

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

ОТЧЕТ
по учебной практике
Тема: Алгоритм Ярника-Прима-Дейкстры

Студент гр. 8304	_____	Холковский К.В.
Студент гр. 8304	_____	Птухов Д.А.
Руководитель	_____	Жангиров Т.Р.

Санкт-Петербург
2020

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Холковский К.В. группы 8304

Студент Птухов Д.А. группы 8304

Тема практики: Алгоритм Ярника-Прима-Дейкстры

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Алгоритм Ярника-Прима-Дейкстры:

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 10.07.2020

Дата защиты отчета: 10.07.2020

Студент	_____	Холковский К.В.
Студент	_____	Птухов Д.А.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

В основе учебной практики стоит реализация визуализации алгоритма ЯПД на языке программирования Java. Итеративная командная работа обеспечивает обмен опытом как студентов между собой, так и с преподавателем, имеющим реальные практические знания по командной разработке коммерческого продукта. Важной особенностью учебной практики и реализации алгоритма является распределение времени, знаний и умений между всей командой разработки, поиск сильных черт студентов для дальнейшего профессионального и личностного роста.

SUMMARY

The practice is based on the implementation of visualization of the YPD algorithm in the Java programming language. Iterative teamwork provides an exchange of experience both among students and with a teacher who has real practical knowledge of team development of a commercial product. An important feature of educational practice and the implementation of the algorithm is the distribution of time, knowledge and skills between the entire development team, the search for strong features of students for further professional and personal growth.

СОДЕРЖАНИЕ

	Введение	5
1.	Требования к программе	6
1.1.	Исходные требования к программе	6
2.	План разработки и распределение ролей в бригаде	7
2.1.	План разработки	7
2.2.	Распределение ролей в бригаде	7
3.	Особенности реализации	8
3.1.	Структуры данных	8
3.2.	Основные методы	11
4.	Тестирование	12
4.1	Unit - тестирование	12
4.2	Ручное тестирование	12
	Заключение	15
	Список использованных источников	16
	Приложение А.	17

ВВЕДЕНИЕ

Целью данной работы является разработка графического интерфейса и визуализации алгоритма ЯПД на языке программирования Java, при помощи кооперации группы студентов для решения поставленной задачи. Так же преследовалась цель обновить и закрепить знания по учебным дисциплинам: «Алгоритмы и структуры данных», «Построение и анализ алгоритмов», «Объектно ориентированное программирование», «Дискретная математика» и др.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1.Исходные Требования к программе

1.1.1. Требования к вводу исходных данных

Ввод исходных данных должен осуществляться следующим образом:

1.1.1.1. Реализовать одну ячейку сохранения, представляющую собой текстовый файл со строчным представлением вершин и ребер графа.

1.1.1.2. Реализовать функционал добавления и удаления вершин и ребер графа при помощи взаимодействия пользователя с холстом.

1.1.2. Требование к выводу данных

Результат работы алгоритма является выделенный конструктивно подграф исходного графа, являющийся остовом минимального веса.

1.1.3. Требование к алгоритму

Для нахождения остова дерева должен быть использован алгоритм Ярника-Прима-Дейкстры.

1.1.4. Требования к визуализации

Отображение графа, должна быть показана поэтапная работа алгоритма с выделением остова минимального подграфа исходного графа.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

- 4 июля — разработка прототипа программы (пустой интерфейс).
- 5 июля - создание возможности задания вершин и ребер при помощи специальных кнопок в программе.
- 6 июля - реализация самого алгоритма (без подробной визуализации).
- 7 июля — Сдача 1-ой версии программы. Начало перехода от 1-ой версии программы ко 2-ой, предполагающей наличие возможности добавления вершин и ребер при помощи кликов мыши по холсту
- 8 июля - реализация интерактивной работы алгоритма (каждый шаг алгоритма визуализируется). Создание возможности считывания данных из файла.
- 9 июля - отладка и тестирование программы. Рефакторинг кода. Создание отчета.
- 10 июля — сдача готовой программы.

2.2. Распределение ролей в бригаде

Денис: Реализация ввода-вывода, алгоритма Прима, рефакторинг кода и объединение модулей программы, связь с преподавателем.

Константин: Создание основного GUI, тестирование программы, расширение возможностей GUI/, оформление отчета.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Таблица 1 – Описание структур данных

Структура данных	Описание
EdgeMath	<p>Структура данных EdgeMath , описывающий ребро, содержит информацию о вершинах которые оно соединяет, координаты центра ребра и флаг, указывающий на то соединяет ли ребро две эквивалентных вершины.</p> <pre>private final Vertex from; private final Vertex to; private final int xc; private final int yc; protected final boolean isToEqualsFrom;</pre> <p>Методы:</p> <pre>public EdgeMath(Vertex from, Vertex to) – конструирует по 2-м вершинам. public boolean isIncidental(Vertex v1) – возвращает true если ребро инцидентно переданной вершине. public VertexPair getEndings() – возвращает пару соединенных этим ребром вершин. public EdgeLineRepresentation lineRepresentation() – формирует представление ребра в виде линии. public EdgeOvalRepresentation ovalRepresentation() - формирует представление ребра в виде овала. public Vertex getFrom()- гетер 1-й вершины. public Vertex getTo() – гетер 2-й вершины.</pre>

GraphMath	<p>Класс, описывающий граф, хранит список вершин и ребер</p> <pre>protected HashSet<Vertex> vertexList; protected HashSet<Edge> edgeList;</pre> <p>Методы:</p> <pre>public GraphMath()-создает пустые списки. public GraphMath(GraphMath other)-конструктор копирования. public void addVertex(Vertex vertex)-добавляет вершину. public void addEdge(Edge edge)-добавляет ребро. public void removeEdge(Edge e)-удаляет ребро. public void removeVertex(Vertex vertex)-удаляет вершину. public boolean isConnected()-возвращает true если граф-связный. public boolean isEmpty()-возвращает true если граф пустой. public ArrayList<Edge> getIncidentalEdges(Vertex v)-возвращает список инцидентных данной вершине ребер. public Edge getEdge(Vertex v1, Vertex v2)-возвращает ребро по 2-м вершинам, если между ними нет ребра то – null. public Vertex getContainingVertex(int x, int y)-возвращает вершину в данной точке, если точка пуста - null. public boolean isVertexCoordinatesCorrect(Vertex</pre>
------------------	--

	<p>vertex)-проверяет что данная вершина не накладывается на другие.</p> <p>public boolean contains(char name)-проверяет существует ли вершина с таким именем.</p>
VertexMath	<p>Класс, описывающий вершину, хранит координаты и диаметр вершины</p> <pre>private final int x; private final int y; private final int diameter;</pre> <p>Методы:</p> <pre>public VertexMath(int x, int y, int diameter)- конструктор public int radius()-возвращает радиус public boolean contains(int x, int y)-возвращает true если данная точка содержится внутри вершины public int getY()-гетер y public int getX()-гетер x public boolean touch(VertexMath other)-true если вершина накладывается на переданную.</pre>
Algorithm	<p>Класс, описывающий алгоритм, хранит панель графа, для визуализации своей работы, и линейный список состояний.</p> <pre>private GraphPanel graphPanel; private State state</pre> <p>Методы:</p> <pre>public Algorithm(GraphPanel graphPanel)-конструктор. public void initState()-инициализация списка состояний.</pre>

	<p><code>public void FirstVertex()</code>-выполнение 1-го шага алгоритма.</p> <p><code>public void MinEdge()</code>-выполнение 2-го шага алгоритма.</p> <p><code>public void nextStep()</code>-метод выбора следующего шага.</p> <p><code>public void prevStep()</code>-метод перехода к предыдущему шагу.</p> <p><code>public void restart()</code>-перезапуск алгоритма.</p> <p><code>public void forceAns()</code>-метод получение конечного результата алгоритма.</p> <p><code>public boolean isEnd()</code>-возвращает true если алгоритм завершен.</p> <p><code>public boolean isStart()</code>-возвращает true если алгоритм еще не запущен.</p> <p><code>public Graph getDefaultGraph()</code>-возвращает начальную версию графа.</p> <p><code>public void updateGraph(Graph graph)</code>-загружает новую версию графа.</p>
--	--

3.2. Основные методы

Таблица 2 – Описание основных методов

Объявление метода	Описание
<code>public Algorithm(GraphPanel graphPanel)</code>	Конструктор, который принимает панель, которая изменяется в результате алгоритма.
<code>public void nextStep()</code>	Метод выполняющий определенный шаг алгоритма.
<code>public void prevStep()</code>	Метод, загружающий результат предыдущего шага.
<code>public void restart()</code>	Метод, загружающий начальное состояние алгоритма.
<code>public void forceAns()</code>	Метод получения результата алгоритма.

4. ТЕСТИРОВАНИЕ

4.1 UNIT-тестирование

Написание Unit Test с помощью библиотеки JUnit. Unit тесты были написаны с целью покрыть основные моменты кода алгоритма для того, чтобы убедиться в корректности работы.

Были написаны тесты для всех реализованных структур данных. Для каждого класса были написаны тесты, покрывающие все его методы, в частности для проверки алгоритма были реализованы различные графы, включая пустой, граф без ребер, полный. Так же производилось тестирование отдельных компонент графа таких как вершины и ребра, были проверены функции отвечающие за взаимодействие этих компонент. Проверка осуществлялась с помощью функции `Assertions.assertEquals`.

4.2 Ручное тстирование

Таблица 3 – Тест пустого графа

Входные данные

Алгоритм Прима

Файл

Добавить вершину

Удалить вершину

Добавить ребро

Удалить ребро

Алгоритм Прима - алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже открыт Робертом Примом в 1957 году, и, независимо от них, Дейкстрой в 1959 году.

Алгоритм Прима


Выходные данные		
	<div> <div>Ошибка</div> <div>  <div>Граф не введен!</div> <div>OK</div> </div> </div>	

Таблица 4 – Тест некорректного графа

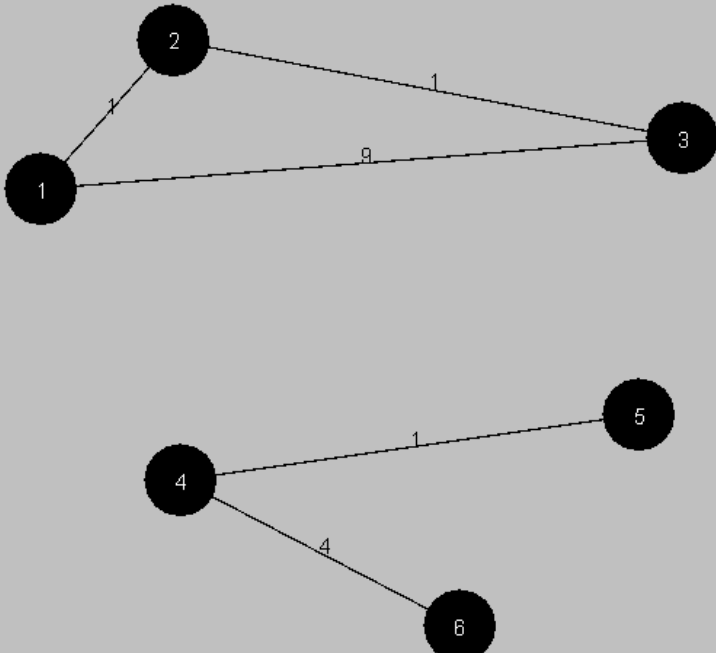

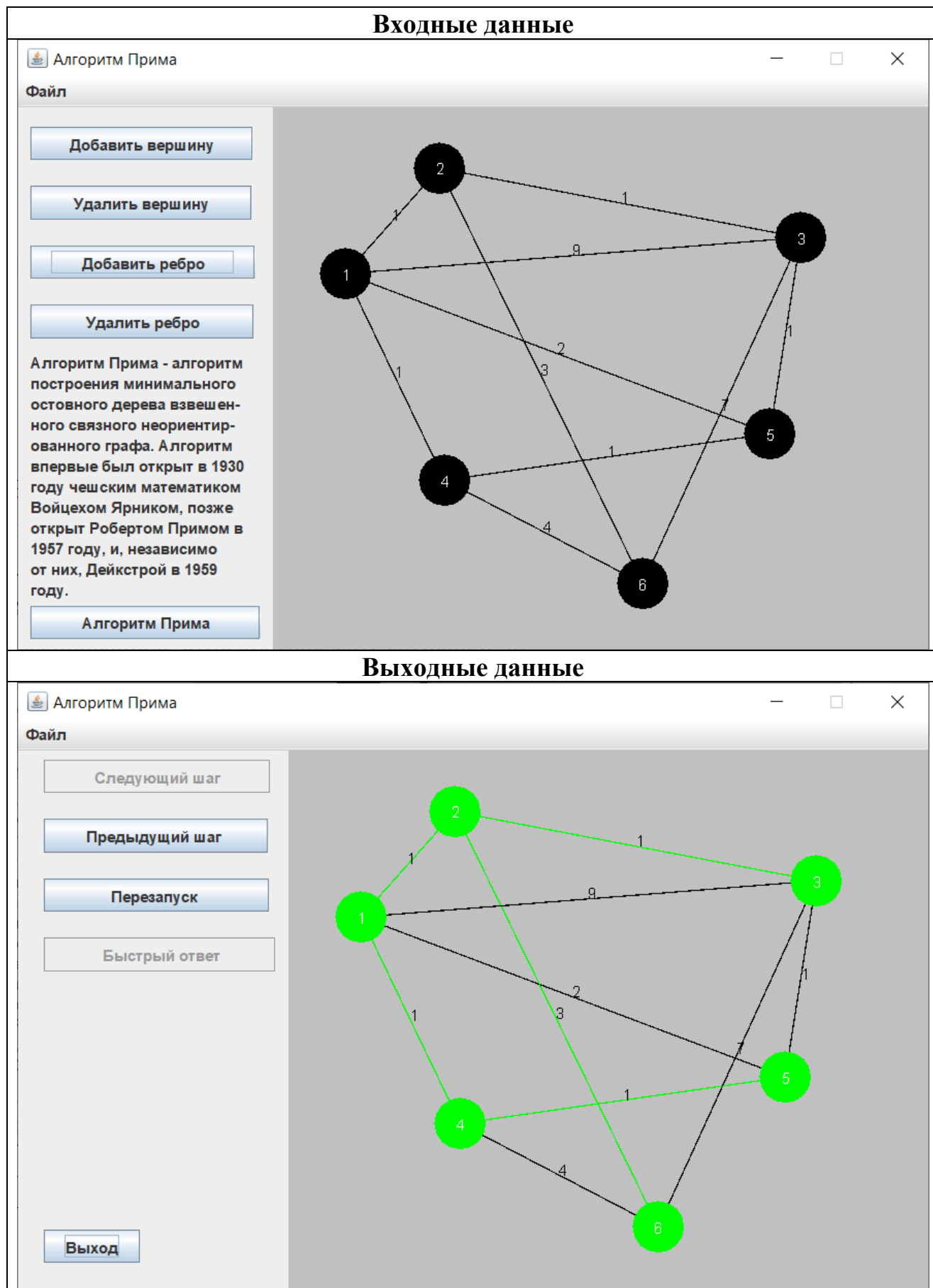
Входные данные		
<div> <div>Алгоритм Прима</div> <div>Файл</div> <div> <div>Добавить вершину</div> <div>Удалить вершину</div> <div>Добавить ребро</div> <div>Удалить ребро</div> </div> <div> <p>Алгоритм Прима - алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже открыт Робертом Примом в 1957 году, и, независимо от них, Дейкстрой в 1959 году.</p> <div>Алгоритм Прима</div> </div> </div>	<div>  </div>	
Выходные данные		
	<div> <div>Ошибка</div> <div>  <div>Граф должен быть связным!</div> <div>OK</div> </div> </div>	

Таблица 5 – Тест корректного графа



ЗАКЛЮЧЕНИЕ

В результате работы был разработан графический интерфейс и визуализация алгоритма ЯПД на языке программирования Java. Так же были закреплены знания по учебным дисциплинам: «Алгоритмы и структуры данных», «Построение и анализ алгоритмов», «Объектно ориентированное программирование», «Дискретная математика» и др.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

<https://javarush.ru/groups/posts/1940-klass-hashmap->
<https://vertex-academy.com/tutorials/ru/filewriter-i-filereader/>
<https://stackoverflow.com/questions/27440701/java-enum-that-registers-as-listener-on-creation>
<https://riptutorial.com/swing/example/498/delay-a-ui-task-for-a-specific-period>
<https://docs.oracle.com/javase/7/docs/api/javax/swing/Timer.html>
<https://javadevblog.com/primer-ispol-zovaniya-java-thread-join.html>
<https://kodejava.org/how-do-i-set-swing-timer-initial-delay/>
https://chortle.ccsu.edu/java5/notes/chap36/ch36_5.html
<http://www.javenue.info/post/36>
<http://java-online.ru/swing-windows.xhtml>
<https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html>
<https://javaswing.wordpress.com/2009/07/19/using-jpanel/>
<https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html>
https://e-maxx.ru/algo/mst_prim
<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>
<https://ru.wikipedia.org/wiki/Model-View-Controller>
<https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>
<https://habr.com/ru/post/181772/>
<https://docs.oracle.com/javase/7/docs/api/java/lang/Class.html>
https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%81%D0%BC%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8%D1%80%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9F%D1%80%D0%B8%D0%BC%D0%B0
<https://docs.oracle.com/javase/7/docs/api/javax/swing/event/AncestorListener.html>
https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%81%D0%BC%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8%D1%80%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9F%D1%80%D0%B8%D0%BC%D0%B0
<https://docs.oracle.com/javase/7/docs/api/java/lang/ClassCastException.html>
<https://brestprog.by/topics/mst/>
https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%81%D0%BC%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8%D1%80%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9F%D1%80%D0%B8%D0%BC%D0%B0
<http://java-online.ru/swing-windows.xhtml>
<https://docs.oracle.com/javase/7/docs/api/javax/swing/JDialog.html>
<https://docs.oracle.com/javase/7/docs/api/java/awt/Window.html>

ПРИЛОЖЕНИЕ А

MAIN.JAVA

```
import GUI.MainWindow;

public class Main {
    public static void main(String[] args) {
        MainWindow window = new MainWindow("Алгоритм Прима");
        window.setVisible(true);
    }
}
```

MAINWINDOW.JAVA

```
package GUI;

import GUI.listeners.GraphPanelMouseListener;
import GUI.listeners.LoadItemListener;
import GUI.listeners.OpenItemListener;
import algorithm.Algorithm;

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

public class MainWindow extends JFrame {

    private GraphPanel graphPanel;
    private final GraphCommandPanel gcp;
    private final AlgorithmCommandPanel acp;
    private final JFrame window;

    private JPanel MainWindowPanel;
    private JMenuBar menuBar;
    private Algorithm algorithm;
    private JMenuItem openItem;

    public MainWindow(String title) {
        super(title);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(720, 480);
        setLocationRelativeTo(null);
        setResizable(false);

        window = this;

        gcp = new GraphCommandPanel();

        initMainWindowPanel();
        initAlgorithm();
        acp = new AlgorithmCommandPanel(algorithm);
        initExitAlgorithmButton();

        initMenuBar();
        setJMenuBar(menuBar);

        setContentPane(MainWindowPanel);
    }

    private void initExitAlgorithmButton()
```

```

{
    JButton exit = new JButton("Выход");
    exit.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            graphPanel.setGraph(algorithm.getDefaultGraph());
            MainWindowPanel.remove(acp);
            MainWindowPanel.add(gcp, 0);
            openItem.setEnabled(true);
        }
    });

    acp.add(exit);
    acp.add(Box.createRigidArea(new Dimension(0, 20)));
}

private void initMenuBar()
{
    menuBar = new JMenuBar();
    JMenu menu = new JMenu("Файл");
    openItem = new JMenuItem("Загрузить граф");
    JMenuItem loadItem = new JMenuItem("Сохранить граф");

    openItem.addActionListener(new OpenItemListener(graphPanel, this));
    loadItem.addActionListener(new LoadItemListener(graphPanel, this));

    menu.add(openItem);
    menu.add(loadItem);

    menuBar.add(menu);
}

private void initAlgorithm()
{
    algorithm = new Algorithm(graphPanel);

    JButton Algorithm = new JButton("Алгоритм Прима");
    Algorithm.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            if (graphPanel.getGraph().isEmpty())
            {
                JOptionPane.showMessageDialog(window, "Граф не введен!",
"Ошибка", JOptionPane.WARNING_MESSAGE);
                return;
            }

            if (!graphPanel.getGraph().isConnected())
            {
                JOptionPane.showMessageDialog(window, "Граф должен быть
связным!", "Ошибка", JOptionPane.ERROR_MESSAGE);
                return;
            }

            MainWindowPanel.remove(gcp);
            MainWindowPanel.add(acp, 0);
            acp.buttonEnabler();
            algorithm.updateGraph(graphPanel.getGraph());
            openItem.setEnabled(false);
        }
    });
}

```

```

        gcp.add(Box.createRigidArea(new Dimension(0,5)));
        gcp.add(Algorithm);
        gcp.add(Box.createRigidArea(new Dimension(0,10)));

    }

    private void initMainWindowPanel()
    {
        MainWindowPanel = new JPanel();
        graphPanel = new GraphPanel(MainWindowPanel, this);
        graphPanel.setBackground(Color.LIGHT_GRAY);

        graphPanel.addMouseListener(new GraphPanelMouseListener(graphPanel, gcp,
this));

        MainWindowPanel.setLayout(new BoxLayout(MainWindowPanel,
BoxLayout.X_AXIS));

        MainWindowPanel.add(gcp);
        MainWindowPanel.add(graphPanel);
    }
}

```

GRAPHPANEL.JAVA

```

package GUI;

import graph.Edge;
import graph.Graph;
import graph.Vertex;

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

public class GraphPanel extends JPanel {

    private final JPanel MainWindowPanel;
    private final MainWindow window;
    private Graph graph;

    public GraphPanel(JPanel MainWindowPanel, MainWindow window) {
        this.MainWindowPanel = MainWindowPanel;
        this.window = window;

        graph = new Graph();
    }

    public void add(Vertex vertex)
    {
        graph.addVertex(vertex);
        MainWindowPanel.repaint();
    }

    public void add(Edge edge)
    {
        Edge e = graph.getEdge(edge.getFrom(), edge.getTo());
        if (e != null) {
            JOptionPane.showMessageDialog(window, "Ребро между выбранными
вершинами уже существует!", "Ошибка", JOptionPane.ERROR_MESSAGE);
            return;
        }
        graph.addEdge(edge);
    }
}

```

```

        MainWindowPanel.repaint();
    }

    public void remove(Vertex v1, Vertex v2)
    {
        Edge e = graph.getEdge(v1, v2);
        if (e == null)
        {
            JOptionPane.showMessageDialog(window, "Ребра между выбранными
вершинами не существует!", "Ошибка", JOptionPane.ERROR_MESSAGE);
            return;
        }

        graph.removeEdge(e);
        MainWindowPanel.repaint();
    }

    public void remove(Edge edge)
    {
        graph.removeEdge(edge);
        MainWindowPanel.repaint();
    }

    public void remove(Vertex vertex)
    {
        graph.removeVertex(vertex);
        MainWindowPanel.repaint();
    }

    @Override
    public void paint(Graphics grph) {
        super.paint(grph);
        graph.draw(grph);
        MainWindowPanel.repaint();
    }

    public Vertex getContainingVertex(int x, int y)
    {
        return graph.getContainingVertex(x, y);
    }

    public void setGraph(Graph graph) {
        this.graph = new Graph(graph);
    }

    public Graph getGraph() {
        return graph;
    }
}

```

GRAPHCOMMANDPANEL.JAVA

```

package GUI;

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

public class GraphCommandPanel extends JPanel {

    private ButtonState buttonState;

```

```

private JButton addVertex;
private JButton removeVertex;
private JButton addEdge;
private JButton removeEdge;

public GraphCommandPanel()
{
    setBorder(BorderFactory.createEmptyBorder(0,10,0,10));
    setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));

    initButtons();
    addButtons();
    addLabels();
}

private void addLabels()
{
    String str = "Алгоритм Прима - алгоритм построения минимального
остовного дерева взвешенного связного неориентированного графа. Алгоритм
впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже
открыт Робертом Примом в 1957 году, и, независимо от них, Дейкстрой в 1959 году.
";
    for (int i = 0 ; i < str.length() - 25; i += 25) add(new
JLabel(str.substring(i, i + 25)));
}

private void addButtons()
{
    add(Box.createRigidArea(new Dimension(0,15)));
    add(addVertex);
    add(Box.createRigidArea(new Dimension(0,20)));
    add(removeVertex);
    add(Box.createRigidArea(new Dimension(0,20)));
    add(addEdge);
    add(Box.createRigidArea(new Dimension(0,20)));
    add(removeEdge);
    add(Box.createRigidArea(new Dimension(0,10)));
}

private void initButtons()
{
    buttonState = ButtonState.noButton;

    addVertex = new JButton(    "    Добавить вершину    ");
    removeVertex = new JButton("    Удалить вершину    ");
    addEdge = new JButton(    "    Добавить ребро    ");
    removeEdge = new JButton(    "    Удалить ребро    ");

    addVertex.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            buttonState = ButtonState.addVertex;
        }
    });
    removeVertex.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            buttonState = ButtonState.removeVertex;
        }
    });
    removeEdge.addActionListener(new ActionListener() {
        @Override

```

```

        public void actionPerformed(ActionEvent e) {
            buttonState = ButtonState.removeEdge;
        }
    });
    addEdge.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            buttonState = ButtonState.addEdge;
        }
    });
}

public ButtonState ButtonState()
{
    return buttonState;
}

public void setButtonState(ButtonState buttonState)
{
    this.buttonState = buttonState;
}
}

```

BUTTONSTATE.JAVA

```

package GUI;

public enum ButtonState{
    addVertex,
    removeVertex,
    removeEdge,
    addEdge,
    noButton
}

```

ALGORITHMCOMMANDPANEL.JAVA

```

package GUI;

import GUI.MainWindow;
import algorithm.Algorithm;

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

public class AlgorithmCommandPanel extends JPanel{

    private JButton NextStep;
    private JButton PrevStep;
    private JButton ForceAns;
    private JButton Restart;
    private JButton Exit;

    private Algorithm algorithm;

    public AlgorithmCommandPanel(Algorithm algorithm)
    {
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
    }
}

```

```

        setBorder(BorderFactory.createEmptyBorder(0,20,0,10));

        this.algorithm = algorithm;
        initButtons();
        addButtons();
    }

    private void addButtons()
    {
        add(Box.createRigidArea(new Dimension(0,5)));
        add(NextStep);
        add(Box.createRigidArea(new Dimension(0,20)));
        add(PrevStep);
        add(Box.createRigidArea(new Dimension(0,20)));
        add(Restart);
        add(Box.createRigidArea(new Dimension(0,20)));
        add(ForceAns);
        add(Box.createRigidArea(new Dimension(0,200)));
    }

    private void initButtons()
    {
        NextStep = new JButton("                Старт                "); // "
        Следующий шаг
        PrevStep = new JButton("                Предыдущий шаг                ");
        Restart = new JButton("                Перезапуск                ");
        ForceAns = new JButton("                Быстрый ответ                ");
        buttonEnabler();

        NextStep.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                algorithm.nextStep();
                buttonEnabler();
            }
        });

        PrevStep.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                algorithm.prevStep();
                buttonEnabler();
            }
        });

        Restart.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                algorithm.restart();
                buttonEnabler();
            }
        });

        ForceAns.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                algorithm.forceAns();
                buttonEnabler();
            }
        });
    }

    public void buttonEnabler()

```

```

    {
        NextStep.setText( "          Следующий шаг          ");
        NextStep.setEnabled(true);
        PrevStep.setEnabled(true);
        ForceAns.setEnabled(true);
        Restart.setEnabled(true);

        if (algorithm.isStart()) {
            NextStep.setText("          Старт          ");
            PrevStep.setEnabled(false);
            Restart.setEnabled(false);
        }

        if (algorithm.isEnd())
        {
            NextStep.setEnabled(false);
            ForceAns.setEnabled(false);
        }
    }
}

```

GRAPHPANELMOUSELISTENER.JAVA

```

package GUI.listeners;

import graph.io.FileGraph;
import GUI.GraphPanel;
import graph.io.IOResult;
import GUI.MainWindow;

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

public class OpenItemListener implements ActionListener {

    private GraphPanel graphPanel;
    private MainWindow window;

    public OpenItemListener(GraphPanel graphPanel, MainWindow window)
    {
        this.graphPanel = graphPanel;
        this.window = window;
    }

    public void actionPerformed(ActionEvent e) {
        FileGraph fw = new FileGraph();
        IOResult result = fw.loadGraph();

        if (result == IOResult.loadDataError || result == IOResult.loadError)
        {
            JOptionPane.showMessageDialog(window, "Ошибка при загрузке!",
"Ошибка", JOptionPane.ERROR_MESSAGE);
            return;
        }

        graphPanel.setGraph(fw.getGraph());
        graphPanel.repaint();
        JOptionPane.showMessageDialog(window, "Загрузка графа прошла успешно!",
"Успех", JOptionPane.INFORMATION_MESSAGE);
    }
}

```



```

    }
}

```

LOADITEMLISTENER.JAVA

```

package GUI.listeners;

import graph.io.FileGraph;
import GUI.GraphPanel;
import graph.io.IOResult;
import GUI.MainWindow;

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

public class LoadItemListener implements ActionListener {

    private GraphPanel graphPanel;
    private MainWindow window;

    public LoadItemListener(GraphPanel graphPanel, MainWindow window)
    {
        this.graphPanel = graphPanel;
        this.window = window;
    }

    public void actionPerformed(ActionEvent e) {
        FileGraph fw = new FileGraph();
        IOResult result = fw.writeGraph(graphPanel.getGraph());

        if (result == IOResult.writeError)
            JOptionPane.showMessageDialog(window, "Ошибка при сохранении!",
"Ошибка", JOptionPane.ERROR_MESSAGE);
        else
            JOptionPane.showMessageDialog(window, "Граф сохранен успешно!",
"Успех", JOptionPane.INFORMATION_MESSAGE);
    }
}

```

OPENITEMLISTENER.JAVA

```

package GUI.listeners;

import GUI.ButtonState;
import GUI.GraphCommandPanel;
import GUI.GraphPanel;
import GUI.MainWindow;
import GUI.dialog.NameDialog;
import GUI.dialog.WeightDialog;
import graph.Edge;
import graph.Vertex;

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

public class GraphPanelMouseListener extends MouseAdapter{
    private final GraphPanel graphPanel;

```

```

private final GraphCommandPanel gcp;
private final MainWindow window;
private Vertex VertexSave = null;

public GraphPanelMouseListener(GraphPanel graphPanel, GraphCommandPanel gcp,
MainWindow window){
    this.graphPanel = graphPanel;
    this.gcp = gcp;
    this.window = window;
}

@Override
public void mousePressed(MouseEvent e) {
    switch (gcp.ButtonState())
    {
        case addVertex:
            addVertexCase(e);
            break;
        case addEdge:
            addEdgeCase(e);
            break;
        case removeEdge:
            removeEdgeCase(e);
            break;
        case removeVertex:
            removeVertexCase(e);
            break;
    }
}

private void removeVertexCase(MouseEvent e)
{
    Vertex removeVertex = graphPanel.getContainingVertex(e.getX(),
e.getY());
    if (removeVertex != null)
    {
        graphPanel.remove(removeVertex);
        gcp.setButtonState(ButtonState.noButton);
    }
}

private void removeEdgeCase(MouseEvent e)
{
    Vertex removeEdgeResult = graphPanel.getContainingVertex(e.getX(),
e.getY());
    if (removeEdgeResult != null)
    {
        if (VertexSave != null) {
            graphPanel.remove(VertexSave, removeEdgeResult);
            gcp.setButtonState(ButtonState.noButton);
            VertexSave = null;
        }
        else
            VertexSave = removeEdgeResult;
    }
}

private void addEdgeCase(MouseEvent e)
{
    Vertex addEdgeResult = graphPanel.getContainingVertex(e.getX(),
e.getY());
    if (addEdgeResult != null)
    {

```

```

        if (VertexSave != null) {

            WeightDialog dialog = new WeightDialog(window);
            dialog.setVisible(true);

            boolean dialogResult = dialog.isOk();
            if (!dialogResult)
            {
                gcp.setButtonState(ButtonState.noButton);
                VertexSave = null;
                return;
            }

            graphPanel.add(new Edge(VertexSave, addEdgeResult, Color.BLACK,
dialog.getWeight()));
            gcp.setButtonState(ButtonState.noButton);
            VertexSave = null;
        }
        else
            VertexSave = addEdgeResult;
    }
}

private void addVertexCase(MouseEvent e)
{
    if (!graphPanel.getGraph().isVertexCoordinatesCorrect(new
Vertex(e.getX(), e.getY(), 40, 'a', Color.BLACK)))
        return;

    NameDialog dialog = new NameDialog(window);
    dialog.setVisible(true);

    boolean dialogResult = dialog.isOk();
    if (dialogResult) {
        char name = dialog.getVertexName();
        if (graphPanel.getGraph().contains(name))
        {
            JOptionPane.showMessageDialog(window, "Вершина с данным именем
уже существует", "Ошибка", JOptionPane.ERROR_MESSAGE);
            gcp.setButtonState(ButtonState.noButton);
            return;
        }

        graphPanel.add(new Vertex(e.getX() - 20, e.getY() - 20, 40, name,
Color.BLACK));
        gcp.setButtonState(ButtonState.noButton);
    }
}
}

```

WEIGHTDIALOG.JAVA

```

package GUI.dialog;

import GUI.MainWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;

```

```

import java.awt.event.WindowEvent;

public class WeightDialog extends DialogBase {
    private int weight;

    public WeightDialog(MainWindow window)
    {
        super(window, "Вес ребра");
    }

    void initTextArea() {
        area = new JTextField(4);
        area.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                loadWeight();
                setVisible(false);
            }
        });
    }

    void initLabel() {
        label = new JLabel("Введите вес ребра: ");
    }

    public void loadWeight() {
        String result = area.getText();

        try
        {
            weight = Integer.parseInt(result);
            if (weight <= 0)
                throw new NumberFormatException();
        }
        catch (NumberFormatException e)
        {
            ok = false;
        }
    }

    public int getWeight()
    {
        return weight;
    }
}

```

NAMEDIALOG.JAVA

```

package GUI.dialog;

import GUI.MainWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class NameDialog extends DialogBase
{

```

```

private char name;

public NameDialog(MainWindow window)
{
    super(window, "Имя вершины");
}

void initTextArea() {
    area = new JTextField(1);
    area.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            loadName();
            setVisible(false);
        }
    });
}

void initLabel() {
    label = new JLabel("Введите имя вершины: ");
}

private void loadName() {
    name = area.getText().toCharArray()[0];
}

public char getVertexName() {
    return name;
}
}

```

DIALOGBASE.JAVA

```

package GUI.dialog;

import GUI.MainWindow;

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

public abstract class DialogBase extends JDialog{
    protected JTextField area;
    protected JLabel label;

    protected boolean ok = true;

    public DialogBase(MainWindow window, String title)
    {
        super(window, title, true);
        setResizable(false);
        setLocationRelativeTo(null);
        setSize(200, 80);
        setLayout(new FlowLayout());

        addWindowListener(new WindowAdapter() {

            @Override
            public void windowClosing(WindowEvent e) {
                ok = false;
                super.windowClosing(e);
            }
        });
    }
}

```

```

        }
    });

    initLabel();
    initTextArea();

    add(label);
    add(area);
}

abstract void initTextArea();
abstract void initLabel();

public boolean isOk()
{
    return ok;
}
}

```

VERTEX.JAVA

```

package graph;

import graph.math.VertexMath;

import java.awt.*;

public class Vertex extends VertexMath
{
    private Color color;
    private char[] name;

    public Vertex(int x, int y, int diameter, char name, Color color)
    {
        super(x, y, diameter);
        this.color = color;
        this.name = new char[]{name};
    }

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

    public void draw(Graphics g)
    {
        g.setColor(color);
        g.fillOval(getX(), getY(), radius() * 2, radius() * 2);
        g.setColor(Color.WHITE);
        g.drawChars(name, 0, 1, getX() + 17, getY() + 25);
    }

    public char getName() {
        return name[0];
    }

    public Color getColor() {
        return color;
    }
}

```

GRAPH.JAVA

```
package graph;

import graph.math.GraphMath;

import java.awt.*;

public class Graph extends GraphMath {

    public Graph()
    {
        super();
    }

    public Graph(Graph other)
    {
        super(other);
    }

    public void draw(Graphics grph) {
        for (Edge e : edgeList)
            e.draw(grph);

        for (Vertex v : vertexList)
            v.draw(grph);
    }

}
```

EDGE.JAVA

```
package graph;

import graph.math.EdgeMath;
import graph.util.EdgeLineRepresentation;
import graph.util.EdgeOvalRepresentation;

import java.awt.*;

public class Edge extends EdgeMath {
    private Color color;
    private final int weight;

    public Edge(Vertex from, Vertex to, Color color, int weight) {
        super(from, to);

        this.color = color;
        this.weight = weight;
    }

    public void draw(Graphics g)
    {
        g.setColor(color);
        String strWeight = String.valueOf(weight);

        if (!isToEqualsFrom) {
            EdgeLineRepresentation edgeLineRep = lineRepresentation();

            g.drawLine(edgeLineRep.getStartPoint().getX(),
edgeLineRep.getStartPoint().getY(),
```

```

        edgeLineRep.getEndPoint().getX(),
edgeLineRep.getEndPoint().getY());

        g.setColor(Color.BLACK);
        g.drawChars(strWeight.toCharArray(), 0, strWeight.length(),
edgeLineRep.getTextCoordinates().getX(),
        edgeLineRep.getTextCoordinates().getY());
    }
    else
    {
        EdgeOvalRepresentation edgeOvalRepr = ovalRepresentation();

        g.drawOval(edgeOvalRepr.getOvalRepr().getX(),
edgeOvalRepr.getOvalRepr().getY(),
        edgeOvalRepr.getOvalRepr().getWidth(),
edgeOvalRepr.getOvalRepr().getHeight());

        g.drawChars(strWeight.toCharArray(), 0, strWeight.length(),
edgeOvalRepr.getTextCoordinates().getX(),
        edgeOvalRepr.getTextCoordinates().getY());

    }
}

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

public int getWeight() {
    return weight;
}

public Color getColor() {
    return color;
}
}

```

VERTEXPAIR.JAVA

```

package graph.util;

import graph.Vertex;

public class VertexPair
{
    public final Vertex from;
    public final Vertex to;

    public VertexPair(Vertex from, Vertex to) {
        this.from = from;
        this.to = to;
    }
}

```

POINT.JAVA

```

package graph.util;

public class Point
{

```



```

private final int x;
private final int y;

public Point(int x, int y)
{
    this.x = x;
    this.y = y;
}

public int getX() {
    return x;
}

public int getY() {
    return y;
}
}

```

OVALREPRESENTATION.JAVA

```

package graph.util;

public class OvalRepresentation
{
    private final int x;
    private final int y;
    private final int height;
    private final int width;

    public OvalRepresentation(int x, int y, int height, int width) {
        this.x = x;
        this.y = y;
        this.height = height;
        this.width = width;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getHeight() {
        return height;
    }

    public int getWidth() {
        return width;
    }
}

```

EDGE OVALREPRESENTATION.JAVA

```

package graph.util;

public class EdgeOvalRepresentation
{
    private final OvalRepresentation ovalRepr;

```

```

        private final Point textCoordinates;

        public EdgeOvalRepresentation(OvalRepresentation ovalRepr, Point
textCoordinates) {
            this.ovalRepr = ovalRepr;
            this.textCoordinates = textCoordinates;
        }

        public OvalRepresentation getOvalRepr() {
            return ovalRepr;
        }

        public Point getTextCoordinates() {
            return textCoordinates;
        }
    }
}

```

EDGELINEREPRESENTATION.JAVA

```

package graph.util;

public class EdgeLineRepresentation
{
    private final Point startPoint;
    private final Point endPoint;

    private final Point textCoordinates;

    public EdgeLineRepresentation(Point startLine, Point endLine, Point
textCoordinates) {
        this.startPoint = startLine;
        this.endPoint = endLine;
        this.textCoordinates = textCoordinates;
    }

    public Point getStartPoint() {
        return startPoint;
    }

    public Point getEndPoint() {
        return endPoint;
    }

    public Point getTextCoordinates() {
        return textCoordinates;
    }
}

```

VERTEXMATH.JAVA

```

package graph.math;

public class VertexMath
{
    private final int x;
    private final int y;
    private final int diameter;

    public VertexMath(int x, int y, int diameter) {

```

```

        this.x = x;
        this.y = y;
        this.diameter = diameter;
    }

    public int radius()
    {
        return diameter / 2;
    }

    public boolean contains(int x, int y)
    {
        return Math.pow(x - (20 + this.x), 2) + Math.pow(y - (20 + this.y), 2)
<= Math.pow(diameter, 2) / 4;
    }

    public int getY() {
        return y;
    }

    public int getX() {
        return x;
    }

    public boolean touch(VertexMath other)
    {
        double distance = Math.sqrt(Math.pow(this.x - other.x, 2) +
Math.pow(this.y - other.y, 2));

        return distance <= radius() * 2 + other.radius();
    }
}

```

EDGEMATH.JAVA

```

package graph.math;

import graph.Edge;
import graph.Vertex;
import graph.util.VertexPair;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;

public class GraphMath {
    protected HashSet<Vertex> vertexList;
    protected HashSet<Edge> edgeList;

    public GraphMath()
    {
        vertexList = new HashSet<Vertex>();
        edgeList = new HashSet<Edge>();
    }

    public GraphMath(GraphMath other)
    {
        vertexList = new HashSet<Vertex>();
        edgeList = new HashSet<Edge>();

        HashMap<Vertex, Vertex> d = new HashMap<Vertex,

```

```

Vertex>(other.vertexList.size());

    for (Vertex v : other.vertexList)
    {
        Vertex copyV = new Vertex(v.getX(), v.getY(), v.radius() * 2,
v.getName(), v.getColor());
        this.addVertex(copyV);
        d.put(v, copyV);
    }

    for (Edge e : other.edgeList)
    {
        VertexPair vertexPair = e.getEndings();
        this.addEdge(new Edge(d.get(vertexPair.from), d.get(vertexPair.to),
e.getColor(), e.getWeight()));
    }

    int lox = 0;
}

public void addVertex(Vertex vertex)
{
    vertexList.add(vertex);
}

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

public void removeEdge(Edge e) {
    edgeList.remove(e);
}

public void removeVertex(Vertex vertex)
{
    ArrayList<Edge> result = getIncidentalEdges(vertex);
    for (Edge edge : result)
        edgeList.remove(edge);

    vertexList.remove(vertex);
}

public boolean isConnected() {
    if (vertexList.size() == 1)
        return true;

    Vertex v = (Vertex) (vertexList.toArray()[0]);
    HashSet<Vertex> result = new HashSet<>();
    getAchievableVertices(v, result);

    return vertexList.size() == result.size();

    /*while (seenVertex.size() != vertexList.size())
    {
        HashSet<Vertex> save = new HashSet<Vertex>(seenVertex);
        for (Vertex cur : seenVertex)
        {
            ArrayList<Edge> tmpEdgeList = getIncidentalEdges(cur);

            for (Edge e : tmpEdgeList) {
                VertexPair curEdgeEndings = e.getEndings();
                Vertex curNewVertex = curEdgeEndings.from;
                if (v == curNewVertex)

```

```

        curNewVertex = curEdgeEndings.to;
        save.add(curNewVertex);
    }
}

if (save.size() == seenVertex.size())
    return false;

seenVertex = save;

}*/

}

private void getAchievableVertices(Vertex v, HashSet<Vertex> result)
{
    ArrayList<Edge> edges = getIncidentalEdges(v);

    for (Edge e : edges)
    {
        VertexPair vertexPair = e.getEndings();
        Vertex newVertex = vertexPair.from;
        if (vertexPair.from == v)
            newVertex = vertexPair.to;
        if (!result.contains(newVertex)) {
            result.add(newVertex);
            getAchievableVertices(newVertex, result);
        }
    }
}

}

public boolean isEmpty()
{
    return vertexList.isEmpty();
}

public ArrayList<Edge> getIncidentalEdges(Vertex v) {
    ArrayList<Edge> result = new ArrayList<>();
    for (Edge e : edgeList) {
        if (e.isIncidental(v))
            result.add(e);
    }

    return result;
}

public Edge getEdge(Vertex v1, Vertex v2) {
    for (Edge e : edgeList) {
        if ((e.isIncidental(v1) && e.isIncidental(v2) && v1 != v2) ||
            (e.isToEqualsFrom && e.isIncidental(v1) && v1 == v2))
            return e;
    }

    return null;
}

public Vertex getContainingVertex(int x, int y) {
    for (Vertex v : vertexList) {
        if (v.contains(x, y))

```

```

        return v;
    }

    return null;
}

public boolean isVertexCoordinatesCorrect(Vertex vertex)
{
    for (Vertex cur : vertexList)
        if (cur.touch(vertex))
            return false;

    return true;
}

public HashSet<Vertex> getVertexList() {
    return vertexList;
}

public HashSet<Edge> getEdgeList() {
    return edgeList;
}

public boolean contains(char name) {
    for (Vertex v : vertexList)
        if (v.getName() == name)
            return true;

    return false;
}
}

```

GRAPHMATH.JAVA

```

package graph.math;

import graph.Vertex;
import graph.util.*;

public class EdgeMath {
    private final Vertex from;
    private final Vertex to;

    private final int xc;
    private final int yc;

    protected final boolean isToEqualsFrom;

    public EdgeMath(Vertex from, Vertex to) {
        this.from = from;
        this.to = to;

        xc = (from.getX() + to.getX() + 40) / 2;
        yc = (from.getY() + to.getY() + 40) / 2;

        isToEqualsFrom = to.equals(from);
    }

    public boolean isIncidental(Vertex v1) {

```

```

        return from == v1 || to == v1;
    }

    public VertexPair getEndings()
    {
        return new VertexPair(from, to);
    }

    public EdgeLineRepresentation lineRepresentation()
    {
        return new EdgeLineRepresentation(new Point(from.getX() + 20,
from.getY() + 20),
new Point(to.getX() + 20, to.getY() +
20),
new Point(xc, yc));
    }

    public EdgeOvalRepresentation ovalRepresentation()
    {
        int ovalX = 0;
        int ovalY = from.getY();
        int ovalWidth = 70;
        int ovalHeight = 30;

        if (from.getX() + from.radius() + ovalWidth / 2 >= 480)
            ovalX = from.getX() - from.radius() - ovalWidth / 2;
        else
            ovalX = from.getX() + from.radius();

        OvalRepresentation ovalRepr = new OvalRepresentation(ovalX, ovalY,
ovalHeight, ovalWidth);

        int distanceBetweenLetterAndOval = 3;
        int letterHeight = 5;

        int textX = ovalX + ovalWidth / 2;
        int textY = 0;
        if (from.getY() - distanceBetweenLetterAndOval <= letterHeight)
            textY = from.getY() + from.radius() * 2 +
distanceBetweenLetterAndOval;
        else
            textY = from.getY() - distanceBetweenLetterAndOval;

        Point textCoordinates = new Point(textX, textY);

        return new EdgeOvalRepresentation(ovalRepr, textCoordinates);
    }

    public Vertex getFrom() {
        return from;
    }

    public Vertex getTo() {
        return to;
    }
}

```

IORESULT.JAVA

```

package graph.io;

public enum IOResult

```

```

{
    loadError,
    loadSuccess,
    writeError,
    loadDataError,
    writeSuccess
}

```

FILEGRAPH.JAVA

```

package graph.io;

import graph.Edge;
import graph.Graph;
import graph.Vertex;

import java.awt.*;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Scanner;

public class FileGraph {

    Graph graph = new Graph();

    public IOResult loadGraph(){

        String vertexDataPattern = ".x\\d{3}y\\d{3}";
        String edgeDataPattern = "w\\d+f.t.";
        HashMap<Character, Vertex> d = new HashMap<Character, Vertex>();

        FileReader fileReader;
        try {
            fileReader = new FileReader("save.txt");
        }
        catch (IOException e) {
            return IOResult.loadError;
        }

        Scanner scanner = new Scanner(fileReader);
        Graph graph = new Graph();
        while (scanner.hasNextLine())
        {
            String cur = scanner.nextLine();
            if (cur.equals("."))
                break;

            if (cur.matches(vertexDataPattern))
            {
                try {
                    Vertex vertex = new Vertex(Integer.parseInt(cur.substring(2,
5)),
                    Integer.parseInt(cur.substring(6,
9)), 40, cur.toCharArray()[0], Color.BLACK);
                    d.put(cur.toCharArray()[0], vertex);
                    graph.addVertex(vertex);
                }
                catch (NumberFormatException e)
                {

```



```

        return IOResult.loadDataError;
    }
}
else {
    return IOResult.loadDataError;
}
}

while (scanner.hasNextLine())
{
    String cur = scanner.nextLine();

    if (cur.matches(edgeDataPattern))
    {
        try {
            char nameFrom = cur.toCharArray()[cur.length() - 3];
            char nameTo = cur.toCharArray()[cur.length() - 1];

            if (!d.containsKey(nameFrom) || !d.containsKey(nameTo))
                return IOResult.loadDataError;

            graph.addEdge(new Edge(d.get(nameFrom), d.get(nameTo),
Color.BLACK, Integer.parseInt(cur.substring(1, cur.length() - 4))));
        }
        catch (NumberFormatException e)
        {
            return IOResult.loadDataError;
        }
    }
}

this.graph = graph;
return IOResult.loadSuccess;
}

public IOResult writeGraph(Graph graph) {
    FileWriter wr;
    try {
        wr = new FileWriter("save.txt");
    }
    catch (IOException e) {
        return IOResult.writeError;
    }

    HashSet<Vertex> vertices = graph.getVertexList();
    StringBuilder graphInfo = new StringBuilder();
    for (Vertex v : vertices) {
        StringBuilder vertexInfo = new StringBuilder();
        String xCoordinate = coordinateConverter(v.getX());
        String yCoordinate = coordinateConverter(v.getY());

        vertexInfo.append(v.getName());
        vertexInfo.append("x");
        vertexInfo.append(xCoordinate);
        vertexInfo.append("y");
        vertexInfo.append(yCoordinate);
        vertexInfo.append("\n");

        graphInfo.append(vertexInfo);
    }
    graphInfo.append(".\n");

    HashSet<Edge> edges = graph.getEdgeList();

```

```

    for (Edge e : edges)
    {
        StringBuilder edgeInfo = new StringBuilder();

        edgeInfo.append("w");
        edgeInfo.append(e.getWeight());
        edgeInfo.append("f");
        edgeInfo.append(e.getFrom().getName());
        edgeInfo.append("t");
        edgeInfo.append(e.getTo().getName());
        edgeInfo.append("\n");

        graphInfo.append(edgeInfo);
    }

    try {
        wr.write(graphInfo.toString());
        wr.close();
    }
    catch (IOException e)
    {
        return IOResult.writeError;
    }

    return IOResult.writeSuccess;
}

private String coordinateConverter(int n) {
    String nStringRepresentation = String.valueOf(n);
    StringBuilder result = new StringBuilder();

    int zerosCnt = 3 - nStringRepresentation.length();
    while (zerosCnt != 0)
    {
        result.append(0);
        --zerosCnt;
    }

    result.append(nStringRepresentation);

    return result.toString();
}

public Graph getGraph() {
    return graph;
}
}

```

STATE.JAVA

```

package algorithm;

import graph.Graph;

public class State {
    final State prev;
    final AlgorithmState next;
    final Graph graph;

    public State()
    {

```

```

        prev = null;
        graph = null;
        next = null;
    }

    public State(State prev, Graph graph, AlgorithmState next)
    {
        this.prev = prev;
        this.graph = graph;
        this.next = next;
    }

    public Graph getGraph() {
        return graph;
    }
}

```

ALGORITHMSTATE.JAVA

```

package algorithm;

public enum AlgorithmState
{
    FirstVertex,
    MinEdge,
    End
}

```

ALGORITHM.JAVA

```

package algorithm;

import GUI.GraphPanel;
import GUI.MainWindow;
import graph.Edge;
import graph.Graph;
import graph.Vertex;
import graph.util.VertexPair;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Random;

public class Algorithm {
    private GraphPanel graphPanel;
    private Graph graph;

    private State state = new State(null, null, AlgorithmState.FirstVertex);

    public Algorithm(GraphPanel graphPanel) {
        this.graphPanel = graphPanel;
        this.graph = graphPanel.getGraph();
    }

    public void initState()
    {

```

```

        state = new State(null, new Graph(graph), AlgorithmState.FirstVertex);
    }

    public void FirstVertex() {
        state = new State(state, new Graph(state.graph),
AlgorithmState.MinEdge);

        Random random = new Random();

        int startVertexInd = random.nextInt(state.graph.getVertexList().size());
        int i = 0;
        Vertex startVertex = null;
        for (Vertex v : state.graph.getVertexList())
        {
            if (i == startVertexInd)
            {
                startVertex = v;
                break;
            }
            ++i;
        }
        startVertex.setColor(Color.GREEN);

        if (state.graph.getVertexList().size() == 1)
            state = new State(state.prev, new Graph(state.graph),
AlgorithmState.End);
    }

    public void MinEdge() {
        state = new State(state, new Graph(state.graph),
AlgorithmState.MinEdge);

        ArrayList<Vertex> curVertexList = new ArrayList<Vertex>();
        for (Vertex v : state.graph.getVertexList())
            if (v.getColor() == Color.GREEN)
                curVertexList.add(v);

        HashSet<Edge> curTreeEdge = new HashSet<Edge>();

        for (Vertex v : curVertexList) {
            ArrayList<Edge> tmpEdgeList = state.graph.getIncidentalEdges(v);

            for (Edge e : tmpEdgeList) {
                VertexPair curEdgeEndings = e.getEndings();
                Vertex curNewVertex = curEdgeEndings.from;
                if (v == curNewVertex)
                    curNewVertex = curEdgeEndings.to;

                if (!curVertexList.contains(curNewVertex))
                    curTreeEdge.add(e);
            }
        }

        Edge min = null;
        for (Edge e : curTreeEdge) {
            if (min == null || e.getWeight() < min.getWeight())
                min = e;
        }

        min.changeColor(Color.GREEN);
        VertexPair endings = min.getEndings();

        Vertex curNewVertex = endings.from;

```

```

        if (curVertexList.contains(curNewVertex))
            curNewVertex = endings.to;

        curNewVertex.setColor(Color.GREEN);
        curVertexList.add(curNewVertex);

        if (curVertexList.size() == graph.getVertexList().size())
            state = new State(state.prev, new Graph(state.graph),
AlgorithmState.End);
    }

    public void nextStep()
    {
        if (state.next == AlgorithmState.FirstVertex) {
            initState();
            FirstVertex();
        }
        else if (state.next == AlgorithmState.MinEdge)
            MinEdge();

        graphPanel.setGraph(state.graph);
        graphPanel.repaint();
    }

    public void prevStep()
    {
        state = state.prev;
        graphPanel.setGraph(state.graph);
        graphPanel.repaint();
    }

    public void restart()
    {
        initState();
        graphPanel.setGraph(state.graph);
        graphPanel.repaint();
    }

    public void forceAns()
    {
        while (state.next != AlgorithmState.End)
            nextStep();
    }

    public boolean isEnd()
    {
        return state.next == AlgorithmState.End;
    }

    public boolean isStart()
    {
        return state.next == AlgorithmState.FirstVertex;
    }

    public Graph getDefaultGraph() {
        initState();
        return graph;
    }

    public void updateGraph(Graph graph)
    {
        this.graph = graph;
        initState();
    }

```

} }