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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритма ЯПД поиска минимального остовного дерева

Студент гр. 7383 _____ Власов Р.А.

Студент гр. 7383 _____ Сычевский Р.А.

Студентка гр. 7383 _____ Иолшина В.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2019

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Власов Р.А. группы 7383

Студент Сычевский Р.А. группы 7383

Студентка Иолшина В. группы 7383

Тема практики: Визуализация алгоритма ЯПД поиска минимального остовного дерева

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: алгоритм ЯПД поиска минимального остовного дерева.

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчета: 12.07.2019

Дата защиты отчета: 12.07.2019

Студент гр. 7383	_____	Власов Р.А.
Студент гр. 7383	_____	Сычевский Р.А.
Студентка гр. 7383	_____	Иолшина В.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

В результате выполнения данной работы была реализована программа, выполняющая пошаговую визуализацию алгоритма Ярника-Прима-Дейкстры поиска минимального остовного дерева в графе. Программа была разработана на языке Java в среде разработки IntelliJ IDEA. Получая на вход неориентированный граф в текстовом виде или графически, программа подробно показывает процесс построения минимального остовного дерева в этом графе. При этом пользователь может видеть текущее и предыдущее состояния программы, а рассмотренные и обрабатываемые на данном шаге вершины и ребра окрашиваются в разные цвета.

SUMMARY

As a result of this work, a program performing step-by-step visualization of the Dijkstra-Jarnik-Prim algorithm for searching the minimum spanning tree in a graph was implemented. The program has been developed in the Java Language in the IntelliJ IDEA development environment. Receiving as input an undirected graph in text form or graphically, the program shows the process of building the minimum spanning tree in this graph in detail. Specifically, the user can see current and previous program states. And moreover, vertices and edges, which are being processed at this step or have been examined already, are painted in different colors.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью выполнения данной работы является изучение основ объектно-ориентированного программирования на языке Java и их применение для визуализации алгоритмов на графах.

Задачей данной работы является создание графического интерфейса для демонстрации работы алгоритма Ярника-Прима-Дейкстры поиска минимального остовного дерева в графе.

Алгоритм Ярника-Прима-Дейкстры (англ. the Dijkstra-Jarnik-Prim algorithm) — алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость. Алгоритм состоит из следующих шагов:

1. Сначала берется произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево.

2. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих ребер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву.

3. Шаг 2 повторяется 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

На вход программе подается неориентированный граф, который задается в текстовом виде (список ребер и их весов) или графически. На выходе пользователь получает визуализацию минимального остовного дерева на главном экране с возможностью записи результата в файл.

Описание интерфейса:

Главное окно программы (рис. 1) представляет собой окно, содержащее 3 кнопки для ввода графа: ввод из файла (открывается диалоговое окно), ввод с клавиатуры (открывается окно с полем для ввода текста) и кнопку графического ввода (открывается окно графического ввода графа). При неправильном формате ввода программа показывает сообщение с ошибкой, пользователь может ввести граф еще раз. Также окно содержит кнопку сохранения графа в файл и кнопку запуска алгоритма, по нажатию на которую открывается окно, отображающее состояние программы, главное окно скрывается. Также главное окно содержит два поля, отображающих введенный граф и полученный в результате работы алгоритма граф.

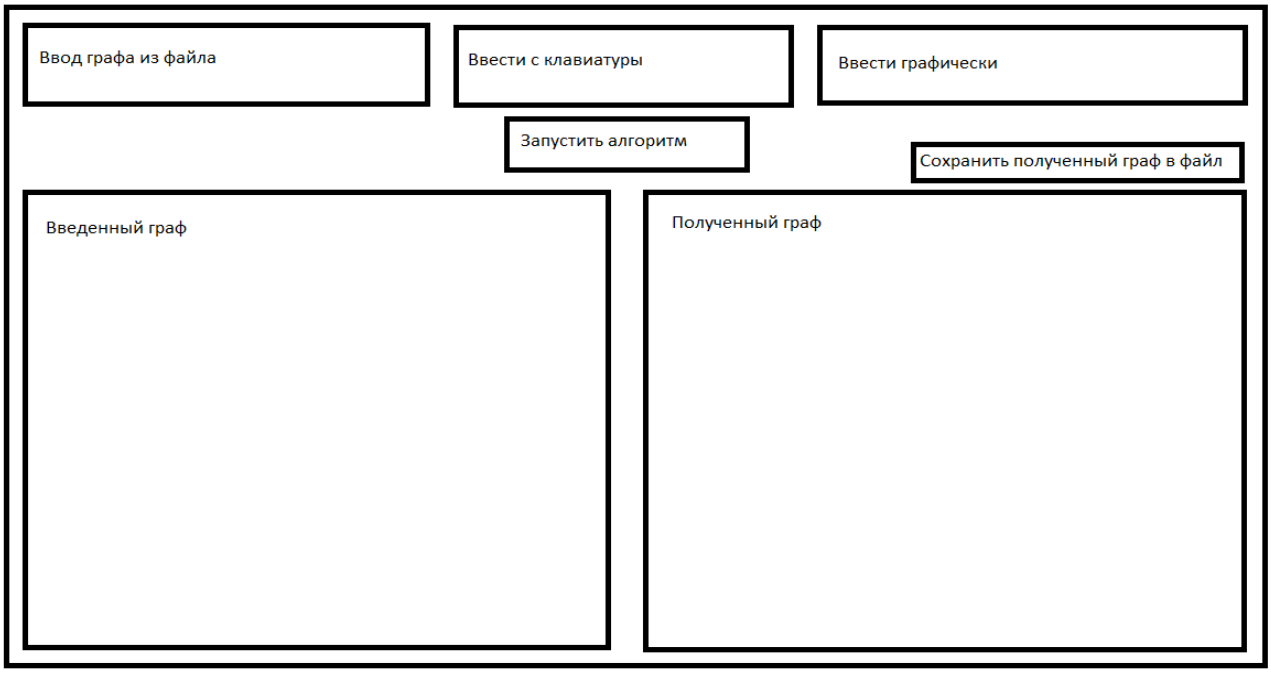


Рисунок 1 - Главное окно программы

Окно, отображающее состояние программы (рис. 2), содержит два поля, в которых отображается состояние алгоритма до и после выполнения текущего шага. В нижней части поля вывода предыдущего состояния расположено поле для вывода комментариев по работе алгоритма. Ребра и вершины, добавленные в дерево, окрашены зеленым цветом, ребра и вершины, которые рассматриваются на данном шаге, окрашены синим цветом, остальные ребра и вершины окрашены черным цветом. В нижней части окна содержатся кнопки “Шаг вперед” и “Шаг назад”, по нажатию на которые программа переходит к следующему шагу алгоритма или возвращается на шаг назад. Также внизу содержится кнопка “Прервать алгоритм”, по нажатию на которую закрывается окно с состоянием программы и отображается главное окно, при этом поле с результатом работы остается пустым. Также внизу расположен таймер, по истечении времени которого автоматически выполняется следующий шаг алгоритма. По умолчанию таймер выставлен на 10 секунд, время можно изменить или остановить таймер. При этом кнопка остановки таймера меняется на кнопку возобновления таймера.

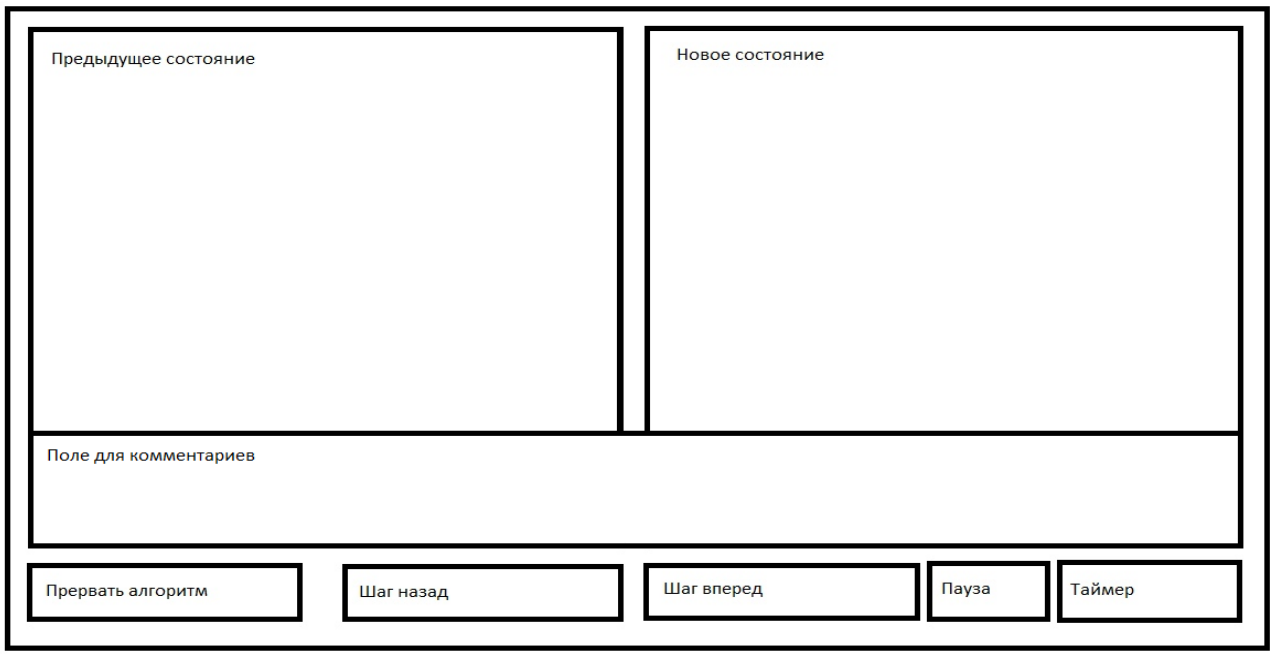


Рисунок 2 - Окно, отображающее состояние программы

Окно графического ввода (рис. 3) содержит рабочую область, пару переключателей (радиокнопок) режима: “Редактирование вершин” и

“Редактирование ребер”, а также кнопки применить и отменить. В рабочей области отображается граф. При активированном режиме редактирования вершин: клик в пустую область добавляет новую вершину, клик по вершине - удаляет вершину и инцидентные ребра. При активированном режиме редактирования ребер: поочередные клики на 2 вершины добавляет между ними ребро, если такого ребра еще нет, или удаляет ребро, если оно существует.

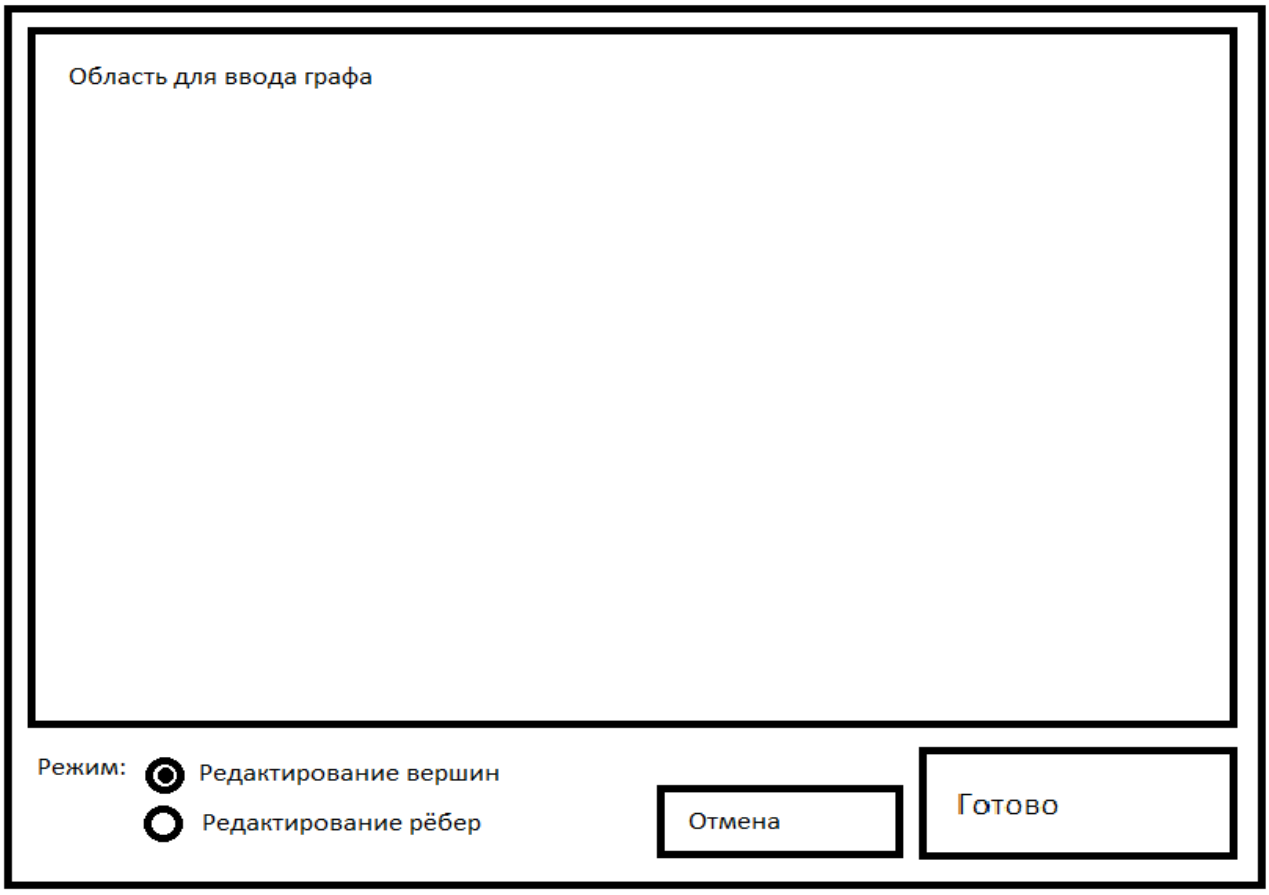


Рисунок 3 - Окно графического ввода графа

1.2. Уточнение требований после первой версии программы

В ходе разработки было принято решение заменить кнопки ввода графа с помощью клавиатуры и графического интерфейса на кнопки редактирование графа с помощью клавиатуры и графического интерфейса. Было принято добавить кнопку для сохранения введенного графа в файл. В ходе выполнения алгоритма, ребра, которые рассматривались на предыдущем шаге, но не были

выбраны, будут окрашиваться в голубой цвет, ребра, не попавшие в минимальное остовное дерево будут окрашены в белый цвет.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Спецификация:

Определены требования к программе.

Дата сдачи – 03.07.2019

Прототип:

Реализованы все окна программы. Все кнопки интерфейса реализуют только открытие или закрытие соответствующих им окон, остальные кнопки отключены.

Дата сдачи – 05.07.2019

Итерация 1:

Реализован ввод графа с клавиатуры или из файла и сохранение графа в файл. Реализован пошаговый проход алгоритма с комментариями в соответствующей области без возможности отмены действий.

Дата сдачи – 08.07.2019

Итерация 2:

Реализовано отображение графа в соответствующих полях на главном окне и на окне, отображающем текущее состояние. Добавлен ввод графа с помощью графического интерфейса.

Дата сдачи – 10.07.2019

Итерация 3:

Добавлен автоматический переход к следующему шагу алгоритма по таймеру, реализовано изменение времени таймера и его отключение. Реализована функция возврата к предыдущему шагу алгоритма (до 32 шагов).

Дата сдачи – 12.07.2019

2.2. Распределение ролей в бригаде

Сычевский Радимир – реализация связи между алгоритмом и интерфейсом, тестирование графического интерфейса

Иолшина Валерия – реализация графического интерфейса, написание отчета

Власов Роман – реализация алгоритма и визуализации графа, тестирование алгоритма

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Использованные структуры данных

При создании графического интерфейса были использованы возможности библиотек `Swing` и `AWT`, в том числе наследование от класса JFrame, представляющего собой окно с рамкой и строкой заголовка, и использование интерфейсов `ActionListener`, `ComponentListener`, `MouseListener` для обработки событий. Создание главного окна, окна состояния графа, окна графического ввода графа и окна текстового ввода графа, а также их обработчики событий описаны в файлах MainWindow.java, StateWindow.java, GraphGUI_Input.java, GraphText_Input.java соответственно.

Связь алгоритма и графического интерфейса осуществляется при помощи интерфейса `AlgorithmControl`, предоставляющего доступ к методам управления выполнением алгоритма.

Основной класс, отвечающий за выполнение алгоритма и реализующий интерфейс `AlgorithmControl`, - класс `DJPAlgorithm`. Для хранения графа в алгоритме используется класс `Graph`.

Класс `Graph` содержит список ребер на основе массива, каждое из которых является объектом класа `Edge`, а также список ребер, попавших в минимальное остовное дерево.

Класс `Edge` содержит две вершины, являющуюся объектами класса `Vertex`, вес ребра и его цвет.

Класс `Vertex` содержит строку с именем вершины и её цвет.

3.2. Основные методы

algorithm.DJPAlgorithm:

Методы `public void init (Graph graph)` и `public void init (Graph graph, String startVertexName)` инициализируют объект алгоритма графом и именем вершины, с которой необходимо начать выполнение алгоритма.

Метод `public void makeStep()` совершает шаг алгоритма.

Метод `public boolean isFinished()` проверяет, завершил ли алгоритм свою работу.

Метод `public String getComment()` возвращает строку с комментарием по последнему шагу алгоритма.

Метод `public Graph getCurrentGraphState()` возвращает текущее состояние графа.

Метод `public boolean canBeUndone()` проверяет, имеются ли действия, которые можно отменить.

Метод `public mxGraphComponent undo()` отменяет предыдущий шаг алгоритма.

Метод `private ArrayList<Edge> getAllIncedentEdges()` возвращает список всех ребер, которые должны быть рассмотрены.

Метод `private boolean isVisited(Vertex v)` проверяет, добавлена ли вершина в минимальное остовное дерево на данный момент.

Классы `class actStep0` и `class actStep1` реализуют функциональный интерфейс `CanBeUndone` с единственным методом `mxGraphComponent undo()`, отвечающим за отмену сохраненного в объекте действия и возврат компоненты для отображения предыдущего состояния графа.

algorithm.Graph:

Методы `public void addEdge(Edge e)` и `public void addEdge(String name1, String name2, cost)` отвечают за добавление ребра в граф.

Метод `public void removeEdge(String name1, String name2)` из графа ребро. Строки `name1` и `name2` содержат имена вершин на концах ребра, подлежащего удалению.

Метод `int countFinalCost()` вычисляет вес минимального остовного дерева.

Метод `public ArrayList<Edge> getIncedentEdges(Vertex vertex)` возвращает список ребер, инцидентных с указанной вершиной.

Методы `public mxGraphComponent createGraphComponent()` и `public mxGraphComponent updateGraphComponent()` отвечают за создание и обновление компоненты для отображения графа.

Метод `public static Boolean isValid(String str)` проверяет граф, содержащийся в строке, на связность.

Метод `public static Graph getGraphFromString(String str)` создает объект класса граф по строке.

algorithm.Edge:

Метод `public void setColor(Color color)` устанавливает цвет ребра.

Метод `public Color getColor()` возвращает цвет ребра.

Метод `public int getCost()` возвращает вес ребра.

Метод `public boolean isAdjacent(Vertex v)` проверяет, является ли ребро инцидентным данной вершине.

Метод `public static Edge getCheapestEdge(ArrayList<Edge> edges)` возвращает ребро с наименьшим весом из данного списка.

algorithm.Vertex:

Метод `public void setColor(Color color)` устанавливает цвет вершины.

Метод `public Color getColor()` возвращает цвет вершины.

Метод `public String getName()` возвращает имя вершины.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса и системное тестирование

Было проведено системное тестирование программы. Все элементы графического интерфейса работают корректно. Были проверены различные сценарии использования программы, включая некорректные сценарии. Все функции работают корректно согласно спецификации. Все ошибки, выявленные в ходе тестирования, были исправлены. В приложении А приведены скриншоты, показывающие корректность работы программы.

4.2. Тестирование алгоритма

Было проведено модульное тестирование с помощью библиотеки `JUnit` для проверки корректности работы отдельных методов классов `Edge`, `Vertex` и `Graph` на корректных данных. Исходный код unit-тестов приведён в приложении Б, результаты модульного тестирования показаны на рисунке 4. Результат всех тестов положительный. Также было проведено системное тестирование программы на корректных данных. Ошибок обнаружено не было.

Рисунок 4 – Результат модульного тестирования

ЗАКЛЮЧЕНИЕ

В ходе данной учебной практики были изучены основы объектно-ориентированного программирования на языке Java, пройден интерактивный курс «Java. Базовый курс» на образовательной платформе Stepik. После чего была разработана программа с графическим интерфейсом, которая с помощью алгоритма Ярника-Прима-Дейкстры находит минимальное остовное дерево в графе, наглядно демонстрируя процесс выполнения алгоритма пользователю.

Программа соответствует требованиям спецификации, все выявленные в процессе тестирования ошибки исправлены. Поставленная задача выполнена и проект завершен успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование на языке JAVA. Лабораторный практикум. / Герасимова Т.В. СПб.: 2014.

2. Учебное пособие по программированию на языке JAVA / Герасимова Т.В. СПб.: 2006.

3. Java Базовый курс. URL: https://stepik.org/course/Java-Базовый-курс-187/syllabus

4. The Java Tutorials. URL: https://docs.oracle.com/javase/tutorial/index.html

5. mxGraph Tutorials. URL: https://jgraph.github.io/mxgraph/

6. Prim' algorithm. URL: https://en.wikipedia.org/wiki/Prim%27s_algorithm

ПРИЛОЖЕНИЕ А

СКРИНШОТЫ РАБОТЫ ПРОГРАММЫ

На рисунке 5 представлено главное окно программы

Рисунок 5

Загрузим граф из файла. Результат представлен на рисунке 6. Содержимое файла:

a b 1

a c 3

a d 5

a e 7

b c 2

b d 23

b e 1

c d 17

c e 5

d e 8

e g 13

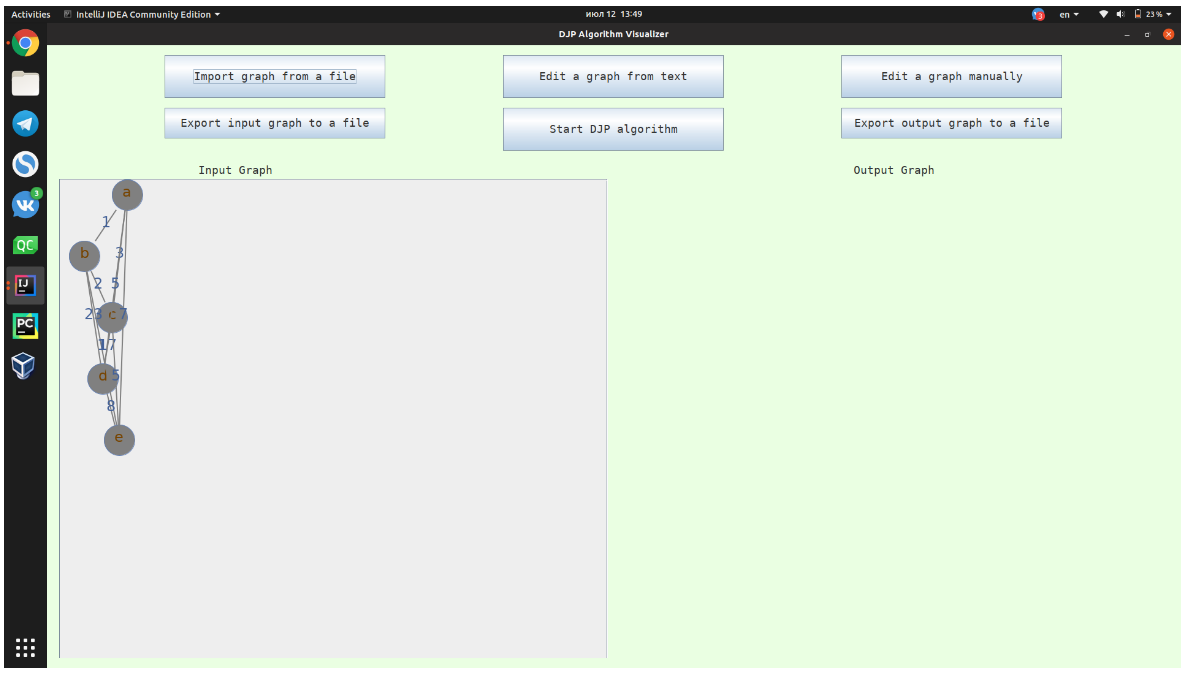


Рисунок 6

Введенный граф можно отредактировать в текстовом виде. Пример показан на рисунке 7.

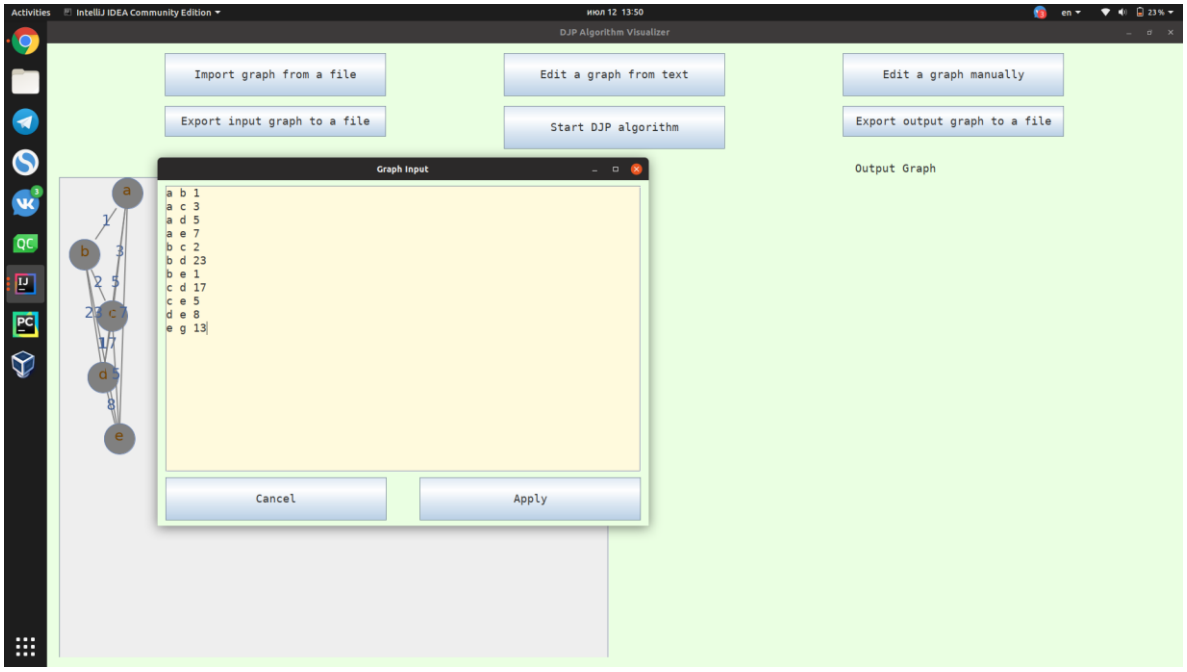


Рисунок 7

Граф можно редактировать в графическом виде. Переместим вершины, удалим одну вершину. Инцидентные ей ребра удалились автоматически. Результат показан на рисунке 8.

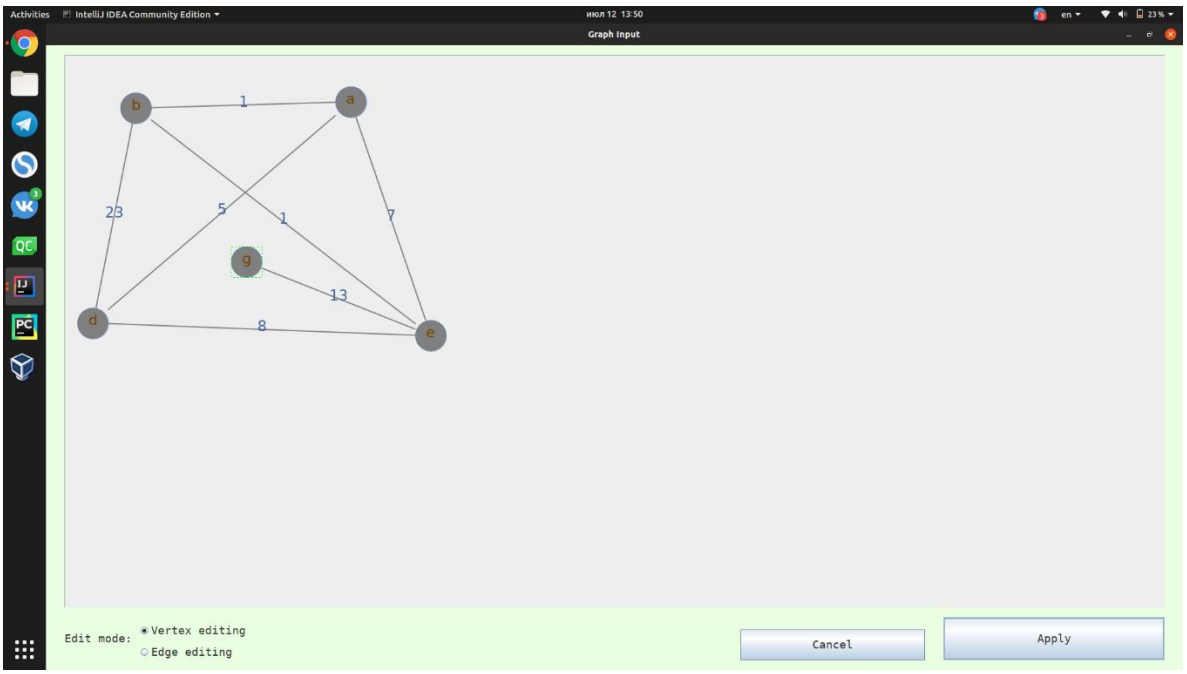


Рисунок 8

Попробуем сохранить граф с результатом работы программы в файл. Программа сообщает об ошибке, так как граф ещё не получен (рисунок 9).

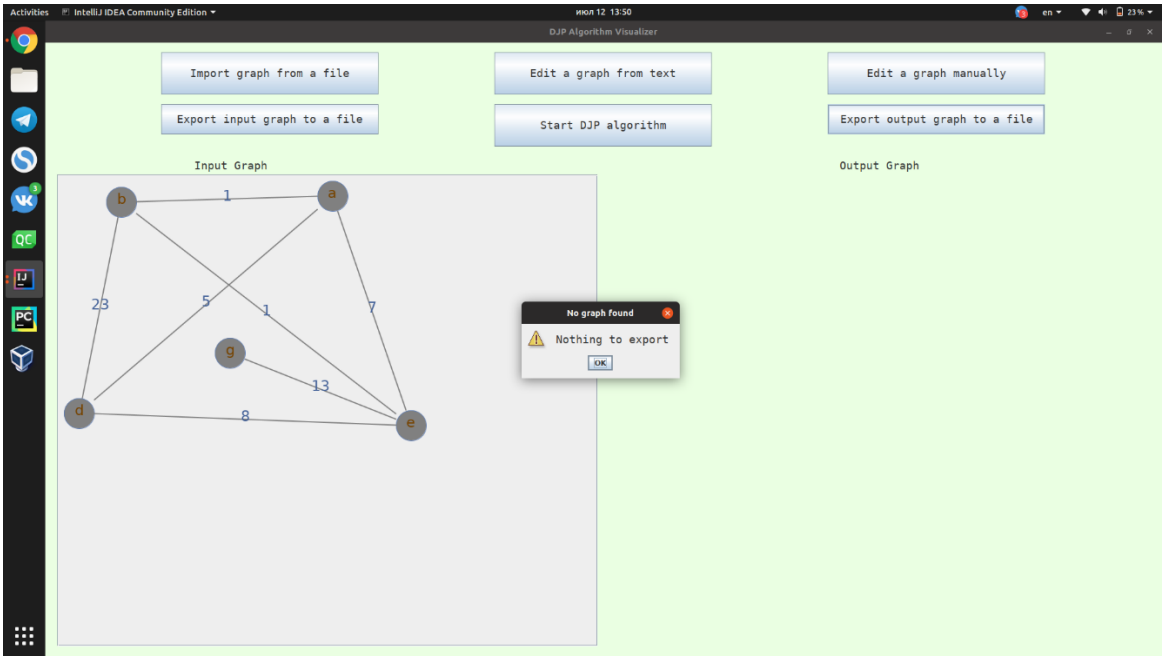


Рисунок 9

Запустим алгоритм и выберем стартовую вершину. Алгоритм корректно начинает работу с выбраной вершины (рисунки 10 и 11).

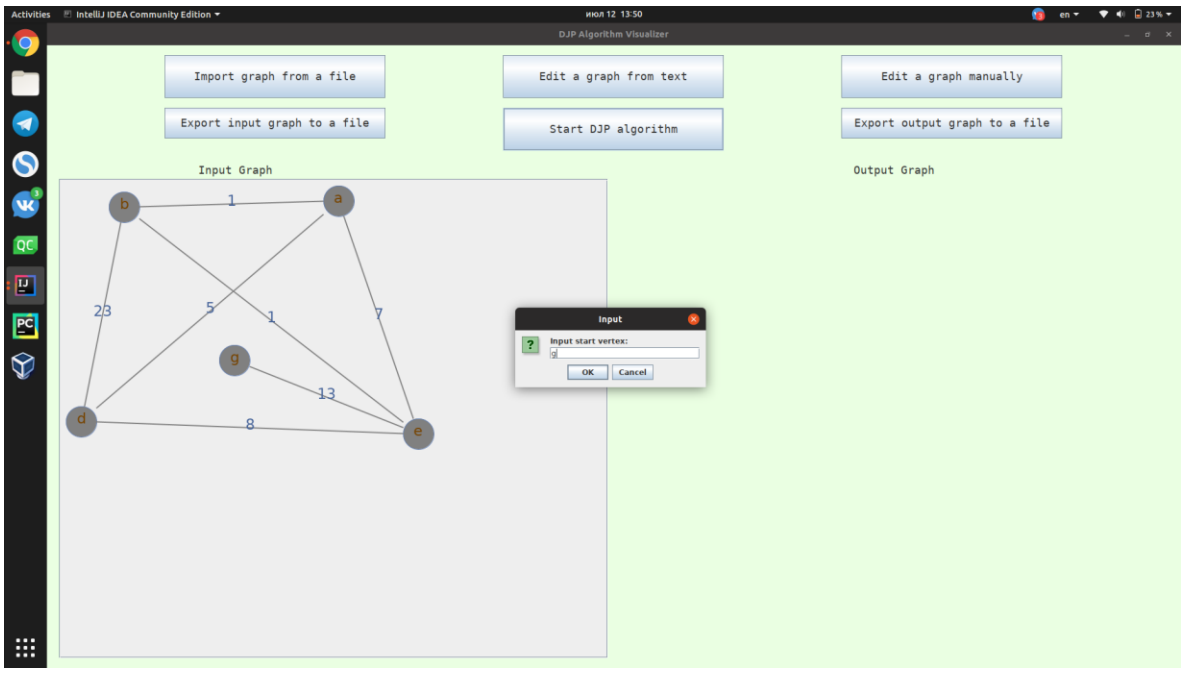


Рисунок 10

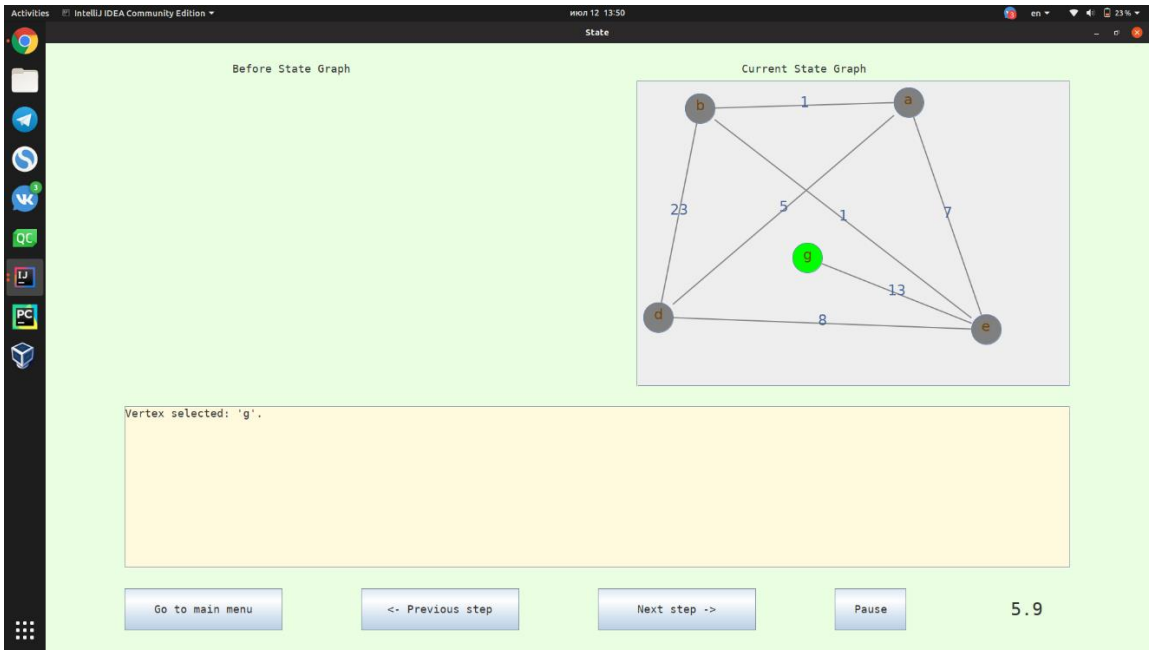


Рисунок 11

На рисунке 12 приведен один из этапов работы программы. Все вершины и ребра окрашены в корректные цвета.

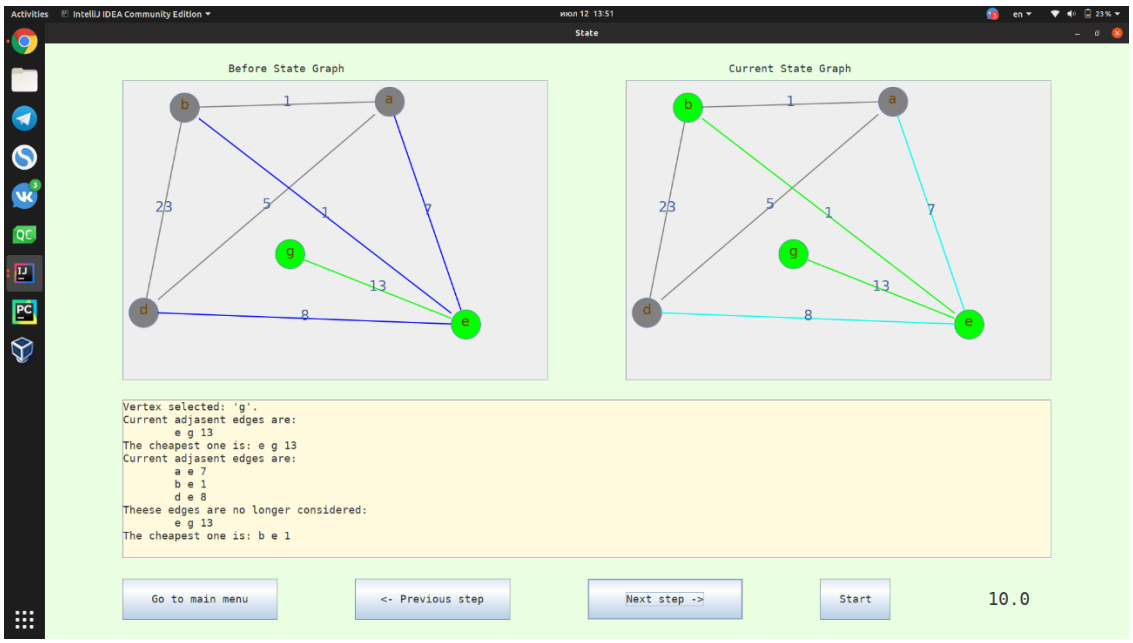
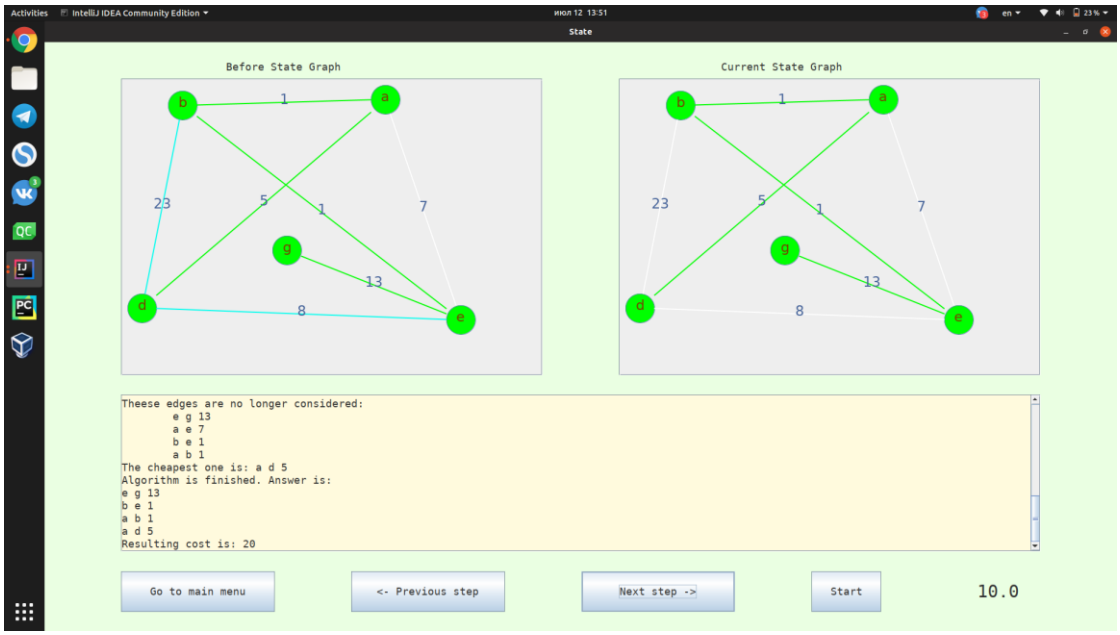


Рисунок 12

На рисунке 13 показан последний этап алгоритма. В логе программа показывает суммарный вес ребер, попавших в минимальное остовое дерево. Действительно, найденное остовное дерево имеет минимальный вес.



Рисуок 13

Алгоритм завершается корректно, на главном окне программы отображается результирующий граф (рисунок 14).

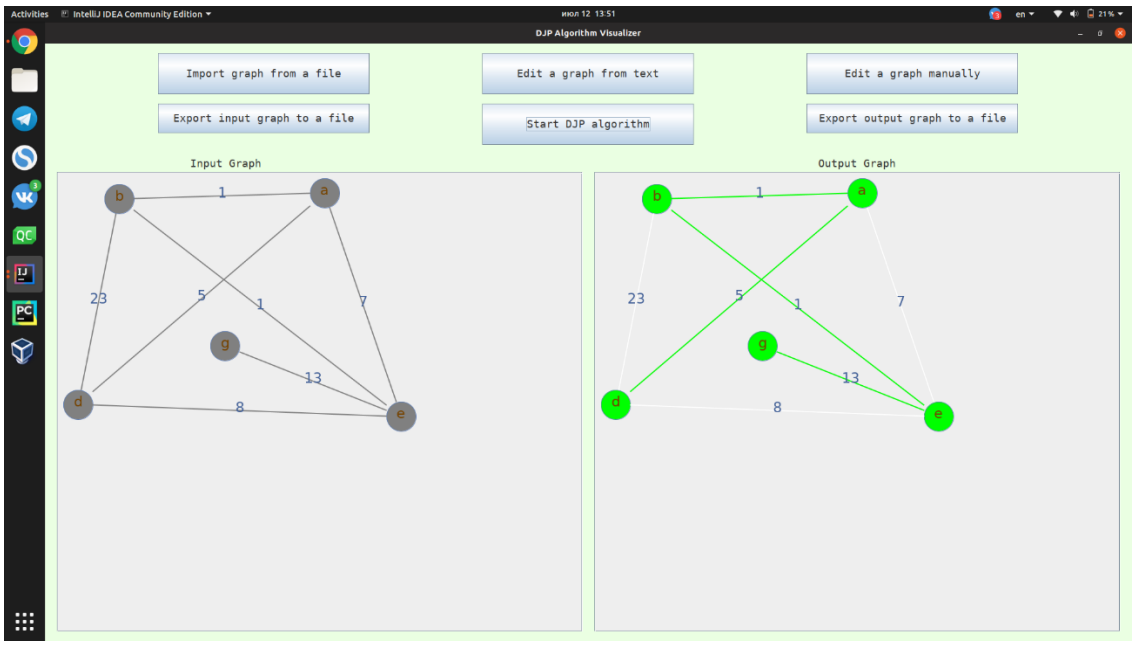


Рисунок 14

Сохраним граф в файл. Результат сохранения:

e g 13
b e 1
a b 1
d a 5

ПРИЛОЖЕНИЕ Б

ТЕСТОВЫЕ СЛУЧАИ

EdgeTest.java

```
import algorithm.Edge;
import algorithm.Vertex;
import org.junit.Assert;
import org.junit.Test;

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

import static org.junit.Assert.*;

public class EdgeTest {

    @Test
    public void isAdjacent() {
        String nameV1 = "Name 1";
        String nameV2 = "Name 2";
        Edge edge = new Edge(nameV1, nameV2, 13);
        Assert.assertEquals(true, edge.isAdjacent(new Vertex(nameV1)));
        Assert.assertEquals(true, edge.isAdjacent(new Vertex(nameV2)));
        Assert.assertEquals(false, edge.isAdjacent(new Vertex("Name
3")));

    }

    @Test
    public void getCost() {
        int cost = 100;
        Edge edge = new Edge("N1", "N2", cost);
```

```
    Assert.assertEquals(cost, edge.getCost());
}

@Test
public void getColor() {
    Edge edge = new Edge("N1", "N2", 100);
    edge.setColor(Color.BLACK);
    Assert.assertEquals(Color.BLACK, edge.getColor());
    edge.setColor(Color.WHITE);
    Assert.assertEquals(Color.WHITE, edge.getColor());
}

@Test
public void getCheapestEdge() {
    Edge e1 = new Edge("N11", "N12", 30);
    Edge e2 = new Edge("N21", "N22", 100);
    Edge e3 = new Edge("N31", "N32", 10);
    ArrayList<Edge> list = new ArrayList<>();
    list.add(e1);
    list.add(e2);
    list.add(e3);
    Assert.assertEquals(e3, Edge.getCheapestEdge(list));
}

@Test
public void getListAsString() {
    Edge e1 = new Edge("N11", "N12", 30);
    Edge e2 = new Edge("N21", "N22", 100);
    Edge e3 = new Edge("N31", "N32", 10);
    ArrayList<Edge> list = new ArrayList<>();
    list.add(e1);
    list.add(e2);
    list.add(e3);
    String expected = "\tN11 N12 30\n" +
```

```
                "\tN21 N22 100\n" +
                "\tN31 N32 10\n";
        Assert.assertEquals(expected, Edge.getListAsString(list));
    }
}
```

VertexTest.java

```
import algorithm.Vertex;
import org.junit.Assert;
import org.junit.Test;

import java.awt.*;

import static org.junit.Assert.*;

public class VertexTest {

    @Test
    public void getColor() {
        Vertex v = new Vertex("Test");
        v.setColor(Color.BLACK);
        Assert.assertEquals(Color.BLACK, v.getColor());
        v.setColor(Color.WHITE);
        Assert.assertEquals(Color.WHITE, v.getColor());
    }

    @Test
    public void getName() {
        String name = "Test name";
        Vertex v = new Vertex(name);
        Assert.assertEquals(name, v.getName());
    }

    @Test
```

```
    public void equals1() {
        String name1 = "Name 1";
        String name2 = "Name 2";
        Vertex v1 = new Vertex(name1);
        Vertex v2 = new Vertex(name2);
        Assert.assertEquals(false, v1.equals(v2));
        Assert.assertEquals(true, v1.equals(new Vertex(name1)));
    }
}
```

GraphTest.java

```
import algorithm.DJPAlgorithm;
import algorithm.Graph;
import org.junit.Assert;
import org.junit.Test;

import static org.junit.Assert.*;

public class GraphTest {

    @Test
    public void countFinalCost() {
        String graph = "1 6 5\n" +
                "2 6 20\n" +
                "11 10 3\n" +
                "8 11 7\n" +
                "9 12 4\n" +
                "11 12 14\n" +
                "8 9 5\n" +
                "8 7 31\n" +
                "7 10 22\n" +
                "10 8 6\n" +
                "4 8 11\n" +
```

```
            "4 7 13\n" +
            "4 3 2\n" +
            "7 3 5\n" +
            "1 3 9\n" +
            "4 5 2\n" +
            "5 6 1\n" +
            "2 5 9\n" +
            "4 6 10\n" +
            "4 9 4\n" +
            "9 11 15\n" +
            "5 9 17\n" +
            "1 4 12";
    Graph g = Graph.getGraphFromString(graph);
    g.createGraphComponent();
    Assert.assertEquals(0, g.countFinalCost());
    DJPAlgorithm alg = new DJPAlgorithm();
    alg.init(g);
    while (!alg.isFinished())
    {
        alg.makeStep();
    }
    Assert.assertEquals(46, g.countFinalCost());
    alg = new DJPAlgorithm();
    alg.init(g);
    while (!alg.isFinished())
    {
        alg.makeStep();
    }
    Assert.assertEquals(46, g.countFinalCost());

}

@Test
public void isValid() {
```

```
    String testStr = "1 6 5\n" +
            "2 6 20\n" +
            "11 10 3\n" +
            "8 11 7\n" +
            "9 12 4\n" +
            "11 12 14\n" +
            "8 9 5\n" +
            "8 7 31\n" +
            "7 10 22\n" +
            "10 8 6\n" +
            "4 8 11\n" +
            "4 7 13\n" +
            "4 3 2\n" +
            "7 3 5\n" +
            "1 3 9\n" +
            "4 5 2\n" +
            "5 6 1\n" +
            "2 5 9\n" +
            "4 6 10\n" +
            "4 9 4\n" +
            "9 11 15\n" +
            "5 9 17\n" +
            "1 4 12";
    Assert.assertEquals(true, Graph.isValid(testStr));
    testStr = "1 2 3\n" +
            "2 3 4\n" +
            "1 4 5\n" +
            "1 7 8\n" +
            "9 10 3\n" +
            "1 3 4";
    Assert.assertEquals(false, Graph.isValid(testStr));
}

@Test
```

```
public void toString1() {
    String testStr = "1 6 5\n" +
            "2 6 20\n" +
            "11 10 3\n" +
            "8 11 7\n" +
            "9 12 4\n" +
            "11 12 14\n" +
            "8 9 5\n" +
            "8 7 31\n" +
            "7 10 22\n" +
            "10 8 6\n" +
            "4 8 11\n" +
            "4 7 13\n" +
            "4 3 2\n" +
            "7 3 5\n" +
            "1 3 9\n" +
            "4 5 2\n" +
            "5 6 1\n" +
            "2 5 9\n" +
            "4 6 10\n" +
            "4 9 4\n" +
            "9 11 15\n" +
            "5 9 17\n" +
            "1 4 12";
    Graph g = Graph.getGraphFromString(testStr);
    Assert.assertEquals(testStr + "\n", g.toString());
    testStr = "1 2 3\n" +
            "2 3 4\n" +
            "1 4 5\n" +
            "1 7 8\n" +
            "9 10 3\n" +
            "1 3 4";
    g = Graph.getGraphFromString(testStr);
    Assert.assertEquals(testStr + "\n", g.toString());
```

```
        }
    }
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД

AlgorithmControl.java

```
package algorithm;

import com.mxgraph.swing.mxGraphComponent;

public interface AlgorithmControl {
   void init(Graph graph);
   void init(Graph graph, String startVertexName);
   Graph getCurrentGraphState();
   String getComment();
   void makeStep();
   boolean isFinished();
   mxGraphComponent undo();
   boolean canBeUndone();
}
```

Vertex.java

```
package algorithm;

import java.awt.Color;

public class Vertex {
   private final String name;
   private Color color = Color.GRAY;

   public Vertex(String value) {
       this.name = value;
   }

   public void setColor(Color color) {
       this.color = color;
```

```
    }

    public Color getColor() {
        return color;
    }

    public String getName()
    {
        return name;
    }

    public boolean equals(Object o)
    {
        if (o == this)
            return true;
        if (!(o instanceof Vertex))
            return false;
        Vertex v = (Vertex)o;
        return this.name.equals(v.name);
    }
}
```

Edge.java

```
package algorithm;

import java.awt.Color;
import java.util.ArrayList;

public class Edge {
    private Vertex v1;
    private Vertex v2;
    private int cost;
    private Color color = Color.GRAY;
```

```
public Edge(String name1, String name2, int cost) {
    v1 = new Vertex(name1);
    v2 = new Vertex(name2);
    this.cost = cost;
}

public boolean isAdjacent(Vertex v)
{
    if (v.equals(v1) || v.equals(v2))
        return true;
    return false;
}

public Vertex[] getAdjasent()
{
    return new Vertex[] {v1, v2};
}

public int getCost()
{
    return cost;
}

public void setColor(Color color) {
    this.color = color;
}

public Color getColor() {
    return color;
}

public static Edge getCheapestEdge(ArrayList<Edge> edges)
{
    Edge cheapest = edges.get(0);
```

```
        int min = cheapest.getCost();
        for (Edge e : edges)
        {
            if (e.getCost() < min)
            {
                min = e.getCost();
                cheapest = e;
            }
        }
        return cheapest;
    }


    public String toString()
    {
        return v1.getName() + " " + v2.getName() + " " + cost +
System.lineSeparator();
    }

    public static String getListAsString(ArrayList<Edge> edges)
    {
        StringBuilder str = new StringBuilder();
        for (Edge e : edges)
        {
            str.append("\t" + e.toString());
        }
        return str.toString();
    }
}
```

Graph.java

```
package algorithm;

import java.awt.*;
```

```
import java.util.ArrayList;
import java.util.Iterator;

import com.mxgraph.model.mxCell;
import com.mxgraph.model.mxGeometry;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.view.mxGraph;
import com.mxgraph.layout.hierarchical.*;


public class Graph {
   private mxGraph graph;


   private ArrayList<Edge> edgeList;
   private ArrayList<Edge> answerEdgesList;

   public Graph() {
       edgeList = new ArrayList<Edge>();
       answerEdgesList = new ArrayList<Edge>();
   }

   public Graph(ArrayList<Edge> edgeList) {
       this.edgeList = edgeList;
       answerEdgesList = new ArrayList<Edge>();
   }

   public Graph(ArrayList<Edge> edgeList, mxGraph graph)
   {
       this.edgeList = edgeList;
       this.graph = graph;
       answerEdgesList = new ArrayList<Edge>();
   }
```

```
    public void addEdge(Edge e)
    {
        edgeList.add(e);
    }

    public void addEdge(String name1, String name2, int cost)
    {
        edgeList.add(new Edge(name1, name2, cost));
    }

    public void resetToStart()
    {
        answerEdgesList = new ArrayList<>();
    }

    public int countFinalCost()
    {
        int cost = 0;
        for (Edge e : answerEdgesList)
        {
            cost += e.getCost();
        }
        return cost;
    }

    public void removeEdge(String name1, String name2)
    {
        for (Iterator<Edge> it = edgeList.iterator(); it.hasNext();)
        {
            Edge e = it.next();
            Vertex[] v = e.getAdjasent();
            if (name1.equals(v[0].getName()) ||
name1.equals(v[1].getName()))
```

```
                if (name2.equals(v[0].getName()) ||
name2.equals(v[1].getName()))
                    it.remove();
        }
    }

    public void removeVertex(String name)
    {
        for(Iterator<Edge> it = edgeList.iterator(); it.hasNext();)
        {
            Edge e = it.next();
            Vertex[] v = e.getAdjasent();
            if (name.equals(v[0].getName()) ||
name.equals(v[1].getName()))
                it.remove();
        }
    }

    public void addToAnswer(Edge e)
    {
        Vertex[] v = e.getAdjasent();
        for (Edge ed : edgeList)
        {
            Vertex[] x = ed.getAdjasent();
            if (x[0].equals(v[0]) || x[0].equals(v[1]))
                x[0].setColor(Color.GREEN);
            if (x[1].equals(v[0]) || x[1].equals(v[1]))
                x[1].setColor(Color.GREEN);
        }
        answerEdgesList.add(e);
    }

    public void removeFromAnswer(Edge e)
    {
```

```
        answerEdgesList.remove(e);
    }

    public void paintVisited(ArrayList<Vertex> visited)
    {
        for (Edge e : edgeList)
        {
            Vertex[] v = e.getAdjasent();
            if (visited.contains(v[0]) && visited.contains(v[1]))
            {
                if (e.getColor() != Color.GREEN)
                    e.setColor(Color.WHITE);
            }
            else {
                if (!visited.contains(v[0]))
                    v[0].setColor(Color.GRAY);
                if (!visited.contains(v[1]))
                    v[1].setColor(Color.GRAY);
            }
        }
    }

    public Edge getFirstEdge()
    {
        return edgeList.get(0);
    }

    public ArrayList<Edge> getIncedentEdges(Vertex vertex) //Возвращает
список ребер, инцедентных с вершиной
    {
        ArrayList<Edge> incedentEdges = new ArrayList<>();
        for (Edge e : edgeList)
        {
            if (e.isAdjacent(vertex))
```

```
                incedentEdges.add(e);
            }
            return incedentEdges;
        }

        private ArrayList<Vertex> getVertexList()
        {
            ArrayList<Vertex> list= new ArrayList<>();
            for (Edge e : edgeList)
            {
                Vertex[] v = e.getAdjasent();
                if (!list.contains(v[0])) {
                    list.add(v[0]);
                }
                if (!list.contains(v[1]))
                {
                    list.add(v[1]);
                }
            }

            return list;
        }

        public mxGraphComponent createGraphComponent() //Создает компоненту
отображения графа
        {
            graph = new mxGraph();
            Object grParent = graph.getDefaultParent();
            graph.getModel().beginUpdate();
            try {
                ArrayList<Vertex> vertex = getVertexList();
                Object[] vertexObj = new Object[vertex.size()];
                for (int i = 0; i < vertex.size(); i++) {
                    Color c = vertex.get(i).getColor();
```

```
                String styleVertex = "shape=ellipse;fontSize=24;" +
                        String.format("fillColor=#%02x%02x%02x",
c.getRed(), c.getGreen(), c.getBlue());
                vertexObj[i] = graph.insertVertex(grParent, null,
vertex.get(i).getName(), 50, 100, 50, 50,
                        styleVertex);
            }

            for (Edge e : edgeList) {
                Color c = e.getColor();
                String costStr = String.valueOf(e.getCost());
                String styleEdge =
"align=center;strokeWidth=2;startArrow=none;endArrow=none;fontSize=24;" +
                        String.format("strokeColor=#%02x%02x%02x",
c.getRed(), c.getGreen(), c.getBlue());

                Vertex[] adjV = e.getAdjasent();

                graph.insertEdge(grParent, null, costStr,
vertexObj[vertex.indexOf(adjV[0])],
                        vertexObj[vertex.indexOf(adjV[1])],
                        styleEdge);
            }

        } finally {
            graph.getModel().endUpdate();
        }
        var layout = new mxHierarchicalLayout(graph);
        layout.execute(grParent);

        graph.setAllowDanglingEdges(false);
        graph.setCellsEditable(false);
        graph.setCellsDisconnectable(false);
        graph.setConnectableEdges(false);
```

```
        graph.setVertexLabelsMovable(false);
        graph.setEdgeLabelsMovable(false);
        graph.setResetEdgesOnMove(true);
        graph.setCellsResizable(false);

        mxGraphComponent graphComponent = new mxGraphComponent(graph);
        graphComponent.setVisible(true);
        graphComponent.setConnectable(false);
        return graphComponent;
    }

    public mxGraphComponent updateGraphComponent() //Обновляет компоненту
отображения графа
    {
        Object[] vertices =
graph.getChildVertices(graph.getDefaultParent());
        Object[] newVertices = new Object[vertices.length];
        int i = 0;
        mxGraph tmp = new mxGraph();
        Object grParent = tmp.getDefaultParent();
        ArrayList<Vertex> vertex = getVertexList();


        tmp.getModel().beginUpdate();
        try{
            for (Object x : vertices)
            {
                mxCell v = (mxCell)x;
                int ind = vertex.indexOf(new
Vertex(v.getValue().toString()));
                if (ind < 0)
                    continue;
                Color c = vertex.get(ind).getColor();
```

```
                String styleVertex = "shape=ellipse;fontSize=24;" +
                        String.format("fillColor=#%02x%02x%02x",
c.getRed(), c.getGreen(), c.getBlue());
                mxGeometry g = v.getGeometry();
                newVertices[i] = tmp.insertVertex(grParent, null,
v.getValue(), g.getX(), g.getY(), g.getWidth(), g.getHeight(),
styleVertex);
                i++;
            }

            for (Edge e : edgeList) {
                Color c = e.getColor();
                String costStr = String.valueOf(e.getCost());
                String styleEdge =
"align=center;strokeWidth=2;startArrow=none;endArrow=none;fontSize=24;" +
                        String.format("strokeColor=#%02x%02x%02x",
c.getRed(), c.getGreen(), c.getBlue());

                Vertex[] adjV = e.getAdjasent();
                Object v1 = null;
                Object v2 = null;

                for (Object o : newVertices)
                {
                    if (o == null)
                        continue;
                    mxCell v = (mxCell)o;
                    if (adjV[0].equals(new
Vertex(v.getValue().toString()))) {
                        v1 = v;
                    }
                    else if (adjV[1].equals(new
Vertex(v.getValue().toString()))) {
                        v2 = v;
```

```
                }

            }
            tmp.insertEdge(grParent, null, costStr, v1, v2,
styleEdge);
        }

    }
    finally {
        tmp.getModel().endUpdate();
    }

    tmp.setAllowDanglingEdges(false);
    tmp.setCellsEditable(false);
    tmp.setCellsDisconnectable(false);
    tmp.setConnectableEdges(false);

    tmp.setVertexLabelsMovable(false);
    tmp.setEdgeLabelsMovable(false);
    tmp.setResetEdgesOnMove(true);
    tmp.setCellsResizable(false);

    mxGraphComponent graphComponent = new mxGraphComponent(tmp);
    graphComponent.setVisible(true);
    graphComponent.setEnabled(false);
    graphComponent.setConnectable(false);
    return graphComponent;
}

public static Graph getGraphFromString(String str)
{
    Graph gr = new Graph();
    String[] substr = str.split("\n");
    for (String s : substr)
```

```
        {
            String[] elements = s.split(" ");
            gr.addEdge(elements[0], elements[1],
Integer.parseInt(elements[2]));
        }
        return gr;
    }

    public String toString()
    {
        StringBuilder str = new StringBuilder();
        for (Edge e : edgeList)
        {
            str.append(e.toString());
        }
        return str.toString();
    }

    public String answerToString()
    {
        StringBuilder str = new StringBuilder();
        for (Edge e : answerEdgesList)
        {
            str.append(e.toString());
        }
        return str.toString();
    }

    public Graph backupGraph()
    {
        ArrayList<Edge> tmp = new ArrayList<>();
        tmp.addAll(edgeList);
        return new Graph(tmp, graph);
    }
```

```
public void setMXGraph(mxGraph graph){
    this.graph = graph;
}

public void resetColors()
{
    for (Edge e : edgeList)
    {
        Vertex[] v = e.getAdjasent();
        v[0].setColor(Color.GRAY);
        v[1].setColor(Color.GRAY);
        e.setColor(Color.GRAY);
    }
}

public static boolean isValid(String str) {
    String str1 = str.replaceAll("\n ", "\n");
    String[] subStr;
    String delimeter1 = "\n";
    String delimeter2 = " ";
    subStr = str1.split(delimeter1);
    String[][] arr = new String[2][subStr.length];
    String[] tmpStr;
    for (int i = 0; i < subStr.length; i++) {
        tmpStr = subStr[i].split(delimeter2);
        arr[0][i] = tmpStr[0];
        arr[1][i] = tmpStr[1];
    }

    boolean[] checked = new boolean[subStr.length];
    checked[0] = true;
    processHelper.goInside(arr, checked, 0);
```

```
        for (int i = 0; i < subStr.length; i++) {
            if (!checked[i])
                return false;
        }
        return true;
    }
}

class processHelper {
    public static void goInside(String[][] arr, boolean[] checked, int
tmp){
        for(int i = 0; i < checked.length; i++){
            if(checked[i])
                continue;
            if(arr[0][i].equals(arr[0][tmp]) ||
arr[1][i].equals(arr[0][tmp]) ||
                    arr[0][i].equals(arr[1][tmp]) ||
arr[1][i].equals(arr[1][tmp])){
                checked[i] = true;
                goInside(arr, checked, i);
            }
        }

    }
}
```

CanBeUndone.java

```
package algorithm;

import com.mxgraph.swing.mxGraphComponent;

@FunctionalInterface
public interface CanBeUndone {
    mxGraphComponent undo();
```

```
}
```

DJPAlgorithm.java

```
package algorithm;

import com.mxgraph.swing.mxGraphComponent;

import java.awt.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Stack;

public class DJPAlgorithm implements AlgorithmControl {
   private Graph gr;
   private ArrayList<Vertex> visited;
   private byte phase; // 0 - find incedent Edges, 1 - choose and add
cheapest
   private ArrayList<Edge> incedentEdgesList;
   private boolean finished;
   private String comment;
   private String bothVisited;
   private Stack<CanBeUndone> acts;

   private mxGraphComponent beforeStep0;
   private mxGraphComponent beforeStep1;


   @Override
   public void init(Graph graph) {
       gr = graph;
       gr.resetToStart();
       acts = new Stack<>();
       visited = new ArrayList<>();
       Edge e = graph.getFirstEdge();
```

```
        Vertex[] v = e.getAdjasent();
        visited.add(v[0]);
        v[0].setColor(Color.GREEN);
        phase = 0;
        finished = false;
        beforeStep0 = gr.updateGraphComponent();
        beforeStep1 = gr.updateGraphComponent();
        incedentEdgesList = new ArrayList<>();

        comment = "Arbitrary Vertex selected: '" + v[0].getName() +
"'.\n";
    }

    @Override
    public void init(Graph graph, String startVertexName) {
        gr = graph;
        gr.resetToStart();
        acts = new Stack<>();
        visited = new ArrayList<>();
        Vertex add;
        String tmpStr = "";
        ArrayList<Edge> e = gr.getIncedentEdges(new
Vertex(startVertexName));
        if (e.size() != 0) {
            Vertex[] v = e.get(0).getAdjasent();
            if (v[0].getName().equals(startVertexName)) {
                visited.add(v[0]);
                v[0].setColor(Color.GREEN);
                add = v[0];
            }
            else {
                visited.add(v[1]);
                v[1].setColor(Color.GREEN);
                add = v[1];
```

```
        }
    }
    else
    {
        Edge m = graph.getFirstEdge();
        Vertex[] v = m.getAdjasent();
        visited.add(v[0]);
        v[0].setColor(Color.GREEN);
        add = v[0];
        tmpStr = "Arbitraty ";
    }
    phase = 0;
    finished = false;
    beforeStep0 = gr.updateGraphComponent();
    beforeStep1 = gr.updateGraphComponent();
    incedentEdgesList = new ArrayList<>();

    comment = tmpStr + "Vertex selected: '" + add.getName() + "'.\n";
}

@Override
public Graph getCurrentGraphState() {
    return gr;
}

@Override
public void makeStep() {
    gr.paintVisited(visited);
    if (phase == 0)
    {
        actStep0 action = new actStep0();
        beforeStep0 = gr.updateGraphComponent();

        if (incedentEdgesList != null){
```

```
                for (Edge e : incedentEdgesList)
                {
                    if (e.getColor() != Color.GREEN)
                        e.setColor(Color.GRAY);
                }
            }
            incedentEdgesList = getAllIncedentEdges();
            action.setNewIncedentEdges(incedentEdgesList);
            for (Edge e : incedentEdgesList)
            {
                e.setColor(Color.BLUE);
            }
            acts.push(action);
            phase = 1;
            comment = "Current adjasent edges are:\n" +
Edge.getListAsString(incedentEdgesList) +
                    (bothVisited.length() == 0 ? "" :
                      "Theese edges are no longer considered:\n" +
bothVisited);
            bothVisited = null;
            if (incedentEdgesList.size() == 0) {
                comment = "Algorithm is finished. Answer is:\n" +
gr.answerToString() + "Resulting cost is: " +
                            gr.countFinalCost();
                finished = true;
            }
        }
        else if (phase == 1)
        {
            actStep1 action = new actStep1();
            beforeStep1 = gr.updateGraphComponent();

            if (incedentEdgesList.size() == 0) {
                return;
```

```
        }

        Edge cheapest = Edge.getCheapestEdge(incedentEdgesList);
        action.setIncedentEdges(incedentEdgesList);
        action.setAddedToAnswerEdge(cheapest);
        comment = "The cheapest one is: " + cheapest.toString();
        cheapest.setColor(Color.GREEN);
        gr.addToAnswer(cheapest);
        Vertex[] v = cheapest.getAdjasent();


        for (Edge e : incedentEdgesList)
        {
            Color c = new Color(0, 255, 238); //#00FFEE
            if (e.getColor() == Color.BLUE)
                e.setColor(c);
        }
        if (!isVisited(v[0])) {
            visited.add(v[0]);
            v[0].setColor(Color.GREEN);
            action.setAddedToAnswerVertex(v[0]);

        }
        if (!isVisited(v[1])) {
            visited.add(v[1]);
            v[1].setColor(Color.GREEN);
            action.setAddedToAnswerVertex(v[1]);
        }
        acts.push(action);
        phase = 0;
    }


}
```

```
@Override
public boolean isFinished() {
    return finished;
}

@Override
public String getComment() {
    return comment;
}

@Override
public mxGraphComponent undo() {
    CanBeUndone action = acts.pop();
    mxGraphComponent comp = action.undo();
    gr.paintVisited(visited);
    return comp;
}

@Override
public boolean canBeUndone() {
    return !acts.isEmpty();
}

private boolean isVisited(Vertex v)
{
    for (Vertex x : visited)
    {
        if (v.equals(x))
            return true;
    }
    return false;
}
```

```
private ArrayList<Edge> getAllIncedentEdges()
{
    StringBuilder str = new StringBuilder();
    ArrayList<Edge> incedent = new ArrayList<>();
    for (Vertex v : visited)
    {
        incedent.addAll(gr.getIncedentEdges(v));
    }

    for(Iterator<Edge> it = incedent.iterator(); it.hasNext();)
    {
        Edge e = it.next();
        Vertex[] c = e.getAdjasent();
        if (isVisited(c[0]) && isVisited(c[1])) {
            if (e.getColor() == Color.GRAY)
                e.setColor(Color.WHITE);
            if (!str.toString().contains(e.toString()))
                str.append("\t" + e.toString());
            it.remove();
        }
    }
    bothVisited = str.toString();

    return incedent;
}

public mxGraphComponent getPrevState()
{
    if (phase == 0)
        return beforeStep1;
    else
        return beforeStep0;
}
```

```
class actStep0 implements CanBeUndone {
    private String lastComment;
    private Edge addedToAnswer = null;
    private ArrayList<Edge> incedentEdges;
    private ArrayList<Edge> newIncedentEdges;
    private mxGraphComponent before0;
    private mxGraphComponent before1;

    actStep0()
    {
        lastComment = comment;
        before0 = beforeStep0;
        before1 = beforeStep1;
        incedentEdges = incedentEdgesList;
        if (incedentEdges != null)
        {
            for (Edge e : incedentEdges)
            {
                if (e.getColor() == Color.GREEN)
                    addedToAnswer = e;
            }
        }



    }

    public void setNewIncedentEdges(ArrayList<Edge> list)
    {
        newIncedentEdges = list;
    }

    @Override
```

```
    public mxGraphComponent undo() {
        beforeStep0 = before0;
        beforeStep1 = before1;
        incedentEdgesList = newIncedentEdges;
        comment = lastComment;
        finished = false;
        for (Edge e : newIncedentEdges)
        {
            e.setColor(Color.GRAY);
        }
        if (incedentEdges != null)
            for (Edge e : incedentEdges)
            {
                if (e == addedToAnswer)
                    e.setColor(Color.GREEN);
                else
                {
                    Color c;
                    if (addedToAnswer == null)
                        c = Color.GRAY;
                    else
                        c = new Color(0, 255, 238);
                    e.setColor(c);
                }
            }



        phase = 0;
        return before1;
    }
}

class actStep1 implements CanBeUndone{
```

```
private String lastComment;
private Vertex addedToAnswerVertex;
private Edge addedToAnswerEdge;
private ArrayList<Edge> incedentEdges;
private mxGraphComponent before0;
private mxGraphComponent before1;

actStep1()
{
    lastComment = comment;
    before0 = beforeStep0;
    before1 = beforeStep1;
}

public void setAddedToAnswerVertex(Vertex e)
{
    addedToAnswerVertex = e;
}
public void setAddedToAnswerEdge(Edge e) {
    addedToAnswerEdge = e;
}
public void setIncedentEdges(ArrayList<Edge> list)
{
    incedentEdges = list;
}

@Override
public mxGraphComponent undo() {
    beforeStep0 = before0;
    beforeStep1 = before1;
    comment = lastComment;
    visited.remove(addedToAnswerVertex);
    addedToAnswerVertex.setColor(Color.GRAY);
    incedentEdgesList = incedentEdges;
```

```
            gr.removeFromAnswer(addedToAnswerEdge);
            for (Edge e : incedentEdges)
            {
                e.setColor(Color.BLUE);
            }
            phase = 1;
            return before0;
        }
    }
}
```

MainWindow.java

```
package gui;

import algorithm.DJPAlgorithm;
import algorithm.Graph;
import com.mxgraph.swing.mxGraphComponent;
import com.sun.tools.javac.Main;

import java.awt.*;
import java.awt.event.*;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.Scanner;
import javax.swing.*;

public class MainWindow extends JFrame {
    private JPanel mainPanel                = new JPanel();
    private JButton getGraphFromFile        = new JButton("Import graph
from a file");
    private JButton getGraphFromKeyboard    = new JButton("Edit a graph
from text");
    private JButton getGraphFromGUI         = new JButton("Edit a graph
manually");
```

```
   private JButton saveOutputGraphToFile   = new JButton("Export output
graph to a file");
   private JButton saveInputGraphToFile    = new JButton("Export input
graph to a file");
   private JButton runAlgorithm            = new JButton("Start DJP
algorithm");
   private JPanel inputGraphPanel          = new JPanel();
   private JPanel outputGraphPanel         = new JPanel();

   private JLabel labelInputGraph          = new JLabel("Input Graph",
SwingConstants.CENTER);
   private JLabel labelOutputGraph         = new JLabel("Output Graph",
SwingConstants.CENTER);
   private Graph graph = null;

   private mxGraphComponent inpGraphComp;
   private mxGraphComponent outpGraphComp;


   public MainWindow() {
       super("DJP Algorithm Visualizer");
       setBounds(150, 150, 1210, 900);
       Dimension d = new Dimension();
       d.setSize(1080, 700);
       setMinimumSize(d);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setVisible(true);

       Color c = new Color(234, 255, 226); //#F6FFF8
       mainPanel.setBackground(c);
       mainPanel.setLayout(null);
       add(mainPanel);

       mainPanel.add(getGraphFromFile);
```

```
mainPanel.add(getGraphFromKeyboard);
mainPanel.add(getGraphFromGUI);
mainPanel.add(saveOutputGraphToFile);
mainPanel.add(saveInputGraphToFile);
mainPanel.add(runAlgorithm);
mainPanel.add(inputGraphPanel);
mainPanel.add(outputGraphPanel);

getGraphFromFile.setBounds      (30, 30, 360, 70);
getGraphFromKeyboard.setBounds  (420, 30, 360, 70);
getGraphFromGUI.setBounds       (810, 30, 360, 70);
saveOutputGraphToFile.setBounds (810, 130,360, 50);
saveInputGraphToFile.setBounds  (30, 130, 360, 50);
runAlgorithm.setBounds          (420, 130, 360, 70);
inputGraphPanel.setBounds       (30, 210, 555, 630);
outputGraphPanel.setBounds      (615 ,210 ,555, 630);

inputGraphPanel.add(labelInputGraph);
labelInputGraph.setBounds(0, 0, 555, 20);

outputGraphPanel.add(labelOutputGraph);
labelOutputGraph.setBounds(0,0, 555, 20);

Font f = new Font("Monospaced", Font.PLAIN, 18);
getGraphFromFile.setFont(f);
getGraphFromKeyboard.setFont(f);
getGraphFromGUI.setFont(f);
saveOutputGraphToFile.setFont(f);
saveInputGraphToFile.setFont(f);
runAlgorithm.setFont(f);
labelInputGraph.setFont(f);
labelOutputGraph.setFont(f);

inputGraphPanel.setBackground(c);
```

```
        outputGraphPanel.setBackground(c);

        eventHandler eH = new eventHandler();
        runAlgorithm.addActionListener(eH);
        getGraphFromGUI.addActionListener(eH);
        getGraphFromKeyboard.addActionListener(eH);
        getGraphFromFile.addActionListener(eH);
        saveOutputGraphToFile.addActionListener(eH);
        saveInputGraphToFile.addActionListener(eH);
        mainPanel.addComponentListener(eH);
        setFocusable(true);
        addKeyListener(eH);
    }

    public static void main(String[] args) {
        MainWindow window = new MainWindow();
    }


    class eventHandler implements ActionListener, ComponentListener,
KeyListener {
        @Override
        public void actionPerformed(ActionEvent actionEvent) {
            if(actionEvent.getSource() == runAlgorithm) {
                if (graph == null)
                    return;
                DJPAlgorithm algorithm = new DJPAlgorithm();
                //Graph bu = graph.backupGraph();
                //graph = Graph.getGraphFromString(graph.toString());
                //graph.restoreMXGraph(bu);
                graph.resetColors();
                String inp = JOptionPane.showInputDialog(MainWindow.this,
                        "Input start vertex: ", "");
                if (inp == null) {
```

```
                algorithm.init(graph);
            }
            else
            {
                algorithm.init(graph, inp);
            }
            StateWindow state = new StateWindow(MainWindow.this, this,
algorithm);
            outputGraphPanel.removeAll();
            outputGraphPanel.add(labelOutputGraph);
        }
        if(actionEvent.getSource() == getGraphFromGUI){
            GraphGUI_Input graphGUI_input = new
GraphGUI_Input(MainWindow.this, this, graph);
        }
        if(actionEvent.getSource() == getGraphFromFile)
        {
            String str = "";
            FileDialog fd = new FileDialog(MainWindow.this, "Choose a
file", FileDialog.LOAD);
            fd.setDirectory(System.getProperty("user.home"));
            fd.setFilenameFilter((file, s) -> s.endsWith(".txt"));
            fd.setVisible(true);

            String filename = fd.getDirectory();
            if (filename == null)
                return;
            filename += fd.getFile();
            try {
                FileReader fr = new FileReader(filename);
                Scanner scan = new Scanner(fr);
                while (scan.hasNextLine()) {
                    str += scan.nextLine();
                    str += "\n";
```

```
                    }
                    fr.close();
                } catch (Exception e) {
                    JLabel errMsg = new JLabel("An error has occured");
                    errMsg.setFont(new Font("Monospaced", Font.PLAIN,
18));
                    JOptionPane.showMessageDialog(null, errMsg, "ERROR",
JOptionPane.WARNING_MESSAGE);
                }
                if (!Graph.isValid(str)) {
                    JLabel msg = new JLabel("The graph must be
connected");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(MainWindow.this, msg,
                            "Invalid input", JOptionPane.WARNING_MESSAGE);
                    return;
                }
                graph = Graph.getGraphFromString(str);
                inputGraphPanel.removeAll();
                inputGraphPanel.add(labelInputGraph);
                inputGraphPanel.setLayout(null);
                graph.createGraphComponent();
                mxGraphComponent grComp = graph.updateGraphComponent();
                grComp.setEnabled(false);
                grComp.setBounds(0, 30 , inputGraphPanel.getWidth(),
inputGraphPanel.getHeight());
                inputGraphPanel.add(grComp);
                inputGraphPanel.updateUI();
            }
            if(actionEvent.getSource() == getGraphFromKeyboard)
            {
                GraphText_Input GUIGraph = new
GraphText_Input(MainWindow.this, this,
```

```
                                                                graph !=
null ? graph.toString() : "");
            }
            if(actionEvent.getSource() == saveOutputGraphToFile)
            {
                if (graph == null || graph.answerToString().length() == 0)
                {
                    JLabel noGraph = new JLabel("Nothing to export");
                    noGraph.setFont(new Font("Monospaced", Font.PLAIN,
18));
                    JOptionPane.showMessageDialog(null, noGraph, "No graph
found", JOptionPane.WARNING_MESSAGE);
                    return;
                }
                FileDialog fd = new FileDialog(MainWindow.this, "Choose a
file", FileDialog.LOAD);
                fd.setDirectory(System.getProperty("user.home"));
                fd.setFilenameFilter((file, s) -> s.endsWith(".txt"));
                fd.setVisible(true);
                String filename = fd.getDirectory();
                if (filename == null)
                    return;
                String newGraph = "";
                filename += fd.getFile();
                try {
                    FileWriter newFile = new FileWriter(filename);
                    newGraph += graph.answerToString();
                    newFile.write(newGraph);

                    newFile.close();
                }
                catch (Exception e){
                    JLabel errMsg = new JLabel("An error has occured");
```

```
                errMsg.setFont(new Font("Monospaced", Font.PLAIN,
18));
                JOptionPane.showMessageDialog(null, errMsg, "Error",
JOptionPane.WARNING_MESSAGE);
            }
        }
        if (actionEvent.getSource() == saveInputGraphToFile)
        {
            if (graph == null || graph.toString().length() == 0)
            {
                JLabel noGraph = new JLabel("Nothing to export");
                noGraph.setFont(new Font("Monospaced", Font.PLAIN,
18));
                JOptionPane.showMessageDialog(null, noGraph, "No graph
found", JOptionPane.WARNING_MESSAGE);
                return;
            }
            FileDialog fd = new FileDialog(MainWindow.this, "Choose a
file", FileDialog.LOAD);
            fd.setDirectory(System.getProperty("user.home"));
            fd.setFilenameFilter((file, s) -> s.endsWith(".txt"));
            fd.setVisible(true);
            String filename = fd.getDirectory();
            if (filename == null)
                return;
            String newGraph = "";
            filename += fd.getFile();
            try {
                FileWriter newFile = new FileWriter(filename);
                newGraph += graph.toString();
                newFile.write(newGraph);

                newFile.close();
            }
```

```
                catch (Exception e){
                    JLabel errMsg = new JLabel("An error has occured");
                    errMsg.setFont(new Font("Monospaced", Font.PLAIN,
18));
                    JOptionPane.showMessageDialog(null, errMsg, "Error",
JOptionPane.WARNING_MESSAGE);
                }
            }
            if(actionEvent.getSource() instanceof GraphText_Input){
                String str = actionEvent.getActionCommand();
                graph = Graph.getGraphFromString(str);
                inputGraphPanel.removeAll();
                inputGraphPanel.add(labelInputGraph);
                inputGraphPanel.setLayout(null);
                graph.createGraphComponent();
                mxGraphComponent grComp = graph.updateGraphComponent();
                grComp.setBounds(0, 30,  inputGraphPanel.getWidth(),
inputGraphPanel.getHeight());
                grComp.setEnabled(false);
                inputGraphPanel.add(grComp);
                inputGraphPanel.updateUI();
                inpGraphComp = graph.updateGraphComponent();

            }
            if(actionEvent.getSource() instanceof Graph){
                if (actionEvent.getActionCommand().equals("Finished")) {
                    outputGraphPanel.removeAll();
                    outputGraphPanel.add(labelOutputGraph);
                    outputGraphPanel.setLayout(null);
                    Graph tmp = (Graph) actionEvent.getSource();
                    mxGraphComponent grComp = tmp.updateGraphComponent();
                    grComp.setBounds(0, 30, outputGraphPanel.getWidth(),
outputGraphPanel.getHeight() - 30);
                    outputGraphPanel.add(grComp);
```

```
                    outputGraphPanel.updateUI();
                    outpGraphComp = graph.updateGraphComponent();

                }
                if (actionEvent.getActionCommand().equals("Edited") ||
actionEvent.getActionCommand().equals("Not changed"))
                {
                    if (actionEvent.getSource() == null ||
actionEvent.getSource().toString().length() == 0)
                        return;
                    inputGraphPanel.removeAll();
                    inputGraphPanel.add(labelInputGraph);
                    inputGraphPanel.setLayout(null);
                    graph = (Graph) actionEvent.getSource();
                    mxGraphComponent grComp =
graph.updateGraphComponent();
                    grComp.setEnabled(false);
                    grComp.setBounds(0, 30, inputGraphPanel.getWidth(),
inputGraphPanel.getHeight() - 30);
                    inputGraphPanel.add(grComp);
                    inputGraphPanel.updateUI();
                    inpGraphComp = graph.updateGraphComponent();
                }
            }
        }

        @Override
        public void componentResized(ComponentEvent componentEvent) {
            if (componentEvent.getComponent() == mainPanel)
            {
                int w = mainPanel.getWidth();
                int h = mainPanel.getHeight();
                w = (w - 360 * 3) / 4;
                h = h / 60;
```

```
                getGraphFromFile.setBounds      (w, h, 360, 70);
                getGraphFromKeyboard.setBounds  (360 + 2 * w, h, 360, 70);
                getGraphFromGUI.setBounds       (720 + 3 * w, h, 360, 70);
                saveOutputGraphToFile.setBounds (720 + 3 * w, 70 + 2 * h ,
360, 50);
                saveInputGraphToFile.setBounds  (w, 70 + 2 * h , 360, 50);
                runAlgorithm.setBounds          (360 + 2 * w, 70 + 2 * h ,
360, 70);
                int width = mainPanel.getWidth();
                width -= 60;
                width /= 2;
                int height = mainPanel.getHeight();
                height = height - 140 - 4 * h;
                inputGraphPanel.setBounds       (20         , 140 + 3 * h,
width, height);
                outputGraphPanel.setBounds      (width + 40, 140 + 3 * h,
width, height);

                for (int i = 0; i < inputGraphPanel.getComponentCount();
i++)
                {
                    if (inputGraphPanel.getComponent(i) instanceof
mxGraphComponent)
                        inputGraphPanel.getComponent(i).setBounds(0, 30,
width, height - 30);
                }
                for (int i = 0; i < outputGraphPanel.getComponentCount();
i++)
                {
                    if (outputGraphPanel.getComponent(i) instanceof
mxGraphComponent)
                        outputGraphPanel.getComponent(i).setBounds(0, 30,
width, height - 30);
                }
```

```
                if (inpGraphComp != null) {
                    inputGraphPanel.removeAll();
                    inputGraphPanel.add(labelInputGraph);
                    inpGraphComp.setBounds(0, 30,
inputGraphPanel.getWidth(), inputGraphPanel.getHeight() - 30);
                    inputGraphPanel.add(inpGraphComp);
                    inputGraphPanel.updateUI();
                }
                if (outpGraphComp != null)
                {
                    outputGraphPanel.removeAll();
                    outputGraphPanel.add(labelOutputGraph);
                    outpGraphComp.setBounds(0, 30,
outputGraphPanel.getWidth(), outputGraphPanel.getHeight() - 30);
                    outputGraphPanel.add(outpGraphComp);
                    outputGraphPanel.updateUI();
                }

            }
        }

        @Override
        public void componentMoved(ComponentEvent componentEvent) {

        }

        @Override
        public void componentShown(ComponentEvent componentEvent) {

        }

        @Override
        public void componentHidden(ComponentEvent componentEvent) {
```

```
        }

        @Override
        public void keyTyped(KeyEvent keyEvent) {
        }

        @Override
        public void keyPressed(KeyEvent keyEvent) {
            int keyCode = keyEvent.getExtendedKeyCode();
            if(keyCode == KeyEvent.VK_F1)
            {
                AboutWindow aboutWindow = new
AboutWindow(MainWindow.this);
            }
        }

        @Override
        public void keyReleased(KeyEvent keyEvent) {

        }
    }

}
```

StateWindow.java

```
package gui;

import algorithm.DJPAlgorithm;
import com.mxgraph.swing.mxGraphComponent;

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

```
public class StateWindow  extends JFrame {
   private final MainWindow parent;
   private int timerTime               = 10000;
   private JPanel mainPanel            = new JPanel();
   private JButton nextStep            = new JButton("Next step ->");
   private JButton prevStep            = new JButton("<- Previous step");
   private JButton interruptAlgorithm  = new JButton("Go to main menu");
   private JButton startPauseTimer     = new JButton("Pause");
   private JLabel timeCounter          = new
JLabel(Integer.toString(timerTime/1000), SwingConstants.CENTER);
   private JPanel prevGraphPanel       = new JPanel();
   private JPanel nextGraphPanel       = new JPanel();
   private JTextArea log               = new JTextArea();
   private JScrollPane scroll;

   private JLabel labelBeforeState     = new JLabel("Before State Graph",
SwingConstants.CENTER);
   private JLabel labelCurrentState    = new JLabel("Current State
Graph", SwingConstants.CENTER);
   private DJPAlgorithm algorithm;
   private ActionListener recipient;
   private boolean timerFlaag          = true;
   private int currentTime             = timerTime;
   private javax.swing.Timer timerLabel= new javax.swing.Timer(100, y->{
       currentTime -= 100;
       if(currentTime <= 0) {
           if(algorithm.isFinished()){
               currentTime = timerTime;
               startPauseTimer.doClick();
           }
           else {
               nextStep.doClick();
               currentTime = timerTime;
           }
```

```
        }
        timeCounter.setText(Integer.toString(currentTime/1000) + "." +
Integer.toString(currentTime%1000/100));
        timeCounter.updateUI();
    });

    public StateWindow(MainWindow  mainWindow, ActionListener recipient,
DJPAlgorithm algorithm) {
        super("State");
        this.algorithm = algorithm;
        this.recipient = recipient;
        parent = mainWindow;
        parent.setVisible(false);
        setBounds(150, 150, 1210, 900);
        Dimension d = new Dimension();
        d.setSize(1080, 700);
        setMinimumSize(d);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new WindowAdapter()
                        {
                            @Override
                            public void windowClosing(WindowEvent e) {
                                parent.setVisible(true);
                                dispose();
                            }
                        });
        setVisible(true);

        Color c = new Color(234, 255, 226); //#F6FFF8
        mainPanel.setBackground(c);
        mainPanel.setLayout(null);
        add(mainPanel);

        scroll = new JScrollPane(log);
```

```
        mainPanel.add(prevGraphPanel);
        mainPanel.add(nextGraphPanel);
        mainPanel.add(scroll);
        mainPanel.add(interruptAlgorithm);
        mainPanel.add(prevStep);
        mainPanel.add(nextStep);
        mainPanel.add(startPauseTimer);
        mainPanel.add(timeCounter);

        prevGraphPanel.setBounds    (30, 30, 560, 540);
        nextGraphPanel.setBounds    (620, 30, 560, 540);
        scroll.setBounds            (30, 585, 1150, 165);
        interruptAlgorithm.setBounds(30, 780,265, 70);
        prevStep.setBounds          (325, 780, 265, 70);
        nextStep.setBounds          (620, 780, 265, 70);
        startPauseTimer.setBounds   (910, 780, 120, 70);
        timeCounter.setBounds       (1040, 780, 140, 70);

        prevGraphPanel.add(labelBeforeState);
        labelBeforeState.setBounds(0, 0, 560, 20);
        prevGraphPanel.setLayout(null);

        nextGraphPanel.add(labelCurrentState);
        labelCurrentState.setBounds(0,0,560,20);
        nextGraphPanel.setLayout(null);
        mxGraphComponent grComp =
algorithm.getCurrentGraphState().updateGraphComponent();

        grComp.setBounds(0, 30, nextGraphPanel.getWidth(),
nextGraphPanel.getHeight() - 30);
        nextGraphPanel.add(grComp);
        nextGraphPanel.updateUI();
```

```
        Font f = new Font("Monospaced", Font.PLAIN, 18);
        interruptAlgorithm.setFont(f);
        prevStep.setFont(f);
        nextStep.setFont(f);
        startPauseTimer.setFont(f);
        timeCounter.setFont(new Font("Monospaced", Font.PLAIN, 30));
        labelCurrentState.setFont(f);
        labelBeforeState.setFont(f);
        log.setFont(f);

        prevGraphPanel.setBackground(c);
        nextGraphPanel.setBackground(c);
        log.setBackground(new Color(255, 250, 221)); //#FFFADD
        log.setLineWrap(true);
        log.setWrapStyleWord(true);

        eventHandler eH = new eventHandler();
        interruptAlgorithm.addActionListener(eH);
        nextStep.addActionListener(eH);
        prevStep.addActionListener(eH);
        startPauseTimer.addActionListener(eH);
        mainPanel.addComponentListener(eH);

        log.setEditable(false);
        timerLabel.start();
        startPauseTimer.setText("Pause");
        timeCounter.addMouseListener(eH);
        log.setText(algorithm.getComment());

    }

    class eventHandler implements ActionListener, ComponentListener,
MouseListener {
        @Override
```

```
        public void actionPerformed (ActionEvent actionEvent){
            if(actionEvent.getSource() == interruptAlgorithm) {
                timerLabel.stop();
                parent.setVisible(true);
                dispose();
            }
            if(actionEvent.getSource() == nextStep){
                if(!algorithm.isFinished()){
                    prevGraphPanel.removeAll();
                    prevGraphPanel.add(labelBeforeState);
                    mxGraphComponent grComp =
algorithm.getCurrentGraphState().updateGraphComponent();
                    grComp.setBounds(0, 30, prevGraphPanel.getWidth(),
prevGraphPanel.getHeight() - 30);
                    prevGraphPanel.add(grComp);
                    prevGraphPanel.updateUI();
                    algorithm.makeStep();
                    log.setText(log.getText() + algorithm.getComment());
                    nextGraphPanel.removeAll();
                    nextGraphPanel.add(labelCurrentState);
                    grComp =
algorithm.getCurrentGraphState().updateGraphComponent();
                    grComp.setBounds(0, 30, nextGraphPanel.getWidth(),
nextGraphPanel.getHeight() - 30);
                    nextGraphPanel.add(grComp);
                    nextGraphPanel.updateUI();
                    currentTime = timerTime;
                    timeCounter.setText(Integer.toString(currentTime/1000)
+ "." + Integer.toString(currentTime%1000/100));
                    timeCounter.updateUI();
                }
                else{
                    parent.setVisible(true);
                    timerLabel.stop();
```

```
                    int id = (int)System.currentTimeMillis();
                    ActionEvent message = new
ActionEvent(algorithm.getCurrentGraphState(), id, "Finished");
                    recipient.actionPerformed(message);
                    dispose();
                }
            }
            if(actionEvent.getSource() == prevStep)
            {
                timerLabel.stop();
                timerFlaag = false;
                startPauseTimer.setText("Start");
                currentTime = timerTime;
                timeCounter.setText(Integer.toString(currentTime/1000) +
"." + Integer.toString(currentTime%1000/100));
                timeCounter.updateUI();
                if (algorithm.canBeUndone()) {

log.setText(log.getText().replaceAll(algorithm.getComment(), ""));
                    mxGraphComponent  tmp = algorithm.undo();

                    nextGraphPanel.removeAll();
                    nextGraphPanel.add(labelCurrentState);
                    mxGraphComponent grComp =
algorithm.getCurrentGraphState().updateGraphComponent();
                    grComp.setBounds(0, 30, nextGraphPanel.getWidth(),
nextGraphPanel.getHeight() - 30);
                    nextGraphPanel.add(grComp);
                    nextGraphPanel.updateUI();

                    prevGraphPanel.removeAll();
                    prevGraphPanel.add(labelBeforeState);
                    tmp.setBounds(0, 30, prevGraphPanel.getWidth(),
prevGraphPanel.getHeight() - 30);
```

```
                prevGraphPanel.add(tmp);
                prevGraphPanel.updateUI();
            }

        }
        if(actionEvent.getSource() == startPauseTimer)
        {
            if(timerFlaag){
                timerLabel.stop();
                startPauseTimer.setText("Start");
                timerFlaag = false;
            }
            else{
                timerLabel.start();
                startPauseTimer.setText("Pause");
                timerFlaag = true;
            }
        }
    }


    @Override
    public void componentResized(ComponentEvent componentEvent) {
        if (componentEvent.getComponent() == mainPanel)
        {
            int w = mainPanel.getWidth();
            w = (w - 1055) / 6;
            int h = mainPanel.getHeight();
            h /= 30;
            int width = mainPanel.getWidth();
            int height = mainPanel.getHeight() - 4 * h - 70;

            prevGraphPanel.setBounds    (w, h, (width - 3 * w) / 2,
height * 2 / 3);
```

```
                nextGraphPanel.setBounds    ((width - 3 * w) / 2 + 2 * w,
h, (width - 3 * w) / 2, height * 2 / 3);
                scroll.setBounds            (w, height * 2 / 3 + 2 * h,
mainPanel.getWidth() - 2 * w, height / 3);
                interruptAlgorithm.setBounds(w, mainPanel.getHeight() - 70
- h,265, 70);
                prevStep.setBounds          (2 * w + 265,
mainPanel.getHeight() - 70 - h, 265, 70);
                nextStep.setBounds          (3 * w + 530,
mainPanel.getHeight() - 70 - h, 265, 70);
                startPauseTimer.setBounds   (4 * w + 795,
mainPanel.getHeight() - 70 - h, 120, 70);
                timeCounter.setBounds       (5 * w + 915,
mainPanel.getHeight() - 70 - h, 140, 70);

                for (int i = 0; i < prevGraphPanel.getComponentCount();
i++)
                {
                    if (prevGraphPanel.getComponent(i) instanceof
mxGraphComponent) {
                        prevGraphPanel.getComponent(i).setBounds(0, 30,
width, height - 30);
                    }

                }
                for (int i = 0; i < nextGraphPanel.getComponentCount();
i++) {
                    if (nextGraphPanel.getComponent(i) instanceof
mxGraphComponent) {
                        nextGraphPanel.getComponent(i).setBounds(0, 30,
width, height - 30);
                    }
                }
                mxGraphComponent grComp = algorithm.getPrevState();
```

```
                if(algorithm.canBeUndone()) {
                    prevGraphPanel.removeAll();
                    prevGraphPanel.add(labelBeforeState);
                    grComp.setBounds(0, 30, prevGraphPanel.getWidth(),
prevGraphPanel.getHeight() - 30);
                    prevGraphPanel.add(grComp);
                    prevGraphPanel.updateUI();
                }
                grComp =
algorithm.getCurrentGraphState().updateGraphComponent();
                nextGraphPanel.removeAll();
                nextGraphPanel.add(labelCurrentState);
                grComp =
algorithm.getCurrentGraphState().updateGraphComponent();
                grComp.setBounds(0, 30, nextGraphPanel.getWidth(),
nextGraphPanel.getHeight() - 30);
                nextGraphPanel.add(grComp);
                nextGraphPanel.updateUI();
            }
        }

        @Override
        public void componentMoved(ComponentEvent componentEvent) {

        }

        @Override
        public void componentShown(ComponentEvent componentEvent) {

        }

        @Override
        public void componentHidden(ComponentEvent componentEvent) {
```

```
        }

        @Override
        public void mouseClicked(MouseEvent e) {
            boolean continueOnClose = timerFlaag;
            timerLabel.stop();
            timerFlaag = false;
            try {
                String inp = JOptionPane.showInputDialog(StateWindow.this,
                        "Input new timer time: ",
String.valueOf(currentTime/1000));
                if (inp == null) {
                    timerTime = 10000;
                    if(continueOnClose) {
                        timerFlaag = true;
                        timerLabel.start();
                    }
                    return;
                }
                if (!inp.matches("^[0-9]+$"))
                {
                    JLabel msg = new JLabel("Input a number");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(StateWindow.this, msg,
                            "Invalid input", JOptionPane.WARNING_MESSAGE);
                    if(continueOnClose) {
                        timerFlaag = true;
                        timerLabel.start();
                    }
                    return;
                }
                int tmpTime = Integer.parseInt(inp);
                if (tmpTime > 0) {
                    if(tmpTime < currentTime)
```

```
                    currentTime = tmpTime * 1000;
                timerTime = tmpTime * 1000;
            }

        }catch (Exception e1){
            timerTime = 10000;
        }
        timeCounter.setText(Integer.toString(currentTime/1000) + "." +
Integer.toString(currentTime%1000/100));
        timeCounter.updateUI();
        if(continueOnClose) {
            timerFlaag = true;
            timerLabel.start();
        }
    }

    @Override
    public void mousePressed(MouseEvent e) {

    }

    @Override
    public void mouseReleased(MouseEvent e) {

    }

    @Override
    public void mouseEntered(MouseEvent e) {

    }

    @Override
    public void mouseExited(MouseEvent e) {
```

```
        }
    }
}
```

GraphText_Input.java

```
package gui;

import algorithm.Graph;

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

public class GraphText_Input extends JFrame {
    private final MainWindow parent;
    private JPanel mainPanel            = new JPanel();
    private JTextArea text              = new JTextArea();
    private JScrollPane scroll;
    private JButton applyInputGraph     = new JButton("Apply");
    private JButton cancelInputGraph    = new JButton("Cancel");
    private ActionListener recipient;

    public GraphText_Input(MainWindow mainWindow, ActionListener
recipient, String str) {
        super("Graph Input");
        this.recipient = recipient;
        parent = mainWindow;
        parent.setEnabled(false);
        text.setText(str);
        setBounds(250, 250, 800, 600);
        Dimension d = new Dimension();
        d.setSize(760, 300);
        setMinimumSize(d);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

```
addWindowListener(new WindowAdapter()
                {
                    @Override
                    public void windowClosing(WindowEvent e) {
                        parent.setEnabled(true);
                        dispose();
                    }
                });
setVisible(true);

Color mainColor = new Color(234, 255, 226); //#FFFADD

mainPanel.setBackground(mainColor); //#F6FFF8
mainPanel.setLayout(null);
add(mainPanel);

scroll = new JScrollPane(text);
text.setLineWrap(true);
text.setWrapStyleWord(true);

mainPanel.add(scroll);
mainPanel.add(applyInputGraph);
mainPanel.add(cancelInputGraph);

scroll.setBounds            (30,30, 730, 430);
cancelInputGraph.setBounds  (30, 480, 360, 70);
applyInputGraph.setBounds   (400, 480, 360, 70);

Font f = new Font("Monospaced", Font.PLAIN, 18);
applyInputGraph.setFont(f);
cancelInputGraph.setFont(f);
text.setFont(f);

Color c = new Color(255, 250, 221); //#FFFADD
```

```
        text.setBackground(c);

        eventHandler eH = new eventHandler();
        cancelInputGraph.addActionListener(eH);
        applyInputGraph.addActionListener(eH);
        mainPanel.addComponentListener(eH);

    }

    class eventHandler implements ActionListener, ComponentListener{
        @Override
        public void actionPerformed (ActionEvent actionEvent){
            if(actionEvent.getSource() == cancelInputGraph) {
                parent.setEnabled(true);
                dispose();
            }
            if(actionEvent.getSource() == applyInputGraph)
            {
                int id = (int)System.currentTimeMillis();

                String str = text.getText();
                if (str.length() == 0)
                {
                    JLabel msg = new JLabel("The graph must contain at
least one edge");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(GraphText_Input.this,
msg,
                            "Invalid input", JOptionPane.WARNING_MESSAGE);
                    return;
                }
                str += "\n";
                str = str.replaceAll("\n\n", "\n");
                str = str.replaceAll("  ", " ");
```

```
                str = str.replaceAll("\n ", "\n");
                if (str.startsWith(" ")) {
                    JLabel msg = new JLabel("Text starts with invalid
symbol");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(GraphText_Input.this,
msg,
                                                  "Invalid input",
JOptionPane.WARNING_MESSAGE);
                    return;
                }
                if (!Graph.isValid(str)) {
                    JLabel msg = new JLabel("The graph must be
connected");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(GraphText_Input.this,
msg,
                            "Invalid input", JOptionPane.WARNING_MESSAGE);
                    return;
                }


                ActionEvent message = new
ActionEvent(GraphText_Input.this, id, str);
                recipient.actionPerformed(message);
                parent.setEnabled(true);
                dispose();
            }
        }

        @Override
        public void componentResized(ComponentEvent componentEvent) {
            if (componentEvent.getComponent() == mainPanel)
            {
```

```
                int w = mainPanel.getWidth();
                int width = w;
                int h = mainPanel.getHeight();
                int height = h;
                w /= 60;
                width -= 2 * w;
                h = h / 60;
                height -= 3 * h + 70;
                scroll.setBounds            (w,h, width, height);
                cancelInputGraph.setBounds  (w, mainPanel.getHeight() - 70
-  h, 360, 70);
                applyInputGraph.setBounds   (mainPanel.getWidth() - 360 -
w, mainPanel.getHeight() - 70 -  h, 360, 70);
            }
        }

        @Override
        public void componentMoved(ComponentEvent componentEvent) {

        }

        @Override
        public void componentShown(ComponentEvent componentEvent) {

        }

        @Override
        public void componentHidden(ComponentEvent componentEvent) {

        }
    }
}
```

GraphGUI_Input.java

```
package gui;

import algorithm.Graph;
import com.mxgraph.model.mxCell;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.view.mxGraph;

import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Arrays;
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class GraphGUI_Input extends JFrame {
   private final MainWindow parent;
   private JPanel mainPanel            = new JPanel();
   private JPanel inputGraphPanel      = new JPanel();
   private JButton applyInputGraph     = new JButton("Apply");
   private JButton cancelInputGraph    = new JButton("Cancel");
   private JLabel inputMode            = new JLabel("Edit mode:");
   private JRadioButton modeVertex     = new JRadioButton("Vertex
editing");
   private JRadioButton modeEdge       = new JRadioButton("Edge
editing");
   private ActionListener recipient;
   private Graph graph;
   private final Graph backUp;
   private mxGraphComponent comp;

   private Object selected;
```

```
   public GraphGUI_Input(MainWindow mainWindow, ActionListener recipient,
Graph gr){
        super("Graph Input");
        parent = mainWindow;
        this.recipient = recipient;
        this.graph = gr;
        if (gr != null)
            backUp = gr.backupGraph();
        else
            backUp = new Graph();
        parent.setVisible(false);
        setBounds(150, 150, 1210, 910);
        Dimension d = new Dimension();
        d.setSize(1050, 600);
        setMinimumSize(d);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new WindowAdapter()
                        {
                            @Override
                            public void windowClosing(WindowEvent e) {
                                parent.setVisible(true);
                                dispose();
                            }
                        });
        setVisible(true);

        Color mainColor = new Color(234, 255, 226); //#FFFADD

        mainPanel.setBackground(mainColor); //#F6FFF8
        mainPanel.setLayout(null);
        add(mainPanel);

        mainPanel.add(inputGraphPanel);
```

```
mainPanel.add(applyInputGraph);
mainPanel.add(cancelInputGraph);
mainPanel.add(inputMode);
mainPanel.add(modeVertex);
mainPanel.add(modeEdge);

inputGraphPanel.setBounds   (30,30, 1140, 740);
inputMode.setBounds         (30, 790, 120, 70);
cancelInputGraph.setBounds  (480, 810, 300, 50);
applyInputGraph.setBounds   (810, 790, 360, 70);
modeVertex.setBounds        (140, 790, 200, 45);
modeEdge.setBounds          (140, 820, 200, 45);

Font f = new Font("Monospaced", Font.PLAIN, 18);
inputMode.setFont(f);
cancelInputGraph.setFont(f);
applyInputGraph.setFont(f);
modeEdge.setFont(f);
modeVertex.setFont(f);

ButtonGroup group = new ButtonGroup();
group.add(modeVertex);
group.add(modeEdge);
modeVertex.setSelected(true);

Color c = new Color(255, 250, 221); //#FFFADD
inputGraphPanel.setBackground(c);
modeVertex.setBackground(mainColor);
modeEdge.setBackground(mainColor);

eventHandler eH = new eventHandler();
modeVertex.addChangeListener(eH);
modeEdge.addChangeListener(eH);
cancelInputGraph.addActionListener(eH);
```

```
        applyInputGraph.addActionListener(eH);
        mainPanel.addComponentListener(eH);


        if (graph != null)
        {
            graph.resetColors();
            comp = graph.updateGraphComponent();
        }
        else
        {
            mxGraph tmp = new mxGraph();
            tmp.setAllowDanglingEdges(false);
            tmp.setCellsEditable(false);
            tmp.setCellsDisconnectable(false);
            tmp.setConnectableEdges(false);
            tmp.setVertexLabelsMovable(false);
            tmp.setEdgeLabelsMovable(false);
            tmp.setResetEdgesOnMove(true);
            tmp.setCellsResizable(false);
            graph = new Graph();
            graph.setMXGraph(tmp);
            comp = new mxGraphComponent(tmp);
        }
        inputGraphPanel.add(comp);
        inputGraphPanel.setLayout(null);
        inputGraphPanel.updateUI();
        comp.setVisible(true);
        comp.setEnabled(true);
        comp.setConnectable(false);
        comp.setBounds(0, 0,  inputGraphPanel.getWidth(),
inputGraphPanel.getHeight());
        comp.setDragEnabled(false);
        comp.getGraphControl().addMouseListener(eH);
```

```
    }

    class eventHandler implements ActionListener, ChangeListener,
ComponentListener, MouseListener {
        @Override
        public void actionPerformed (ActionEvent actionEvent){
            if(actionEvent.getSource() == cancelInputGraph) {
                int id = (int)System.currentTimeMillis();
                if(backUp != null)
                {
                    ActionEvent message = new ActionEvent(backUp, id, "Not
changed");
                    recipient.actionPerformed(message);
                }
                parent.setVisible(true);
                dispose();
            }
            if(actionEvent.getSource() == applyInputGraph)
            {
                int id = (int)System.currentTimeMillis();
                if (graph == null)
                {
                    JLabel msg = new JLabel("No graph entered");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(GraphGUI_Input.this,
msg,
                            "Invalid input", JOptionPane.WARNING_MESSAGE);
                    return;
                }
                String str = graph.toString();
                if (str.length() == 0)
                {
```

```
                    JLabel msg = new JLabel("The graph must contain at
least one edge");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(GraphGUI_Input.this,
msg,
                            "Invalid input", JOptionPane.WARNING_MESSAGE);
                    return;
                }

                if (!Graph.isValid(str)) {
                    JLabel msg = new JLabel("The graph must be
connected");
                    msg.setFont( new Font("Monospaced", Font.PLAIN, 18));
                    JOptionPane.showMessageDialog(GraphGUI_Input.this,
msg,
                            "Invalid input", JOptionPane.WARNING_MESSAGE);
                    return;
                }


comp.getGraph().moveCells(comp.getGraph().getChildCells(comp.getGraph().g
etDefaultParent()),
                        getDistanceX(), getDistanceY());
                graph.setMXGraph(comp.getGraph());
                ActionEvent message = new ActionEvent(graph, id,
"Edited");
                recipient.actionPerformed(message);
                parent.setVisible(true);
                dispose();
            }
        }

        @Override
        public void stateChanged(ChangeEvent changeEvent) {
```

```
            selected = null;
        }

        private double getDistanceX ()
        {
            double leftMost = Double.MAX_VALUE;
            Object[] verticess =
comp.getGraph().getChildVertices(comp.getGraph().getDefaultParent());
            for (Object c : verticess)
            {
                mxCell v = (mxCell)c;
                if (v.getGeometry().getX() < leftMost)
                    leftMost = v.getGeometry().getX();
            }
            return 10 - leftMost;
        }
        private double getDistanceY ()
        {
            double leftMost = Double.MAX_VALUE;
            Object[] verticess =
comp.getGraph().getChildVertices(comp.getGraph().getDefaultParent());
            for (Object c : verticess)
            {
                mxCell v = (mxCell)c;
                if (v.getGeometry().getY() < leftMost)
                    leftMost = v.getGeometry().getY();
            }
            return 10 - leftMost;
        }

        @Override
        public void componentResized(ComponentEvent componentEvent) {
            if (componentEvent.getComponent() == mainPanel)
            {
```

```
                int w = mainPanel.getWidth();
                int width = w;
                w /= 60;
                int h = mainPanel.getHeight();
                int height = h;
                h /=60;
                inputGraphPanel.setBounds   (w,h, mainPanel.getWidth() - 2
* w, mainPanel.getHeight() - 70 - 3 * h);
                inputMode.setBounds         (w, mainPanel.getHeight() - h
- 70, 120, 70);
                cancelInputGraph.setBounds  (mainPanel.getWidth() - 2 * w
- 660, mainPanel.getHeight() - h - 50, 300, 50);
                applyInputGraph.setBounds   (mainPanel.getWidth() - w -
360, mainPanel.getHeight() - h - 70, 360, 70);
                modeVertex.setBounds        (120 + w,
mainPanel.getHeight() - h - 70, 200, 45);
                modeEdge.setBounds          (120 + w,
mainPanel.getHeight() - h - 35, 200, 45);

                for (int i = 0; i < inputGraphPanel.getComponentCount();
i++)
                {
                    if (inputGraphPanel.getComponent(i) instanceof
mxGraphComponent)
                        inputGraphPanel.getComponent(i).setBounds(0, 0,
width, height);
                }
            }
        }

        @Override
        public void componentMoved(ComponentEvent componentEvent) {

        }
```

```
@Override
public void componentShown(ComponentEvent componentEvent) {

}

@Override
public void componentHidden(ComponentEvent componentEvent) {
}

@Override
public void mousePressed(MouseEvent mouseEvent) {

}

@Override
public void mouseReleased(MouseEvent mouseEvent) {

}

@Override
public void mouseEntered(MouseEvent mouseEvent) {

}

@Override
public void mouseExited(MouseEvent mouseEvent) {

}

@Override
public void mouseClicked(MouseEvent mouseEvent) {
    double x = mouseEvent.getPoint().getX();
    double y = mouseEvent.getPoint().getY();
```

```
            ArrayList<Object> vertexList = new ArrayList<Object>

(Arrays.asList(comp.getGraph().getChildVertices(comp.getGraph().getDefaul
tParent())));
            Object o = comp.getCellAt((int)x, (int)y);

            if (o != null && o instanceof mxCell &&
vertexList.contains(o)){
                if (selected == null)
                {
                    if (modeVertex.isSelected()) {

graph.removeVertex(((mxCell)o).getValue().toString());
                        comp.getGraph().getModel().remove(o);
                    }
                    else{
                        selected = o;
                    }
                    return;
                }
                else if (selected == o)
                {
                    selected = null;
                    return;

                }
                else
                {
                    try {
                        if (modeEdge.isSelected()) {
                            // if Edge exists - remove it and return
                            for (int i = 0; i <
comp.getGraph().getModel().getEdgeCount(comp.getGraph().getSelectionCell(
)); i++) {
```

```
                                if (selected == ((mxCell)
comp.getGraph().getModel().getEdgeAt(comp.getGraph().getSelectionCell(),
i)).getSource()
                                        || selected == ((mxCell)
comp.getGraph().getModel().getEdgeAt(comp.getGraph().getSelectionCell(),
i)).getTarget()) {
                                    if (o == ((mxCell)
comp.getGraph().getModel().getEdgeAt(comp.getGraph().getSelectionCell(),
i)).getSource()
                                            || o == ((mxCell)
comp.getGraph().getModel().getEdgeAt(comp.getGraph().getSelectionCell(),
i)).getTarget()) {

comp.getGraph().getModel().remove(comp.getGraph().getModel().getEdgeAt(co
mp.getGraph().getSelectionCell(), i));

graph.removeEdge(((mxCell)selected).getValue().toString(),

((mxCell)o).getValue().toString());
                                        return;
                                    }
                                }
                            }
                            // else add new Edge
                            comp.getGraph().getModel().beginUpdate();
                            try {
                                Color c = Color.GRAY;
                                String inp =
JOptionPane.showInputDialog(GraphGUI_Input.this, "Input edge cost:",
"10");
                                if (inp == null)
                                    return;
                                if (!inp.matches("^[0-9]+$"))
                                {
```

```
                                    JLabel msg = new JLabel("Input a
number");
                                    msg.setFont( new Font("Monospaced",
Font.PLAIN, 18));

JOptionPane.showMessageDialog(GraphGUI_Input.this, msg,
                                            "Invalid input",
JOptionPane.WARNING_MESSAGE);
                                    return;
                                }
                                int cost = Integer.parseInt(inp);
                                String style =
"align=center;strokeWidth=2;startArrow=none;endArrow=none;fontSize=24;" +

String.format("strokeColor=#%02x%02x%02x", c.getRed(), c.getGreen(),
c.getBlue());

comp.getGraph().insertEdge(comp.getGraph().getDefaultParent(), null,
                                        inp, selected, o, style);

graph.addEdge(((mxCell)selected).getValue().toString(),
                                        ((mxCell)o).getValue().toString(),
cost);
                            } finally {
                                comp.getGraph().getModel().endUpdate();
                            }
                            return;
                        }
                    }
                    finally {
                        selected = null;
                    }
                }
            }
```

```
            else {
                if (modeVertex.isSelected()) {
                    comp.getGraph().getModel().beginUpdate();
                    try {
                        int i = 0;
                        boolean valid = false;
                        while (!valid) {
                            i++;
                            for (Object p : vertexList) {
                                mxCell px = (mxCell) p;
                                valid |=
px.getValue().toString().equals(String.valueOf(i));
                            }
                            valid = !valid;
                        }


                        Color c = Color.GRAY;
                        String styleVertex = "shape=ellipse;fontSize=24;"
+
                                String.format("fillColor=#%02x%02x%02x",
c.getRed(), c.getGreen(), c.getBlue());

comp.getGraph().insertVertex(comp.getGraph().getDefaultParent(), null,
String.valueOf(i),
                                  x - 25, y - 25, 50, 50, styleVertex);
                    } finally {
                        comp.getGraph().getModel().endUpdate();
                    }
                }
            }
        }

    }
```

```
}
```

AboutWindow.java

```
package gui;

import com.sun.tools.javac.Main;

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

public class AboutWindow extends JFrame {
   private JPanel mainPanel = new JPanel();
   private JLabel theme = new JLabel("Dijkstra-Jarnik-Prim algorithm");
   private JTextArea description = new JTextArea("The program implements
step-by-step visualization of the " +
           "Dijkstra-Jarnik-Prim algorithm for finding the minimum
spanning tree in a graph. The input to the program " +
           "is an undirected graph, which is defined graphically, or in
text form (a list of edges and their weights). " +
           "After the completion of the algorithm, the resulting spanning
tree is displayed on the main window, it is " +
           "also possible to save the result in a text file. During the
execution of the algorithm, the user can see " +
           "the previous and current state of the program.");
   private JButton descriptionButton = new JButton("Algorithm
description");
   private JTextArea students = new JTextArea("Authors:\n" +
           "Vlasov Roman\n"+
           "Sychevsky Radimir\n"+
           "Iolshina Valeria");
   private JButton close = new JButton("OK");
```

```
private MainWindow parent;


AboutWindow(MainWindow parent) {
    super("Pathetic Attempt");
    setVisible(true);
    this.parent = parent;
    parent.setEnabled(false);
    setResizable(false);
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    addWindowListener(new WindowAdapter()
    {
        @Override
        public void windowClosing(WindowEvent e) {
            parent.setEnabled(true);
            dispose();
        }
    });
    setBounds(150, 150, 900, 450);

    Color c = new Color(234, 255, 226);
    mainPanel.setBackground(c);
    mainPanel.setLayout(null);
    add(mainPanel);

    description.setEditable(false);
    students.setEditable(false);

    mainPanel.add(theme);
    mainPanel.add(description);
    mainPanel.add(descriptionButton);
    mainPanel.add(students);
    mainPanel.add(close);
```

```
        theme.setBounds             (280,30,500,50);
        description.setBounds       (30, 90, 840, 200);
        descriptionButton.setBounds (510, 310, 360, 30);
        students.setBounds          (30, 300, 250, 100);
        close.setBounds             (510, 350, 360, 30);

        Font f = new Font("Monospaced", Font.PLAIN, 18);
        theme.setFont(f);
        description.setFont(f);
        descriptionButton.setFont(f);
        students.setFont(f);
        close.setFont(f);

        description.setBackground(c);
        students.setBackground(c);

        description.setLineWrap(true);
        description.setWrapStyleWord(true);

        close.addActionListener(new TestActionListener());
        descriptionButton.addActionListener(new TestActionListener());
    }

    class TestActionListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            if(e.getSource() == close) {
                parent.setEnabled(true);
                dispose();
            }
            if(e.getSource() == descriptionButton) {
                String description = "<html>The algorithm may informally
be described as performing the following steps:<br>" +
                        "<br>" +
```

```
                        "Initialize a tree with a single vertex, chosen
arbitrarily from the graph.<br>" +
                        "Grow the tree by one edge: of the edges that
connect the tree to vertices not <br>" +
                        "yet in the tree, find the minimum-weight edge,
and transfer it to the tree.<br>" +
                        "Repeat step 2 (until all vertices are in the
tree).</html>";
                JLabel message = new JLabel(description);

                message.setFont(new Font("Monospaced", Font.PLAIN, 18));
                JOptionPane.showMessageDialog(new JFrame(), message,
"Algorithm description", JOptionPane.PLAIN_MESSAGE);
            }
        }
    }
}
```