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

ОТЧЕТ

по учебной практике

Тема: Минимальное остовное дерево. Алгоритм Краскала

Студент гр. 1304 _____ Байков Е.С.

Студент гр. 1303 _____ Депрейс А.С.

Студент гр. 1304 _____ Маркуш А.Е.

Руководитель _____ Токарев А.П.

Санкт-Петербург

2023

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Байков Е.С. группы 1304

Студент Депрейс А.С. группы 1303

Студент Маркуш А.Е. группы 1304

Тема практики: Минимальное остовное дерево. Алгоритм Краскала

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Минимальное остовное дерево. Алгоритм Краскала

Сроки прохождения практики: 30.06.2020 – 13.07.2020

Дата сдачи отчета: 12.07.2020

Дата защиты отчета: 12.07.2020

Студент гр. 1304	_____	Байков Е.С.
Студент гр. 1303	_____	Депрейс А.С.
Студент гр. 1304	_____	Маркуш А.Е.
Руководитель	_____	Токарев А.П.

АННОТАЦИЯ

Задачей учебной практики является получение опыта командной работы над проектом. Практика заключается в разработке приложения — визуализатора указанного в задании алгоритма на языке Java. В данном случае предстоит визуализация алгоритма Краскала для поиска минимального остовного дерева. Разработка ведётся итеративно: студентами и их руководителем согласовывается спецификация, чётко определяющая предполагаемое поведение программы, после чего создаются несколько версий проекта, каждая из которых расширяет функционал предыдущей. В отчёте приведена информация о ходе выполнения практической работы.

SUMMARY

The objective of the training practice is to gain experience in teamwork on a project. The practice consists in developing an application - visualizing the congestion of the algorithm specified in the task in the Java language. In this case, it is necessary to visualize the Kraskal algorithm for finding the minimum spanning tree. The development is carried out iteratively: students and their supervisor agree on a specification that clearly defines the intended behavior of the program, after which several versions of the project are created, each of which extends the functionality of the previous one. The report provides information on the progress of practical work.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель практики – визуализация предоставленного алгоритма на языке Java. Для выполнения цели практики следует решить следующие задачи:

1. Изучить алгоритм, который предстоит реализовать.
2. Составить и согласовать спецификацию разработки приложения.
3. Составить план разработки и распределить роли при реализации приложения.
4. Написать и отладить модули проекта.
5. Скомпилировать модули в один проект и отладить полное приложение.
6. Предоставить проект руководителю.

Алгоритм Краскала – эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм работает работает за линейное время на отсортированном множестве рёбер, а если учесть, что сортировка имеет сложность O(E × log(E)), то общее время работы алгоритма будет O(E × log(E)).

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Исходную структуру программы можно представить в качесте UML диаграммы, изображённой на рис. 1.

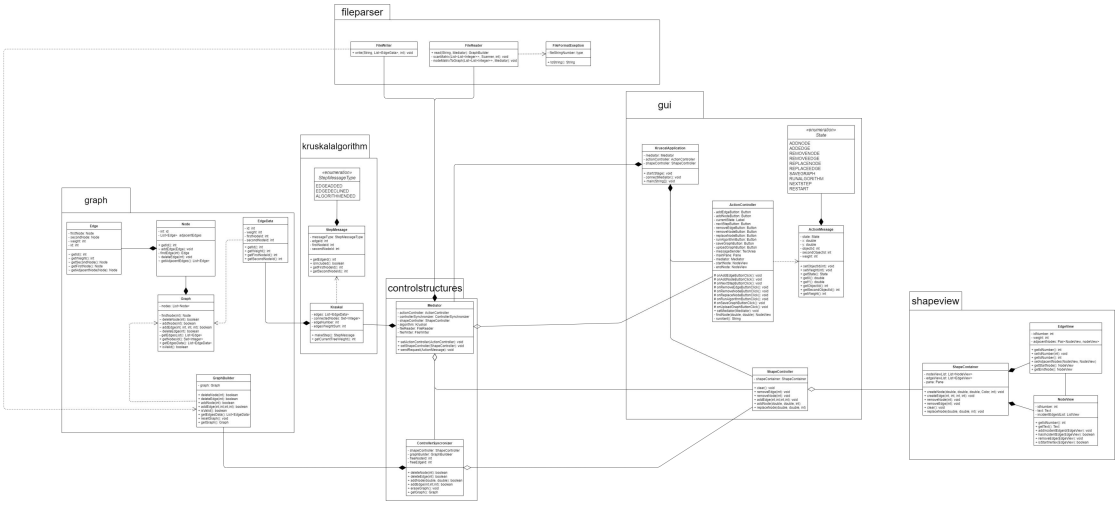

Рисунок 1 – UML-диаграмма классов разрабатываемой программы

1.1.1. Требования к визуализации работы алгоритма

В данной программе визуализирована работа алгоритма Краскала по поиску минимального остовного дерева. Вначале стоится граф с помощью графического интерфейса либо загружается из файла. Затем начинается работа самого алгоритма и пошагово проходится его реализация, визуализируя шаги алгоритма.

1.1.2. Требования к визуализации пользовательского интерфейса

Интерфейс представлен на рис. 2.

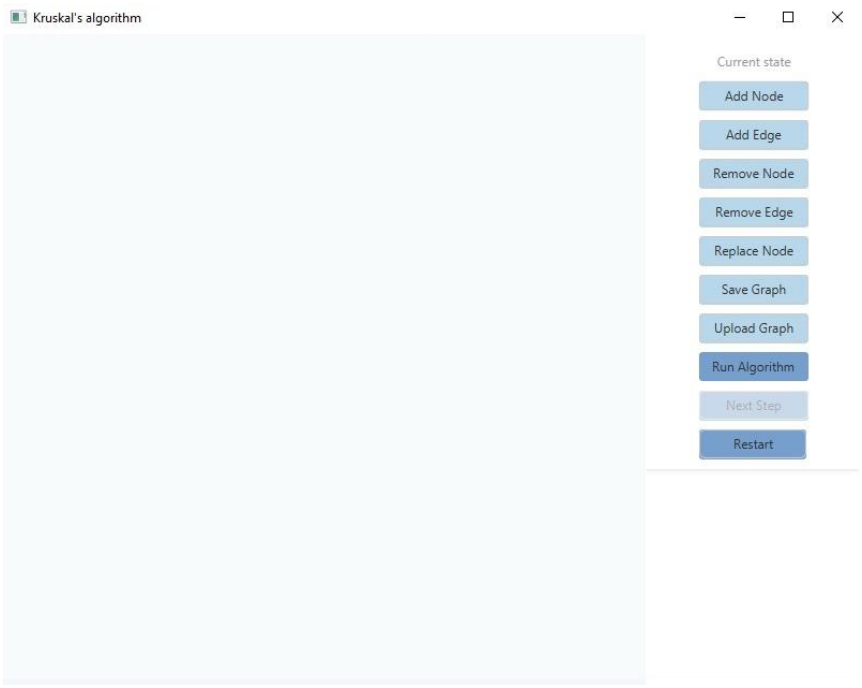

Рисунок 2 – Пользовательский интерфейс приложения в начале его работы

В начале запускается окно, в котором большую часть занимает зона для отрисовки, где будет находится граф, справа находится панель управления с кнопками, в правом нижнем углу находится окно для предоставления информационных сообщений при исключениях, а так же при непосредственной работе алгоритма.

При создании графа, кнопка “Next Step” заблокирована, т.к. алгоритм ещё не начал своей работы. Кнопка “Save Graph” позволяет сохранить граф в файл текстовый файл. Кнопки “Add Node”, “Add Edge” переводят программу в режим добавления вершин либо рёбер соответственно, т.е. при нажатии на рабочее поле в режиме добавления вершин будет появляться вершина в указанной точке, а при нажатии на две вершины в режиме добавления рёбер будет появляться ребро между выбранными вершинами. Кнопки “Remove Node” и “Remove Edge” переводят программу в режим удаления вершин или рёбер. При нахождении в режиме удаления нажатие на объект соответствующий режиму приведёт к удалению этого объекта. При удалении вершины, все инцидентные ей рёбра тоже удаляются. Формат представления графа в файле будет представлен в п.1.1.3. После того, как пользователь нажмёт кнопку “Run Algorithm”, кнопки связанные с созданием и сохранением графа

будут заблокированы для пользователя, а кнопка “Next Step” станет доступной. Это можно увидеть на рис. 2.

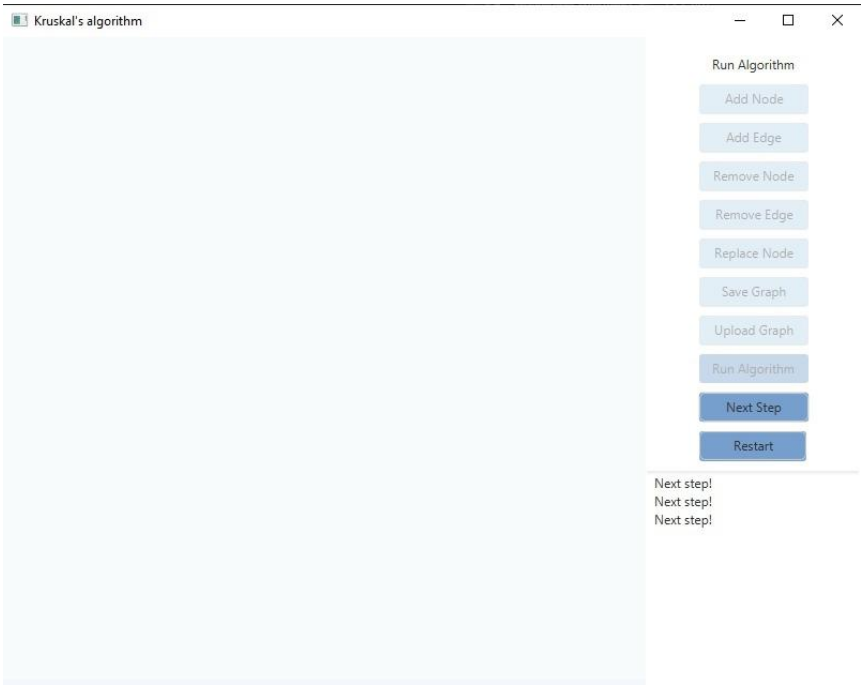

Рисунок 3 – Пользовательский интерфейс, после начала работы алгоритма

Кнопка “Restart” доступна всегда, она возвращает программу в начальное состояние как на рис. 2.

Как можно заметить на рис. 3 при работе алгоритма в окне сообщений выводится информация о пройденном шаге алгоритма. Вместо сообщения “Next step!” будет выводиться сообщение, поясняющее данный шаг алгоритма.

1.1.3. Требования к входным данным

Представления графа может осуществляться двумя способами:

1. Загрузка графа из .txt файла, по нажатию кнопки “Upload File”, в котором задано число вершин графа и построчно записана матрица смежности.

2. Создания графа с помощью инструментов, доступных в пользовательском интерфейсе.

1.2. Уточнение требований после сдачи первой версии

1. Сделать возможность шага назад.
2. Добавление отображения веса уже найденного дерева при работе алгоритма.
3. Задать расположение вершин по зоне отображения графа при его загрузке из файла.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки и распределение ролей в бригаде

1. Составление спецификации. Исполнители: Маркуш А.Е., Депрейс А.С., Байков Е.С. Скрок сдачи: 06.07
2. Составление плана разработки. Исполнители: Маркуш А.Е., Депрейс, А.С. Байков Е.С. Скрок сдачи: 06.07
3. Реализация графического интерфейса:
    a) Реализация области отображения графа. Исполнитель: Байков Е.С. Срок сдачи: 08.07
    b) Реализация графического интерфейса для редактирование графа и работы алгоритма. Исполнитель: Байков Е.С. Срок сдачи: 08.07
4. Реализация работы с файлом:
    a) Реализация сохранения графа в файл. Исполнитель: Маркуш А.Е. Срок сдачи: 08.07
    b) Реализация загрузки графа из файла. Исполнитель: Маркуш А.Е. Срок сдачи: 08.07
5. Реализация бизнес логики программы.
    a) Реализация алгоритма Краскала. Исполнитель: Депрейс А.С. Срок сдачи: 08.07
    b) Реализация класса графа и его создания. Исполнитель: Депрейс А.С. Срок сдачи: 08.07
    c) Реализация классов для вершины и дуги. Исполнитель: Депрейс А.С. Срок сдачи: 08.07
    d) Создание объектов для передачи сообщений о работе алгоритма. Исполнитель: Депрейс А.С. Срок сдачи: 08.07
6. Реализация связующих структур между графической и логической частью программы. Исполнители: Маркуш А.Е., Депрейс А.С. Срок сдачи: 10.07
7. Реализация визуализации работы алгоритма:

a) Добавление возможности запуска алгоритма и просмотра каждого его шага. Исполнитель: Байков Е.С. Срок сдачи: 10.07
b) Реализация API для связи с логикой программы. Исполнитель: Байков Е.С. Скрок сдачи: 10.07

8. Внесение корректировок, исправление багов и тестирование программы:
   a) Тестирование и исправление графической части программы. Исполнитель: Байков Е.С. Скрок сдачи: 12.07
   b) Тестирование и исправление сохранения/загрузки графа. Исполнитель: Маркуш А.Е. Скрок сдачи: 12.07
   c) Тестирование и исправление бизнес логики программы. Исполнитель: Депрейс А.С. Скрок сдачи: 12.07

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

В данном проекте структуры данных можно разбить на 6 условных групп: реализация алгоритма Краскала, построение и хранение графа для внутренней логики, построение и хранение графа для отрисовки, работа с файлами, графический интерфейс, связь интерфейса с остальной программой.

Классы реализующие алгоритм Краскала:

- Kruskal – класс, в котором происходит пошаговая реализация самого алгоритма.
- StepMessege – класс, формирующий сообщение, которое будет передано для дальнейшего представления в окне информации графического интерфейса.
- StepMessegeType – перечисление, в котором содержатся типы сообщений, которые может сформировать StepMessege.

Классы отвечающие за представление графа для логики программы:

- Edge – класс, хранящий информацию о ребре графа.
- EdgeData – класс, хранящий информацию о ребре графа и предоставляющий её другим другим классам.
- Node – класс, хранящий информацию о вершине графа.
- Graph – класс, который хранит информацию о всём графе и имеет методы для построения графа, доступные только классам из того же пакета.
- GraphBuilder – класс, позволяющий строить граф и получать информацию о нём, для классов их других пакетов.
- GraphConnectednessExeption – класс исключение, которое выбрасывается при неподходящей структуре (несвязности) графа при запуске алгоритма.

Классы отвечающие за представление графа для графического интерфейса:

- EdgeView – класс, хранящий информацию, для отрисовки рёбер.
- NodeView – класс, хранящий информацию, для отрисовки вершин.

- ShapeContainer – класс, хранящий информацию о всех рёбрах и вершинах, которые надо отобразить, а так же предоставляющий методы для редактирования информации о графе.

Классы отвечающие за работу с файлами:

- FileReader – класс, реализущий считывание информации о графе из файла, проверку корректности формата файла и отправку сигналов, для построения графа, по считанной информации.
- FileWriter – класс, реализующий запись графа в файл, в корректном формате.
- FileFormatExeption – класс исключение, которое выбрасывается классами FileWriter и FileReader, при проблемах с файлом или его содержимым.

Классы реализующие графический интерфейс программы:

- KruscalApplication – класс, запускающий программу и создающий основные объекты для её дальнейшей работы.
- ActionController – класс, отвечающий за работу кнопок управления и окна с информацией.
- ActionMessage – класс, реализующий сообщения, с помощью которых компоненты программы передают информацию друг другу.
- ShapeController – класс, для управления информацией в ShapeContainer.
- State – перечисление, которое содержит типы состояний, в которых может находится программа

Классы, реализующие связь компонентов программы:

- Mediator – класс, который вызывает методы внутренней логики программы, в зависимости от состояния программы.
- ControllerSynchronizer – класс, позволяющий синхронизировать обновления информации о графе в логической и графических частях, с помощью вызовов соответствующих методов у ShapeController и GraphBuilder.

3.2. Основные методы

- start – метода класса KruscalApplication, создающий основыне объекты программы, после её запуска.

- onRunAlgorithmButtonClick, onPreviousStepButtonClick, onUploadGraphButtonClick, onSaveGraphButtonClicked, onNextStepButtonClick, onRestartButtonClick, onAddNodeButtonClick, onAddEdgeButtonClick, onRemoveNodeButtonClick,, onRemoveEdgeButtonClicked, onReplaceNodeButtonClicked – методы класса ActionController, обрабатывающие нажатия на соответствующие кнопки в графическом интерфейсе.
- createErrorAlertMessage, runAlert – методы методы класса ActionController, вызывающие алерты при вводе веса ребра, либо обработке исключений.
- printMessage – метод формирующий сообщения, которое выводится в информационном окне.
- класс ActionMessage имеет множество переопределений конструктора, для универсальности передающихся между компонентами программы сообщений.
- removeEdge, removeNode, addEdge, addNode, replaceNode – методы в классах ShapeController и ShapeContainer, обновляющие информация для изображения графа с помощью соответствующих действий. Вызов метода во внешнем коде объекта класса ShapeController вызывает тот же метод у объекта класса ShapeContainer.
- paintEdge – метод класса ShapeController, изменяющий цвет ребра в соответствии с результатами работы алгоритма Краскала с помощью метода colorEdge объекта типа ShapeContainer.
- deleteNode, deleteEdge, addNode, addEdge – методы классов GraphBuilder и Graph, обновляющие информация для хранения логического представления графа с помощью соответствующих действий. Внешний код вызывает методы объекта типа GraphBuilder, который вызывает те же методы у объекта типа Graph, хранящегося в нём. Метод getGraph позволяет получить объект типа Graph из объекта GraphBuilder.

- read – метод, реализующий считывание информации о графе и передачи команд для его построения с помощью методов scanMatrix и nodeMatrixToGraph.
- write – метод, реализующий запись информации о графе в файл.
- makeStep, stepBack – методы класса Kruskal, который реализует следующий и предыдущий шаги алгоритма Краскала.
- sendRequest – метод класса Mediator, который определяет вызов методов классов FileReader, FileWriter, ControllerSynchronizer и ShapeController в зависимости от состояния, которое получил метод sendRequest.
- addNode, addEdge, deleteNode, deleteEdge, eraseGraph – методы класса ControllerSynchronizer, который вызывает методы с соответствующим функционалом у объектов классов GraphBuilder и ShapeController.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

На рисунках 4 - 10 в приложении В представлена демонстрация тестирования функционала модификации графа с помощью графического интерфейса. Были проверены все заявленные модификации.

4.2. Тестирование сохранения и загрузки графа

На рисунках 11 - 18 в приложении В представлена демонстрация тестирования сохранения и загрузки графа. Загрузка производилась в том числе из файлов с неверным форматом.

4.3 Тестирования визуализации алгоритма

На рисунках 19 - 21 в приложении В представлена демонстрация тестирования визуализации алгоритма на различных графах.

ЗАКЛЮЧЕНИЕ

В ходе учебной практики был изучен строго типизированный объектно-ориентированный язык программирования Java.

Перед выполнением основной задачи учебной практики был изучен алгоритм Краскала. Главным преимуществом алгоритма Прима является простота реализации. Недостаток алгоритма заключается в том, что он требует сортировку рёбер графа по возрастанию, что может замедлить работу алгоритма на больших графах.

В процессе выполнения учебной практики была рассмотрена библиотека JavaFX языка программирования Java для реализации графического интерфейса.

Результатом практической работы бригады является приложение, реализованное на языке программирования Java, которое визуализирует рассматриваемый алгоритм для поиска остовного дерева связного взешенного ненаправленного графа. Приложение позволяет пользователю ввести граф как через интерфейс приложения, так и в виде файла с расширением .txt. Кнопки интерфейса обеспечивают пошаговую визуализацию рассматриваемого алгоритма. В результате тестирования были выявлены неточности в работе приложения, которые были успешно исправлены.

Полученный результат соответствует поставленным целям учебной практики. Функционал приложения соответствует согласованной спецификации с учётом замечаний преподавателя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java документация от Oracle // Oracle. URL:
2. Metanit. URL: https://metanit.com/java/javafx
3. Stackoverflow. URL: https://stackoverflow.com/
4. Wikipedia. URL: https://ru.wikipedia.org/wiki/Шаблон_проектирования
5. Habr. URL: https://habr.com/ru/articles/474982/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: KruscalApplication.java

```
package com.kruskal.gui;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;


import java.io.IOException;

import com.kruskal.controlstructures.Mediator;

public class KruskalApplication extends Application {
   private Mediator mediator;
   private ActionController actionController;
   private ShapeController shapeController;
   @Override
   public void start(Stage stage) throws IOException
   {
       FXMLLoader fxmlLoader = new
FXMLLoader(KruskalApplication.class.getResource("application-view.fxml"));
       AnchorPane root = fxmlLoader.load();
       actionController = fxmlLoader.getController();
       actionController.setStage(stage);
       shapeController = new ShapeController((Pane)root.getChildren().get(1));
       connectMediator();
       Image icon = new
Image(getClass().getResourceAsStream("/icons/graph_icon.png"));
       Scene scene = new Scene(root);
       stage.getIcons().add(icon);
       stage.setTitle("Kruskal's algorithm");
       stage.setScene(scene);
       stage.show();
   }

   private void connectMediator() {
       mediator = new Mediator();
       mediator.setActionController(actionController);
       mediator.setShapeController(shapeController);
       actionController.setMediator(mediator);
   }
   public static void main(String[] args) {
       launch();
   }
}
```

Файл: ActionMessege.java

```
package com.kruskal.gui;


public class ActionMessage {
   private State state;
```

```
    private double x;
    private double y;
    private int objectId;
    private int secondObjectId;
    private int weight;

    public String getFileName() {
        return fileName;
    }

    private String fileName;

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public State getState() {
        return state;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public int getObjectId() {
        return objectId;
    }

    public int getSecondObjectId() {
        return secondObjectId;
    }

    public int getWeight() {
        return weight;
    }

    public ActionMessage(State state, double x, double y, int objectId, int
secondObjectId, int weight, String fileName) {
        this.state = state;
        this.x = x;
        this.y = y;
        this.objectId = objectId;
        this.secondObjectId = secondObjectId;
        this.weight = weight;
        this.fileName = fileName;
    }
    public ActionMessage(State state, double x, double y) {
        this(state, x, y, -1, -1, -1, null);
    }

    public ActionMessage(State state) {
        this(state, -1, -1, -1, -1, -1, null);
    }

    public ActionMessage(State state, int objectId, int secondObjectId, int
weight) {
        this(state, -1, -1, objectId, secondObjectId, weight, null);
    }
```

```
    public ActionMessage(State state, int objectId) {
        this(state, -1, -1, objectId, -1, -1, null);
    }

    public ActionMessage(State state, double x, double y, int objectId) {
        this(state, x, y, objectId, -1, -1, null);
    }

    public ActionMessage(State state, String fileName) {
        this(state, -1, -1, -1, -1, -1, fileName);
    }
    public ActionMessage(State state, double x, double y,String fileName) {
        this(state, x, y, -1, -1, -1, fileName);
    }
}
```

Файл: ActionConroller.java

```
package com.kruskal.gui;

import com.kruskal.controlstructures.Mediator;
import com.kruskal.fileparser.FileFormatException;
import com.kruskal.graph.GraphConnectednessExeption;
import com.kruskal.kruskalalgorithm.StepMessage;
import com.kruskal.shapeview.EdgeView;
import com.kruskal.shapeview.NodeView;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class ActionController {
    @FXML
    private Button addEdgeButton;
    @FXML
    private Button addNodeButton;
    @FXML
    private Label currentState;
    @FXML
    private Button nextStepButton;
    @FXML
    private Button previousStepButton;
    @FXML
    private Button removeEdgeButton;
    @FXML
    private Button removeNodeButton;
    @FXML
    private Button replaceNodeButton;
    @FXML
    private Button runAlgorithmButton;
    @FXML
    private Button saveGraphButton;
```

```
    @FXML
    private Button uploadGraphButton;
    @FXML
    private TextArea messageSender;
    @FXML
    private Pane mainPane;
    @FXML
    private Label treeWeightInfo;
    private Mediator mediator;
    private NodeView startNode;
    private NodeView endNode;
    private Stage stage;

    public void setStage(Stage stage) {
        this.stage = stage;
    }

    @FXML
    protected void onRunAlgorithmButtonClick() {
        try {
            mediator.sendRequest(new ActionMessage(State.RUNALGORITHM));
            currentState.setText("Run Algorithm");
            currentState.setOpacity(1);
            nextStepButton.setDisable(false);
            setDisability(true);
        } catch (GraphConnectednessExeption exception) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setHeaderText(exception.getMessage());
            alert.showAndWait();
        }
    }

    @FXML
    protected void onUploadGraphButtonClick() {
        currentState.setText("Current state");
        currentState.setOpacity(0.5d);
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open File");
        File file = fileChooser.showOpenDialog(stage);
        if (file != null) {
            try {
                mediator.sendRequest(new ActionMessage(State.UPLOADGRAPH,
mainPane.getWidth(), mainPane.getHeight(), file.getAbsolutePath()));
                messageSender.appendText("Graph has been uploaded\n");
            } catch (FileFormatException exception) {
                createErrorAlertMessage(exception.toString());
            } catch (Exception exception) {
                createErrorAlertMessage(exception.getMessage());
            }
        } else {
            createErrorAlertMessage("Не удалось открыть файл");
        }
    }

    @FXML
    protected void onSaveGraphButtonClicked() {
        currentState.setText("Current state");
        currentState.setOpacity(0.5d);
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open File");
        File file = fileChooser.showSaveDialog(stage);
        if (file != null) {
            try {
```

```
                mediator.sendRequest(new ActionMessage(State.SAVEGRAPH,
file.getAbsolutePath()));
                messageSender.appendText("Graph has been saved\n");
            } catch (FileFormatException exception) {
                createErrorAlertMessage(exception.toString());
            }
        } else {
            createErrorAlertMessage("Не удалось открыть файл");
        }
    }

    @FXML
    protected void onNextStepButtonClick() {
        if (previousStepButton.isDisable()) {
            previousStepButton.setDisable(false);
        }
        try {
            mediator.sendRequest(new ActionMessage(State.NEXTSTEP));
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onPreviousStepButtonClick() {
        if (nextStepButton.isDisable()) {
            nextStepButton.setDisable(false);
        }
        try {
            mediator.sendRequest(new ActionMessage(State.PREVIOUSSTEP));
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onRestartButtonClick() {
        setDisability(false);
        treeWeightInfo.setText("0");
        nextStepButton.setDisable(true);
        previousStepButton.setDisable(true);
        messageSender.clear();
        currentState.setText("Current state");
        currentState.setOpacity(0.5d);
        mainPane.setOnMouseClicked(event -> {});
        mediator.sendRequest(new ActionMessage(State.RESTART));
    }

    private void setDisability(boolean disability) {
        addEdgeButton.setDisable(disability);
        addNodeButton.setDisable(disability);
        removeEdgeButton.setDisable(disability);
        removeNodeButton.setDisable(disability);
        saveGraphButton.setDisable(disability);
        uploadGraphButton.setDisable(disability);
        runAlgorithmButton.setDisable(disability);
        replaceNodeButton.setDisable(disability);
    }

    @FXML
    protected void onAddNodeButtonClick() {
        currentState.setText("Add Node");
        currentState.setOpacity(1);
```

```
        try {
            mainPane.setOnMouseClicked(event -> {
                if (mainPane.contains(event.getX() + 40, event.getY())) {
                    mediator.sendRequest(new ActionMessage(State.ADDNODE,
event.getX(), event.getY()));
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onAddEdgeButtonClick() {
        currentState.setText("Add Edge");
        currentState.setOpacity(1);
        try {
            mainPane.setOnMouseClicked(event -> {
                if (startNode == null) {
                    startNode = findNode(event.getX(), event.getY());
                } else {
                    endNode = findNode(event.getX(), event.getY());
                    if (endNode != null && !endNode.equals(startNode)) {
                        String inputWeight = runAlert();
                        int weight = !inputWeight.equals("") ?
Integer.parseInt(inputWeight) : 1;
                        mediator.sendRequest(new ActionMessage(State.ADDEDGE,
startNode.getIdNumber(), endNode.getIdNumber(), weight));
                        startNode = null;
                        endNode = null;
                    }
                }
            });
        }
        catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onRemoveNodeButtonClick() {
        currentState.setText("Remove Node");
        currentState.setOpacity(1);
        try {
            mainPane.setOnMouseClicked(event -> {
                NodeView removingNode = findNode(event.getX(), event.getY());
                if (removingNode != null) {
                    mediator.sendRequest(new ActionMessage(State.REMOVENODE,
removingNode.getIdNumber()));
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onRemoveEdgeButtonClicked() {
        currentState.setText("Remove Edge");
        currentState.setOpacity(1);
        try {
            mainPane.setOnMouseClicked(event -> {
                for (int i = 0; i < mainPane.getChildren().size(); ++i) {
```

```
                    if (mainPane.getChildren().get(i) instanceof Line edge) {
                        if (edge.contains(event.getX(), event.getY())) {
                            mediator.sendRequest(new
ActionMessage(State.REMOVEEDGE, ((EdgeView) edge).getIdNumber()));
                        }
                    }
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onReplaceNodeButtonClicked() {
        currentState.setText("Replace Node");
        currentState.setOpacity(1);
        startNode = null;
        try {
            mainPane.setOnMouseClicked(event -> {
                if (event.getButton() == MouseButton.PRIMARY) {
                    if (startNode == null) {
                        startNode = findNode(event.getX(), event.getY());
                    } else {
                        mediator.sendRequest(new ActionMessage(State.REPLACENODE,
event.getX(), event.getY(), startNode.getIdNumber()));
                        startNode = null;
                    }
                } else if (event.getButton() == MouseButton.SECONDARY) {
                    startNode = null;
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    private NodeView findNode(double x, double y) {
        for (int i = 0; i < mainPane.getChildren().size(); ++i) {
            if (mainPane.getChildren().get(i) instanceof Circle node) {
                if (node.contains(x, y)) return (NodeView) node;
            }
        }
        return null;
    }

    private String runAlert() {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Text field");
        alert.setHeaderText("Введите вес ребра");
        TextField inputField = new TextField();
        inputField.setPrefWidth(Integer.toString(Integer.MAX_VALUE).length());
        inputField.textProperty().addListener((observable, oldValue, newValue) ->
{
            if (newValue.startsWith("0") || !newValue.matches("\\d*") ||
newValue.equals("") ||
                    Integer.toString(Integer.MAX_VALUE).length() <
newValue.length() ||
                    Double.parseDouble(newValue) > Integer.MAX_VALUE) {
```

```
                inputField.setStyle("-fx-border-color: red");
                Button okButton = (Button)
alert.getDialogPane().lookupButton(ButtonType.OK);
                okButton.setDisable(true);
            } else {
                inputField.setStyle("-fx-border-color: blue");
                Button okButton = (Button)
alert.getDialogPane().lookupButton(ButtonType.OK);
                okButton.setDisable(false);
            }
        });
        VBox vBox = new VBox();
        vBox.getChildren().addAll(inputField);
        alert.getDialogPane().setContent(vBox);
        alert.showAndWait();
        return inputField.getText();
    }

    public void printMessage(StepMessage message) {
        switch (message.getType()) {
            case EDGEADDED -> messageSender.appendText("Edge between " +
message.getFirstNodeId()
                    + " and " + message.getSecondNodeId() + " were added\n");
            case EDGEDECLINED -> messageSender.appendText("Edge between " +
message.getFirstNodeId()
                    + " and " + message.getSecondNodeId() + " were declined\n");
            case ALGORITHMENDED -> {
                messageSender.appendText("Algorithm ended\n");
                nextStepButton.setDisable(true);
            }
        }
    }

    public void deleteLastMessage() {
        List<String> sentences = new
ArrayList<>(List.of(messageSender.getText().split("\n")));
        if (sentences.size() > 0 && sentences.contains("Algorithm ended")) {
            sentences.remove(sentences.size() - 1);
            if (sentences.size() > 0) {
                sentences.set(sentences.size() - 1, "");
            }
        } else {
            sentences.set(sentences.size() - 1, "");
        }
        if (!sentences.isEmpty()) {
            messageSender.setText(String.join("\n", sentences));
        } else {
            messageSender.setText("");
        }
    }

    public void blockPreviousStepButton() {
        previousStepButton.setDisable(true);
    }

    public void printTreeWeight(int treeWeight) {
        treeWeightInfo.setText(Integer.toString(treeWeight));
    }

    private void createErrorAlertMessage(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(message);
        alert.showAndWait();
```

```
        mediator.sendRequest(new ActionMessage(State.RESTART));
    }
}
```

Файл ShapeController.java

```
package com.kruskal.gui;

import com.kruskal.kruskalalgorithm.StepMessage;
import com.kruskal.shapeview.ShapeContainer;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;


public class ShapeController {
    private final ShapeContainer shapeContainer;


    public ShapeController(Pane pane) {
        this.shapeContainer = new ShapeContainer(pane);
    }


    public void clear() {
        shapeContainer.clear();
    }


    public void removeEdge(int edgeId) {
        shapeContainer.removeEdge(edgeId);
    }

    public void removeNode(int nodeId) {
        shapeContainer.removeNode(nodeId);
    }

    public void addEdge(int startNodeId, int endNodeId, int weight, int edgeId) {
        shapeContainer.createEdge(startNodeId, endNodeId, weight, edgeId);
    }

    public void addNode(double x, double y, int nodeId) {
        shapeContainer.createNode(x, y,
                30, Color.rgb(161, 227, 255), nodeId);
    }

    public void replaceNode(double x, double y, int objectId) {
        shapeContainer.replaceNode(x, y, objectId);
    }
    public void paintEdge(StepMessage stepMessage) {
        switch (stepMessage.getType()) {
            case EDGEADDED -> shapeContainer.colorEdge(stepMessage.getEdgeId(),
Color.LIME);
            case EDGEDECLINED -> shapeContainer.colorEdge(stepMessage.getEdgeId(),
Color.RED);
        }
    }
}
```

Файл: State.java

```
package com.kruskal.gui;
```

```
public enum State {
   ADDNODE,
   ADDEDGE,
   REMOVENODE,
   REMOVEEDGE,
   REPLACENODE,
   SAVEGRAPH,
   UPLOADGRAPH,
   RUNALGORITHM,
   NEXTSTEP,
   RESTART,
   PREVIOUSSTEP
}
```

Файл: Kruskal.java

```
package com.kruskal.kruskalalgorithm;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import com.kruskal.graph.EdgeData;
import com.kruskal.graph.Graph;

public class Kruskal {
   private final List<StepMessage> steps;
   private int currentStep;
   private final List<EdgeData> edges;
   private final List<Set<Integer>> connectedNodes;
   private int edgeNumber;
   private int edgesWeightSum;

   public Kruskal(Graph graph){
       steps = new ArrayList<StepMessage>();
       currentStep = 0;
       Set<Integer> nodes = graph.getNodesId();
       edges = graph.getEdgesData();

       Comparator<EdgeData> comparator = new Comparator<EdgeData>() {
           public int compare(EdgeData first, EdgeData second){
               return
Integer.valueOf(first.getWeight()).compareTo(Integer.valueOf(second.getWeight()))
;
           }
       };
       edges.sort(comparator);

       connectedNodes = new ArrayList<Set<Integer>>();
       for (Integer node: nodes) {
           Set<Integer> newSet = new HashSet<Integer>();
           newSet.add(node);
           connectedNodes.add(newSet);
       }

       edgeNumber = 0;
       edgesWeightSum = 0;
   }

   public StepMessage makeStep(){
       if(currentStep < edgeNumber){
           currentStep++;
```

```
            edgesWeightSum += steps.get(currentStep - 1).getWeightShift();
            return steps.get(currentStep - 1);
        }

        if(edgeNumber == edges.size()){
            return new StepMessage(StepMessageType.ALGORITHMENDED);
        }

        if(connectedNodes.size() == 1){
            return new StepMessage(StepMessageType.ALGORITHMENDED);
        }

        EdgeData currentEdge = edges.get(edgeNumber);
        edgeNumber++;

        Set<Integer> firstNodeSet = null;
        Set<Integer> secondNodeSet = null;

        for (Set<Integer> set : connectedNodes) {
            if(firstNodeSet == null){
                if(set.contains(currentEdge.getFirstNodeId())){
                    firstNodeSet = set;
                }
            }

            if(secondNodeSet == null){
                if(set.contains(currentEdge.getSecondNodeId())){
                    secondNodeSet = set;
                }
            }
        }

        if(firstNodeSet.equals(secondNodeSet)){
            steps.add(new StepMessage(currentEdge, StepMessageType.EDGEDECLINED,
0));
            currentStep++;
            return steps.get(currentStep - 1);
        }

        firstNodeSet.addAll(secondNodeSet);
        connectedNodes.remove(secondNodeSet);
        edgesWeightSum += currentEdge.getWeight();

        steps.add(new StepMessage(currentEdge, StepMessageType.EDGEADDED,
currentEdge.getWeight()));
        currentStep++;
        return steps.get(currentStep - 1);
    }

    public StepMessage stepBack(){
        if(currentStep == 0){
            return null;
        }

        currentStep--;
        edgesWeightSum -= steps.get(currentStep).getWeightShift();
        return steps.get(currentStep);
    }

    public int getCurrentTreeWeight(){
        return edgesWeightSum;
    }
```

```
}
```

Файл: StepMessage.java

```
package com.kruskal.kruskalalgorithm;
import com.kruskal.graph.EdgeData;

public class StepMessage {
   private final StepMessageType messageType;
   private final int firstNodeId;
   private final int secondNodeId;
   private final int edgeId;
   private final int weightShift;

   public StepMessage(EdgeData edgeData, StepMessageType messageType, int
weightShift){
       this.weightShift = weightShift;
       this.messageType = messageType;
       firstNodeId = edgeData.getFirstNodeId();
       secondNodeId = edgeData.getSecondNodeId();
       edgeId = edgeData.getId();
   }

   public StepMessage(StepMessageType messageType){
       this.messageType = messageType;
       firstNodeId = -1;
       secondNodeId = -1;
       edgeId = -1;
       weightShift = -1;
   }

   public StepMessageType getType(){
       return messageType;
   }

   public int getFirstNodeId(){
       return firstNodeId;
   }

   public int getEdgeId() {
       return edgeId;
   }

   public int getSecondNodeId() {
       return secondNodeId;
   }

   int getWeightShift(){
       return weightShift;
   }
}
```

Файл: StepMessageType.java

```
package com.kruskal.kruskalalgorithm;

public enum StepMessageType {
   EDGEADDED,
   EDGEDECLINED,
```

```
   ALGORITHMENDED
}
```

Файл: EdgeView.java

```
package com.kruskal.shapeview;

import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;
import javafx.scene.text.TextFlow;
import javafx.util.Pair;

public class EdgeView extends Line {
   private int idNumber;
   private final int weight;
   private Text weightText;
   private final TextFlow textFlow = new TextFlow();

   public TextFlow getTextFlow() {
       return textFlow;
   }

   private Pair<NodeView, NodeView> adjacentNodes;

   public int getIdNumber() {
       return idNumber;
   }

   public void setIdNumber(int idNumber) {
       this.idNumber = idNumber;
   }

   public Text getWeightText() {
       return weightText;
   }

   public EdgeView(double v, double v1, double v2, double v3, int idNumber, int
weight) {
       super(v, v1, v2, v3);
       this.idNumber = idNumber;
       this.weight = weight;
       this.weightText = new Text(Integer.toString(weight));
   }

   public EdgeView(int idNumber, int weight) {
       super();
       this.idNumber = idNumber;
       this.weight = weight;
       this.weightText = new Text(Integer.toString(weight));
       this.weightText.setStyle("-fx-fill: black; -fx-font-weight: bold");
       this.textFlow.setStyle("-fx-background-color: white");
       this.textFlow.setPrefWidth(40);
       this.textFlow.setPrefHeight(20);
       this.textFlow.setTextAlignment(TextAlignment.CENTER);
   }
   public void setAdjacentNodes(NodeView startNode, NodeView endNode) {
       adjacentNodes = new Pair<>(startNode, endNode);
   }

   public NodeView getStartNode() {
       return adjacentNodes.getKey();
   }
```

```
    public NodeView getEndNode() {
        return adjacentNodes.getValue();
    }
}
```

Файл: NodeView.java

```
package com.kruskal.shapeview;

import javafx.scene.paint.Paint;
import javafx.scene.shape.Circle;
import javafx.scene.text.Text;
import javafx.util.Pair;

import java.util.ArrayList;
import java.util.List;

public class NodeView extends Circle {
    private final int idNumber;
    private final Text text;
    private final List<EdgeView> incidentEdgeIdList = new ArrayList<>();

    public int getIdNumber() {
        return idNumber;
    }

    public NodeView(double x, double y, double radius, Paint paint, int idNumber)
{
        super(x, y, radius, paint);
        this.idNumber = idNumber;
        this.text = new Text(x, y, Integer.toString(idNumber));
        this.text.setStyle("-fx-font-size: 20");
    }

    public Text getText() {
        return text;
    }

    public void addIncidentEdgeId(EdgeView edge) {
        incidentEdgeIdList.add(edge);
    }

    public boolean hasIncidentEdge(EdgeView edge) {
        return incidentEdgeIdList.contains(edge);
    }

    public void removeEdge(EdgeView edge) {
        incidentEdgeIdList.remove(edge);
    }

    public boolean isStartVertex(EdgeView edge) {
        return edge.getStartNode().equals(this);
    }
}
```

Файл: ShapeContainer.java

```
package com.kruskal.shapeview;

import com.kruskal.graph.Edge;
import com.kruskal.graph.Node;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
```

```
import java.util.ArrayList;
import java.util.List;

public class ShapeContainer {
   private final List<NodeView> nodeViewList = new ArrayList<>();
   private final List<EdgeView> edgeViewList = new ArrayList<>();
   private final Pane pane;


   public ShapeContainer(Pane pane) {
       this.pane = pane;
   }

   public void createNode(double xCoordinate, double yCoordinate, double radius,
Color color, int nodeId) {
       NodeView node = new NodeView(xCoordinate, yCoordinate, radius, color,
nodeId);
       nodeViewList.add(node);
       pane.getChildren().addAll(node, node.getText());
   }

   public void createEdge(int startNodeId, int endNodeId, int weight, int edgeId)
{
       EdgeView edgeView = new EdgeView(edgeId, weight);
       NodeView startNode = null, endNode = null;
       for (NodeView node : nodeViewList) {
           if (node.getIdNumber() == startNodeId) {
               startNode = node;
               edgeView.setStartX(node.getCenterX());
               edgeView.setStartY(node.getCenterY());
               node.addIncidentEdgeId(edgeView);
           } else if (node.getIdNumber() == endNodeId) {
               endNode = node;
               edgeView.setEndX(node.getCenterX());
               edgeView.setEndY(node.getCenterY());
               node.addIncidentEdgeId(edgeView);
           }
       }
       edgeView.setAdjacentNodes(startNode, endNode);
       edgeView.setStroke(Color.BLACK);
       edgeView.setStrokeWidth(5d);
       edgeView.getTextFlow().setLayoutX((edgeView.getEndX() +
edgeView.getStartX()) / 2 - 20);
       edgeView.getTextFlow().setLayoutY((edgeView.getEndY() +
edgeView.getStartY()) / 2 - 10);
       edgeView.getTextFlow().getChildren().add(edgeView.getWeightText());
       edgeViewList.add(edgeView);
       pane.getChildren().addAll(edgeView, edgeView.getTextFlow());
   }

   public void removeNode(int nodeId) {
       for (NodeView node : nodeViewList) {
           if (nodeId == node.getIdNumber()) {
               List<EdgeView> needToBeDeletedEdge = new ArrayList<>();
               for (EdgeView edge : edgeViewList) {
                   if (node.hasIncidentEdge(edge)) {
                       if (!node.equals(edge.getStartNode())) {
                           edge.getStartNode().removeEdge(edge);
                       } else {
                           edge.getEndNode().removeEdge(edge);
                       }
                       pane.getChildren().remove(edge.getTextFlow());
```

```
                        pane.getChildren().remove(edge);
                        needToBeDeletedEdge.add(edge);
                        node.removeEdge(edge);
                    }
                }
                for (EdgeView edge : needToBeDeletedEdge) {
                    if (edgeViewList.contains(edge)) {
                        edgeViewList.remove(edge);
                    }
                }
                nodeViewList.remove(node);
                pane.getChildren().remove(node.getText());
                pane.getChildren().remove(node);
                break;
            }
        }
    }

    public void removeEdge(int edgeId) {
        for (EdgeView edge : edgeViewList) {
            if (edge.getIdNumber() == edgeId) {
                edge.getStartNode().removeEdge(edge);
                edge.getEndNode().removeEdge(edge);
                edgeViewList.remove(edge);
                pane.getChildren().remove(edge.getTextFlow());
                pane.getChildren().remove(edge);
                break;
            }
        }
    }

    public void clear() {
        nodeViewList.clear();
        edgeViewList.clear();
        pane.getChildren().clear();
    }

    public void replaceNode(double x, double y, int objectId) {
        for (NodeView node : nodeViewList) {
            if (node.getIdNumber() == objectId) {
                node.setCenterX(x);
                node.setCenterY(y);
                node.getText().setX(x);
                node.getText().setY(y);
                for (EdgeView edge : edgeViewList) {
                    if (node.hasIncidentEdge(edge) && node.isStartVertex(edge)) {
                        edge.setStartX(x);
                        edge.setStartY(y);
                        edge.getTextFlow().setLayoutX((edge.getEndX() +
edge.getStartX()) / 2 - 10);
                        edge.getTextFlow().setLayoutY((edge.getEndY() +
edge.getStartY()) / 2 - 10);
                    } else if (node.hasIncidentEdge(edge)) {
                        edge.setEndX(x);
                        edge.setEndY(y);
                        edge.getTextFlow().setLayoutX((edge.getEndX() +
edge.getStartX()) / 2 - 10);
                        edge.getTextFlow().setLayoutY((edge.getEndY() +
edge.getStartY()) / 2 - 10);
                    }
                }
                break;
            }
```

```
        }
    }

    public void colorEdge(int edgeId, Color color) {
        for (EdgeView edge : edgeViewList) {
            if (edge.getIdNumber() == edgeId) {
                edge.setStroke(color);
            }
        }
    }
}
```

Файл: Edge.java

```
package com.kruskal.graph;

public class Edge {
    private final Node firstNode;
    private final Node secondNode;
    private final int weight;
    private final int id;

    Edge(Node first, Node second, int weight, int id){
        this.firstNode = first;
        this.secondNode = second;
        this.weight = weight;
        this.id = id;
    }

    @Override
    public boolean equals(Object obj){
        if(this == obj){
            return true;
        }
        if(obj instanceof Edge){
            if(this.firstNode.equals(((Edge)obj).firstNode)){
                if(this.secondNode.equals(((Edge)obj).secondNode)){
                    return true;
                }
            }

            if(this.firstNode.equals(((Edge)obj).secondNode)){
                if(this.secondNode.equals(((Edge)obj).firstNode)){
                    return true;
                }
            }
        }
        return false;
    }

    int getWeight() {
        return weight;
    }

    Node getFirstNode() {
        return firstNode;
    }

    Node getSecondNode() {
        return secondNode;
    }

    int getId() {
```

```
        return id;
    }

    Node getAdjacentNode(Node start){
        if(firstNode.equals(start)){
            return secondNode;
        }
        return firstNode;
    }

    @Override
    public String toString(){
        return "(EdgeId:"+ id +" FirstNodeId:"+ firstNode.getId()+"
SecondNodeId:"+secondNode.getId() +" Weight"+ weight + ")";
    }
}
```

Файл: EdgeData.java

```
package com.kruskal.graph;

public class EdgeData {
    private final int id;
    private final int weight;
    private final int firstNodeId;
    private final int secondNodeId;

    EdgeData(Edge edge){
        this.id = edge.getId();
        this.weight = edge.getWeight();
        this.firstNodeId = edge.getFirstNode().getId();
        this.secondNodeId = edge.getSecondNode().getId();
    }

    public int getId(){
        return id;
    }

    public int getWeight() {
        return weight;
    }

    public int getFirstNodeId() {
        return firstNodeId;
    }

    public int getSecondNodeId() {
        return secondNodeId;
    }
}
```

Файл: Graph.java

```
package com.kruskal.graph;

import java.util.List;
import java.util.Set;
import java.util.ArrayList;
import java.util.HashSet;

public class Graph {
    private final List<Node> nodes;

    Graph(){
```

```
    nodes = new ArrayList<Node>();
};

private Node findNode(int nodeId){
    for(Node currentNode: nodes){
        if(currentNode.getId() == nodeId){
            return currentNode;
        }
    }
    return null;
}

boolean deleteNode(int nodeId){
    Node currentNode = findNode(nodeId);
    if(currentNode != null){

        List<Edge> edges = currentNode.getAdjacentEdges();
        while (!edges.isEmpty()) {
            Edge target = edges.get(0);
            currentNode.deleteEdge(target.getId());
        }

        nodes.remove(currentNode);
        return true;
    }
    return false;
}

boolean addNode(int newNodeId){
    Node newNode = new Node(newNodeId);
    if(findNode(newNode.getId()) == null){
        nodes.add(newNode);
        return true;
    }
    return false;
}

boolean addEdge(int firstNodeId, int secondNodeId, int weight, int edgeId){
    List<Edge> allEdges = getEdgesList();
    for(Edge edge: allEdges){
        if(edge.getId() == edgeId){
            return false;
        }
    }

    Node firstNode = findNode(firstNodeId);
    if(firstNode == null){
        return false;
    }

    Node secondNode = findNode(secondNodeId);
    if(secondNode == null){
        return false;
    }

    Edge newEdge = new Edge(firstNode, secondNode, weight, edgeId);

    if(newEdge.getFirstNode().getAdjacentEdges().contains(newEdge)){
        return false;
    }
    newEdge.getFirstNode().addEdge(newEdge);
    return true;
}
```

```
boolean deleteEdge(int edgeId){
    List<Edge> allEdges = getEdgesList();
    for(Edge edge: allEdges){
        if(edge.getId() == edgeId){
            edge.getFirstNode().deleteEdge(edgeId);
            return true;
        }
    }
    return false;
}

private List<Edge> getEdgesList(){
    List<Edge> allEdges = new ArrayList<Edge>();
    Set<Node> viewedNodes = new HashSet<Node>();
    List<Node> nodesToView = new ArrayList<Node>();

    if(nodes.isEmpty()){
        return allEdges;
    }

    nodesToView.add(nodes.get(0));
    while(!nodesToView.isEmpty()){
        Node currentNode = nodesToView.get(0);
        nodesToView.remove(0);

        if(viewedNodes.contains(currentNode)){
            continue;
        }

        viewedNodes.add(currentNode);

        for (Edge adjacent: currentNode.getAdjacentEdges()){
            if(!allEdges.contains(adjacent)){
                allEdges.add(adjacent);
            }

            nodesToView.add(adjacent.getAdjacentNode(currentNode));

        }
    }

    return allEdges;
}

public void printGraph(){
    for (Node node: nodes) {
        System.out.println(node);
    }
}

public boolean isValid(){
    Set<Node> viewedNodes = new HashSet<Node>();
    List<Node> nodesToView = new ArrayList<Node>();

    if(nodes.isEmpty()){
        return true;
    }

    nodesToView.add(nodes.get(0));
    while(!nodesToView.isEmpty()){
        Node currentNode = nodesToView.get(0);
        nodesToView.remove(0);
```

```
            if(viewedNodes.contains(currentNode)){
                continue;
            }

            viewedNodes.add(currentNode);

            for (Edge adjacent: currentNode.getAdjacentEdges()){
                nodesToView.add(adjacent.getAdjacentNode(currentNode));
            }
        }

        for (Node node : nodes){
            if(!viewedNodes.contains(node)){
                return false;
            }
        }

        return true;
    }

    public Set<Integer> getNodesId(){
        Set<Integer> set = new HashSet<>();
        for (Node node : nodes) {
            set.add(node.getId());
        }
        return set;
    }

    public List<EdgeData> getEdgesData(){
        List<EdgeData> data = new ArrayList<EdgeData>();
        for (Edge edge : getEdgesList()) {
            data.add(new EdgeData(edge));
        }
        return data;
    }
}
```

Файл: GraphBuilder.java

```
P                       ackage com.kruskal.graph;

import java.util.List;


public class GraphBuilder {

    private Graph graph;
    public GraphBuilder(){
            graph = new Graph();
    };

    public boolean deleteNode(int nodeId){
        return graph.deleteNode(nodeId);
    }

    public boolean deleteEdge(int edgeId){
        return graph.deleteEdge(edgeId);
    }

    public boolean addNode(int newNodeId){
        return graph.addNode(newNodeId);
    }
```

```
   public boolean addEdge(int firstNodeId, int secondNodeId, int weight, int
edgeId){
        if(weight > 0){
            return graph.addEdge(firstNodeId, secondNodeId, weight, edgeId);
        }
        return false;
   }

   public boolean isValid(){
        return graph.isValid();
   }

   public void printGraph(){
        graph.printGraph();
   }

   public List<EdgeData> getEdgesData(){
        return graph.getEdgesData();
   }

   public void resetGraph(){
        graph = new Graph();
   }

   public Graph getGraph() {
        return graph;
   }
}
```

Файл: GraphConnectednessExeption.java

```
package com.kruskal.graph;

public class GraphConnectednessExeption extends RuntimeException{
   public GraphConnectednessExeption(String message){
        super(message);
   }

   @Override
   public String toString() {
        return "FileFormatException{" + getMessage() + "}";
   }
}
```

Файл: Node.java

```
package com.kruskal.graph;

import java.util.ArrayList;
import java.util.List;


public class Node {
   private final int id;
   private final List<Edge> adjacentEdges;
   Node(int id){
        this.id = id;
```

```
        adjacentEdges = new ArrayList<Edge>();
    };

    public int getId() {
        return id;
    }

    void addEdge(Edge newEdge){
        adjacentEdges.add(newEdge);
        newEdge.getAdjacentNode(this).adjacentEdges.add(newEdge);
    }

    private Edge findEdge(int edgeId){
        for(Edge currentEdge: adjacentEdges){
            if(currentEdge.getId() == edgeId){
                return currentEdge;
            }
        }
        return null;
    }

    void deleteEdge(int edgeId){
        Edge currentEdge = findEdge(edgeId);
        if(currentEdge != null){
            currentEdge.getFirstNode().adjacentEdges.remove(currentEdge);
            currentEdge.getSecondNode().adjacentEdges.remove(currentEdge);
        }
    }

    List<Edge> getAdjacentEdges() {
        return adjacentEdges;
    }

    @Override
    public boolean equals(Object obj){
        if(this == obj){
            return true;
        }
        if(obj instanceof Node){
            if (this.id == ((Node)obj).id){
                return true;
            }
        }
        return false;
    }

    @Override
    public String toString(){
        StringBuilder builder = new StringBuilder();
        builder.append("NodeId:" + id + ",  Edges:\n");
        for (Edge edge : adjacentEdges) {
            builder.append("\t" + edge + "\n");
        }
        return builder.toString();
    }
}
```

Файл: FileFormatException.java

```
package com.kruskal.fileparser;
public class FileFormatException extends RuntimeException{

    private final int fileStringNumber;
```

```
    public FileFormatException(String message, int fileStringNumber){
        super(message);
        this.fileStringNumber = fileStringNumber;
    }

    @Override
    public String toString() {
        return "FileFormatException{" + getMessage() + " (Строка файла: " +
fileStringNumber + ")}";
    }
}
```

Файл: FileReader.java

```
package com.kruskal.fileparser;

import com.kruskal.controlstructures.Mediator;
import com.kruskal.gui.ActionMessage;
import com.kruskal.gui.State;

import java.net.URLConnection;
import java.nio.file.Paths;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class FileReader {

    private enum ReadSymbol{
        SPACE,
        NUMBER
    }

    public void read(String inputFileName, double screenWidth, double
screenHeight, Mediator mediator){

        Path path = Paths.get(inputFileName);
        Scanner scanner;
        int nodeNumber;
        List<List<Integer>> nodeMatrix = new ArrayList<>();
        String type = URLConnection.guessContentTypeFromName(path.toString());
        if(!type.equals("text/plain")){
            throw new FileFormatException("Неверный тип файла", 0);
        }
        try{
            scanner = new Scanner(path);
        } catch (java.io.IOException e){
            throw new FileFormatException("Не удалось открыть файл для чтения",
0);
        }

        String line = scanner.nextLine();
        Scanner lineScanner = new Scanner(line);
        if(lineScanner.hasNextInt()){
            nodeNumber = lineScanner.nextInt();
            if (lineScanner.hasNext()){
                throw new FileFormatException("Неверно задана строка с размером
графа", 1);
            }
        } else{
            throw new FileFormatException("Неверно задана строка с размером
графа", 1);
```

```
        }
        if(nodeNumber > 100){
            throw new FileFormatException("Размер графа превышает 80 вершин", 1);
        }
        nodeReader(nodeNumber, screenWidth, screenHeight, mediator, nodeMatrix);
        scanMatrix(nodeMatrix, scanner, nodeNumber);
        nodeMatrixToGraph(nodeMatrix, mediator);
    }

    private void scanMatrix(List<List<Integer>> nodeMatrix, Scanner scanner, int
nodeNumber){
        for(int i = 0; i < nodeNumber; i++){
            if(!scanner.hasNextLine()) {
                throw new FileFormatException("Задана матрица меньшего размера,
чем указано в строке с размером", i+2);
            }
            String line = scanner.nextLine();
            Scanner lineScanner = new Scanner(line).useDelimiter("");
            ReadSymbol readSymbol = ReadSymbol.NUMBER;
            int numberCount = 0;
            for(int j = 1; j <= line.length(); j++) {
                if((lineScanner.hasNextInt() && (readSymbol == ReadSymbol.SPACE))
||
                        (!lineScanner.hasNextInt() && (readSymbol ==
ReadSymbol.NUMBER))){
                    throw new FileFormatException("Символ вместо ожидаемого
числа", i+2);
                }
                if(readSymbol == ReadSymbol.NUMBER) {
                    int weight = 0;
                    while (lineScanner.hasNextInt()) {
                        weight = weight * 10 + lineScanner.nextInt();
                        if(weight >= 10){
                            j++;
                        }
                    }

                    if(i == numberCount && weight != 0){
                        throw new FileFormatException("Значение на диалогнали
матрицы отличное от 0", i+2);
                    }

                    nodeMatrix.get(i).add(weight);
                    if(numberCount < i
&& !(nodeMatrix.get(i).get(numberCount).equals(nodeMatrix.get(numberCount).get(i)
))){
                        throw new FileFormatException("Матрица не симметричная",
i+2);
                    }
                    numberCount++;
                } else{
                    String next = lineScanner.next();
                    if(!next.equals(" ")) {
                        throw new FileFormatException("Присутствуют символы кроме
разделительного пробела", i+2);
                    }
                }
                if(readSymbol == ReadSymbol.NUMBER){
                    readSymbol = ReadSymbol.SPACE;
                } else {
                    readSymbol = ReadSymbol.NUMBER;
                }
            }
```

```
            if(numberCount != nodeNumber){
                throw new FileFormatException("В строке не то количество чисел,
которое ожидалось", i+2);
            }
        }
        if(scanner.hasNextLine()){
            throw new FileFormatException("В файле больше строк, чем ожидалось",
nodeNumber+2);
        }
        scanner.close();
    }

    private void nodeMatrixToGraph(List<List<Integer>> nodeMatrix, Mediator
mediator){
        for(int raw = 1; raw <= nodeMatrix.size(); raw++){
            for(int column = 1; column <= nodeMatrix.size(); column++){
                if(nodeMatrix.get(raw-1).get(column-1) != 0){
                    mediator.sendRequest(new ActionMessage(State.ADDEDGE, raw,
column, nodeMatrix.get(raw-1).get(column-1)));
                }
            }
        }
    }

    private void nodeReader(int nodeNumber, double screenWidth, double
screenHeight, Mediator mediator, List<List<Integer>> nodeMatrix){
        double radius = Double.min((screenHeight - 80)/2, (screenWidth - 80)/2);
        double centreY = screenHeight/2;
        double centreX = screenWidth/2;
        int layerNodeNumber = 40;
        if (nodeNumber < layerNodeNumber){
            layerNodeNumber = nodeNumber;
        }
        double degreeStep = Math.toRadians((double) 360/(layerNodeNumber));
        int layerNodeIterator = 0;
        for(int i = 1; i <= nodeNumber; i++){
            mediator.sendRequest(new ActionMessage(State.ADDNODE, centreX -
Math.sin(degreeStep*layerNodeIterator)*radius,
                    centreY - Math.cos(degreeStep*layerNodeIterator)*radius));
            nodeMatrix.add(new ArrayList<>());
            layerNodeIterator++;
            if(layerNodeIterator == layerNodeNumber){
                layerNodeNumber -= 10;
                layerNodeIterator = 0;
                if (nodeNumber - i < layerNodeNumber){
                    layerNodeNumber = nodeNumber - i;
                }
                degreeStep =  Math.toRadians((double) 360/(layerNodeNumber));
                radius = radius - 60;

            }
        }
    }
}
```

Файл: FileWriter.java

```
package com.kruskal.fileparser;

import com.kruskal.graph.EdgeData;

import java.io.File;
import java.io.FileNotFoundException;
```

```
import java.io.PrintWriter;
import java.net.URLConnection;
import java.util.*;

public class FileWriter {

   public void write(String outFileName, List<EdgeData> edgesData, Set<Integer>
nodesId){
       int nodesNumber = nodesId.size();
       File file;
       String type = URLConnection.guessContentTypeFromName(outFileName);
       if(!type.equals("text/plain")){
           throw new FileFormatException("Неверный тип файла", 0);
       }
       try {
           file = new File(outFileName);
           file.createNewFile();
       }
       catch (Exception e) {
           throw new FileFormatException("Неудалось создать/открыть файл для
записи", 0);
       }
       PrintWriter printWriter;
       try {
           printWriter = new PrintWriter(file);
       } catch (FileNotFoundException e) {
           throw new FileFormatException("Неудалось найти файл для записи", 0);
       }

       printWriter.println(nodesNumber);
       IdConverter idConverter = new IdConverter();
       Iterator<Integer> iterator = nodesId.iterator();
       for(int i = 1; i <= nodesNumber; i++){
           idConverter.addId(iterator.next(), i);
       }
       for(int firstNodeIndex = 1; firstNodeIndex <= nodesNumber;
firstNodeIndex++){
           for(int secondNodeIndex = 1; secondNodeIndex <= nodesNumber;
secondNodeIndex++){
               boolean edgeExists = false;
               for (EdgeData edgeData : edgesData) {
                   if (!((idConverter.getId(firstNodeIndex) ==
edgeData.getFirstNodeId() && idConverter.getId(secondNodeIndex) ==
edgeData.getSecondNodeId())
                           || (idConverter.getId(secondNodeIndex) ==
edgeData.getFirstNodeId() && idConverter.getId(firstNodeIndex) ==
edgeData.getSecondNodeId()))) {
                       continue;
                   }
                   printWriter.print(edgeData.getWeight());
                   edgeExists = true;
                   if (secondNodeIndex != nodesNumber) {
                       printWriter.print(" ");
                   }
               }
               if(!edgeExists) {
                   printWriter.print(0);
                   if (secondNodeIndex != nodesNumber) {
                       printWriter.print(" ");
                   }
               }
           }
           printWriter.println();
```

```
        }
        printWriter.close();
    }
}

class IdConverter{
    private Map<Integer, Integer> idMap;

    public IdConverter(){
        idMap = new HashMap<>();
    }

    public void addId(int id, int value){
        idMap.put(value, id);
    }

    public int getId(int value){
        return idMap.get(value);
    }
}
```

Файл: ControllerSynchronizer.java

```
package com.kruskal.controlstructures;

import com.kruskal.gui.ShapeController;
import com.kruskal.graph.Graph;
import com.kruskal.graph.GraphBuilder;

public class ControllerSynchronizer {
    private final ShapeController shapeController;
    private final GraphBuilder graphBuilder;
    private int freeNodeId;
    private int freeEdgeId;

    public ControllerSynchronizer(ShapeController shapeController, GraphBuilder
graphBuilder){
        this.graphBuilder = graphBuilder;
        this.shapeController = shapeController;
        freeNodeId = 1;
        freeEdgeId = 1;
    }

    public boolean addNode(double xCoordinate, double yCoordinate){
        if(graphBuilder.addNode(freeNodeId)){
            shapeController.addNode(xCoordinate, yCoordinate, freeNodeId);
            freeNodeId++;
            return true;
        }
        return false;
    }

    public boolean addEdge(int firstNodeId, int secondNodeId, int weight){
        if(graphBuilder.addEdge(firstNodeId, secondNodeId, weight, freeEdgeId)){
            shapeController.addEdge(firstNodeId, secondNodeId, weight,
freeEdgeId);
            freeEdgeId++;
            return true;
        }
        return false;
    }

    public boolean deleteNode(int nodeId){
```

```
        if(graphBuilder.deleteNode(nodeId)){
            shapeController.removeNode(nodeId);
            return true;
        }
        return false;
    }

    public boolean deleteEdge(int edgeId){
        if(graphBuilder.deleteEdge(edgeId)){
            shapeController.removeEdge(edgeId);
            return true;
        }
        return false;
    }

    public void eraseGraph(){
        freeEdgeId = 1;
        freeNodeId = 1;
        graphBuilder.resetGraph();
        shapeController.clear();
    }

    public Graph getGraph(){
        return graphBuilder.getGraph();
    }
}
```

Файл: Mediator.java

```
package com.kruskal.controlstructures;

import com.kruskal.fileparser.FileReader;
import com.kruskal.fileparser.FileWriter;
import com.kruskal.graph.Graph;
import com.kruskal.graph.GraphBuilder;
import com.kruskal.graph.GraphConnectednessExeption;
import com.kruskal.gui.ActionController;
import com.kruskal.gui.ActionMessage;
import com.kruskal.gui.ShapeController;
import com.kruskal.kruskalalgorithm.Kruskal;
import com.kruskal.kruskalalgorithm.StepMessage;

public class Mediator {
    private ActionController actionController;
    private ControllerSynchronizer controllerSynchronizer;
    private ShapeController shapeController;
    private Kruskal algorithm;
    private final FileReader fileReader;
    private final FileWriter fileWriter;

    public Mediator() {
        fileWriter = new FileWriter();
        fileReader = new FileReader();
    }

    public void setActionController(ActionController controller) {
        this.actionController = controller;
    }

    public void setShapeController(ShapeController shapeController){
        this.shapeController = shapeController;
        this.controllerSynchronizer = new ControllerSynchronizer(shapeController,
new GraphBuilder());
```

```
    }

    public void sendRequest(ActionMessage actionMessage) {
        switch (actionMessage.getState()) {
            case ADDNODE -> controllerSynchronizer.addNode(actionMessage.getX(),
actionMessage.getY());
            case ADDEDGE ->
controllerSynchronizer.addEdge(actionMessage.getObjectId(),
actionMessage.getSecondObjectId(), actionMessage.getWeight());
            case REMOVEEDGE ->
controllerSynchronizer.deleteEdge(actionMessage.getObjectId());
            case REMOVENODE ->
controllerSynchronizer.deleteNode(actionMessage.getObjectId());
            case RESTART -> controllerSynchronizer.eraseGraph();
            case REPLACENODE -> shapeController.replaceNode(actionMessage.getX(),
actionMessage.getY(), actionMessage.getObjectId());
            case UPLOADGRAPH -> {
                controllerSynchronizer.eraseGraph();
                fileReader.read(actionMessage.getFileName(), actionMessage.getX(),
actionMessage.getY(), this);
            }
            case SAVEGRAPH -> fileWriter.write(actionMessage.getFileName(),
controllerSynchronizer.getGraph().getEdgesData(),
controllerSynchronizer.getGraph().getNodesId());
            case RUNALGORITHM -> {
                Graph graph = controllerSynchronizer.getGraph();
                if(graph.isValid()){
                    algorithm = new Kruskal(graph);
                }else{
                    throw new GraphConnectednessExeption("Graph is not
connected");
                }
            }
            case NEXTSTEP -> {
                StepMessage stepMessage = algorithm.makeStep();
                actionController.printMessage(stepMessage);
                shapeController.paintEdge(stepMessage);

actionController.printTreeWeight(algorithm.getCurrentTreeWeight());
            }
            case PREVIOUSSTEP -> {
                StepMessage stepMessage = algorithm.stepBack();
                if (stepMessage != null) {
                    actionController.deleteLastMessage();
                    shapeController.paintEdgeDefault(stepMessage);

actionController.printTreeWeight(algorithm.getCurrentTreeWeight());
                } else {
                    actionController.blockPreviousStepButton();
                    actionController.deleteLastMessage();
                }
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ А

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

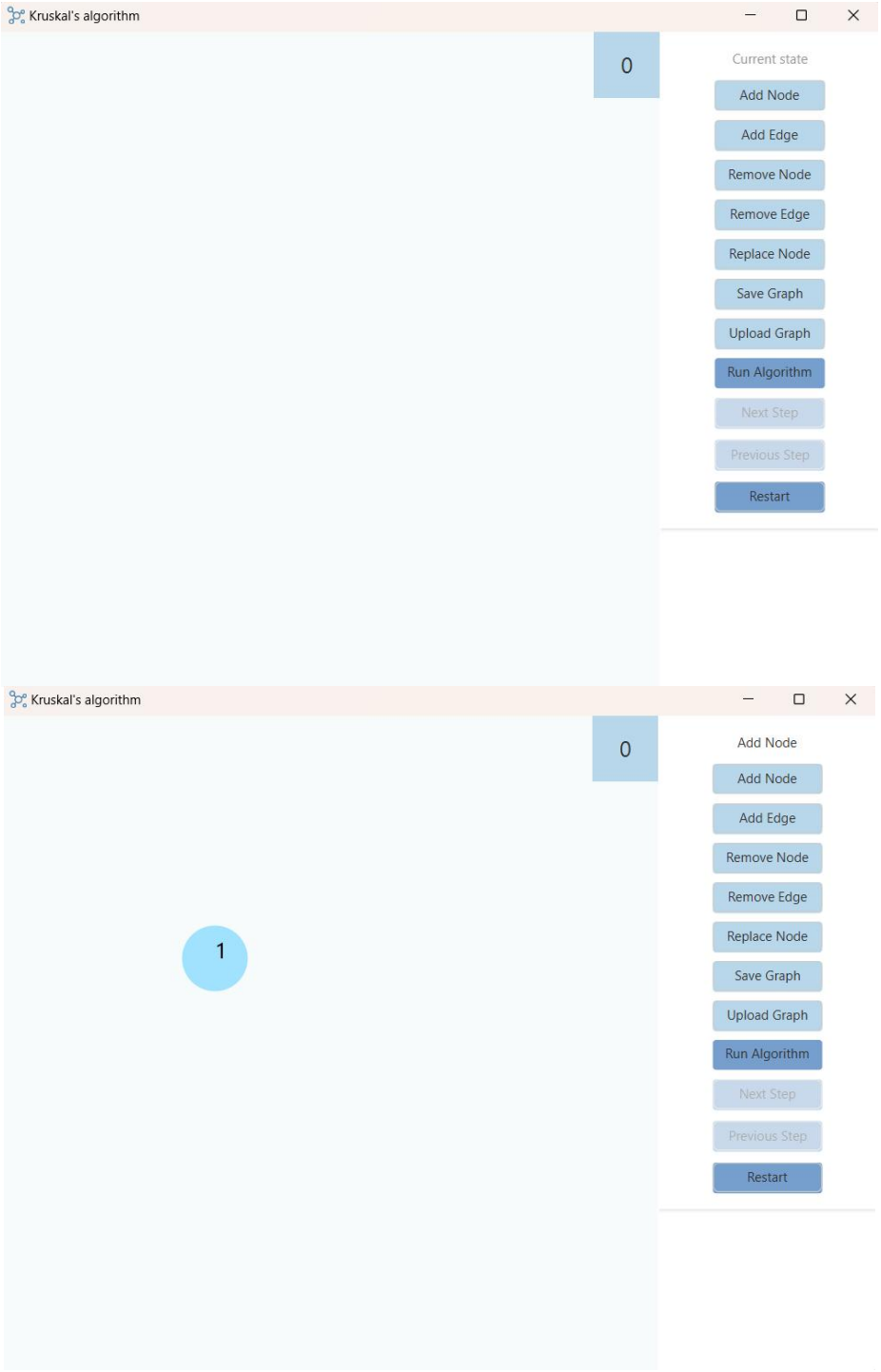

Рисунок 4 – Создание вершины графа

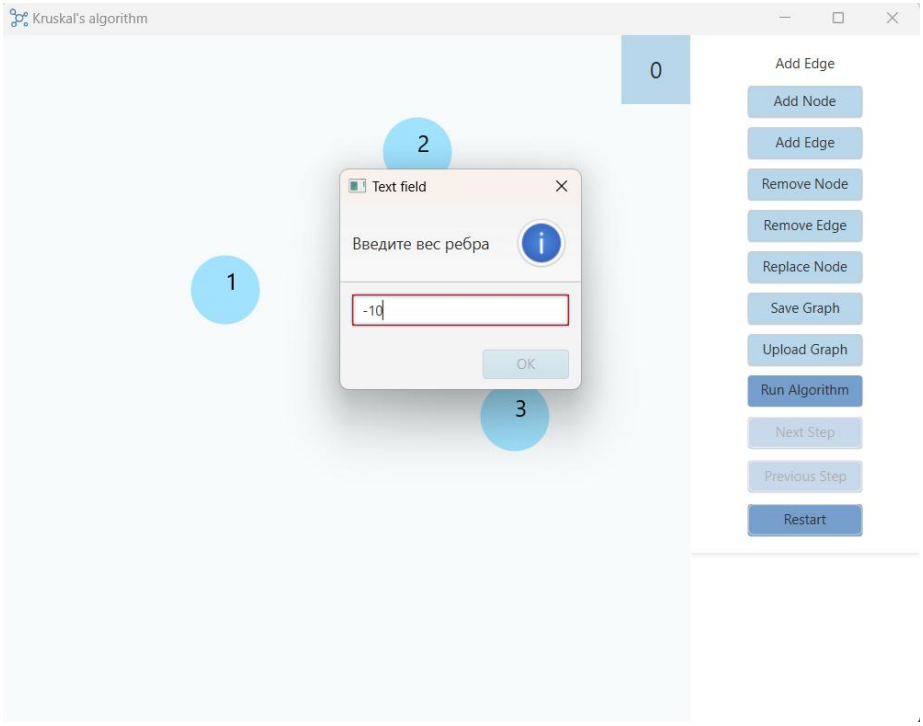

Рисунок 5 – Попытка создать ребро с отрицательным весом

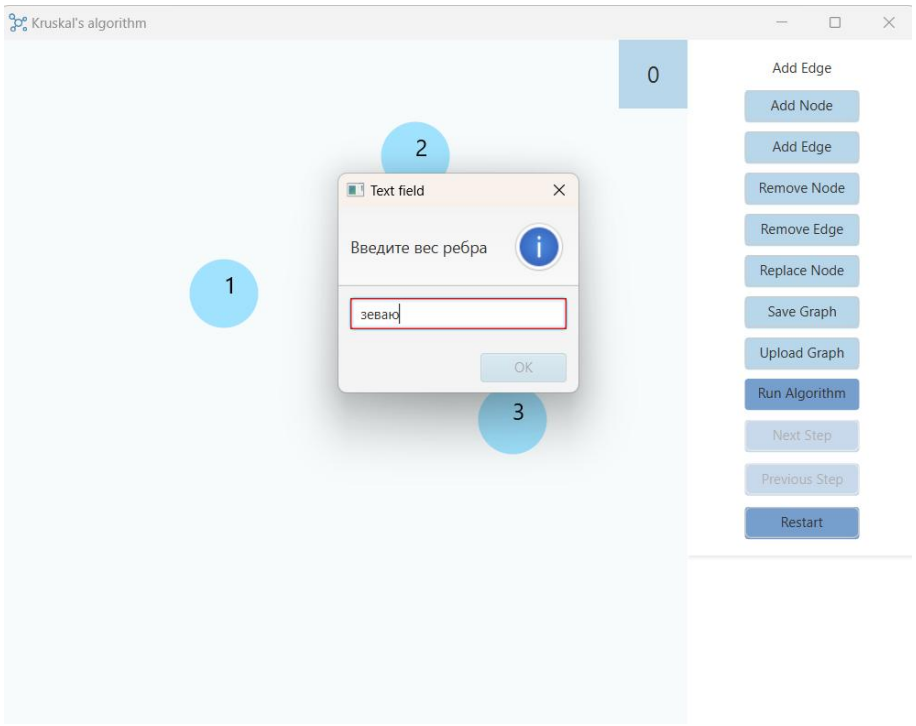

Рисунок 6 – Попытка создать ребро с некорректными данными вместо веса

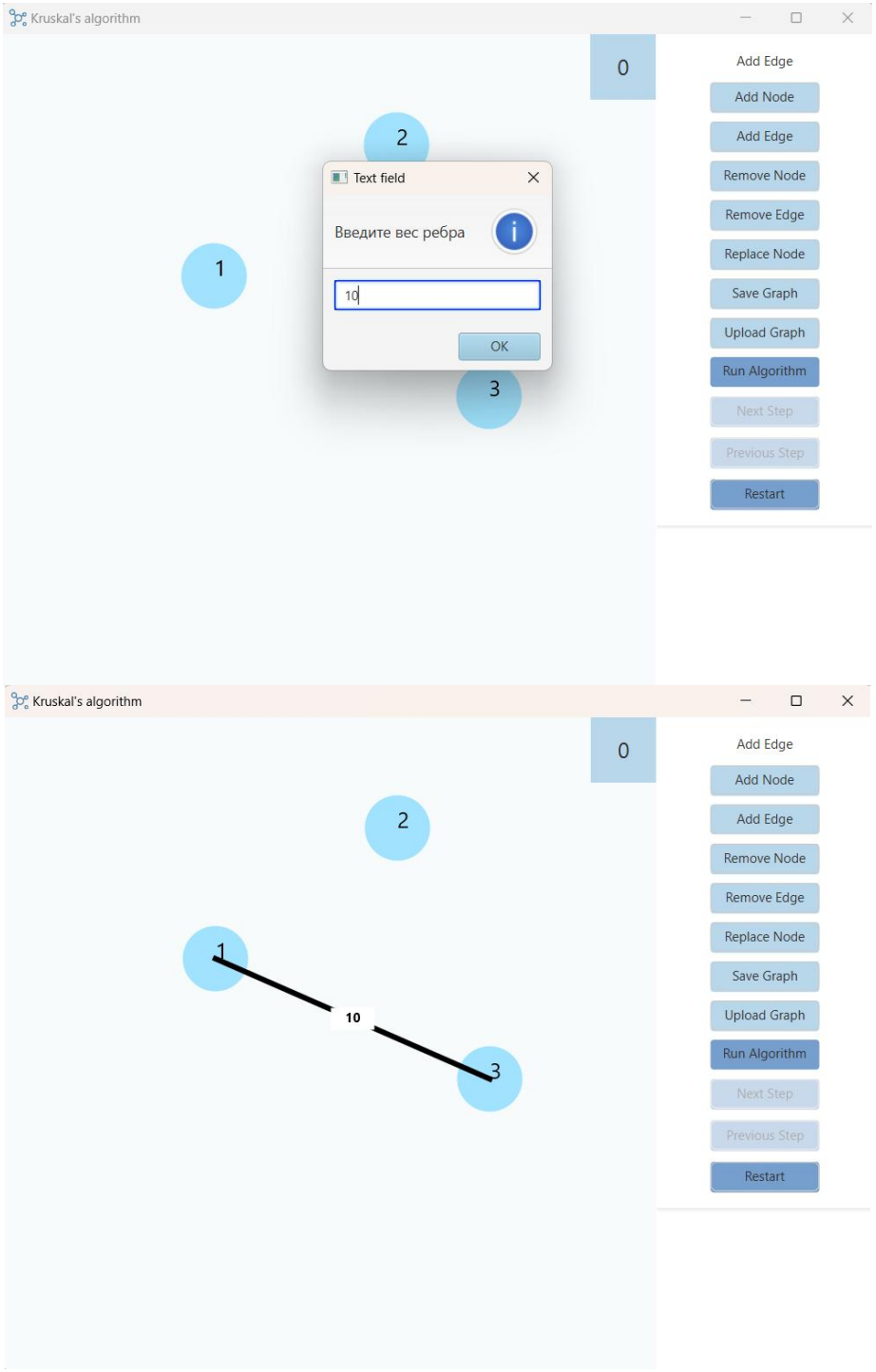

Рисунок 7 – Успешное создание корректного ребра

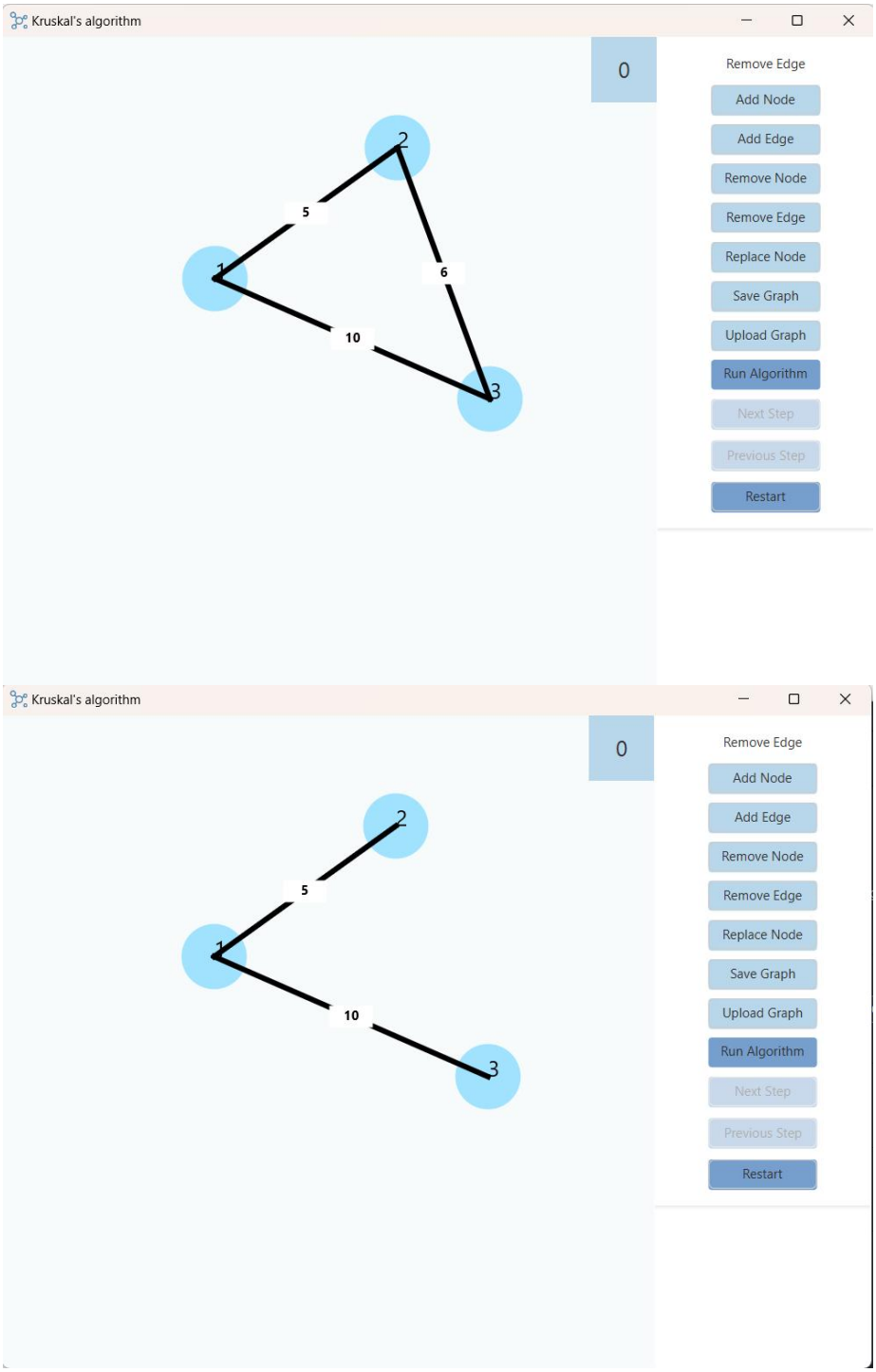

Рисунок 8 – Удаление ребра

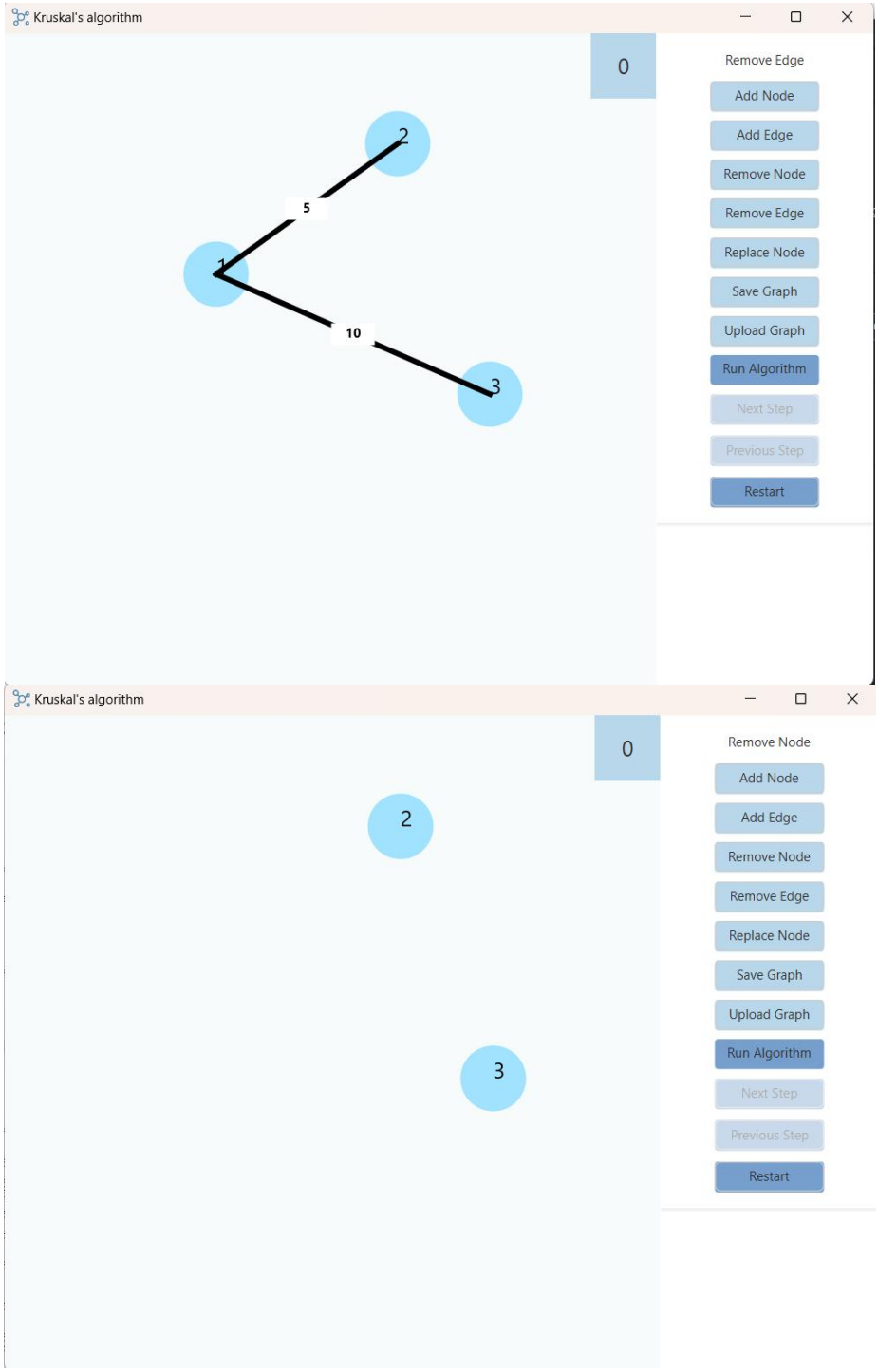

Рисунок 9 – Удаление вершины

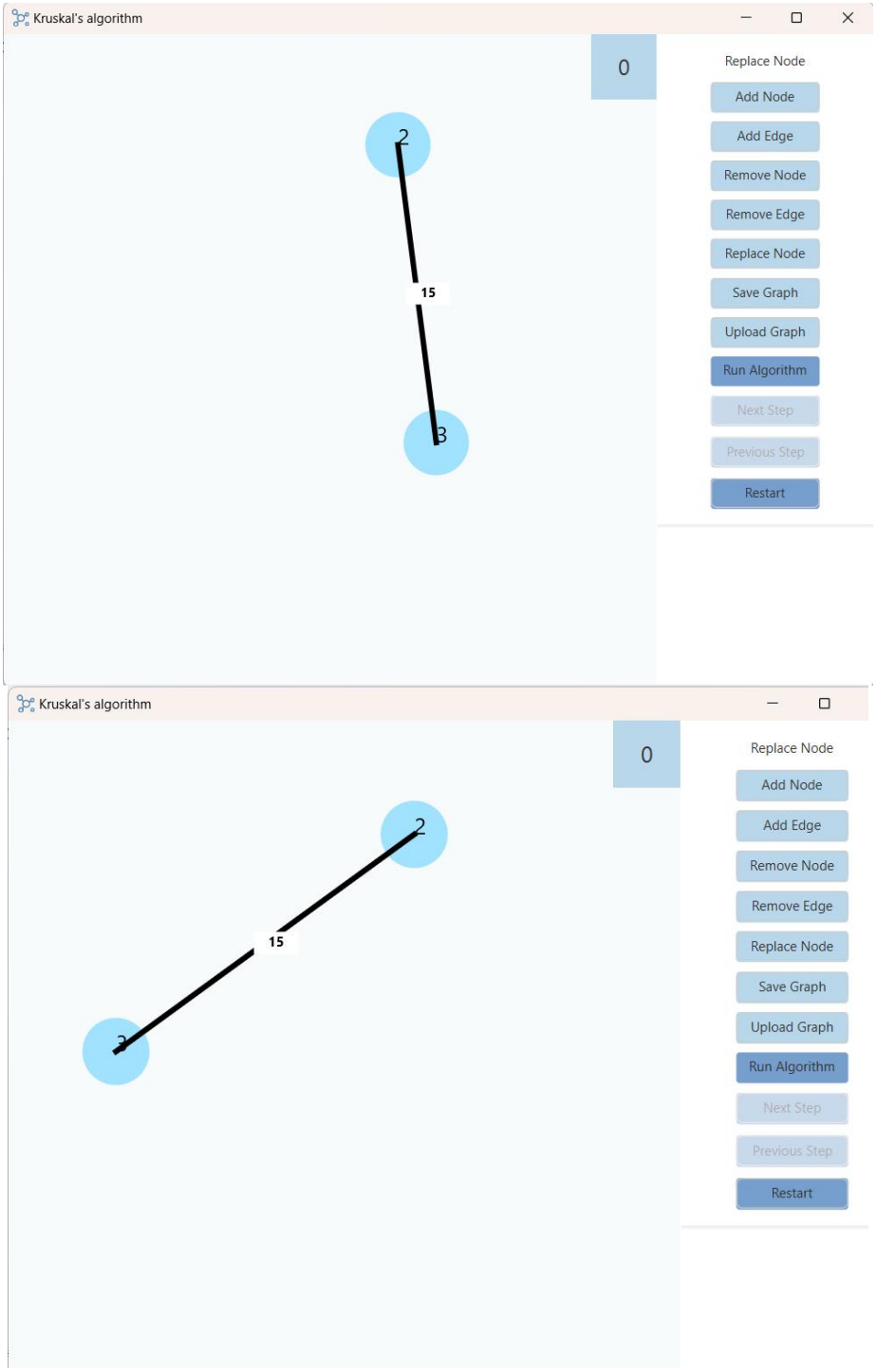

Рисунок 10 – Перемещение вершины

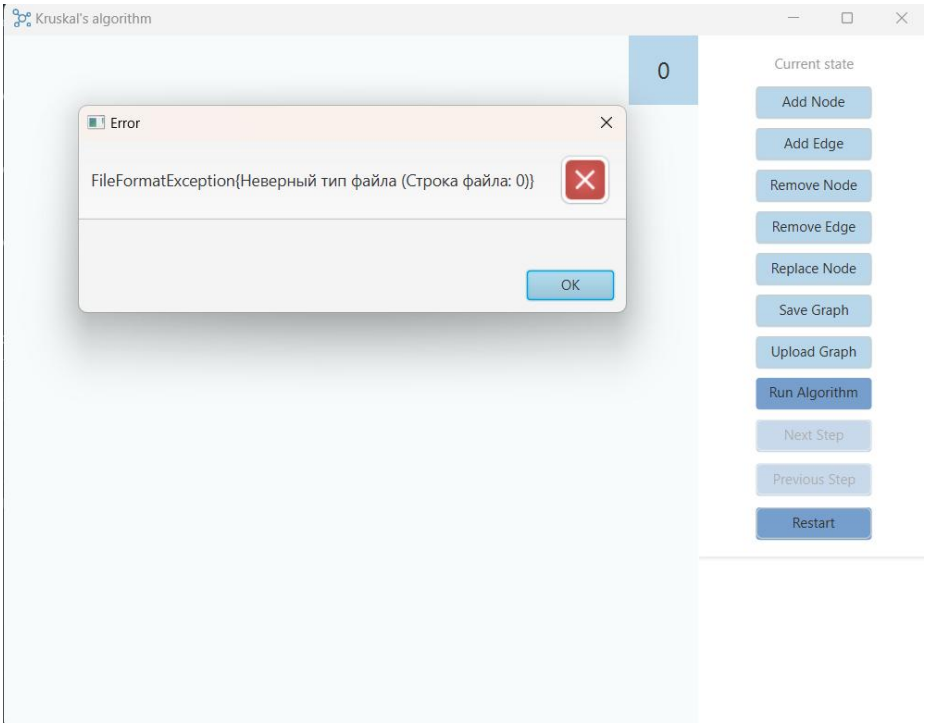

Рисунок 11 – Передан некорректный для считывания формат файла (не .txt)

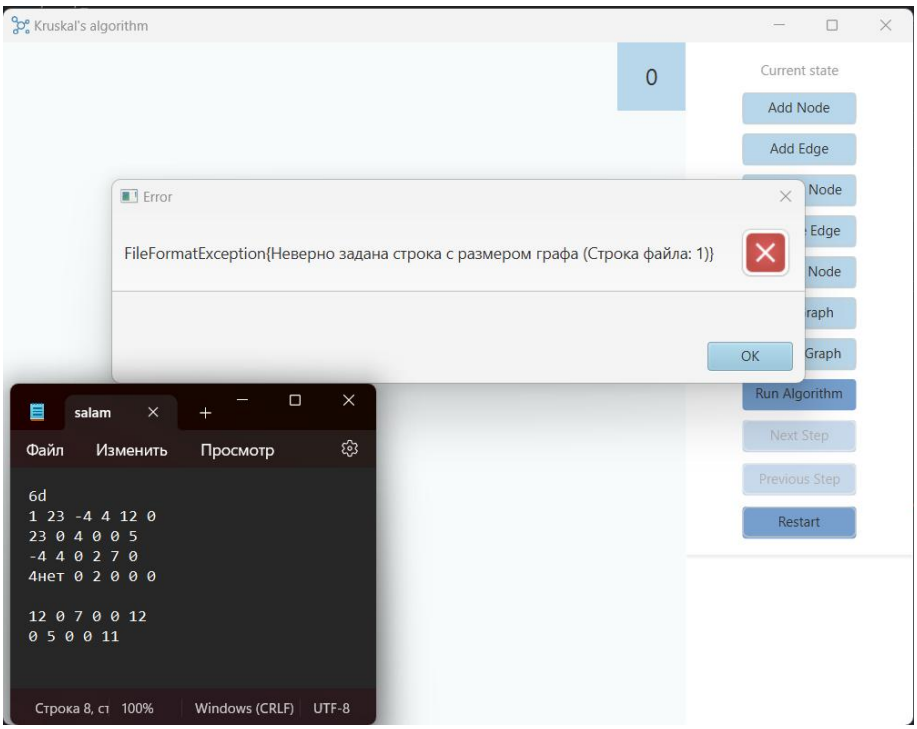

Рисунок 12 – Некорректная запись в строке с количеством вершин

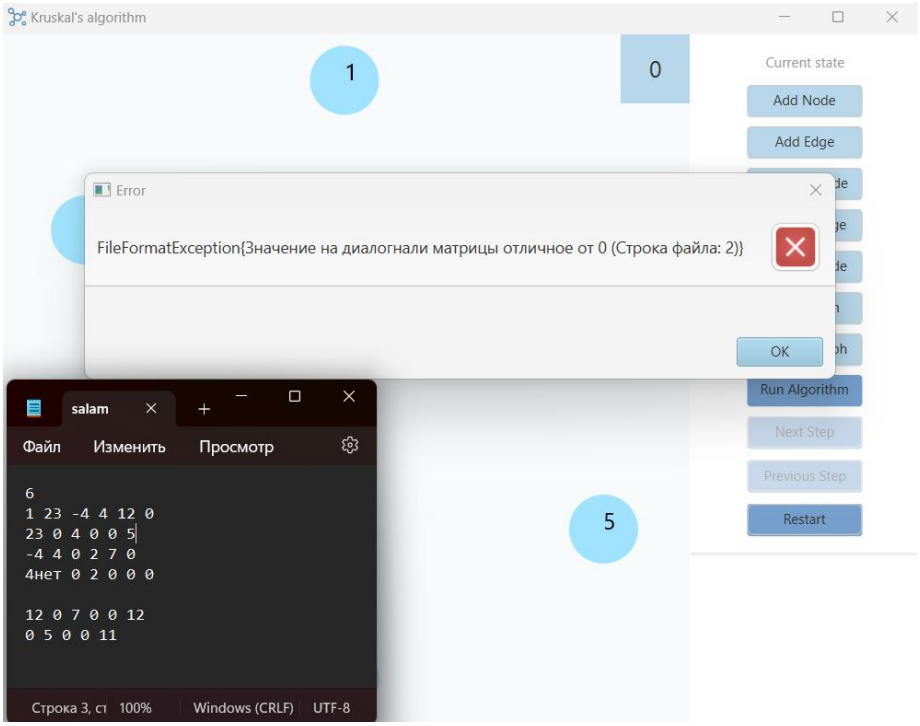

Рисунок 13 – На диагонали находится не 0 вес ребра, т.е. в гарфе есть петля

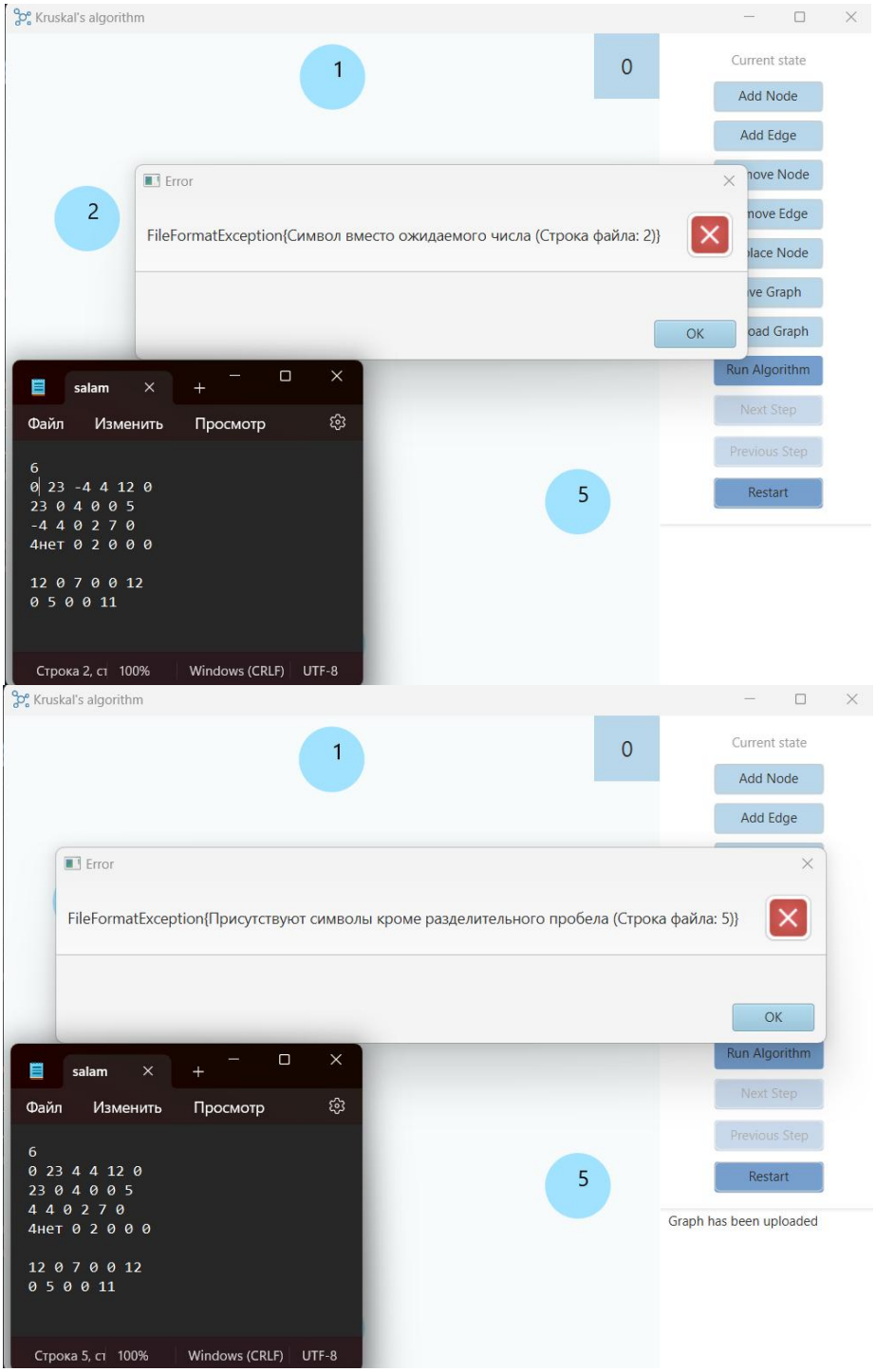

Рисунок 14 – Наличие символов кроме пробела

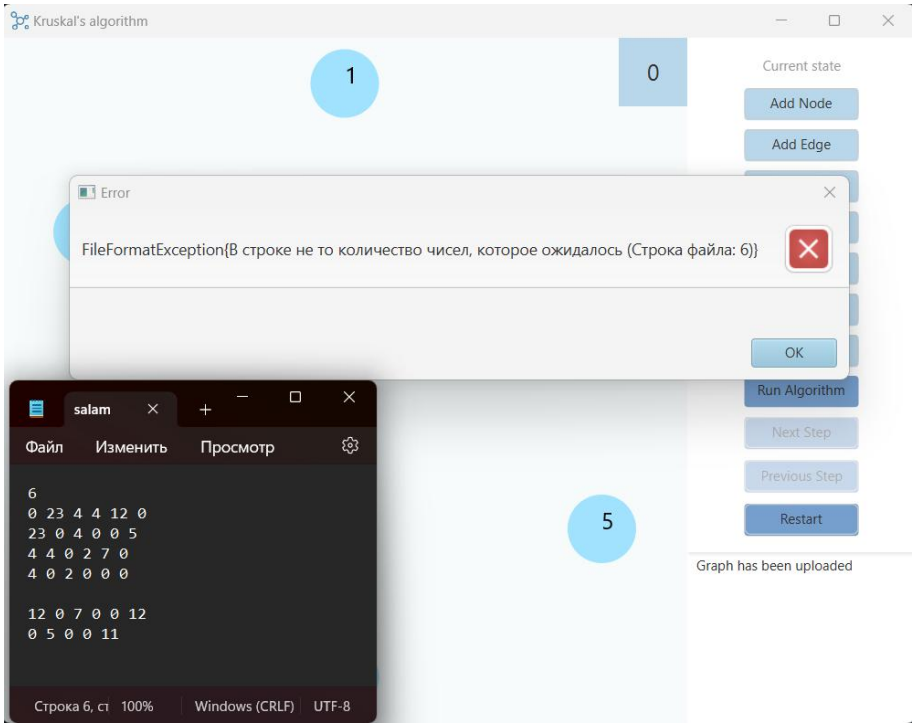

Рисунок 15 – Пустая строка посреди матрицы

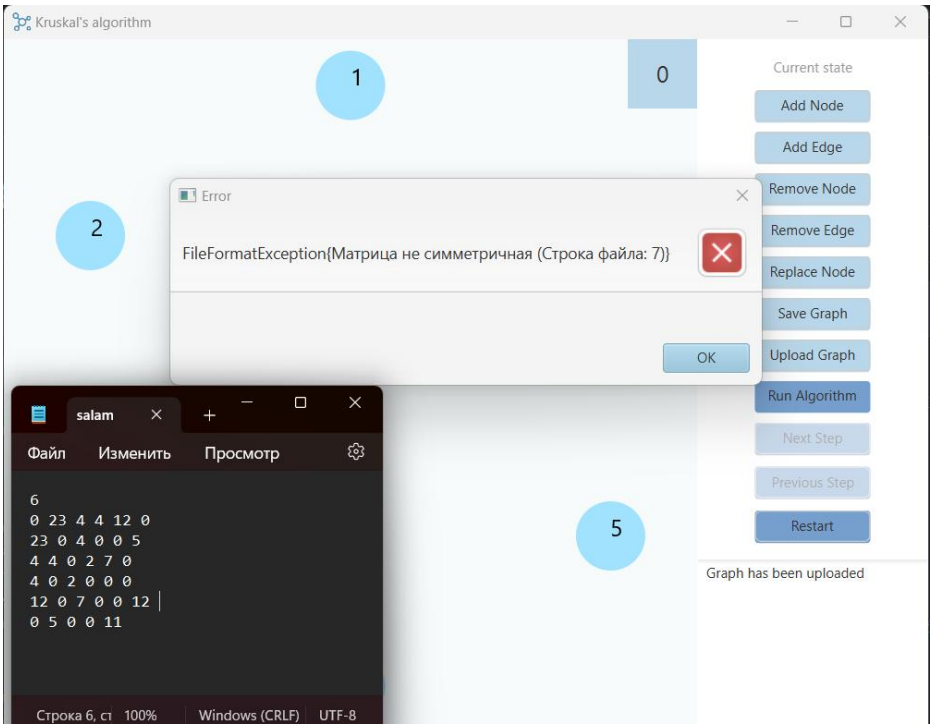

Рисунок 16 – Асимметрия матрицы рёбер

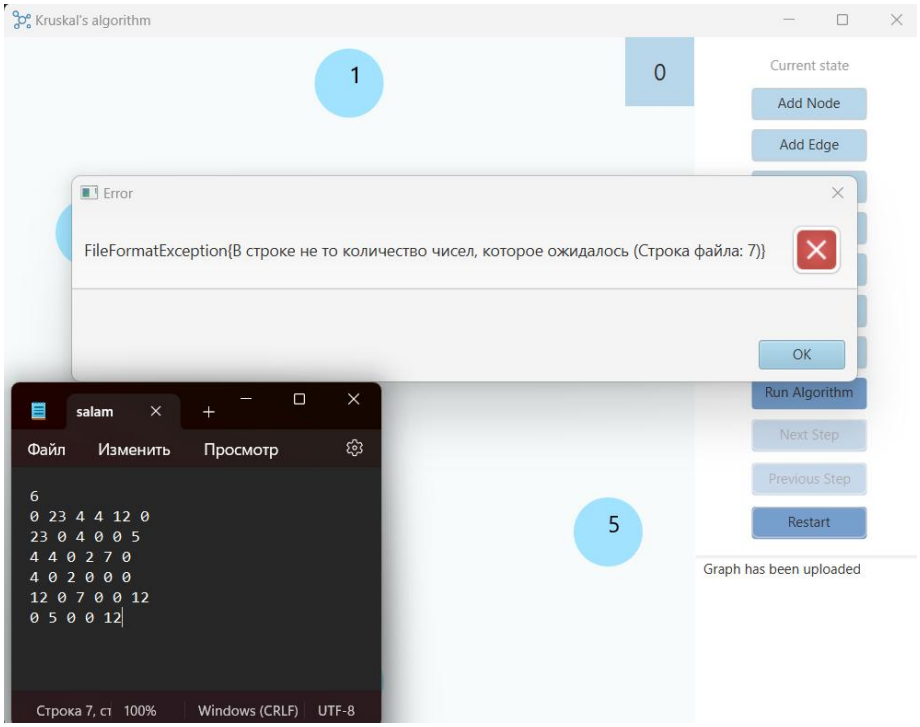

Рисунок 17 – Число рёбер в строке не совпадает с числом вершин

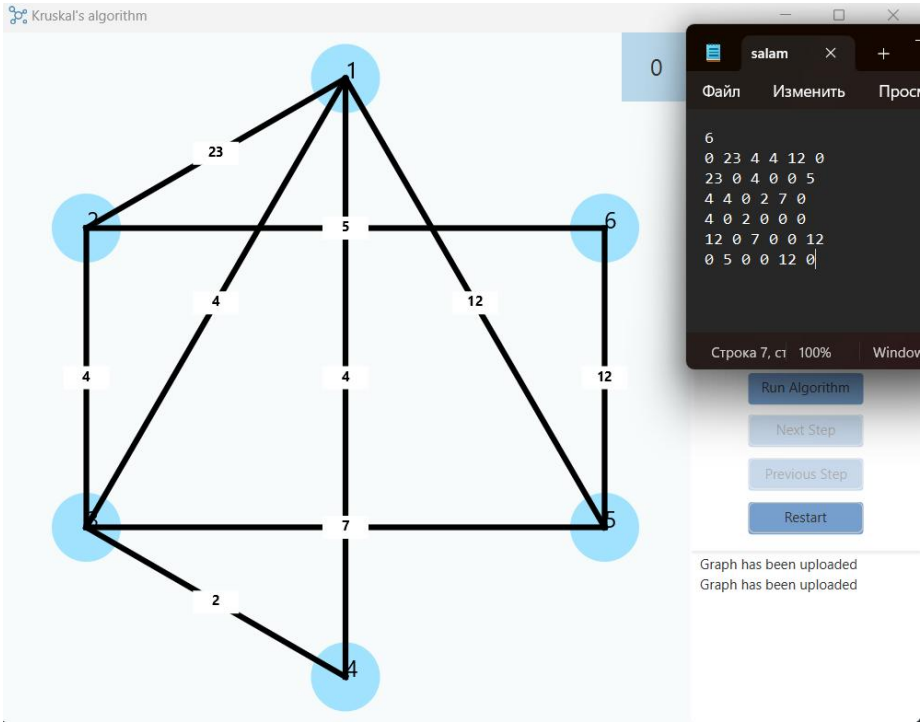

Рисунок 18 – Успешная загрузка графа из корректного файла

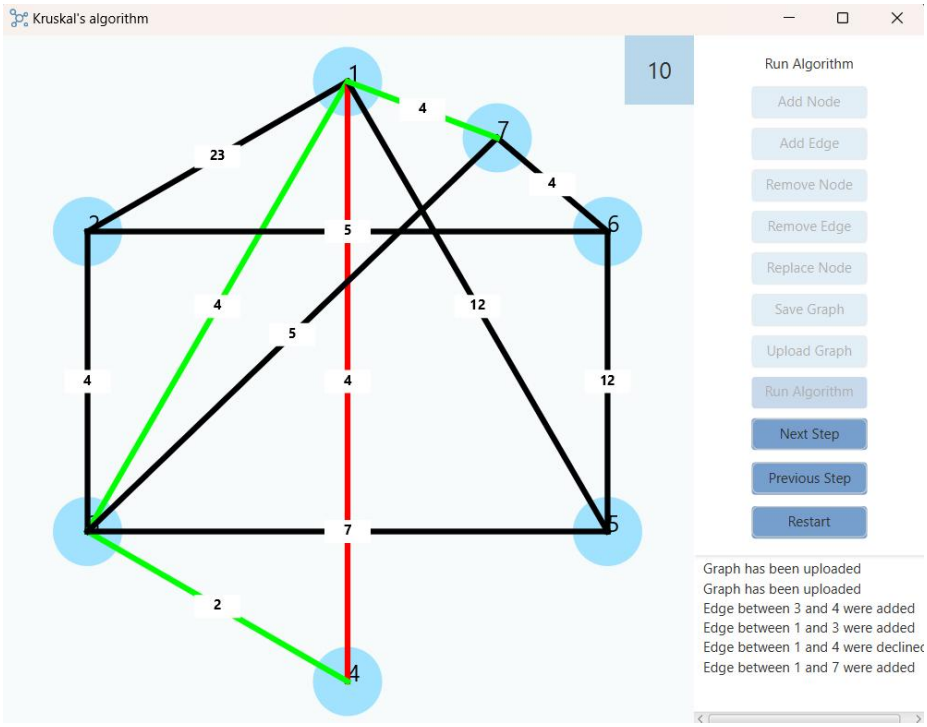

Рисунок 19 – Запуск алгоритма и прохождение нескольких шагов

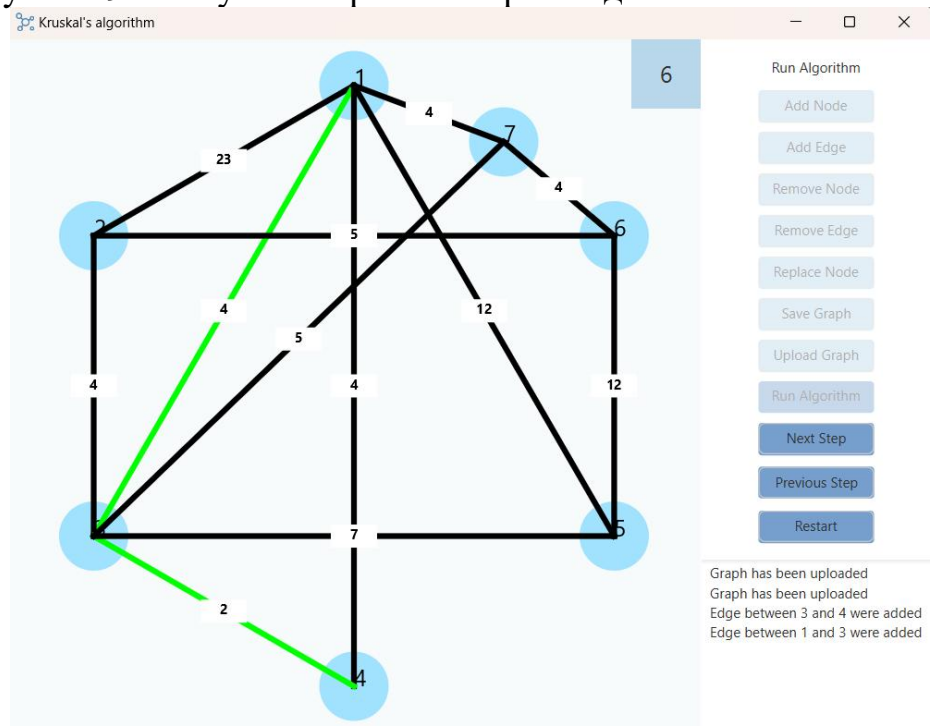

Рисунок 20 – Возвращение на 2 шага назад

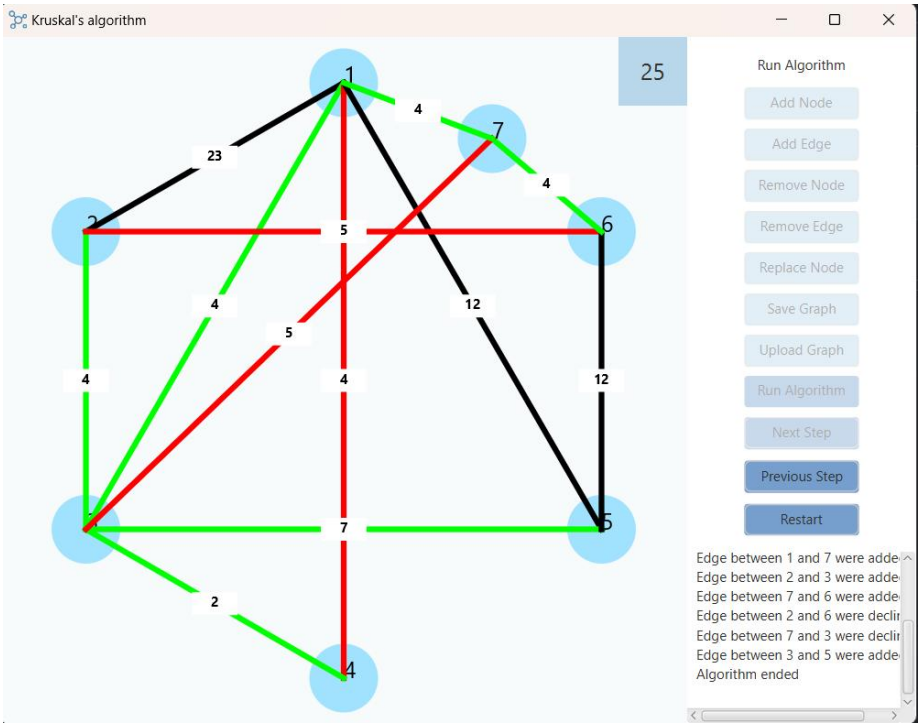

Рисунок 21 – Полностью отработавший алгоритм