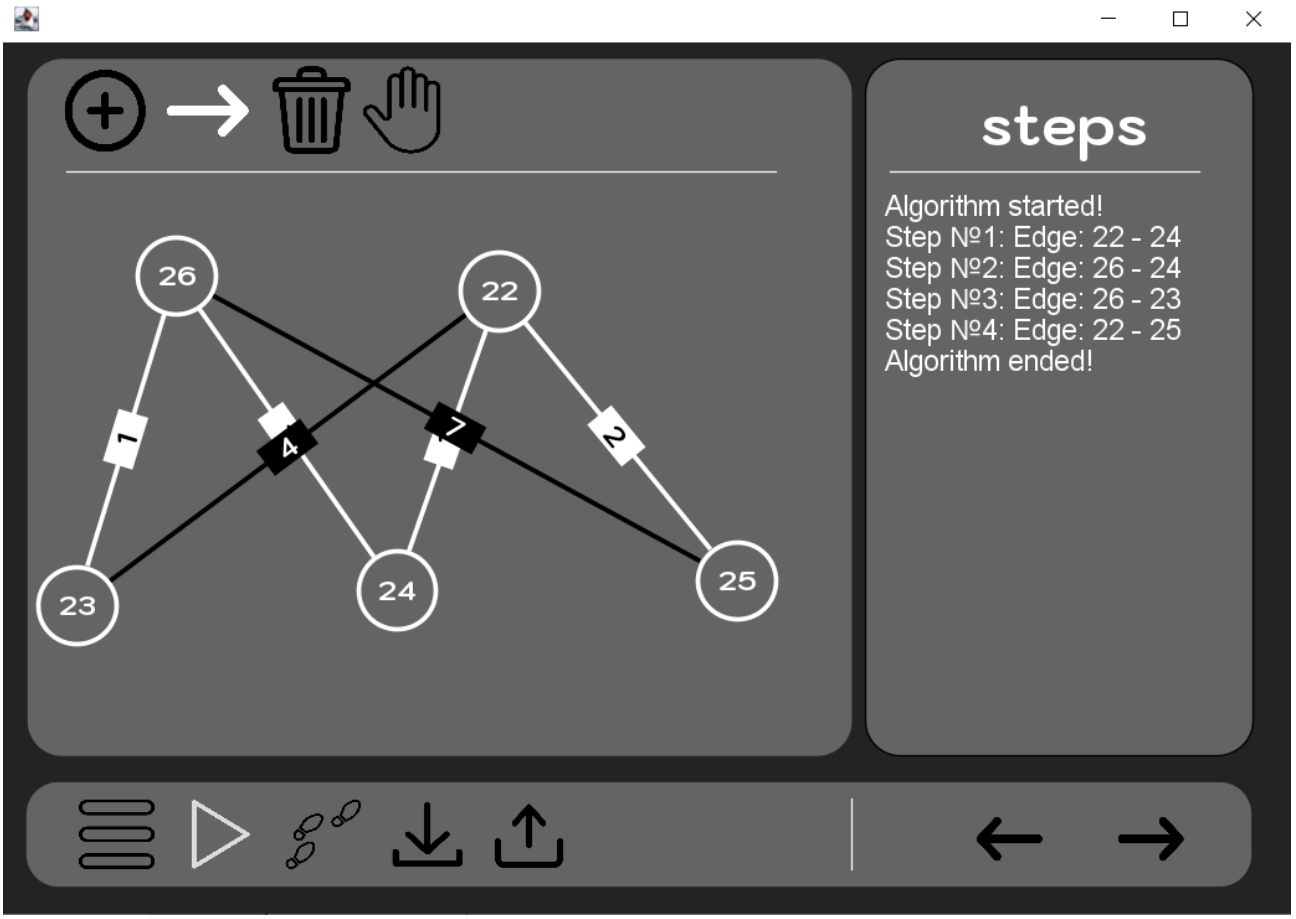

Рисунок 14 - выполнение алгоритма на двудольном графе

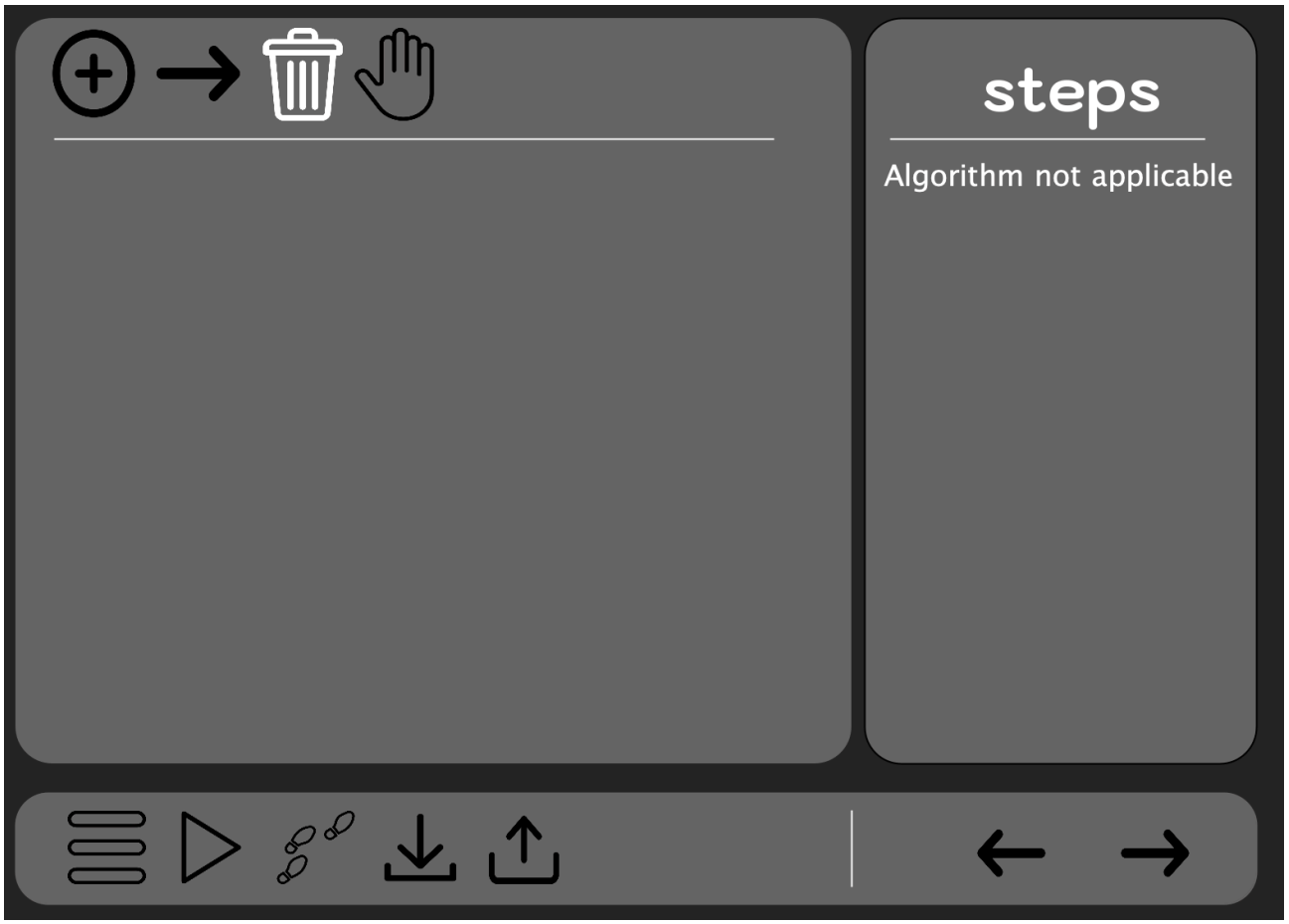

Рисунок 15 - алгоритм не применим к пустому графу

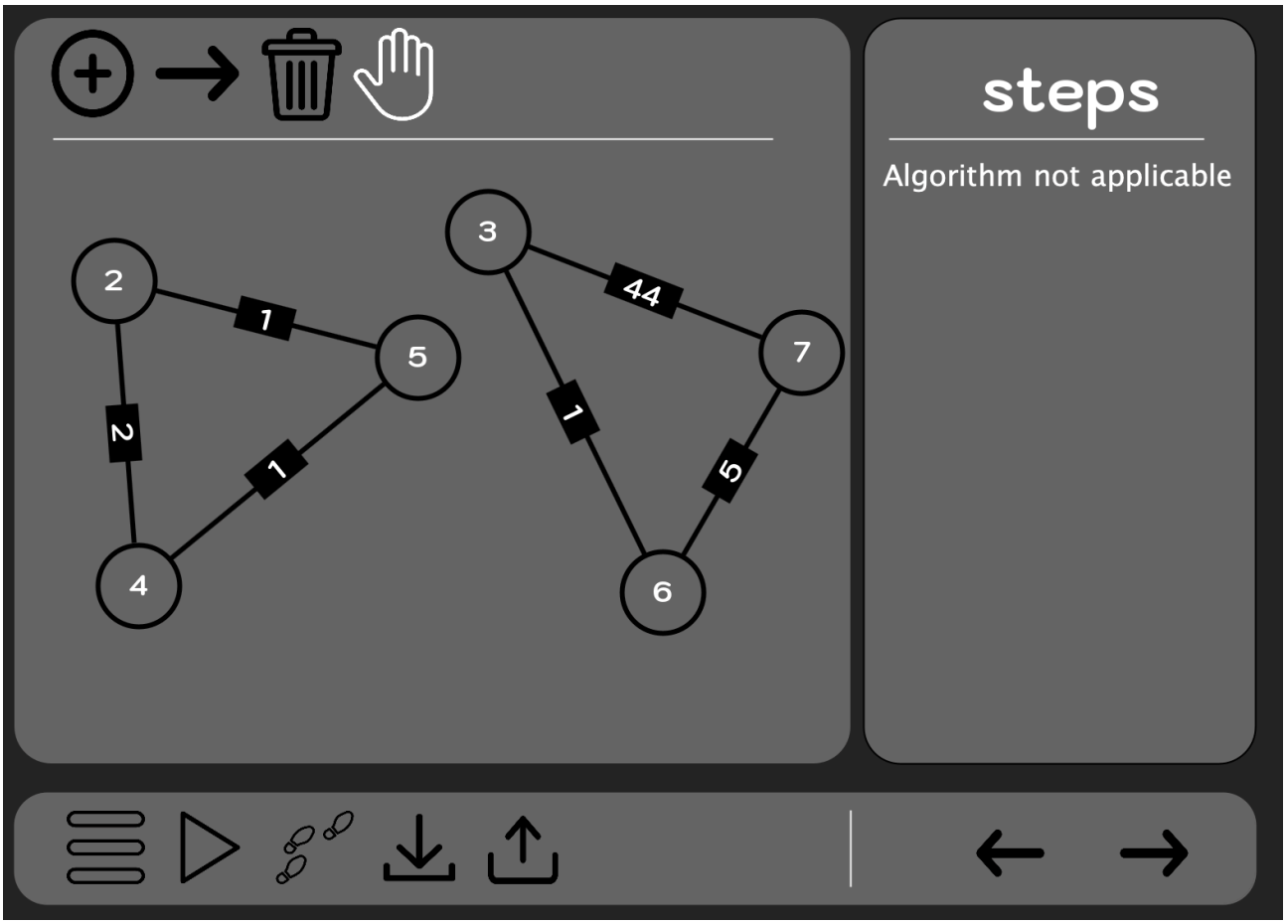

Рисунок 16 - алгоритм не применим к несвязному графу

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД

CalculatorNavigation.java:

```
package com.example;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.RescaleOp;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;


public class CalculatorNavigation extends JComponent {

    private BufferedImage handImage;
    private BufferedImage plusImage;
    private BufferedImage urnImage;
    private BufferedImage arrowImage;

    private BufferedImage handImageHovered;
    private BufferedImage plusImageHovered;
    private BufferedImage urnImageHovered;
    private BufferedImage arrowImageHovered;

    private BufferedImage handImagePressed;
    private BufferedImage plusImagePressed;
    private BufferedImage arrowImagePressed;
    private BufferedImage urnImagePressed;

    public JButton handButton;
    public JButton plusButton;
    public JButton urnButton;
    public JButton arrowButton;

    private boolean handButtonPressed;
    private boolean plusButtonPressed;
    private boolean urnButtonPressed;
    private boolean arrowButtonPressed;
    private boolean handButtonHovered;
    private boolean plusButtonHovered;
    private boolean urnButtonHovered;
    private boolean arrowButtonHovered;


    public CalculatorNavigation() {
        try {
            handImage = ImageIO.read(new File("./src/main/resources/images/hand.png"));
            plusImage = ImageIO.read(new File("./src/main/resources/images/plus.png"));
            urnImage = ImageIO.read(new File("./src/main/resources/images/urn.png"));
            arrowImage = ImageIO.read(new File("./src/main/resources/images/arrow.png"));

            handImageHovered = ImageIO.read(new File("./src/main/resources/images/handgray.png"));
            plusImageHovered = ImageIO.read(new File("./src/main/resources/images/plusgray.png"));
```

```
    urnImageHovered = ImageIO.read(new File("./src/main/resources/images/urngray.png"));
    arrowImageHovered = ImageIO.read(new File("./src/main/resources/images/arrowgray.png"));

    handImagePressed = ImageIO.read(new File("./src/main/resources/images/handwhite.png"));
    plusImagePressed = ImageIO.read(new File("./src/main/resources/images/pluswhite.png"));
    urnImagePressed = ImageIO.read(new File("./src/main/resources/images/urnwhite.png"));
    arrowImagePressed = ImageIO.read(new File("./src/main/resources/images/arrowwhite.png"));

} catch (IOException e) {
    System.out.println("Failed to load image: " + e.getMessage());
}

setLayout(null); // Use null layout for absolute positioning

// Create buttons
handButton = new JButton();
plusButton = new JButton();
urnButton = new JButton();
arrowButton = new JButton();

// Set the buttons to be transparent
handButton.setOpaque(false);
handButton.setContentAreaFilled(false);
handButton.setBorderPainted(false);
handButton.setFocusPainted(false);
handButton.setBorder(BorderFactory.createEmptyBorder());

plusButton.setOpaque(false);
plusButton.setContentAreaFilled(false);
plusButton.setBorderPainted(false);
plusButton.setFocusPainted(false);
plusButton.setBorder(BorderFactory.createEmptyBorder());

urnButton.setOpaque(false);
urnButton.setContentAreaFilled(false);
urnButton.setBorderPainted(false);
urnButton.setFocusPainted(false);
urnButton.setBorder(BorderFactory.createEmptyBorder());

arrowButton.setOpaque(false);
arrowButton.setContentAreaFilled(false);
arrowButton.setBorderPainted(false);
arrowButton.setFocusPainted(false);
arrowButton.setBorder(BorderFactory.createEmptyBorder());

// Add action listeners to buttons
ButtonActionListener buttonListener = new ButtonActionListener();
handButton.addActionListener(buttonListener);
plusButton.addActionListener(buttonListener);
urnButton.addActionListener(buttonListener);
arrowButton.addActionListener(buttonListener);

// Add mouse listeners to buttons for hover effect
ButtonHoverListener hoverListener = new ButtonHoverListener();
handButton.addMouseListener(hoverListener);
plusButton.addMouseListener(hoverListener);
urnButton.addMouseListener(hoverListener);
arrowButton.addMouseListener(hoverListener);



// Add buttons to the component
```

```
    add(handButton);
    add(plusButton);
    add(urnButton);
    add(arrowButton);
}

private class ButtonActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == handButton) {
            handButtonPressed = true;
            plusButtonPressed = false;
            urnButtonPressed = false;
            arrowButtonPressed = false;
        } else if (e.getSource() == plusButton) {
            handButtonPressed = false;
            plusButtonPressed = true;
            urnButtonPressed = false;
            arrowButtonPressed = false;
        } else if (e.getSource() == urnButton) {
            handButtonPressed = false;
            urnButtonPressed = true;
            arrowButtonPressed = false;
            plusButtonPressed = false;
        } else if (e.getSource() == arrowButton) {
            handButtonPressed = false;
            arrowButtonPressed = true;
            plusButtonPressed = false;
            urnButtonPressed = false;
        }
        repaint();
    }
}

private class ButtonHoverListener extends MouseAdapter {
    @Override
    public void mouseEntered(MouseEvent e) {
        if (e.getSource() == handButton) {
            handButtonHovered = true;
        } else if (e.getSource() == plusButton) {
            plusButtonHovered = true;
        } else if (e.getSource() == urnButton) {
            urnButtonHovered = true;
        } else if (e.getSource() == arrowButton) {
            arrowButtonHovered = true;
        }
        repaint();
    }

    @Override
    public void mouseExited(MouseEvent e) {
        if (e.getSource() == handButton) {
            handButtonHovered = false;
        } else if (e.getSource() == plusButton) {
            plusButtonHovered = false;
        } else if (e.getSource() == urnButton) {
            urnButtonHovered = false;
        } else if (e.getSource() == arrowButton) {
            arrowButtonHovered = false;
        }
        repaint();
    }
```

```
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);
        g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BICUBIC);

        int width_screen = getWidth();
        int height_screen = getHeight();

        int handWidth = handImage.getWidth();
        int handHeight = handImage.getHeight();

        int plusWidth = plusImage.getWidth();
        int plusHeight = plusImage.getHeight();

        int urnWidth = urnImage.getWidth();
        int urnHeight = urnImage.getHeight();

        int arrowWidth = arrowImage.getWidth();
        int arrowHeight = arrowImage.getHeight();

        int newHandWidth = (8 * width_screen/100);
        int newHandHeight = (10*height_screen/100);

        int newPlusWidth = (8*width_screen/100);
        int newPlusHeight = (12*height_screen / 100);


        int newUrnWidth = (8 * width_screen/100);
        int newUrnHeight = (10* height_screen/100);

        int newArrowWidth = (8 * width_screen/100);
        int newArrowHeight = (12 * height_screen/ 100);



        int xHand = (27 * width_screen) / 100;
        int yHand = (3*height_screen) / 100;

        int xPlus = (4 * width_screen) / 100;
        int yPlus = (2 * height_screen) / 100;

        int xArrow = (12 * width_screen) / 100;
        int yArrow = (2 * height_screen) / 100;

        int xUrn = (20 * width_screen) / 100;
        int yUrn = (3*height_screen) / 100;

        // Масштабируем изображение с помощью AffineTransform
        AffineTransform atHand = new AffineTransform();
        atHand.translate(xHand, yHand);
        atHand.scale((double) newHandWidth / handWidth, (double) newHandHeight / handHeight);

        AffineTransform atPlus = new AffineTransform();
        atPlus.translate(xPlus, yPlus);
        atPlus.scale((double) newPlusWidth / plusWidth, (double) newPlusHeight / plusHeight);
```

```
        AffineTransform atArrow = new AffineTransform();
        atArrow.translate(xArrow, yArrow);
        atArrow.scale((double) newArrowWidth / arrowWidth, (double) newArrowHeight / arrowHeight);

        AffineTransform atUrn = new AffineTransform();
        atUrn.translate(xUrn, yUrn);
        atUrn.scale((double) newUrnWidth / urnWidth, (double) newUrnHeight / urnHeight);

        BufferedImage handImageToDraw = handImage;
        BufferedImage plusImageToDraw = plusImage;
        BufferedImage arrowImageToDraw = arrowImage;
        BufferedImage urnImageToDraw = urnImage;

        if (handButtonPressed) {
            handImageToDraw = handImagePressed;
        } else if (handButtonHovered) {
            handImageToDraw = handImageHovered;
        }

        if (plusButtonPressed) {
            plusImageToDraw = plusImagePressed;
        } else if (plusButtonHovered) {
            plusImageToDraw = plusImageHovered;
        }

        if (arrowButtonPressed) {
            arrowImageToDraw = arrowImagePressed;
        } else if (arrowButtonHovered) {
            arrowImageToDraw = arrowImageHovered;
        }

        if (urnButtonPressed) {
            urnImageToDraw = urnImagePressed;
        } else if (urnButtonHovered) {
            urnImageToDraw = urnImageHovered;
        }

        g2d.drawRenderedImage(plusImageToDraw, atPlus);
        g2d.drawRenderedImage(urnImageToDraw, atUrn);
        g2d.drawRenderedImage(arrowImageToDraw, atArrow);
        g2d.drawRenderedImage(handImageToDraw, atHand);

        // Update button positions and sizes
        handButton.setBounds(xHand, yHand, newHandWidth, newHandHeight);
        plusButton.setBounds(xPlus, yPlus, newPlusWidth, newPlusHeight);
        arrowButton.setBounds(xArrow, yArrow, newArrowWidth, newArrowHeight);
        urnButton.setBounds(xUrn, yUrn, newUrnWidth, newUrnHeight);
    }

//    private static BufferedImage changeImageColor(BufferedImage src, Color color) {
//        BufferedImage result = new BufferedImage(src.getWidth(), src.getHeight(), BufferedImage.TYPE_INT_ARGB);
//
//
//        // Rescale operation to change the image color
//        float[] scales = { color.getRed()/255f, color.getGreen()/255f, color.getBlue()/255f, color.getAlpha()/255f};
//        float[] offsets = {255f, 255f, 255f, 0f};
//
//
//        RescaleOp op = new RescaleOp(scales, offsets, null);
//        op.filter(src, result);
//
//        return result;
```

```
//    }

    public boolean isHandButtonPressed() {
        return handButtonPressed;
    }

    public boolean isPlusButtonPressed() {
        return plusButtonPressed;
    }

    public boolean isUrnButtonPressed() {
        return urnButtonPressed;
    }

    public boolean isArrowButtonPressed() {
        return arrowButtonPressed;
    }

}
```

DelimiterLineCalculator.java:

```
package com.example;

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

public class DelimiterLineCalculator extends JComponent {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setColor(Color.WHITE);

        int screen_width = getWidth();
        int screen_height = getHeight();


        g2d.setStroke(new BasicStroke(1));
        g2d.drawLine(5*getWidth()/100, 15*getHeight()/100, 60*getWidth()/100, 15*getHeight()/100);
    }
}
```

DelimiterLineSteps.java:

```
package com.example;

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

public class DelimiterLineSteps extends JComponent {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setColor(Color.WHITE);

        int screen_width = getWidth();
        int screen_height = getHeight();
```

```
        g2d.setStroke(new BasicStroke(1));
        g2d.drawLine(69*getWidth()/100, 15*getHeight()/100, 93*getWidth()/100, 15*getHeight()/100);
    }
}
```

DelimiterLineToolBar.java:

```
package com.example;

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

public class DelimiterLineToolBar extends JComponent {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setColor(Color.WHITE);

        int screen_width = getWidth();
        int screen_height = getHeight();


        g2d.setStroke(new BasicStroke(1));
        g2d.drawLine(66*getWidth()/100, 87*getHeight()/100, 66*getWidth()/100, 95*getHeight()/100);
    }
}
```

Edge.java:

```
package com.example;

import java.awt.*;
import java.io.Serializable;

public class Edge {
    Vertex start;
    Vertex end;
    String colorE = "Black";
    int weight;
    int screenWidth;
    int screenHeight;
    private boolean isInMst = false;

    Edge(Vertex start, Vertex end, int weight, int screenWidth, int screenHeight) {
        this.start = start;
        this.end = end;
        this.weight = weight;
    }

    void updateAbsoluteCoordinates(int screenWidth, int screenHeight) {
        // Обновляем только координаты стартовой и конечной вершины
        this.start.updateAbsoluteCoordinates(screenWidth, screenHeight);
        this.end.updateAbsoluteCoordinates(screenWidth, screenHeight);
    }

    Point getStart() {
        return start.getAbsoluteLocation(screenWidth, screenHeight);
    }

    Point getEnd() {
```

```
        return end.getAbsoluteLocation(screenWidth, screenHeight);
    }

    double ptSegDist(Point p) {
        Point startLoc = start.getAbsoluteLocation(screenWidth, screenHeight);
        Point endLoc = end.getAbsoluteLocation(screenWidth, screenHeight);

        double dx = endLoc.x - startLoc.x;
        double dy = endLoc.y - startLoc.y;
        double len = Math.sqrt(dx * dx + dy * dy);
        if (len == 0.0) {
            return p.distance(startLoc);
        }
        double t = ((p.x - startLoc.x) * dx + (p.y - startLoc.y) * dy) / (len * len);
        if (t < 0.0) {
            return p.distance(startLoc);
        }
        if (t > 1.0) {
            return p.distance(endLoc);
        }
        double closestX = startLoc.x + t * dx;
        double closestY = startLoc.y + t * dy;
        return p.distance(closestX, closestY);
    }
    Vertex getStartVertex() {
        return start;
    }
    Vertex getEndVertex() {
        return end;
    }
    int getWeight() {
        return weight;
    }

    String getColorE() {
        return colorE;
    }

    void setColorE(String colorE) {
        this.colorE = colorE;
    }

    public void setInMst(boolean isInMst) {
        this.isInMst = isInMst;
    }

    public boolean isInMst() {
        return isInMst;
    }

}
```

Graph.java:

```
package com.example;

import java.awt.*;
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.*;

public class Graph implements Serializable {
```

```
    private ArrayList<Vertex> vertices = new ArrayList<>();
    private ArrayList<Edge> edges = new ArrayList<>();

    public Graph(ArrayList<Vertex>verticesIn, ArrayList<Edge>edgesIn) {
        this.vertices = verticesIn;
        this.edges = edgesIn;
    }

    public List<Vertex> getVertices() {
        return vertices;
    }

    public List<Edge> getEdges() {
        return edges;
    }

    public ArrayList<Edge> runPrimsAlgorithm() {
        if (vertices.isEmpty()) {
            return new ArrayList<>();
        }

        ArrayList<Edge> mst = new ArrayList<>();
        Set<Vertex> visited = new HashSet<>();
        ArrayList<Edge> availableEdges = new ArrayList<>();

        Vertex startVertex = vertices.get(0);
        visited.add(startVertex);

        while (visited.size() < vertices.size()) {
            availableEdges.clear(); // Очищаем список доступных рёбер для каждой итерации

            for (Vertex vertex : visited) {
                for (Edge edge : edges) {
                    if ((edge.getStartVertex().equals(vertex) && !visited.contains(edge.getEndVertex())) ||
                            (edge.getEndVertex().equals(vertex) && !visited.contains(edge.getStartVertex()))) {
                        availableEdges.add(edge);
                    }
                }
            }

            Edge minEdge = null;
            for (Edge edge : availableEdges) {
                if (minEdge == null || edge.getWeight() < minEdge.getWeight()) {
                    minEdge = edge;
                }
            }

            if (minEdge != null) {
                mst.add(minEdge);
                Vertex newVertex = visited.contains(minEdge.getStartVertex()) ? minEdge.getEndVertex() :
minEdge.getStartVertex();
                visited.add(newVertex);
            } else {
                break; // Если нет доступных рёбер, алгоритм завершен
            }
        }

        return mst;
    }
}
```

Main.java:

```
package com.example;

public class Main {
    public static void main(String[] args) {
        MainWindow.run();
    }
}
```

MainWindow.java:

```
package com.example;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MainWindow {
    private static final Logger LOGGER = Logger.getLogger(MainWindow.class.getName());

    public static void run() {
        JFrame frame = new JFrame();
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension screenSize = toolkit.getScreenSize(); // Getting screen size

        Color backgroundColor = new Color(37, 37, 37);  // background color
        // frame size and bounds
        frame.setBounds(screenSize.width / 2 - (screenSize.width * 55) / 200, screenSize.height / 2 - (screenSize.height *
58) / 200, (screenSize.width * 55) / 100, (screenSize.height * 58) / 100);
        frame.setMinimumSize(new Dimension(800, 600));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        // Path to the font file
        String fontPath = "./src/main/resources/fonts/Kodchasan/Kodchasan-Bold.ttf";

        try {
            // Load the custom font
            Font customFont = Font.createFont(Font.TRUETYPE_FONT, new File(fontPath)).deriveFont(24f);

            // Register the font
            GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
            ge.registerFont(customFont);

        } catch (FontFormatException | IOException e) {
            LOGGER.log(Level.SEVERE, "Failed to load custom font", e);
        }

        // Create a panel with OverlayLayout to hold the components
        JLayeredPane overlayPanel = new JLayeredPane();
        overlayPanel.setLayout(new OverlayLayout(overlayPanel));

        JPanel backgroundPanel = new JPanel();
        backgroundPanel.setBackground(backgroundColor);
        overlayPanel.add(backgroundPanel, JLayeredPane.DEFAULT_LAYER);

        CalculatorNavigation calculatorNavigation = new CalculatorNavigation();
        RoundedRectangleCalculatorComponent roundedRectangleCalculatorComponent = new
RoundedRectangleCalculatorComponent(calculatorNavigation);
```

```
        ToolBarItems toolBarItems = new ToolBarItems(roundedRectangleCalculatorComponent);
        RoundedRectangleStepComponent roundedRectangleStepComponent = new RoundedRectangleStepComponent();
        RoundedRectangleTollBarComponent roundedRectangleTollBarComponent = new
RoundedRectangleTollBarComponent();
        DelimiterLineCalculator delimiterLineCalculatorLine = new DelimiterLineCalculator();
        DelimiterLineSteps delimiterLineSteps = new DelimiterLineSteps();
        DelimiterLineToolBar delimiterLineToolBar = new DelimiterLineToolBar();
        StepsText stepsText = new StepsText();

        overlayPanel.add(roundedRectangleCalculatorComponent, JLayeredPane.PALETTE_LAYER);
        overlayPanel.add(roundedRectangleStepComponent, JLayeredPane.PALETTE_LAYER);
        overlayPanel.add(roundedRectangleTollBarComponent, JLayeredPane.PALETTE_LAYER);
        overlayPanel.add(delimiterLineCalculatorLine, JLayeredPane.MODAL_LAYER);
        overlayPanel.add(delimiterLineSteps, JLayeredPane.MODAL_LAYER);
        overlayPanel.add(delimiterLineToolBar, JLayeredPane.MODAL_LAYER);
        overlayPanel.add(calculatorNavigation, JLayeredPane.MODAL_LAYER);
        overlayPanel.add(toolBarItems, JLayeredPane.MODAL_LAYER);
        overlayPanel.add(stepsText, JLayeredPane.MODAL_LAYER);



        // Add the panel to the frame
        frame.add(overlayPanel);

        frame.setVisible(true);
    }


}
```

RoundedRectangleCalculatorComponent.java:

```
package com.example;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.AffineTransform;
import java.util.ArrayList;
import java.io.*;

public class RoundedRectangleCalculatorComponent extends JComponent {

    private ArrayList<Vertex> vertices = new ArrayList<>();
    private ArrayList<Edge> edges = new ArrayList<>();
    private ArrayList<String> steps = new ArrayList<>();
    public Graph graph;
    public ArrayList<Edge> mst = new ArrayList<>();;
    private Vertex hoveredVertex = null;
    private Vertex selectedVertex = null;
    private Edge selectedEdge = null;
    private Edge hoveredEdge = null;
    private CalculatorNavigation navigation;
    private ToolBarItems toolBarItems;
    private int vertexCounter = 0;
    private static final int VERTEX_RADIUS = 25;
    private static final double EDGE_ADJUSTMENT_FACTOR = 1.1;
    public boolean actionMenu = false;
    public boolean actionRun = false;
    public boolean actionSteps = false;
    public boolean actionDownload = false;
    public boolean actionSave = false;
```

```
    public boolean actionPreviousStep = false;
    public boolean actionNextStep = false;
    public int num = 0;
    private boolean algPrimaSelected = false;

    public RoundedRectangleCalculatorComponent(CalculatorNavigation navigation) {
        this.navigation = navigation;


        addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                handleMousePressed(e);
            }

            @Override
            public void mouseReleased(MouseEvent e) {
                handleMouseReleased(e);
            }
        });

        addMouseMotionListener(new MouseAdapter() {
            @Override
            public void mouseDragged(MouseEvent e) {
                handleMouseDragged(e);
            }

            @Override
            public void mouseMoved(MouseEvent e) {
                handleMouseMoved(e);
            }
        });

        addComponentListener(new java.awt.event.ComponentAdapter() {
            public void componentResized(java.awt.event.ComponentEvent evt) {
                handleResize();
            }
        });
    }

    private void handleMousePressed(MouseEvent e) {
        int screenWidth = getWidth();
        int screenHeight = getHeight();

        if (e.getSource() == navigation.plusButton || e.getSource() == navigation.urnButton || e.getSource() ==
navigation.arrowButton) {
            resetMst();
        }

        if (navigation.isPlusButtonPressed()) {
            if (e.getX() > (int) (getWidth() * 5 / 100) && e.getY() > (int) (19.5 * getHeight() / 100) && (e.getX() < (int)
(getWidth() * 62.5 / 100) && e.getY() < (int) (79.5 * getHeight() / 100))) {
                Vertex vertex = new Vertex(e.getPoint(), ++vertexCounter, screenWidth, screenHeight);
                vertices.add(vertex);
                repaint();
            }
        } else if (navigation.isUrnButtonPressed()) {
            resetMst();
            Vertex vertexToRemove = null;
            for (Vertex vertex : vertices) {
                if (vertex.contains(e.getPoint(), screenWidth, screenHeight)) {
                    vertexToRemove = vertex;
```

```
                    break;
                }
            }
            if (vertexToRemove != null) {
                final Vertex finalVertexToRemove = vertexToRemove; // Делаем final
                vertices.remove(finalVertexToRemove);
                edges.removeIf(edge -> edge.start.equals(finalVertexToRemove) || edge.end.equals(finalVertexToRemove));
                repaint();
            } else {
                Edge edgeToRemove = null;
                for (Edge edge : edges) {
                    if (isPointOnEdge(e.getPoint(), edge)) {
                        edgeToRemove = edge;
                        break;
                    }
                }
                if (edgeToRemove != null) {
                    edges.remove(edgeToRemove);
                    repaint();
                }
            }
        } else if (navigation.isHandButtonPressed()) {
            for (Vertex vertex : vertices) {
                if (vertex.contains(e.getPoint(), screenWidth, screenHeight)) {
                    selectedVertex = vertex;
                    break;
                }
            }
        } else if (navigation.isArrowButtonPressed()) {
            resetMst();
            boolean vertexClicked = false;
            for (Vertex vertex : vertices) {
                if (vertex.contains(e.getPoint(), screenWidth, screenHeight)) {
                    if (selectedVertex == null) {
                        selectedVertex = vertex;
                        vertexClicked = true; // Указываем, что была найдена вершина
                        break;
                    } else {
                        int weight = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter weight for the edge:", "Edge
Weight", JOptionPane.PLAIN_MESSAGE));
                        edges.add(new Edge(selectedVertex, vertex, weight, screenWidth, screenHeight));
                        selectedVertex = null;
                        repaint();
                        vertexClicked = true; // Указываем, что была найдена вершина
                        break;
                    }
                }
            }

            // Если не было найдено ни одной вершины при клике, просто выходим из условия
            if (!vertexClicked) {
                selectedVertex = null;
            }
        } else {
            selectedVertex = null;
            resetMst(); // Сбрасываем MST при нажатии на пустое место
        }
    }



    private void handleMouseReleased(MouseEvent e) {
```

```
    if (navigation.isHandButtonPressed()) {
        selectedVertex = null;
    } else {
        resetMst(); // Сбрасываем MST при отпускании мыши, если не выбрана кнопка "Рука"
    }
}


private void handleMouseDragged(MouseEvent e) {
    int screenWidth = getWidth();
    int screenHeight = getHeight();

    if (selectedVertex != null && navigation.isHandButtonPressed()
            && e.getX() > (getWidth() * 5 / 100) && e.getY() > (int) (19.5 * getHeight() / 100)
            && (e.getX() < (int) (getWidth() * 62.5 / 100) && e.getY() < (int) (79.5 * getHeight() / 100))) {

        selectedVertex.setLocation(e.getPoint(), screenWidth, screenHeight);

        // Обновляем только координаты рёбер, которые связаны с перемещаемой вершиной
        for (Edge edge : edges) {
            if (edge.start.equals(selectedVertex) || edge.end.equals(selectedVertex)) {
                edge.updateAbsoluteCoordinates(screenWidth, screenHeight);
            }
        }
        repaint();
    }
}


private void handleMouseMoved(MouseEvent e) {
    int screenWidth = getWidth();
    int screenHeight = getHeight();

    hoveredVertex = null;
    hoveredEdge = null;

    for (Vertex vertex : vertices) {
        if (vertex.contains(e.getPoint(), screenWidth, screenHeight)) {
            hoveredVertex = vertex;
            break;
        }
    }

    if (hoveredVertex == null) {
        for (Edge edge : edges) {
            if (isPointOnEdge(e.getPoint(), edge)) {
                hoveredEdge = edge;
                break;
            }
        }
    }

    repaint();
}


private void handleResize() {
    int screenWidth = getWidth();
    int screenHeight = getHeight();

    for (Vertex vertex : vertices) {
        vertex.updateAbsoluteCoordinates(screenWidth, screenHeight);
```

```
    }

    for (Edge edge : edges) {
        edge.updateAbsoluteCoordinates(screenWidth, screenHeight);
    }

    repaint();
}


private boolean isPointOnEdge(Point p, Edge edge) {
    Point start = calculateIntersection(edge.start, edge.end);
    Point end = calculateIntersection(edge.end, edge.start);

    double distance = ptSegDist(start.x, start.y, end.x, end.y, p.x, p.y);
    return distance <= 3.0;
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
    g2d.setFont(new Font("Kodchasan", Font.BOLD, 18));

    int width_screen = getWidth();
    int height_screen = getHeight();

    int x = (2 * width_screen) / 100;
    int y = (2 * height_screen) / 100;

    int width = (64 * width_screen) / 100;
    int height = (80 * height_screen) / 100;

    int arcWidth = 45;
    int arcHeight = 45;

    g2d.setColor(new Color(102, 101, 101)); // Полупрозрачный цвет фона
    g2d.fillRoundRect(x, y, width, height, arcWidth, arcHeight);

    for (Edge edge : edges) {
        Point start = calculateIntersection(edge.start, edge.end);
        Point end = calculateIntersection(edge.end, edge.start);

        if (edge.equals(hoveredEdge)) {
            g2d.setColor(Color.LIGHT_GRAY);
        } else if (edge.isInMst()) {
            g2d.setColor(Color.WHITE); // Рисуем MST рёбра белым цветом
        } else {
            g2d.setColor(Color.BLACK);
        }
        g2d.setStroke(new BasicStroke(3));
        g2d.drawLine(start.x, start.y, end.x, end.y);

        if (edge.weight > 0) {
            int rectWidth = calculateRectWidth(edge.weight);
            int rectHeight = 20;
            int rectX = (start.x + end.x - rectWidth) / 2;
            int rectY = (start.y + end.y - rectHeight) / 2;

            double angle = Math.atan2(end.y - start.y, end.x - start.x);
```

```
            if (angle > Math.PI / 2 || angle < -Math.PI / 2) {
                angle += Math.PI;
            }

            AffineTransform originalTransform = g2d.getTransform();

            g2d.rotate(angle, rectX + rectWidth / 2, rectY + rectHeight / 2);
            if (edge.equals(hoveredEdge)) {
                g2d.setColor(Color.LIGHT_GRAY);
                g2d.fillRect(rectX, rectY, rectWidth, rectHeight);

                g2d.setColor(Color.BLACK);
                FontMetrics fm = g2d.getFontMetrics();
                String weightStr = String.valueOf(edge.weight);
                int textWidth = fm.stringWidth(weightStr);
                int textHeight = fm.getHeight();
                g2d.drawString(weightStr, rectX + (rectWidth - textWidth) / 2, rectY + (rectHeight + textHeight) / 2 - 5);
            } else if (edge.isInMst()) {
                g2d.setColor(Color.WHITE);
                g2d.fillRect(rectX, rectY, rectWidth, rectHeight);

                g2d.setColor(Color.BLACK);
                FontMetrics fm = g2d.getFontMetrics();
                String weightStr = String.valueOf(edge.weight);
                int textWidth = fm.stringWidth(weightStr);
                int textHeight = fm.getHeight();
                g2d.drawString(weightStr, rectX + (rectWidth - textWidth) / 2, rectY + (rectHeight + textHeight) / 2 - 5);
            } else {
                g2d.setColor(Color.BLACK);
                g2d.fillRect(rectX, rectY, rectWidth, rectHeight);

                g2d.setColor(Color.WHITE);
                FontMetrics fm = g2d.getFontMetrics();
                String weightStr = String.valueOf(edge.weight);
                int textWidth = fm.stringWidth(weightStr);
                int textHeight = fm.getHeight();
                g2d.drawString(weightStr, rectX + (rectWidth - textWidth) / 2, rectY + (rectHeight + textHeight) / 2 - 5);
            }

            g2d.setTransform(originalTransform);
        }
    }

    for (Vertex vertex : vertices) {
        Point absoluteLocation = vertex.getAbsoluteLocation(getWidth(), getHeight());
        if (vertex.equals(hoveredVertex) || vertex.equals(selectedVertex)) {
            g2d.setColor(Color.LIGHT_GRAY);
        } else if (isVertexInMst(vertex)) {
            g2d.setColor(Color.WHITE);
        } else {
            g2d.setColor(Color.BLACK);
        }
        g2d.setStroke(new BasicStroke(3));
        g2d.drawOval(absoluteLocation.x - VERTEX_RADIUS, absoluteLocation.y - VERTEX_RADIUS,
VERTEX_RADIUS * 2, VERTEX_RADIUS * 2);


        g2d.setColor(Color.WHITE);
        FontMetrics fm = g2d.getFontMetrics();
        String numberStr = String.valueOf(vertex.number);
        int textWidth = fm.stringWidth(numberStr);
        int textHeight = fm.getHeight();
```

```
            g2d.drawString(numberStr, absoluteLocation.x - (textWidth / 2), absoluteLocation.y + (textHeight / 4));
        }

        g2d.setColor(Color.WHITE);
        g2d.setFont(new Font("Kodchasan-Bold", Font.PLAIN, 18));
        int yPosition = 20*height_screen/100;  // Начальная вертикальная позиция для текста
        for (String step : steps) {
            g2d.drawString(step, (int) (68.5*width_screen/100), yPosition);  // Рисование строки на экране
            yPosition += 20;  // Смещение вниз для следующей строки
        }
    }



    private Point calculateIntersection(Vertex v1, Vertex v2) {
        Point v1Loc = v1.getAbsoluteLocation(getWidth(), getHeight());
        Point v2Loc = v2.getAbsoluteLocation(getWidth(), getHeight());

        double dx = v2Loc.x - v1Loc.x;
        double dy = v2Loc.y - v1Loc.y;
        double dist = Math.sqrt(dx * dx + dy * dy);
        double scale = VERTEX_RADIUS * EDGE_ADJUSTMENT_FACTOR / dist;
        return new Point((int) (v1Loc.x + dx * scale), (int) (v1Loc.y + dy * scale));
    }

    private int calculateRectWidth(int weight) {
        int digits = String.valueOf(weight).length();
        return digits * 10 + 25;
    }

    public void showAlgorithmSelectionDialog() {
        String[] algorithms = {"Prim's Algorithm"};
        String selectedAlgorithm = (String) JOptionPane.showInputDialog(
                null,
                "Select the algorithm to use:",
                "Algorithm Selection",
                JOptionPane.QUESTION_MESSAGE,
                null,
                algorithms,
                algorithms[0]
        );

        if (selectedAlgorithm != null) {
            JOptionPane.showMessageDialog(null, "You selected: " + selectedAlgorithm);
            // Process the selected algorithm (for now just Prim's Algorithm)
            if (selectedAlgorithm.equals("Prim's Algorithm")) {
                // Call the function that runs Prim's Algorithm
                algPrimaSelected = true;
            }
        }
    }

    public void saveGraphToFile(String filename) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
            // Сохраняем количество вершин
            writer.write(vertices.size() + "\n");
            for (Vertex vertex : vertices) {
                writer.write(vertex.number + " " + vertex.relativeX + " " + vertex.relativeY + "\n");
            }
            // Сохраняем количество рёбер
            writer.write(edges.size() + "\n");
            for (Edge edge : edges) {
```

```
            writer.write(edge.start.number + " " + edge.end.number + " " + edge.weight + "\n");
        }
        writer.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void loadGraphFromFile(String filename) {
    try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
        vertices.clear();
        edges.clear();

        // Считываем количество вершин
        int vertexCount = Integer.parseInt(reader.readLine().trim());
        for (int i = 0; i < vertexCount; i++) {
            String[] vertexData = reader.readLine().split(" ");
            int number = Integer.parseInt(vertexData[0]);
            double relativeX = Double.parseDouble(vertexData[1]);
            double relativeY = Double.parseDouble(vertexData[2]);
            Vertex vertex = new Vertex(new Point((int)(relativeX * getWidth()), (int)(relativeY * getHeight())), number,
getWidth(), getHeight());
            vertices.add(vertex);

        }

        // Считываем количество рёбер
        int edgeCount = Integer.parseInt(reader.readLine().trim());
        for (int i = 0; i < edgeCount; i++) {
            String[] edgeData = reader.readLine().split(" ");
            int startNumber = Integer.parseInt(edgeData[0]);
            int endNumber = Integer.parseInt(edgeData[1]);
            int weight = Integer.parseInt(edgeData[2]);
            Vertex startVertex = findVertexByNumber(startNumber);
            Vertex endVertex = findVertexByNumber(endNumber);
            if (startVertex != null && endVertex != null) {
                edges.add(new Edge(startVertex, endVertex, weight, getWidth(), getHeight()));
            }
        }
        vertexCounter = vertices.get(vertices.size()-1).getNumber();
        repaint();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private Vertex findVertexByNumber(int number) {
    for (Vertex vertex : vertices) {
        if (vertex.number == number) {
            return vertex;
        }
    }
    return null;
}

public Graph getGraph(){
    return graph = new Graph(this.vertices, this.edges);
}

void setMst(ArrayList<Edge> edges){
    mst = edges;
    steps.clear();
```

```
        for (Edge edge : this.edges) {
            edge.setInMst(false); // Сначала сбросим поле для всех рёбер
        }
        for (Edge edge : mst) {
            edge.setInMst(true); // Установим поле для рёбер из MST
        }
        String start = String.format("Algorithm started!");
        steps.add(start);
        for (int i = 0; i < mst.size(); i++) {
            String step = String.format("Step №%d: Edge: %d - %d", i+1, mst.get(i).getStartVertex().getNumber(),
mst.get(i).getEndVertex().getNumber());
            steps.add(step);  // Сохраняем шаги
        }
        if (mst.size() == vertices.size()-1){
            String end = String.format("Algorithm ended!");
            steps.add(end);
        }
        repaint(); // Перерисовать компонент для отображения изменений
    }

    private boolean isVertexInMst(Vertex vertex) {
        for (Edge edge : mst) {
            if (edge.start.equals(vertex) || edge.end.equals(vertex)) {
                return true;
            }
        }
        return false;
    }

    public void resetMst() {
        mst.clear();
        for (Edge edge : edges) {
            edge.setInMst(false);
        }
        repaint();
    }


    public static double ptSegDist(double x1, double y1, double x2, double y2, double px, double py) {
        double x2x1 = x2 - x1;
        double y2y1 = y2 - y1;
        double x1px = x1 - px;
        double y1py = y1 - py;
        double dotprod = x1px * x2x1 + y1py * y2y1;
        double projlenSq = dotprod * dotprod / (x2x1 * x2x1 + y2y1 * y2y1);
        double lenSq = x1px * x1px + y1py * y1py - projlenSq;
        if (projlenSq < 0.0) {
            return Math.sqrt(x1px * x1px + y1py * y1py);
        } else if (projlenSq > x2x1 * x2x1 + y2y1 * y2y1) {
            double x2px = x2 - px;
            double y2py = y2 - py;
            return Math.sqrt(x2px * x2px + y2py * y2py);
        }
        return Math.sqrt(lenSq);
    }

}
```

RoundedRectangleStepComponent.java:

```
package com.example;

import javax.swing.*;
```

```
import java.awt.*;

public class RoundedRectangleStepComponent extends JComponent {

    private static final int PANEL_WIDTH_PERCENTAGE = 30;
    private static final int PANEL_HEIGHT_PERCENTAGE = 80;

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        int width_screen = getWidth();
        int height_screen = getHeight();

        int x = (67 * width_screen) / 100;
        int y = (2 * height_screen) / 100;

        int width = (PANEL_WIDTH_PERCENTAGE * width_screen) / 100;
        int height = (PANEL_HEIGHT_PERCENTAGE * height_screen) / 100;

        int arcWidth = 45;
        int arcHeight = 45;

        g2d.setColor(new Color(102, 101, 101)); // Semi-transparent fill color
        g2d.fillRoundRect(x, y, width, height, arcWidth, arcHeight);

        g2d.setColor(Color.BLACK);
        g2d.drawRoundRect(x, y, width, height, arcWidth, arcHeight);
    }

    @Override
    public Dimension getPreferredSize() {
        return new Dimension((PANEL_WIDTH_PERCENTAGE + 67) * getWidth() / 100,
(PANEL_HEIGHT_PERCENTAGE + 2) * getHeight() / 100);
    }
}
```

RoundedRectangleToolBarComponent.java:

```
package com.example;

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

public class RoundedRectangleTollBarComponent extends JComponent {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        int width_screen = getWidth();
        int height_screen = getHeight();

        int x = (2 * width_screen) / 100;
        int y = (85 * height_screen) / 100;
        int width = (95 * width_screen) / 100;
        int height = (12 * height_screen) / 100;
        int arcWidth = 40;
        int arcHeight = 40;
```

```
        g2d.setColor(new Color(102, 101, 101)); // Semi-transparent fill color
        g2d.fillRoundRect(x, y, width, height, arcWidth, arcHeight);


        g2d.drawRoundRect(x, y, width, height, arcWidth, arcHeight);
    }
}
```

StepsText.java:

```
package com.example;

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

public class StepsText extends JComponent {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        int width_screen = getWidth();
        int height_screen = getHeight();

        g2d.setColor(Color.WHITE);
        int fontSize = Math.min(width_screen, height_screen) / 15;
        g2d.setFont(new Font("Kodchasan", Font.BOLD, fontSize));
        g2d.drawString("steps", (76*width_screen)/100, (12*height_screen)/100);
    }
}
```

ToolBarItems.java:

```
package com.example;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.Line2D;
import java.awt.image.RenderedImage;
import java.awt.image.RescaleOp;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import java.awt.image.AffineTransformOp;
import java.util.ArrayList;


public class ToolBarItems extends JComponent {
    private BufferedImage menuImage;
    private BufferedImage runImage;
    private BufferedImage stepsImage;
    private BufferedImage downloadImage;
    private BufferedImage uploadImage;
    private BufferedImage rightArrowImage;
    private BufferedImage leftArrowImage;
```

```
private BufferedImage menuImageHovered;
private BufferedImage runImageHovered;
private BufferedImage stepsImageHovered;
private BufferedImage downloadImageHovered;
private BufferedImage uploadImageHovered;
private BufferedImage rightArrowImageHovered;
private BufferedImage leftArrowImageHovered;


private BufferedImage menuImagePressed;
private BufferedImage runImagePressed;
private BufferedImage stepsImagePressed;
private BufferedImage downloadImagePressed;
private BufferedImage uploadImagePressed;
private BufferedImage rightArrowImagePressed;
private BufferedImage leftArrowImagePressed;

private JButton menuButton;
private JButton runButton;
private JButton stepsButton;
private JButton downloadButton;
private JButton uploadButton;
private JButton rightArrowButton;
private JButton leftArrowButton;

private boolean menuButtonPressed = false;
private boolean runButtonPressed = false;
private boolean stepsButtonPressed = false;
private boolean downloadButtonPressed = false;
private boolean uploadButtonPressed = false;
private boolean rightArrowButtonPressed = false;
private boolean leftArrowButtonPressed = false;

private boolean menuButtonHovered = false;
private boolean runButtonHovered = false;
private boolean stepsButtonHovered = false;
private boolean downloadButtonHovered = false;
private boolean uploadButtonHovered = false;
private boolean rightArrowButtonHovered = false;
private boolean leftArrowButtonHovered = false;

private int currentIteration = 0;
private ArrayList<Edge> result = new ArrayList<>();
private boolean stepsClicked = false;


private final RoundedRectangleCalculatorComponent roundedRectangleCalculatorComponent;

public ToolBarItems(RoundedRectangleCalculatorComponent roundedRectangleCalculatorComponent) {
   try {
      menuImage = ImageIO.read(new File("./src/main/resources/images/menu.png"));
      runImage = ImageIO.read(new File("./src/main/resources/images/run.png"));
      stepsImage = ImageIO.read(new File("./src/main/resources/images/steps.png"));
      downloadImage = ImageIO.read(new File("./src/main/resources/images/download.png"));
      uploadImage = ImageIO.read(new File("./src/main/resources/images/upload.png"));
      rightArrowImage = ImageIO.read(new File("./src/main/resources/images/arrow.png"));
      leftArrowImage = ImageIO.read(new File("./src/main/resources/images/arrow.png"));

      menuImageHovered = ImageIO.read(new File("./src/main/resources/images/menugray.png"));
      runImageHovered = ImageIO.read(new File("./src/main/resources/images/rungray.png"));
      stepsImageHovered = ImageIO.read(new File("./src/main/resources/images/stepsgray.png"));
```

```
    downloadImageHovered = ImageIO.read(new File("./src/main/resources/images/downloadgray.png"));
    uploadImageHovered = ImageIO.read(new File("./src/main/resources/images/uploadgray.png"));
    rightArrowImageHovered = ImageIO.read(new File("./src/main/resources/images/arrowgray.png"));
    leftArrowImageHovered = ImageIO.read(new File("./src/main/resources/images/arrowgray.png"));

    menuImagePressed = ImageIO.read(new File("./src/main/resources/images/menuwhite.png"));
    runImagePressed = ImageIO.read(new File("./src/main/resources/images/runwhite.png"));
    stepsImagePressed = ImageIO.read(new File("./src/main/resources/images/stepswhite.png"));
    downloadImagePressed = ImageIO.read(new File("./src/main/resources/images/downloadwhite.png"));
    uploadImagePressed = ImageIO.read(new File("./src/main/resources/images/uploadwhite.png"));
    rightArrowImagePressed = ImageIO.read(new File("./src/main/resources/images/arrowwhite.png"));
    leftArrowImagePressed = ImageIO.read(new File("./src/main/resources/images/arrowwhite.png"));



    AffineTransform tx = AffineTransform.getScaleInstance(-1, 1);
    tx.translate(-leftArrowImage.getWidth(null), 0);

    AffineTransformOp op = new AffineTransformOp(tx, AffineTransformOp.TYPE_NEAREST_NEIGHBOR);
    leftArrowImage = op.filter(leftArrowImage, null);

    AffineTransform tx2 = AffineTransform.getScaleInstance(-1, 1);
    tx2.translate(-leftArrowImageHovered.getWidth(null), 0);

    AffineTransform tx3 = AffineTransform.getScaleInstance(-1, 1);
    tx3.translate(-leftArrowImagePressed.getWidth(null), 0);

    AffineTransformOp op2 = new AffineTransformOp(tx2, AffineTransformOp.TYPE_BILINEAR);
    leftArrowImageHovered = op2.filter(leftArrowImageHovered, null);

    AffineTransformOp op3 = new AffineTransformOp(tx3, AffineTransformOp.TYPE_BILINEAR);
    leftArrowImagePressed = op3.filter(leftArrowImagePressed, null);



    // Creating different colors:



} catch (IOException e){
    System.out.println("Failed to load image: " + e.getMessage());
}
this.roundedRectangleCalculatorComponent = roundedRectangleCalculatorComponent;
setLayout(null);


menuButton = new JButton();
runButton = new JButton();
stepsButton = new JButton();
downloadButton = new JButton();
uploadButton = new JButton();
rightArrowButton = new JButton();
leftArrowButton = new JButton();

menuButton.setOpaque(false);
menuButton.setContentAreaFilled(false);
menuButton.setBorderPainted(false);
menuButton.setFocusPainted(false);
menuButton.setBorder(BorderFactory.createEmptyBorder());

runButton.setOpaque(false);
runButton.setContentAreaFilled(false);
```

```
    runButton.setBorderPainted(false);
    runButton.setFocusPainted(false);
    runButton.setBorder(BorderFactory.createEmptyBorder());

    stepsButton.setOpaque(false);
    stepsButton.setContentAreaFilled(false);
    stepsButton.setBorderPainted(false);
    stepsButton.setFocusPainted(false);
    stepsButton.setBorder(BorderFactory.createEmptyBorder());

    downloadButton.setOpaque(false);
    downloadButton.setContentAreaFilled(false);
    downloadButton.setBorderPainted(false);
    downloadButton.setFocusPainted(false);
    downloadButton.setBorder(BorderFactory.createEmptyBorder());

    rightArrowButton.setOpaque(false);
    rightArrowButton.setContentAreaFilled(false);
    rightArrowButton.setBorderPainted(false);
    rightArrowButton.setFocusPainted(false);
    rightArrowButton.setBorder(BorderFactory.createEmptyBorder());

    leftArrowButton.setOpaque(false);
    leftArrowButton.setContentAreaFilled(false);
    leftArrowButton.setBorderPainted(false);
    leftArrowButton.setFocusPainted(false);
    leftArrowButton.setBorder(BorderFactory.createEmptyBorder());

    uploadButton.setOpaque(false);
    uploadButton.setContentAreaFilled(false);
    uploadButton.setBorderPainted(false);
    uploadButton.setFocusPainted(false);
    uploadButton.setBorder(BorderFactory.createEmptyBorder());

    ButtonActionListener buttonListener = new ButtonActionListener();
    menuButton.addActionListener(buttonListener);
    runButton.addActionListener(buttonListener);
    stepsButton.addActionListener(buttonListener);
    downloadButton.addActionListener(buttonListener);
    uploadButton.addActionListener(buttonListener);
    rightArrowButton.addActionListener(buttonListener);
    leftArrowButton.addActionListener(buttonListener);

    ButtonHoverListener hoverListener = new ButtonHoverListener();
    menuButton.addMouseListener(hoverListener);
    runButton.addMouseListener(hoverListener);
    stepsButton.addMouseListener(hoverListener);
    downloadButton.addMouseListener(hoverListener);
    uploadButton.addMouseListener(hoverListener);
    rightArrowButton.addMouseListener(hoverListener);
    leftArrowButton.addMouseListener(hoverListener);


    add(menuButton);
    add(runButton);
    add(stepsButton);
    add(downloadButton);
    add(uploadButton);
    add(rightArrowButton);
    add(leftArrowButton);
}
```

```
private class ButtonActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == menuButton) {
            roundedRectangleCalculatorComponent.resetMst();
            stepsClicked = false;
            menuButtonPressed = true;
            runButtonPressed = false;
            stepsButtonPressed = false;
            downloadButtonPressed = false;
            uploadButtonPressed = false;
            rightArrowButtonPressed = false;
            leftArrowButtonPressed = false;
            roundedRectangleCalculatorComponent.showAlgorithmSelectionDialog();
        } else if (e.getSource() == runButton) {
            roundedRectangleCalculatorComponent.resetMst();
            stepsClicked = false;
            menuButtonPressed = false;
            runButtonPressed = true;
            stepsButtonPressed = false;
            downloadButtonPressed = false;
            uploadButtonPressed = false;
            rightArrowButtonPressed = false;
            leftArrowButtonPressed = false;
            Graph graphPrima = roundedRectangleCalculatorComponent.getGraph();
            result = graphPrima.runPrimsAlgorithm(); // Выполнение алгоритма Прима и получение результата
            roundedRectangleCalculatorComponent.setMst(result); // Передача результата в компонент
        } else if (e.getSource() == stepsButton) {
            result.clear();
            updateMst();
            stepsClicked = true;
            roundedRectangleCalculatorComponent.resetMst();
            menuButtonPressed = false;
            runButtonPressed = false;
            stepsButtonPressed = true;
            downloadButtonPressed = false;
            uploadButtonPressed = false;
            rightArrowButtonPressed = false;
            leftArrowButtonPressed = false;
            Graph graphPrima = roundedRectangleCalculatorComponent.getGraph();
            result = graphPrima.runPrimsAlgorithm();
            currentIteration = 0;

        } else if (e.getSource() == downloadButton) {
            roundedRectangleCalculatorComponent.resetMst();
            stepsClicked = false;
            menuButtonPressed = false;
            runButtonPressed = false;
            stepsButtonPressed = false;
            downloadButtonPressed = true;
            uploadButtonPressed = false;
            rightArrowButtonPressed = false;
            leftArrowButtonPressed = false;
            JFileChooser fileChooser = new JFileChooser();
            int option = fileChooser.showOpenDialog(null);
            if (option == JFileChooser.APPROVE_OPTION) {
                File file = fileChooser.getSelectedFile();
                roundedRectangleCalculatorComponent.loadGraphFromFile(file.getAbsolutePath());
            }
        } else if (e.getSource() == uploadButton) {
            roundedRectangleCalculatorComponent.resetMst();
            menuButtonPressed = false;
```

```
            runButtonPressed = false;
            stepsButtonPressed = false;
            downloadButtonPressed = false;
            uploadButtonPressed = true;
            rightArrowButtonPressed = false;
            leftArrowButtonPressed = false;
            JFileChooser fileChooser = new JFileChooser();
            int option = fileChooser.showSaveDialog(null);
            if (option == JFileChooser.APPROVE_OPTION) {
                File file = fileChooser.getSelectedFile();
                roundedRectangleCalculatorComponent.saveGraphToFile(file.getAbsolutePath());
            }

        } else if (e.getSource() == rightArrowButton) {
            menuButtonPressed = false;
            runButtonPressed = false;
            stepsButtonPressed = false;
            downloadButtonPressed = false;
            uploadButtonPressed = false;
            rightArrowButtonPressed = true;
            leftArrowButtonPressed = false;
            if (stepsClicked){
                if (currentIteration < result.size()) {
                    currentIteration++;
                    updateMst();
                }
            }
        } else if (e.getSource() == leftArrowButton) {
            menuButtonPressed = false;
            runButtonPressed = false;
            stepsButtonPressed = false;
            downloadButtonPressed = false;
            uploadButtonPressed = false;
            rightArrowButtonPressed = false;
            leftArrowButtonPressed = true;
            if (stepsClicked){
                if (currentIteration > 0) {
                    currentIteration--;
                    updateMst();
                }
            }
        }
        repaint();


        Timer timer = new Timer(200, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent evt) {
                menuButtonPressed = false;
                runButtonPressed = false;
                stepsButtonPressed = false;
                downloadButtonPressed = false;
                uploadButtonPressed = false;
                rightArrowButtonPressed = false;
                leftArrowButtonPressed = false;
                repaint();
            }
        });
        timer.setRepeats(false);
        timer.start();
    }
}
```

```
    private class ButtonHoverListener extends MouseAdapter {

        @Override
        public void mouseEntered(MouseEvent e) {
            if (e.getSource() == menuButton) {
                menuButtonHovered = true;
            } else if (e.getSource() == runButton) {
                runButtonHovered = true;
            } else if (e.getSource() == stepsButton) {
                stepsButtonHovered = true;
            } else if (e.getSource() == downloadButton) {
                downloadButtonHovered = true;
            } else if (e.getSource() == uploadButton) {
                uploadButtonHovered = true;
            } else if (e.getSource() == rightArrowButton) {
                rightArrowButtonHovered = true;
            } else if (e.getSource() == leftArrowButton) {
                leftArrowButtonHovered = true;
            }
            repaint();
        }

        @Override
        public void mouseExited(MouseEvent e) {
            if (e.getSource() == menuButton) {
                menuButtonHovered = false;
            } else if (e.getSource() == runButton) {
                runButtonHovered = false;
            } else if (e.getSource() == stepsButton) {
                stepsButtonHovered = false;
            } else if (e.getSource() == downloadButton) {
                downloadButtonHovered = false;
            } else if (e.getSource() == uploadButton) {
                uploadButtonHovered = false;
            } else if (e.getSource() == rightArrowButton) {
                rightArrowButtonHovered = false;
            } else if (e.getSource() == leftArrowButton) {
                leftArrowButtonHovered = false;
            }
            repaint();
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);
        g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BICUBIC);


        int width_screen = getWidth();
        int height_screen = getHeight();

        int menuWidth = menuImage.getWidth();
        int menuHeight = menuImage.getHeight();

        int runWidth = runImage.getWidth();
        int runHeight = runImage.getHeight();
```

```
int stepsWidth = stepsImage.getWidth();
int stepsHeight = stepsImage.getHeight();

int downloadWidth = downloadImage.getWidth();
int downloadHeight = downloadImage.getHeight();

int uploadWidth = uploadImage.getWidth();
int uploadHeight = uploadImage.getHeight();

int rightArrowWidth = rightArrowImage.getWidth();
int rightArrowHeight = rightArrowImage.getHeight();

int leftArrowWidth = leftArrowImage.getWidth();
int leftArrowHeight = leftArrowImage.getHeight();

int newMenuWidth = (6*width_screen)/100;
int newMenuHeight = (8*height_screen)/100;

int newRunWidth = (6*width_screen)/100;
int newRunHeight = (8*height_screen)/100;

int newStepsWidth = (6*width_screen)/100;
int newStepsHeight = (8*height_screen)/100;

int newDownloadWidth = (6*width_screen)/100;
int newDownloadHeight = (8*height_screen)/100;

int newUploadWidth = (int)(5.9*width_screen)/100;
int newUploadHeight = (int)(9.7*height_screen)/100;

int newRightArrowWidth = (int)(6.5*width_screen)/100;
int newRightArrowHeight = (int)(10.2*height_screen)/100;

int newLeftArrowWidth = (int)(6.5*width_screen)/100;
int newLeftArrowHeight = (int)(10.2*height_screen)/100;

int xMenu = (6*width_screen)/100;
int yMenu = (87*height_screen)/100;

int xRun = (14*width_screen)/100;
int yRun = (87*height_screen)/100;

int xSteps = (22*width_screen)/100;
int ySteps = (87*height_screen)/100;

int xDownload = (30*width_screen)/100;
int yDownload = (87*height_screen)/100;

int xUpload = (38*width_screen)/100;
int yUpload = (int)(86.25*height_screen)/100;

int xRightArrow = (int)(86*width_screen)/100;
int yRightArrow = (int)(86.5*height_screen)/100;

int xLeftArrow = (int)(75*width_screen)/100;
int yLeftArrow = (int)(86.5*height_screen)/100;

AffineTransform atMenu = new AffineTransform();
atMenu.translate(xMenu, yMenu);
atMenu.scale((double) newMenuWidth/menuWidth, (double) newMenuHeight/menuHeight);
```

```
        AffineTransform atRun = new AffineTransform();
        atRun.translate(xRun, yRun);
        atRun.scale((double) newRunWidth/runWidth, (double) newRunHeight/runHeight);

        AffineTransform atSteps = new AffineTransform();
        atSteps.translate(xSteps, ySteps);
        atSteps.scale((double) newStepsWidth/stepsWidth, (double) newStepsHeight/stepsHeight);

        AffineTransform atDownload = new AffineTransform();
        atDownload.translate(xDownload, yDownload);
        atDownload.scale((double) newDownloadWidth/downloadWidth, (double)
newDownloadHeight/downloadHeight);

        AffineTransform atUpload = new AffineTransform();
        atUpload.translate(xUpload, yUpload);
        atUpload.scale((double) newUploadWidth/uploadWidth, (double) newUploadHeight/uploadHeight);

        AffineTransform atRightArrow = new AffineTransform();
        atRightArrow.translate(xRightArrow, yRightArrow);
        atRightArrow.scale((double) newRightArrowWidth/rightArrowWidth, (double)
newRightArrowHeight/rightArrowHeight);

        AffineTransform atLeftArrow = new AffineTransform();
        atLeftArrow.translate(xLeftArrow, yLeftArrow);
        atLeftArrow.scale((double) newLeftArrowWidth/leftArrowWidth, (double)
newLeftArrowHeight/leftArrowHeight);

        BufferedImage menuImageToDraw = menuImage;
        BufferedImage runImageToDraw = runImage;
        BufferedImage stepsImageToDraw = stepsImage;
        BufferedImage downloadImageToDraw = downloadImage;
        BufferedImage uploadImageToDraw = uploadImage;
        BufferedImage rightArrowImageToDraw = rightArrowImage;
        BufferedImage leftArrowImageToDraw = leftArrowImage;


        if (menuButtonPressed) {
           menuImageToDraw = menuImagePressed;
        } else if (menuButtonHovered) {
           menuImageToDraw = menuImageHovered;
        }
        if (runButtonPressed) {
           runImageToDraw = runImagePressed;
        } else if (runButtonHovered) {
           runImageToDraw = runImageHovered;
        }
        if (stepsButtonPressed) {
           stepsImageToDraw = stepsImagePressed;
        } else if (stepsButtonHovered) {
           stepsImageToDraw = stepsImageHovered;
        }
        if (downloadButtonPressed) {
           downloadImageToDraw = downloadImagePressed;
        } else if (downloadButtonHovered) {
           downloadImageToDraw = downloadImageHovered;
        }
        if (uploadButtonPressed) {
           uploadImageToDraw = uploadImagePressed;
        } else if (uploadButtonHovered) {
           uploadImageToDraw = uploadImageHovered;
        }
        if (rightArrowButtonPressed) {
```

```
            rightArrowImageToDraw = rightArrowImagePressed;
        } else if (rightArrowButtonHovered) {
            rightArrowImageToDraw = rightArrowImageHovered;
        }
        if (leftArrowButtonPressed) {
            leftArrowImageToDraw = leftArrowImagePressed;
        } else if (leftArrowButtonHovered) {
            leftArrowImageToDraw = leftArrowImageHovered;
        }

        g2d.drawRenderedImage(menuImageToDraw, atMenu);
        g2d.drawRenderedImage(runImageToDraw, atRun);
        g2d.drawRenderedImage(stepsImageToDraw, atSteps);
        g2d.drawRenderedImage(downloadImageToDraw, atDownload);
        g2d.drawRenderedImage(uploadImageToDraw, atUpload);
        g2d.drawRenderedImage(rightArrowImageToDraw, atRightArrow);
        g2d.drawRenderedImage(leftArrowImageToDraw, atLeftArrow);



        menuButton.setBounds(xMenu, yMenu, newMenuWidth, newMenuHeight);
        runButton.setBounds(xRun, yRun, newRunWidth, newRunHeight);
        stepsButton.setBounds(xSteps, ySteps, newStepsWidth, newStepsHeight);
        downloadButton.setBounds(xDownload, yDownload, newDownloadWidth, newDownloadHeight);
        uploadButton.setBounds(xUpload, yUpload, newUploadWidth, newUploadHeight);
        rightArrowButton.setBounds(xRightArrow,yRightArrow, newRightArrowWidth, newRightArrowHeight);
        leftArrowButton.setBounds(xLeftArrow,yLeftArrow, newLeftArrowWidth, newLeftArrowHeight);
    }

    void updateMst() {
        ArrayList<Edge> currentEdges = new ArrayList<>(result.subList(0, currentIteration));
        roundedRectangleCalculatorComponent.setMst(currentEdges);
    }
}
```