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

ОТЧЕТ

по учебной практике

Тема: Алгоритм ЯПД (Прима)

Студент гр. 9304 _____ Попов Д.С.

Студентка гр. 9304 _____ Рослова Л.С.

Студентка гр. 9304 _____ Паутова Ю.В.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург

2021

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Попов Д.С. группы 9304

Студентка Рослова Л.С. группы 9304

Студентка Паутова Ю.В. группы 9304

Тема практики: Алгоритм ЯПД (Прима)

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: ЯПД (Прима).

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 11.07.2021

Дата защиты отчета: 12.07.2021

Студент	_____	Попов Д.С.
Студентка	_____	Рослова Л.С.
Студентка	_____	Паутова Ю.В.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

В период прохождения данной учебной практики реализуется командная итеративная разработка визуализатора алгоритма ЯПД на языке программирования Java с графическим интерфейсом. Алгоритм ЯПД (Прима) – алгоритм построения минимального остового дерева взвешенного связанного неориентированного графа. Разработанное в ходе работы приложение визуализирует пошаговое выполнение данного алгоритма.

SUMMARY

During the course of this training practice, a team iterative development of a visualizer of the YAPD algorithm in the Java programming language with a graphical interface is implemented. The YAPD algorithm (Prima) is an algorithm for constructing a minimal spanning tree of a weighted connected undirected graph. The application developed during the work visualizes the step-by-step execution of this algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной учебной практики является изучение алгоритма ЯПД и итеративная разработка его визуализатора на языке программирования Java с графическим интерфейсом.

Алгоритм ЯПД (Прима) выглядит следующим образом:

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Реализация алгоритма	**Сложность алгоритма**
Тривиальная	$O(\|E\| \cdot \|V\|)$
Сортировка инцидентных рёбер каждой вершины по возрастанию весов	$O(\|V\|^2 + \|E\| \log \|V\|)$
Хранение для каждой невыбранной вершины минимального по весу ребра, соединяющего её с уже выбранной	$O(\|V\|^2)$

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к реализации алгоритма

- Алгоритм должен быть реализован так, чтобы можно было использовать любой тип данных.
- Алгоритм должен поддерживать возможность включения промежуточных выводов и пошагового выполнения.

1.1.2. Требования к проекту

- Возможность запуска через GUI и по желанию CLI (достаточно промежуточных выводов).
- Загрузка данных из файла или ввод через интерфейс.
- GUI должен содержать интерфейс управления работой алгоритма, визуализацию алгоритма, окно с логами работы.
- Возможность запуска алгоритма заново на новых данных без перезапуска программы.
- Возможность продвижения/отката на один шаг, завершения алгоритма до конца.
- Возможность сброса алгоритма в исходное состояние.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Итерация	Дата сдачи	Требования
Сдача прототипа	06.07.2021	• Эскиз или прототип графического интерфейса; • описание архитектуры (UML-диаграммы классов, состояний и последовательностей); • отчёт.
Сдача 1-ой версии	09.07.2021	• Реализация алгоритма; • прототип GUI с частичным функционалом; • отчёт.
Сдача 2-ой версии	12.07.2021	• Полностью рабочий GUI и CLI; • реализация взаимодействия с алгоритмом; • тестирование алгоритма; • отчёт.
Финал	14.07.2021	• Внесение правок/устранение недочетов.

2.2. Распределение ролей в бригаде

Участники	Итерация	Роли
Попов Д.С.	Сдача прототипа	Создание прототипа графического интерфейса, проектирование архитектуры.
	Сдача 1-ой версии	Объединение алгоритма с графическим интерфейсом.
	Сдача 2-ой версии	Доработка GUI.

Рослова Л.С.	Сдача прототипа	Создание UML-диаграмм, проектирование архитектуры.
	Сдача 1-ой версии	Реализация алгоритма.
	Сдача 2-ой версии	Промежуточные выводы (логи), разработка CLI.
Паутова Ю.В.	Сдача прототипа	Создание UML-диаграмм, проектирование архитектуры.
	Сдача 1-ой версии	Разработка визуализации графа для алгоритма.
	Сдача 2-ой версии	Тестирование алгоритма.

3. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС

3.1. Итерация 1: сдача прототипа

На рисунках 1-3 представлен внешний вид пользовательского (графического) интерфейса программы-визуализатора алгоритма ЯПД, разрабатываемого в данном проекте. На рисунке 1 представлено стартовое окно программы, которое открывается при запуске приложения. Здесь пользователь может ознакомиться с программой, нажав на кнопку «О программе», или перейти к вводу данных для визуализации алгоритма, нажав на кнопку «Поехали».

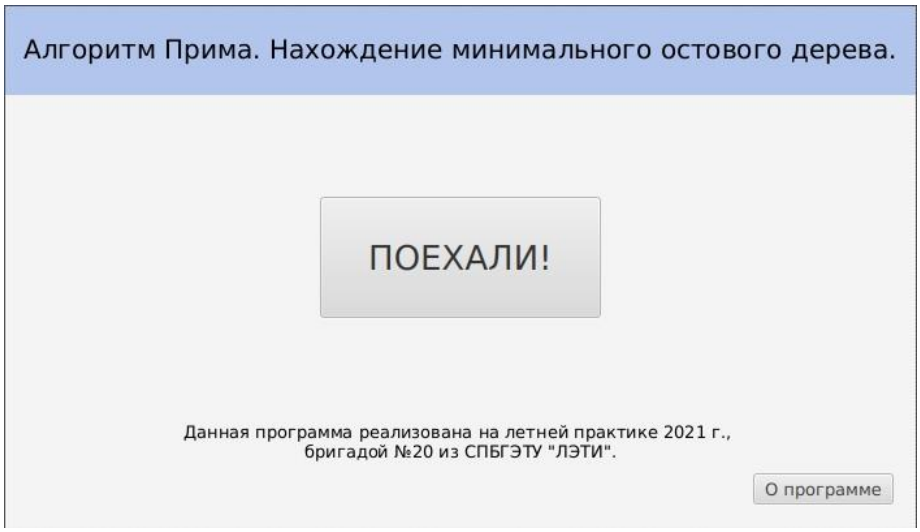

Рисунок 1 – Стартовое окно программы

На рисунке 2 представлено окно ввода данных для алгоритма. Пользователь может ввести данные непосредственно в самом окне или загрузить их из файла, в который заранее занёс данные. Также пользователь может выбрать с какой вершины начать построение минимального остового дерева. После ввода данных пользователь нажимает кнопку «Далее», чтобы перейти к визуальному представлению алгоритма.

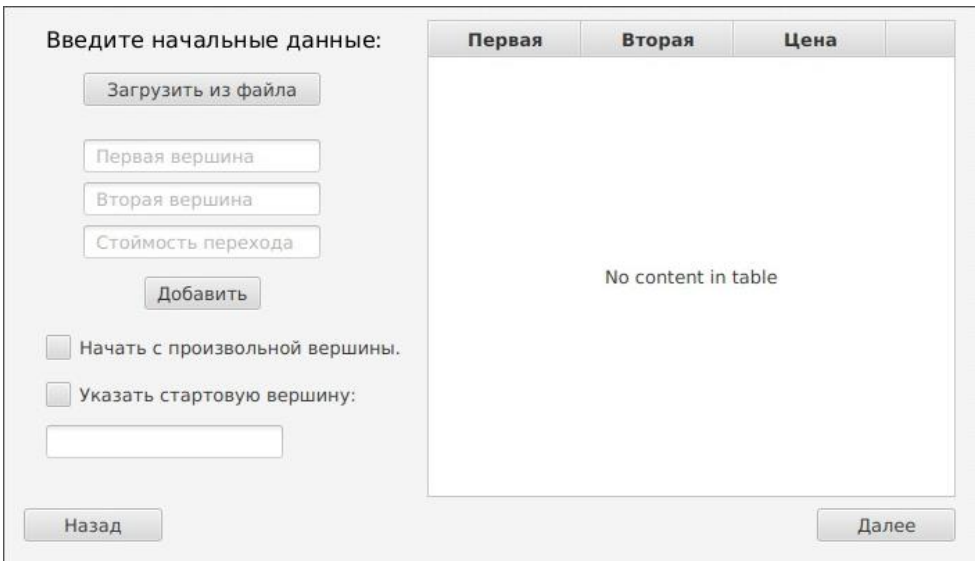

Рисунок 2 – Окно для ввода основных данных

На рисунке 3 представлено окно визуализации. Здесь пользователь может выбирать проходить алгоритм пошагово или до конца, может вернуться к изначальному состоянию графа или же ввести новые данные. Параллельно с визуальным представлением алгоритма в данном окне выводятся логи: какие ребра рассматривались и какое было взято в итоге.

Рисунок 3 – Окно визуализации

3.2. Итерация 2: сдача 1-ой версии

На рисунках 4-6 представлен реализованный функционал пользовательского (графического) интерфейса программы-визуализатора алгоритма ЯПД.

На рисунке 4 представлено окно для ввода данных. В нём был реализован функционал добавления рёбер как через графический интерфейс, так и через передаваемый программе файл. При нажатии кнопки далее происходит инициализация графа и мы переходим на следующее окно.

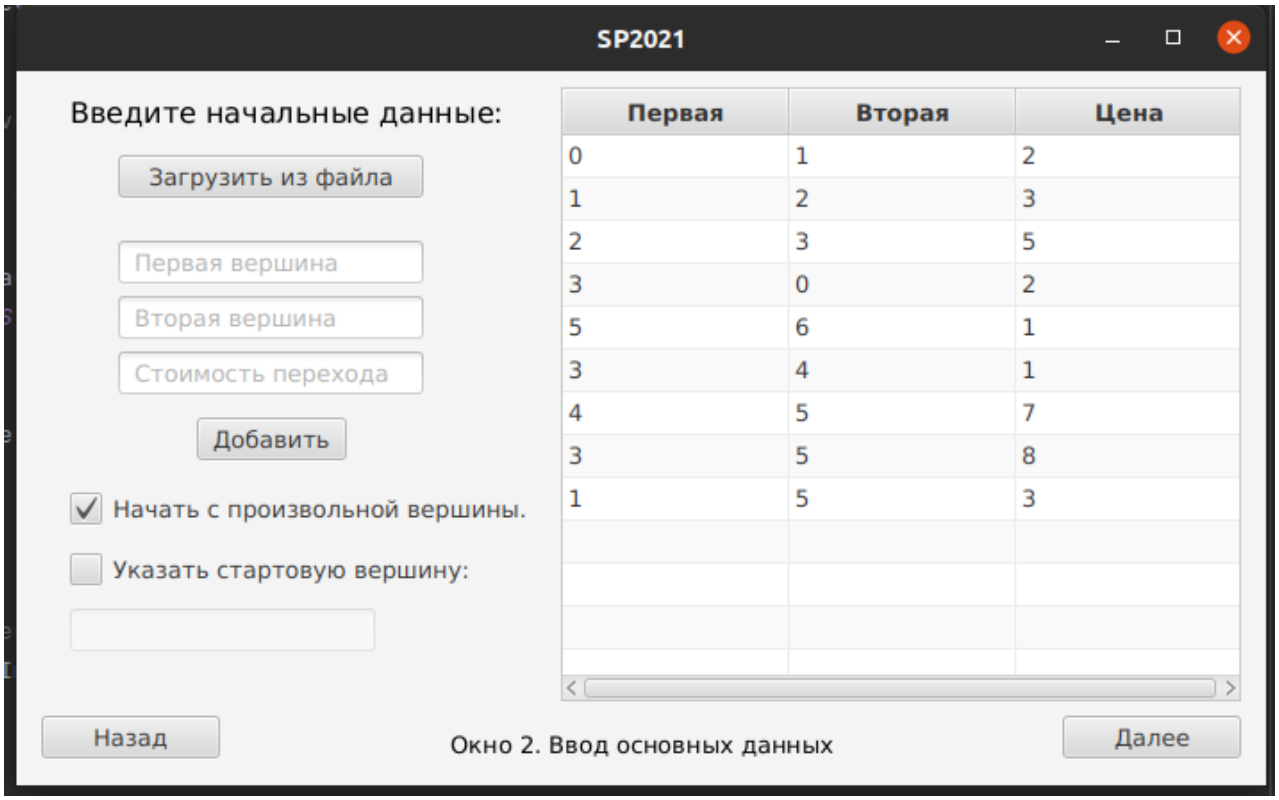

Рисунок 4 – Окно ввода информации

На рисунке 5 представлен граф, который мы инициализировали во 2м окошке программы. Функционал данного окна на данном этапе позволяет двигаться по алгоритму постепенно, откатываться на шаг назад, начать выполнение алгоритма заново.

На рисунке 6 представлен вывод алгоритма на некоторой итерации.

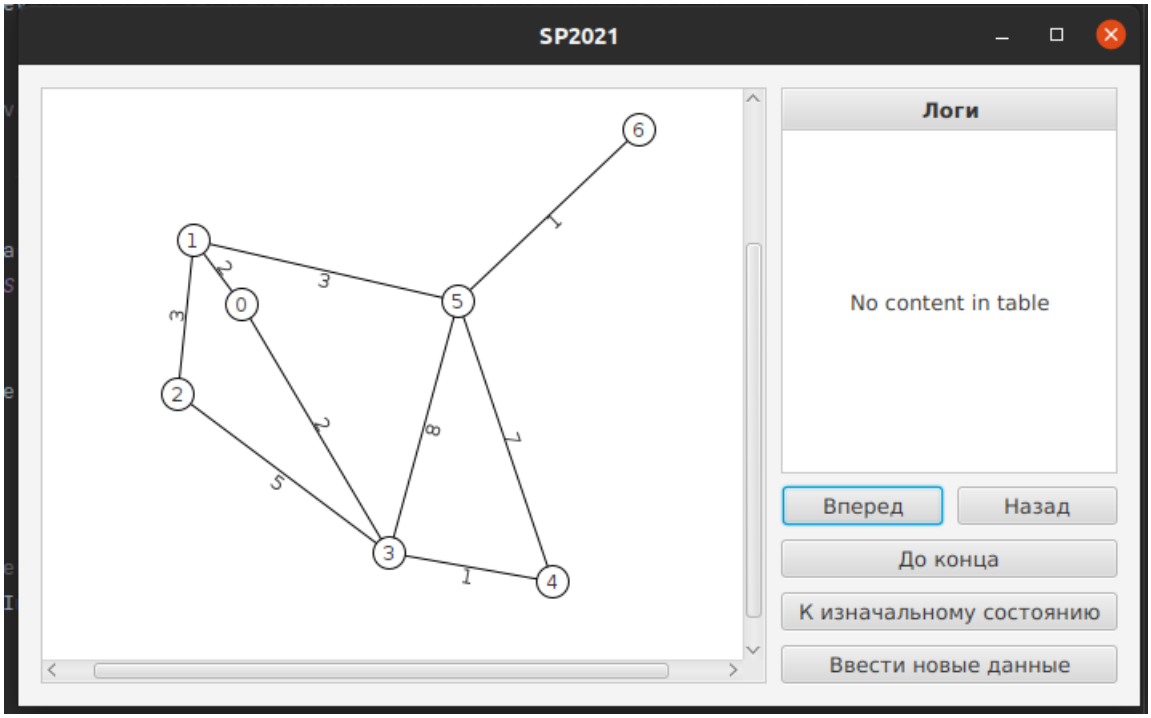

Рисунок 5 — Окно визуализации алгоритма

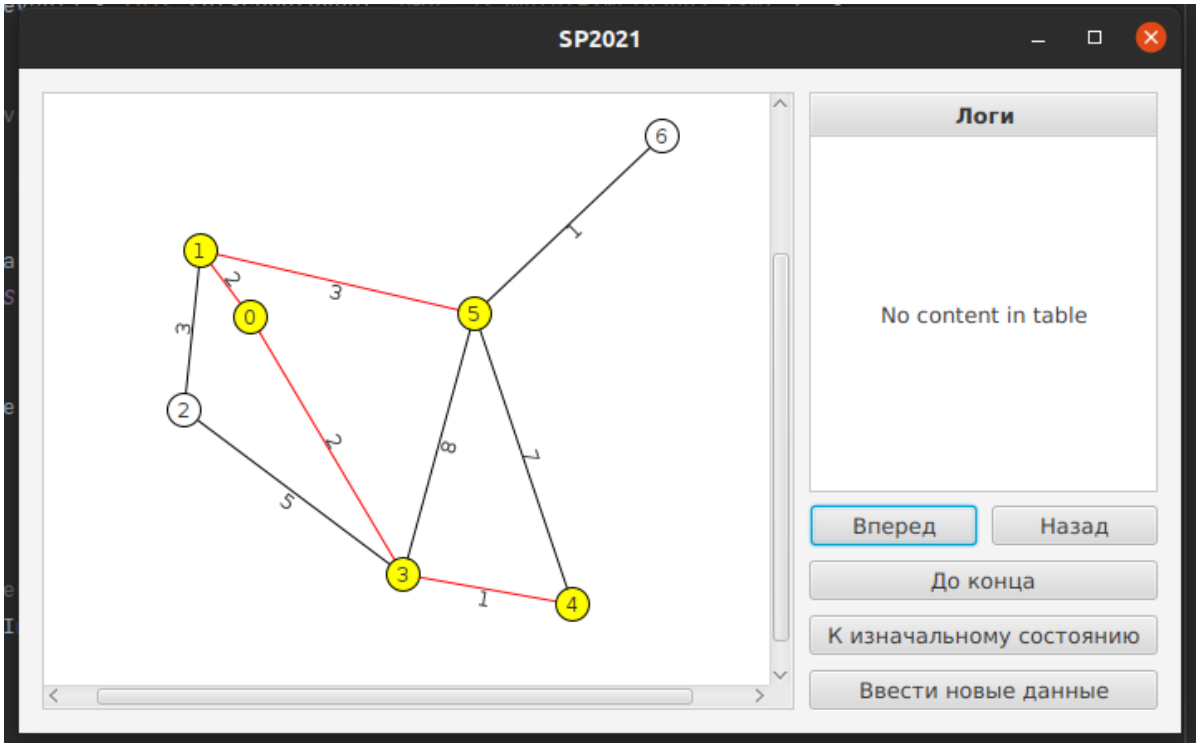

Рисунок 6 — Окно визуализации алгоритма (вывод программы на некоторой итерации)

Для графического представления алгоритма использовалась библиотека JUNG.

3.3. Итерация 3: сдача 2-ой версии

Функционал окна визуализации на данном этапе был дополнен двумя функциями: отображение алгоритма поэтапно после нажатия на кнопку «До конца» и возможность введения новых данных без перезапуска программы. Также добавлено поле для вывода логов программы (рисунок 7).

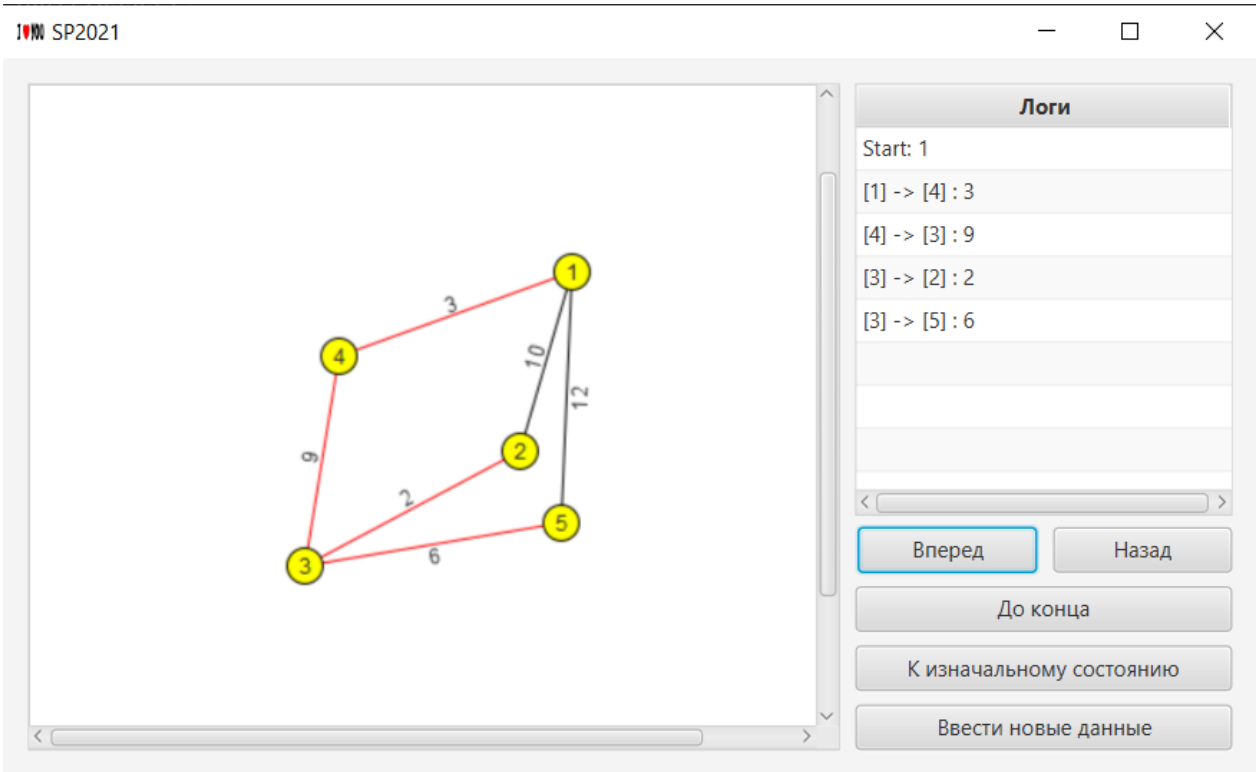

Рисунок 7 – Вывод логов программы

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

4.1. Описание архитектуры

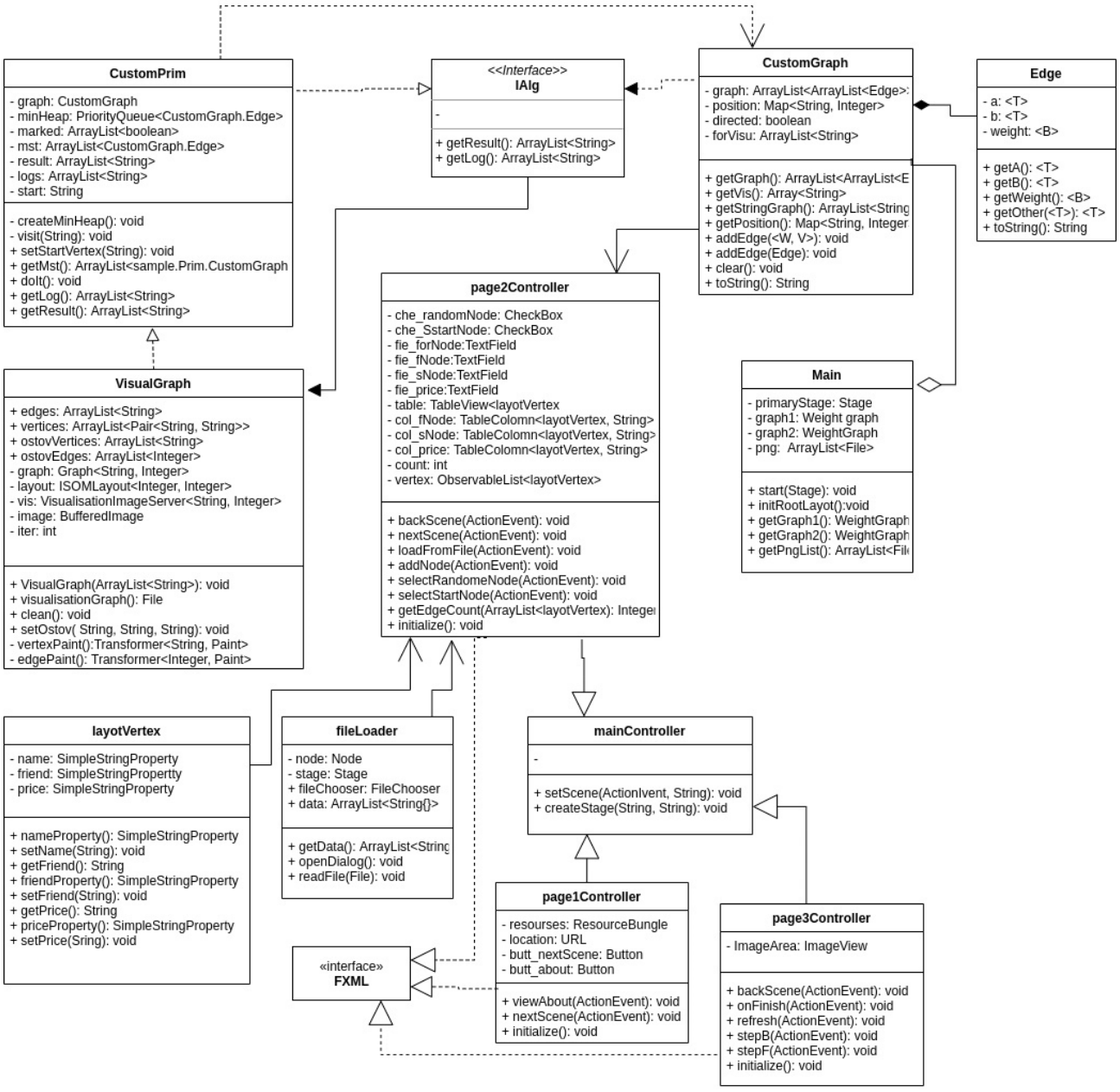

Рисунок 8 – UML-диаграмма классов

Рисунок 8 демонстрирует архитектуру программы на данном этапе реализации проекта. Т.к. инициализация графа для работы программы проходит во 2-м окне приложения, то все основные методы по созданию объектов и присваивания им значений происходят в классе page2Controller, потому что он реализует загрузку и функционал окна ввода данных. В нём происходит

создание считывание значений и создание графа, который передаётся алгоритму.

На рисунке 9 изображена UML-диаграмма состояний. С помощью данной диаграммы описывается алгоритм ЯПД, реализующийся в данном проекте. В начале создается граф по данным, введённым пользователем или считанным из файла, и выбирается вершина, с которой начнется построения минимального остового дерева. После чего запускается алгоритм:

1. У начальной вершины ищется ребро с наименьшей стоимостью. Если ребро не найдено, то в графе одна вершина и алгоритм завершается, иначе добавляем найденное ребро к каркасу.
2. В цикле происходит дополнение остового дерева:
   2.1. Формирование множества рёбер, одна вершина которых помечена, а вторая нет.
   2.2. Выбор из этого множества ребра с наименьшей стоимостью и добавление его к каркасу.
   2.3. Условие завершения цикла: все вершины графа вошли в остовое дерево, то есть были помечены.

Рисунок 9 – UML-диаграмма состояний

На рисунке 10 изображена UML-диаграмма последовательности, которая отображает взаимодействие объектов в динамике. При запуске пользователем программы открывается окно, после чего пользователь вводит данные, которые используются для построения графа. Построенный граф визуализируется и передается графическому интерфейсу для отображения его пользователю. Затем пользователь через интерфейс запускает алгоритм и проходит его пошагово или до конца. Каждая итерация алгоритма визуализируется и отображается пользователю через графический интерфейс.

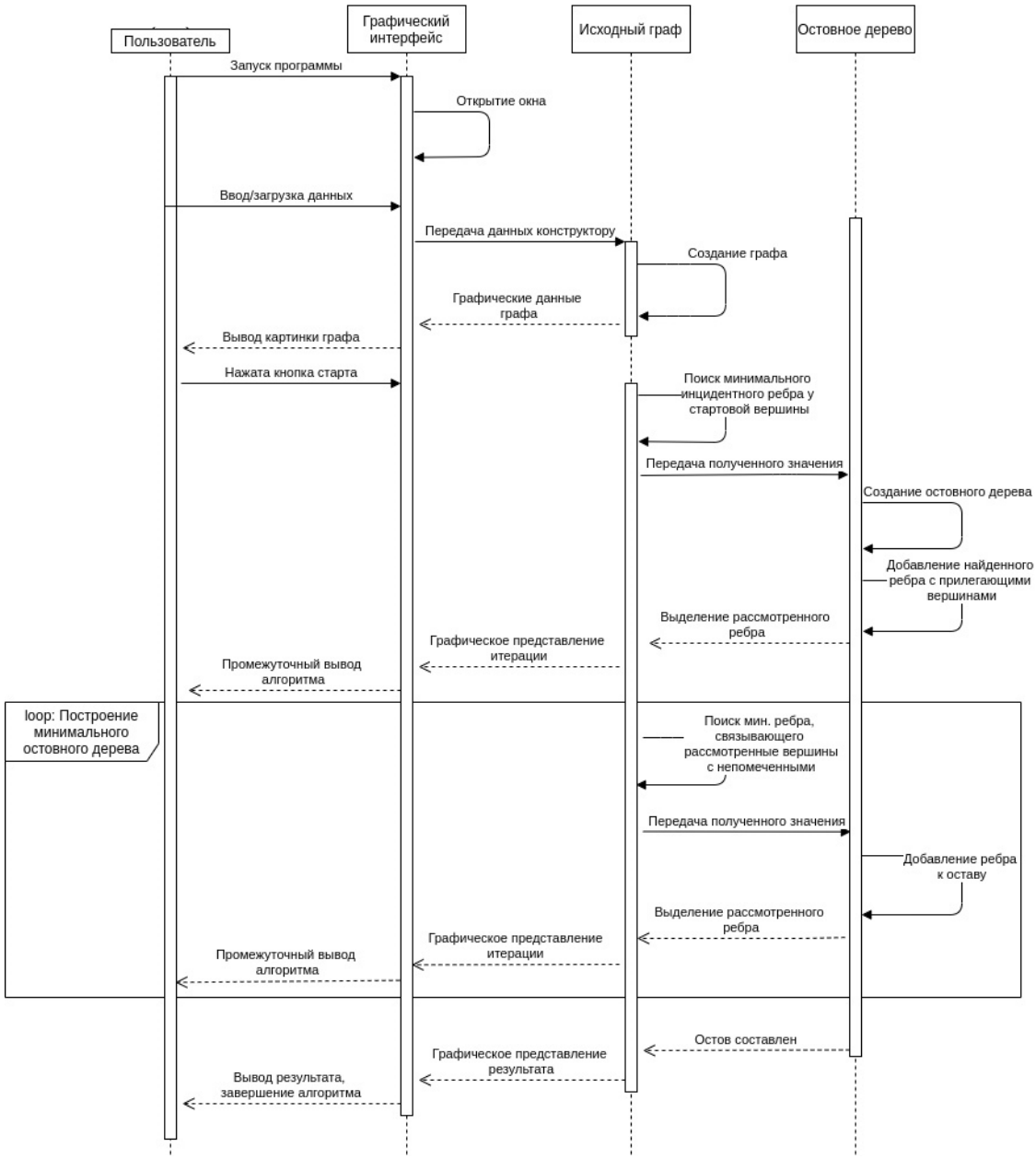

Рисунок 10 – UML-диаграмма последовательности

4.2. Реализация алгоритма Прима

Теорема сегментации: при любой сегментации ребро с наименьшим весом среди поперечных ребер должно принадлежать минимальному остовому дереву.

Придерживаясь этой теоремы, был реализован алгоритм:

1. Начиная с 0-й вершины, записываем посещенные вершины
2. Получаем все смежные ребра вершины (вершина на другом конце ребра не должна быть посещена, иначе ребро не является ребром поперечного сечения) и добавляем вес ребра к минимальной куче
3. Берём текущее наименьшее ребро из наименьшей кучи. Согласно теореме о разбиении, это ребро должно быть ребром MST.
4. Устанавливаем текущую вершину на другую вершину наименьшего ребра и начинаем повторять процесс 1, 2, 3
5. Когда минимальная куча пуста, построение MST завершено.

4.3. Структуры данных и основные методы

4.3.1. Класс CustomGraph

Данный класс является реализацией графа на языке программирования Java.

Поля класса:

- ***private ArrayList<ArrayList<Edge>> graph*** – представление графа;
- ***private Map<String, Integer> position*** – позиция в листе;
- ***private final boolean directed*** – ориентированный ли граф;
- ***private ArrayList<String> forVisu*** – список для построения графа для визуализации.

Методы класса:

- Конструктор ***public CustomGraph(boolean directed)*** – инициализирует поля directed, position, graph, forVisu.

- ***public ArrayList<ArrayList<Edge>> getGraph()*** – возвращает значение поля graph.
- ***public ArrayList<String> getVis()*** – возвращает значение поля forVisu.
- ***public Map<String, Integer> getPosition()*** – возвращает значение поля position.
- ***public <W,V> void addEdge(W w, W v, V weight)*** – создает ребро и добавляет его в граф.
- ***public void addEdge(Edge other)*** – добавляет ребро в граф.
- ***public void clear()*** – очищает поля position, forVisu, graph.
- ***public String toString()*** – переопределение метода toString для данного класса.

4.3.2. Класс Edge

Данный класс является представлением ребра.

Поля класса:

- ***private T a*** – имя первой вершины;
- ***private T b*** – имя второй вершины;
- ***private B weight*** – имя первой вершины;

Методы класса:

- ***public T getA()*** – возвращает значение поля a.
- ***public T getB()*** – возвращает значение поля b.
- ***public T getOther(T other)*** – возвращает смежную вершину.
- ***public B getWeight()*** – возвращает значение поля weight.
- ***public String toString()*** – переопределение метода toString для данного класса.

4.3.3. Класс CustomPrima

Данный класс является реализацией алгоритма ЯПД (Прима) построения минимального остового дерева.

Поля класса:

- ***private PriorityQueue<WeightGraph.Edge> minHeap*** – минимальная куча, откуда будут стягиваться значения;
- ***private ArrayList<CustomGraph.Edge> mst*** – хранение ребер минимального остового дерева;
- ***private ArrayList<Boolean> marked*** – добавлена вершина в остов или нет;
- ***private ArrayList<String> logs*** – хранит список логов;
- ***private CustomGraph graph*** – хранит граф;
- ***private String start*** – хранит стартовую вершину.

Методы класса:

- Конструктор ***public CustomPrim(CustomGraph graph)*** – заполняет поля graph и start, инициализирует поле logs.
- ***private void createMinHeap()*** – создание минимальной кучи.
- ***privet void visit(String str)*** – поиск рёбер, связывающих помеченные вершины и непомеченные.
- ***public void setStartVertex(String newStart)*** – заполняет поле start.
- ***public String getStartVertex()*** – возвращает значение поля start.
- ***public ArrayList<CustomGraph.Edge> getMst()*** – возвращает значение поля mst.
- ***public void doIt()*** – реализация алгоритма Прима.
- ***public ArrayList<String> getLog()*** – возвращает значение поля logs.

4.3.4. Класс VisualGraph

Данный класс является реализацией графического отображения графа.

Поля класса:

- ***private final Graph<String, Integer> graph*** – граф;

- ***private final ISOMLayout<String, Integer> layout*** – макет;
- ***private final VisualizationImageServer<String, integer> vis*** – визуализация графа и создание его изображения;
- ***private BufferedImage image*** – описывает изображение с доступным буфером данных;
- ***private int iter*** – хранит номер итерации выполнения алгоритма;
- ***private static ArrayList<String> edges*** – хранит веса рёбер;
- ***private static ArrayList<Integer> ostovEdges*** – хранит id рёбер, которые вошли в остов;
- ***private static ArrayList<Pair<String, String>> vertices*** – хранит пары вершин, принадлежащих соответствующим рёбрам;
- ***private static ArrayList<String> ostovVertices*** – хранит вершины вошедшие в остов на каждой итерации.

Методы класса:

- Конструктор ***public VisuaGraph(ArrayList<String> input)*** – принимает список введённых рёбер; на его основе создает граф и заполняет поля edges и vertices; создает макет графа; задает размеры и другие параметры изображения.
- ***public File visualizationGraph()*** – окрашивает граф в соответствии с итерацией алгоритма и сохраняет изображение в файл, который возвращает.
- ***public void clean()*** – очищает поля edges, vertices, ostovEdges и ostovVertices.
- ***public void setOstov()*** – на каждой итерации алгоритма добавляет вошедшее ребро с вершинами в ostovEdges и ostovVertices соответственно.

- ***private Transformer<String, Paint> vertexPaint()*** – отвечает за окрашивание вершин.
- ***private Transformer<Integer, Paint> edgePaint()*** – отвечает за окрашивание рёбер.

4.3.5. Классы для работы с графическим интерфейсом

4.3.5.1. mainController

4.3.5.2. page1Controller

4.3.5.3. page2Controller

4.3.5.4. page3Controller

Данные классы реализованы через интерфейс FXML.

4.3.5.5. Класс layotVertex

4.3.5.6. Класс fileLoader

Данные классы производят считывание данных из графического интерфейса/загрузку данных из файла в графический интерфейс для построения графа.

4.3.6. Класс Main

Данный класс является главным классом программы.

Поля класса:

- ***private Stage primaryStage*** – окно программы;
- ***private static CustomGraph graph*** – хранит граф;
- ***private static ArrayList<String> log*** – хранит логи;
- ***public static ArrayList<File> png*** – хранит файлы, в которые сохранены отображения работы алгоритма.

Методы класса:

- ***public void start(Stage primaryStage)*** – создание окна программы.
- ***public void initRootLayot()*** – заполнение окна программы.
- ***public static CustomGraph getGraph()*** – возвращает значение поля graph.

- ***public static Collection getLog()*** – возвращает значение поля log.

Точка входа в программу:

- ***public static void main(String[] args)*** – определяет взаимодействие с программой: через CLI или GUI.

Код описанных структур данных представлен в приложении А.

5. ТЕСТИРОВАНИЕ

5.1. Тестирование кода алгоритма

Тестирование было проведено с использованием библиотеки JUnit. Результаты тестирования приведены на рисунке 11.

При запуске тестов инициализировались поля prim1 и prim2. Первому в конструктор был передан граф, вершины которого заданы числами, второму – граф, вершины которого заданы буквами.

Метод	Описание
throwsExceptionWhenGraphIsNull()	Проверяет, кидает ли конструктор класса CustomPrim исключение при передаче ему пустого графа.
setStartVertex()	Проверяет, было ли заполнено поле start класса CustomPrim при вызове метода setStartVertex.
	Проверяет, что при заполнении стартовой вершины вершиной с именем, которого нет в графе, построение минимального остового дерева начинается с вершины по умолчанию.
getMst()	Сравнивает минимальное остовое дерево, полученное алгоритмом, с ожидаемым результатом.
getMstNotNull()	Проверяет, что при отработке алгоритма возвращается не пустое остовое дерево.
getLog()	Сравнивает логи, полученные алгоритмом, с заданными.

getLogNotNull()	Проверяет, что логи записываются при отработке алгоритма.

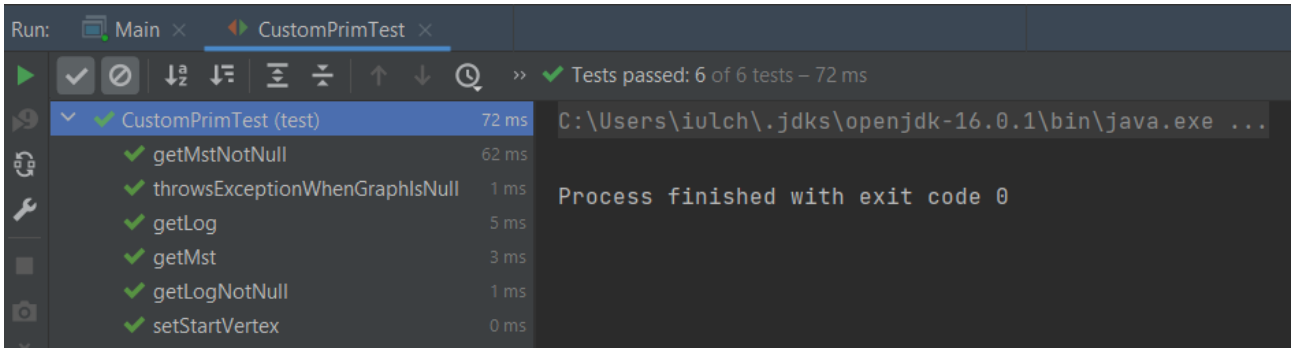

Рисунок 11 – Результат работы тестов

ЗАКЛЮЧЕНИЕ

Разработка проекта учебной практики велась итеративно, то есть создание программы-визуализатора алгоритма ЯПД (Прима) осуществлялось в несколько этапов.

На первом этапе были разработаны архитектура проекта и прототип пользовательского интерфейса.

На втором этапе были реализованы алгоритм, визуальное представление графа и пользовательский интерфейс с частичным функционалом: загрузка данных из файла, ввод данных через интерфейс, отображение пошагового выполнения алгоритма.

На третьем этапе были реализованы GUI с полным функционалом и CLI, была осуществлена параметризация алгоритма, то есть алгоритм стал поддерживать любой тип данных, и было проведено тестирование алгоритма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. AL диджитализация бизнеса // evergreens.com.ua. URL: https://evergreens.com.ua/ru/articles/uml-diagrams.html

2. Свободная энциклопедия // wikipedia.org. URL: https://ru.wikipedia.org/wiki/Алгоритм_Прима

3. JUNG: Java Universal Network/Graph Framework // jung.sourceforge.net. URL: http://jung.sourceforge.net/doc/api/index.html

4. JavaRush – онлайн-курс обучения программированию на Java // javarush.ru. URL: https://javarush.ru/groups/posts/605-junit

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Название файла: ***Main.java***

```
package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Objects;
import sample.Prim.*;

import static java.lang.System.exit;


public class Main extends Application {

    private Stage primaryStage;
    private static CustomGraph graph = new CustomGraph(false);
    private static ArrayList<String> log = new ArrayList();
    public static ArrayList<File> png = new ArrayList<>();

    @Override
    public void start(Stage primaryStage){
        this.primaryStage = primaryStage;
        this.primaryStage.setTitle("SP2021");
        initRootLayot();
    }

    public void initRootLayot(){
        try{
            InputStream iconStream =
getClass().getResourceAsStream("/sample/assets/icon3.png");
            Image image = new Image(Objects.requireNonNull(iconStream));
            primaryStage.getIcons().add(image);
            Parent root =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("/sample/FX
ML/page2.fxml")));
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    public static CustomGraph getGraph(){ return graph; }
    public static Collection getLog(){ return log; }

    public static void main(String[] args) {
```

```
        for(String str : args){
            if(str.equals("CLI")){
                BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
                System.out.println("Введите кол-во ребер.");
                try{
                    String count = reader.readLine();
                    int c = Integer.parseInt(count);
                    System.out.println("Введите данные(Вершина1 Вершина2
Вес)");
                    while(c != 0){
                        String[] reg = reader.readLine().split(" ");
                        graph.addEdge(reg[0], reg[1],
Integer.parseInt(reg[2]));
                        c--;
                    }
                    CustomPrim prim = new CustomPrim(graph);
                    prim.doIt();
                    System.out.println(prim.getLog());
                }catch (Exception e){
                    System.out.println(e.getMessage());
                }
                exit(0);
            }else{
                launch(args);
            }
        }
        launch(args);
    }
}
```

Название файла: ***fileLoader.java***

```
package sample;

import javafx.event.ActionEvent;
import javafx.scene.Node;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import java.io.*;
import java.util.ArrayList;


/* Класс открывает файл и производит считывание данных для графа. */
public class fileLoader {

    private Node node;
    private Stage stage;
    final FileChooser fileChooser;
    ArrayList<String[]> data;


    public fileLoader(ActionEvent event){
        this.node = (Node)event.getSource();
        this.stage = (Stage)node.getScene().getWindow();
        this.fileChooser = new FileChooser();
        this.data = new ArrayList<>();
    }
```

```
    public ArrayList<String[]> getData(){
        return data;
    }

    public void openDialog(){
        FileChooser.ExtensionFilter filter = new
FileChooser.ExtensionFilter("text", "*.txt");
        fileChooser.setTitle("Выберите файл.");
        fileChooser.getExtensionFilters().add(filter);
        fileChooser.setInitialDirectory(new
File(System.getProperty("user.home")));
        File file = fileChooser.showOpenDialog(stage);

        if (file != null) {
            //System.out.println("Файл открылся!");
            readFile(file);
        }
    }

    private void readFile(File file) {
        try {
            FileReader fr = new FileReader(file);
            BufferedReader reader = new BufferedReader(fr);
            String line = reader.readLine();
            while (line != null) {
                //System.out.println(line);
                String[] words = line.split(" ");
                line = reader.readLine();
                if(words.length != 3){
                    continue;
                }
                Integer.parseInt(words[0]);
                Integer.parseInt(words[1]);
                Integer.parseInt(words[2]);
                this.data.add(words);
            }
        }catch(Exception e){
            System.out.println(e.getMessage());
            data.clear();
        }
    }
}
```

Название файла: ***VisualGraph.java***

```
package sample.Prim;

import edu.uci.ics.jung.algorithms.layout.*;
import edu.uci.ics.jung.graph.Graph;
import edu.uci.ics.jung.graph.SparseMultigraph;
import edu.uci.ics.jung.visualization.VisualizationImageServer;
import edu.uci.ics.jung.visualization.decorators.EdgeShape;
import edu.uci.ics.jung.visualization.decorators.ToStringLabeller;
import edu.uci.ics.jung.visualization.renderers.Renderer;
import javafx.util.Pair;
import org.apache.commons.collections15.Transformer;
```

```
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.geom.Point2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;



public class VisualGraph {

    private final Graph<String, Integer> graph;
    private final ISOMLayout<String, Integer> layout;
    private final VisualizationImageServer<String, Integer> vis;
    private BufferedImage image;
    private int iter = 0;

    public VisualGraph(ArrayList<String> input) {
//вес,вершина1,вершина2,...

        this.graph = new SparseMultigraph<String, Integer>();
        int SIZE = input.size();
        int edgeId = 0;
        int i = 0;

        while (i<SIZE) {
            String[] res = input.get(i).split(" ");
            graph.addEdge((Integer) edgeId, res[1], res[2]);
            edges.add(res[0]);
            vertices.add(new Pair<String,String>(res[1], res[2]));
            i += 1;
            edgeId += 1;
        }

        this.layout = new ISOMLayout<String,Integer>(this.graph);
        layout.setSize(new Dimension(500,500));
        this.vis = new VisualizationImageServer<String,
Integer>(layout,new Dimension(500,500));
        vis.setBackground(Color.WHITE); //цвет фона
        vis.getRenderContext().setEdgeLabelTransformer(new
Transformer<Integer, String>() { //Преобразование id ребра на его вес при
рисовании
            public String transform(Integer input) {
                return VisualGraph.edges.get(input);
            }
        });
        vis.getRenderContext().setEdgeShapeTransformer(new
EdgeShape.Line<String, Integer>());
        vis.getRenderContext().setVertexLabelTransformer(new
ToStringLabeller<String>());

vis.getRenderer().getVertexLabelRenderer().setPosition(Renderer.VertexLab
el.Position.CNTR);

    }
```

```
    public File visualizationGraph(){

vis.getRenderContext().setEdgeDrawPaintTransformer(this.edgePaint());

vis.getRenderContext().setVertexFillPaintTransformer(vertexPaint());
        this.image = (BufferedImage) vis.getImage(new Point2D.Double(500
/ 2,500 / 2), new Dimension(500,500));
        File outputFile;
        try {
            outputFile = File.createTempFile("./src/sample/Result/" +
Integer.toString(iter) + ".png", null);
            outputFile.deleteOnExit();
            ImageIO.write(image, "png", outputFile);
        } catch (IOException e) {
            System.out.println(e.getMessage());
            outputFile = null;
        }
        iter += 1;
        return outputFile;
    }

    public void clean(){
        edges.clear();
        vertices.clear();
        ostovEdges.clear();
        ostovVertices.clear();
    }


    public void setOstov(String weight, String v1, String v2){
        String V1 = v1;
        String V2 = v2;
        ostovVertices.add(V1);
        ostovVertices.add(V2);
        String curV1, curV2;
        for(int i = 0; i<edges.size();i++){
            curV1 = vertices.get(i).getKey();
            curV2 = vertices.get(i).getValue();
            if(edges.get(i).equals(weight) && (curV1.equals(V1) |
curV1.equals(V2)) && (curV2.equals(V2) | curV2.equals(V1))){
                ostovEdges.add(i);
            }
        }
    }

    private Transformer<String,Paint> vertexPaint() { //окрашивание
вершин в разные цвета в соотв. с их именами
        return new Transformer<String, Paint>() {
            @Override
            public Paint transform(String str) {
                int SIZE = VisualGraph.ostovVertices.size();
                for (int j = 0; j < SIZE; j++) {
                    if (str.equals(VisualGraph.ostovVertices.get(j))) {
                        return Color.YELLOW;
                    }
                }
                return Color.WHITE;
```

```
            }
        };
    }

    private Transformer<Integer,Paint> edgePaint() { //Окрашивание ребер
в нужный цвет
        return new Transformer<Integer, Paint>() {
            @Override
            public Paint transform(Integer integer) {
                int SIZE = VisualGraph.ostovEdges.size();
                for (int i = 0; i < SIZE; i++) {
                    if (integer == VisualGraph.ostovEdges.get(i)) {
                        return Color.RED;
                    }
                }
                return Color.BLACK;
            }
        };
    }

    private static ArrayList<String> edges = new ArrayList<>();
    private static ArrayList<Pair<String,String>> vertices = new
ArrayList<>();
    private static ArrayList<String> ostovVertices = new ArrayList<>();
    private static ArrayList<Integer> ostovEdges = new ArrayList<>();
}
```

Название файла: ***layotVertex.java***

```
package sample.Prim;

import javafx.beans.property.SimpleStringProperty;

public class layotVertex {
    private SimpleStringProperty name;
    private SimpleStringProperty friend;
    private SimpleStringProperty price;

    public layotVertex(String a, String b, String c){
        this.name = new SimpleStringProperty(a);
        this.friend = new SimpleStringProperty(b);
        this.price = new SimpleStringProperty(c);
    }

    public String getName() {
        return name.get();
    }

    public SimpleStringProperty nameProperty() {
        return name;
    }

    public void setName(String name) {
        this.name.set(name);
    }

    public String getFriend() {
        return friend.get();
```

```
    }

    public SimpleStringProperty friendProperty() {
        return friend;
    }

    public void setFriend(String friend) {
        this.friend.set(friend);
    }

    public String getPrice() {
        return price.get();
    }

    public SimpleStringProperty priceProperty() {
        return price;
    }

    public void setPrice(String price) {
        this.price.set(price);
    }
}
```

Название файла: ***IAlg.java***

```
ackage sample.Prim;

import java.util.ArrayList;

public interface IAlg {
    ArrayList<String> getResult();
    ArrayList<String> getLog();
}
```

Название файла: ***CustomPrim.java***

```
package sample.Prim;

import java.util.ArrayList;
import java.util.PriorityQueue;



public class CustomPrim implements IAlg{

    private PriorityQueue<CustomGraph.Edge> minHeap;
    private ArrayList<CustomGraph.Edge> mst;
    private ArrayList<Boolean> marked;
    private ArrayList<String> result;
    private ArrayList<String> logs;
    private CustomGraph graph;
    private String start;

    public CustomPrim(CustomGraph graph){
        if(graph.getGraph().isEmpty()){
            throw new IllegalArgumentException("CustomPrim: graph is
empty!");
        }
```

```
        this.graph = graph;
        logs = new ArrayList<>();
        this.start = new
String(graph.getGraph().get(0).get(0).getA().toString());       //
Получаем имя самой первой вершины.
        createMinHeap();
    }

    private void createMinHeap(){
        minHeap = new PriorityQueue<>((e1 ,e2)->{
            return (int)e1.getWeight() - (int)e2.getWeight();
        });
        marked = new ArrayList<>();
        for(int i = 0; i < graph.getGraph().size(); ++i){
            marked.add(false);
        }
        mst = new ArrayList<>();
    }

    private void visit(String str){
        int index = graph.getPosition().get(str);
        marked.set(index, true);
        for(CustomGraph.Edge edge : graph.getGraph().get(index)){

if(!marked.get(graph.getPosition().get(edge.getOther(str).toString()))){
                minHeap.offer(edge);
            }
        }
    }

    public void setStartVertex(String newStart){
        if(graph.getPosition().containsKey(newStart)){
            this.start = newStart;
        }
    }

    public String getStartVertex(){return start;}

    public ArrayList<sample.Prim.CustomGraph.Edge> getMst(){
        return mst;
    }

    public void doIt(){
        this.result = new ArrayList<>();
        visit(start);
        logs.add("Start: " + start);
        while (!minHeap.isEmpty()) {
            CustomGraph.Edge minEdge = minHeap.poll();
            if (marked.get(graph.getPosition().get(minEdge.getA())) &&
marked.get(graph.getPosition().get(minEdge.getB()))){
                // Ели посещены обе вершины.
                continue;
            }
            logs.add("[" + minEdge.getA() + "] -> [" + minEdge.getB() +
"] : " + minEdge.getWeight());
            mst.add(minEdge);
            result.add(minEdge.getWeight().toString() + " " +
```

```
minEdge.getA().toString() + " " + minEdge.getB().toString());

            if (!marked.get(graph.getPosition().get(minEdge.getA()))) {
                visit(minEdge.getA().toString());
            } else {
                visit(minEdge.getB().toString());
            }
        }
    }

    @Override
    public ArrayList<String> getLog(){ return logs; }

    @Override
    public ArrayList<String> getResult(){
        return result;
    }
}
```

Название файла: ***CustomGraph.java***

```
package sample.Prim;

import org.apache.commons.collections15.map.HashedMap;
import java.util.ArrayList;
import java.util.Map;


// Класс графа.
public class CustomGraph {

    // Класс ребра.
    public static class Edge<T,B>{

        private T a;        // Имя первой вершины.
        private T b;        // Имя второй вершины.
        private B weight;   // Стоймость перехода.

        public Edge(T a, T b, B weight){
            this.a = a;
            this.b = b;
            this.weight = weight;
        }

        public T getA(){ return a; }

        public T getB(){ return b; }

        public T getOther(T other) { return other.equals(a) ? b : a; }

        public B getWeight(){ return weight; }

        @Override
        public String toString() {
            String outString = new String();
            outString = outString.concat(a + " - " + b + " : " + weight);
            return outString;
        }
```

```
    }

    private ArrayList<ArrayList<Edge>> graph;   // Представление нашего
графа.
    private Map<String, Integer> position;      // Позиция в листе.
    private final boolean directed;             // Ориентированный ли
граф.
    private ArrayList<String> forVisu;

    public CustomGraph(boolean directed){
        this.directed = directed;
        this.position = new HashedMap<String, Integer>();
        this.graph = new ArrayList<ArrayList<Edge>>();
        this.forVisu = new ArrayList<>();
    }

    public ArrayList<ArrayList<Edge>> getGraph(){
        return this.graph;
    }

    public ArrayList<String> getVis(){ return forVisu; }

    public ArrayList<String> getStringGraph(){
        ArrayList<String> out = new ArrayList<>();
        for(ArrayList<CustomGraph.Edge> a : graph){
            for(Edge b : a){
                out.add(new String(b.getWeight().toString() + " " +
b.getA().toString() + " " + b.getB().toString()));
            }
        }
        return out;
    }

    public Map<String, Integer> getPosition(){
        return this.position;
    }

    public <W, V> void addEdge(W w, W v, V weight){
        if(position.containsKey(w)){
            Edge newEdge = new Edge(w, v, weight);
            graph.get(position.get(w)).add(newEdge);
            //forVisu.add(new String(weight.toString() + " " +
w.toString() + " " + v.toString()));
        }else{
            graph.add(new ArrayList<Edge>());
            int index = graph.size() - 1;
            Edge newEdge = new Edge(w, v, weight);
            graph.get(index).add(newEdge);

            position.put(w.toString(), index);
        }
        forVisu.add(new String(weight.toString() + " " + w.toString() + "
" + v.toString()));
        if(!directed){
            //addEdge(v, w, weight);        // Хз как этот момент
поправить.
            if(position.containsKey(v)){
```

```
                Edge newEdge = new Edge(v, w, weight);
                graph.get(position.get(v)).add(newEdge);
            }else{
                graph.add(new ArrayList<Edge>());
                int index = graph.size() - 1;
                Edge newEdge = new Edge(v, w, weight);
                graph.get(index).add(newEdge);
                position.put(v.toString(), index);
            }
        }
    }

    public void addEdge(Edge other) throws NullPointerException{
        if(position.containsKey(other.getA().toString())){
            graph.get(position.get(other.getA().toString())).add(other);
        }else{
            graph.add(new ArrayList<Edge>());
            int index = graph.size() - 1;
            graph.get(index).add(other);
            position.put(other.getA().toString(), index);
        }
        forVisu.add(new String(other.weight.toString() + " " +
other.a.toString() + " " + other.b.toString()));
        if(!directed){
            if(position.containsKey(other.getB().toString())){

graph.get(position.get(other.getB().toString())).add(other);
            }else{
                graph.add(new ArrayList<Edge>());
                int index = graph.size() - 1;
                graph.get(index).add(other);
                position.put(other.getB().toString(), index);
            }
        }
    }

    public void clear(){
        position.clear();
        forVisu.clear();
        graph.clear();
    }

    @Override
    public String toString() {
        String outString = new String("");
        String preString = new String("");
        for(ArrayList<Edge> t : graph){
            for(Edge v : t){
                outString = outString.concat(v.toString() + "\n");
            }
            outString.concat(preString);
        }
        return outString;
    }
}
```

Название файла: ***mainController.java***

```
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.util.Objects;


/* Родительский класс всех контроллеров, осуществляет задачу смену сцен и
создание дополнительных окон. */
public class mainController {

    public void setScene(ActionEvent event, String path){
        Node node = (Node)event.getSource();
        Stage stage = (Stage)node.getScene().getWindow();
        try{
            Parent root =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource(path)));
            Scene scene = new Scene(root);
            stage.setScene(scene);
            stage.show();
        }catch (Exception e){
            e.printStackTrace();
        }
    }

    public Stage createStage(String path, String name){
        Stage newStage = new Stage();
        try{
            FXMLLoader fmxloader = new FXMLLoader();
            fmxloader.setLocation(getClass().getResource(path));
            Scene scene = new Scene(fmxloader.load());
            newStage.setTitle(name);
            newStage.setScene(scene);
        }catch(Exception e){
            e.printStackTrace();
        }
        return newStage;
    }
}
```

Название файла: ***page1Controller.java***

```
package sample.Controllers;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.event.ActionEvent;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.stage.Stage;



public class page1Controller extends mainController{

    @FXML private ResourceBundle resources;
    @FXML private URL location;
    @FXML private Button butt_nextScene;
    @FXML private Button butt_about;
```

```
    private Stage stageAbout;


    @FXML /* По нажатию на "О программе" выводит окно с информацией. */
    void viewAbout(ActionEvent event) {
        if(!stageAbout.isShowing()) {
            stageAbout.showAndWait();
        }
    }

    @FXML /* Смена сцены на page2 по нажатию кнопки далее. */
    void nextScene(ActionEvent event) {
        this.setScene(event,"/sample/FXML/page2.fxml");
    }

    @FXML
    void initialize() { stageAbout =
this.createStage("/sample/FXML/aboutProgram.fxml", "О программе"); }
}
```

Название файла: ***page2Controller.java***

```
package sample.Controllers;

import java.util.ArrayList;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.fxml.FXML;
import javafx.scene.control.cell.PropertyValueFactory;
import sample.*;
import sample.Prim.*;



public class page2Controller extends mainController{

    @FXML private CheckBox che_randomNode;
    @FXML private CheckBox che_startNode;
    @FXML private CheckBox che_debugInfo;
    @FXML private TextField fie_forNode;
    @FXML private TextField fie_fNode;
    @FXML private TextField fie_sNode;
    @FXML private TextField fie_price;
    @FXML private TableView<layotVertex> table;
    @FXML private TableColumn<layotVertex, String> col_fNode;
    @FXML private TableColumn<layotVertex, String> col_sNode;
    @FXML private TableColumn<layotVertex, String> col_price;
    private int count = 0;                          // Подсчет введенных
пользователем вершин.
    private ObservableList<layotVertex> vertex;     // Наблюдаемый лист для
TableView.


    @FXML /* Смена сцены на page1 по нажатию кнопки "Назад". */
    void backScene(ActionEvent event) {
        this.setScene(event,"/sample/FXML/page1.fxml");
```

```
    }


    @FXML /* Смена сцены на page3 по нажатию кнопки "Далее". */
    void nextScene(ActionEvent event) {
        CustomGraph graph = Main.getGraph();
        for(layotVertex layot : this.vertex){
            graph.addEdge(layot.getName(),
                          layot.getFriend(),
                          Integer.parseInt(layot.getPrice()));
        }
        CustomPrim prim = new CustomPrim(graph);
        VisualGraph vis = new VisualGraph(graph.getVis());

        if(this.che_startNode.isSelected()){
            prim.setStartVertex(this.fie_forNode.getText());
        }

        prim.doIt();
        Main.png.add(vis.visualizationGraph());
        //System.out.println(prim.getMst().size());
        for(int i = 0; i < prim.getMst().size(); ++i){
            vis.setOstov(prim.getMst().get(i).getWeight().toString(),
prim.getMst().get(i).getA().toString(), prim.getMst().get(i).getB().toString());
            Main.png.add(vis.visualizationGraph());
        }
        vis.clean();
        Main.getLog().addAll(prim.getLog());
        this.setScene(event,"/sample/FXML/page3.fxml");
    }


    @FXML /* Загрузка из файла по нажатию кнопки "Загрузить из файла". */
    void loadFromFile(ActionEvent event) {
        this.vertex.clear();
        fileLoader file = new fileLoader(event);
        file.openDialog();
        ArrayList<String[]> vertexs = file.getData();
        for(String[] str : vertexs){
            this.vertex.add(new layotVertex(str[0], str[1], str[2]));
        }
    }


    @FXML /* Добавить данные в граф из полей (fie_fNode, fir_sNode, fie_price)
по нажатию кнопки "Добавить". */
    void addNode(ActionEvent event) {
        if(!fie_fNode.getText().isEmpty() || !fie_sNode.getText().isEmpty() ||
!fie_price.getText().isEmpty()){
            try{
                Integer.parseInt(fie_price.getText());
                this.vertex.add(new layotVertex(fie_fNode.getText(),
fie_sNode.getText(), fie_price.getText()));
            }catch(Exception e){
                e.getMessage();
            }
        }
        fie_fNode.clear();
        fie_sNode.clear();
        fie_price.clear();
    }
```

```
    @FXML /* Выбран чек-бокс с рандомной вершиной. */
    void selectRandomNode(ActionEvent event) {
        if(che_randomNode.isSelected()){
            che_startNode.setSelected(false);
            fie_forNode.setDisable(true);
        }else{
            che_randomNode.setSelected(true);
        }
    }


    @FXML /* выбран чек-бокс со стартовой вершиной. */
    void selectStartNode(ActionEvent event) {
        if(che_startNode.isSelected()){
            che_randomNode.setSelected(false);
            fie_forNode.setDisable(false);
        }else{
            che_startNode.setSelected(true);
        }
    }


    @FXML
    void debugInfo(ActionEvent event) {
        if(che_debugInfo.isSelected()){
            che_debugInfo.setSelected(false);
        }else{
            che_debugInfo.setSelected(true);
        }
    }


    @FXML
    void initialize() {
        Main.getGraph().clear();
        Main.getLog().clear();
        this.vertex = FXCollections.observableArrayList();
        col_fNode.setCellValueFactory(new PropertyValueFactory<layotVertex,
String>("name"));
        col_sNode.setCellValueFactory(new PropertyValueFactory<layotVertex,
String>("friend"));
        col_price.setCellValueFactory(new PropertyValueFactory<layotVertex,
String>("price"));
        table.setItems(this.vertex);
    }
}
```

Название файла: ***page3Controller.java***

```
package sample.Controllers;

import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
```

```
import java.util.Collection;
import java.util.concurrent.*;
import sample.*;
import java.io.File;
import java.util.ArrayList;



public class page3Controller extends mainController{

    @FXML private Button butt_stepForward;
    @FXML private Button butt_finish;
    @FXML private Button butt_stepBack;
    @FXML private Button butt_refresh;
    @FXML private TableView<String> logTable;
    @FXML private TableColumn<String, String> col_log;
    @FXML private Button butt_newData;
    @FXML private ImageView imageArea;
    private ArrayList<File> png;        // Лист файлов с рисунками.
    private int iter;                   // Итератор по рисункам.
    private ObservableList<String> details;

    @FXML
    void backScene(ActionEvent event) {
        Main.png.clear();
        this.setScene(event,"/sample/FXML/page2.fxml");
    }

    @FXML
    void onFinish(ActionEvent event) {
        try {
            if(png.size() > this.iter + 1) {
                this.iter += 1;
                imageArea.setImage(new
Image(png.get(iter).toURI().toString()));
                //this.stepF(event);
                //System.out.println(iter);
                TimeUnit.SECONDS.sleep(1);
                details.add(Main.getLog().get(iter));
                this.onFinish(event);
            }
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }

    @FXML
    void refresh(ActionEvent event) {
        imageArea.setImage(new Image(png.get(0).toURI().toString()));
        this.iter = 0;
    }

    @FXML
    void stepB(ActionEvent event) {
        if(iter > 0){
            this.iter--;
            imageArea.setImage(new
```

```
Image(png.get(iter).toURI().toString()));
            details.remove(Main.getLog().get(iter+1));
            logTable.setItems(details);
        }
    }

    @FXML
    void stepF(ActionEvent event) {
        if(png.size() > this.iter + 1){
            this.iter++;
            imageArea.setImage(new
Image(png.get(iter).toURI().toString()));
            details.add(Main.getLog().get(iter));
            logTable.setItems(details);
        }
    }

    @FXML
    void initialize(){
        this.png = Main.png;
        this.iter = 0;
        imageArea.setImage(new Image(png.get(0).toURI().toString()));
        details = FXCollections.observableArrayList();
        details.add(Main.getLog().get(iter));
        TableView<String> tableView = new TableView<>();
        TableColumn<String, String> col1 = new TableColumn<>();
        tableView.getColumns().addAll(col1);
        col_log.setCellValueFactory(data -> new
SimpleStringProperty(data.getValue()));
        logTable.setItems(details);
    }
}
```

Название файла: ***aboutProgram.fxml***

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Text?>


<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" prefHeight="200.0" prefWidth="400.0"
xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Text layoutX="14.0" layoutY="21.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Алгоритм Прима - алгоритм построеия минимального
остовного дерева взвешенного связного неориентированного графа.
Построение начинается с дерева, включающего в себя одну(произвольную)
вершину. В течение работы алгоритма дерево разрастается, пока не охватит
все вершины исходного графа. На каждом шаге алгоритма к текущему дереву
присоединяется самое легкое из ребер, соединяющих вершину из построенного
дерева, и вершину не из дерева. Асимптотика алгоритма зависит от способа
хранения графа и способа хранения вершин, не входящих в дерево."
wrappingWidth="373.7294921875" />
   </children>
</AnchorPane>
```

Название файла: ***page1.fxml***

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane maxHeight="400.0" maxWidth="700.0" minHeight="400.0"
minWidth="700.0" prefHeight="400.0" prefWidth="700.0"
xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="sample.Controllers.page1Controller">
   <children>
      <Button fx:id="butt_nextScene" layoutX="242.0" layoutY="146.0"
mnemonicParsing="false" onAction="#nextScene" prefHeight="93.0"
prefWidth="216.0" text="ПОЕХАЛИ!" AnchorPane.leftAnchor="242.0"
AnchorPane.topAnchor="146.0">
         <font>
            <Font size="26.0" />
         </font></Button>
      <Text layoutX="272.0" layoutY="379.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Окно 1. Вводная часть"
wrappingWidth="156.7294921875" AnchorPane.bottomAnchor="14.0" />
      <AnchorPane prefHeight="68.0" prefWidth="700.0" style="-fx-
background-color: #b2c6ed;" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
         <children>
            <Text layoutX="9.0" layoutY="41.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Алгоритм Прима. Нахождение минимального остового
дерева." textAlignment="CENTER" wrappingWidth="681.7294921875"
AnchorPane.bottomAnchor="22.283203125" AnchorPane.leftAnchor="9.0"
AnchorPane.rightAnchor="9.27050781259"
AnchorPane.topAnchor="21.716796875">
               <font>
                  <Font size="20.0" />
               </font>
            </Text>
         </children>
      </AnchorPane>
      <Text layoutX="17.0" layoutY="332.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Данная программа реализована на летней практике
2021 г., &#10;бригадой №20 из СПБГЭТУ &quot;ЛЭТИ&quot;."
textAlignment="CENTER" wrappingWidth="665.7294921875"
AnchorPane.bottomAnchor="49.80126953125" />
      <Button fx:id="butt_about" layoutX="575.0" layoutY="358.0"
mnemonicParsing="false" onAction="#viewAbout" text="О программе"
AnchorPane.bottomAnchor="14.0" AnchorPane.rightAnchor="14.0" />
   </children>
</AnchorPane>
```

Название файла: ***page2.fxml***

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.CheckBox?>
<?import javafx.scene.control.TableColumn?>
```

```
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane maxHeight="-Infinity" maxWidth="1.7976931348623157E308"
minHeight="400.0" minWidth="700.0" prefHeight="400.0" prefWidth="700.0"
xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="sample.Controllers.page2Controller">
   <children>
      <Button fx:id="butt_nextScene" layoutX="586.0" layoutY="361.0"
mnemonicParsing="false" onAction="#nextScene" prefHeight="24.0"
prefWidth="100.0" text="Далее" AnchorPane.bottomAnchor="15.0"
AnchorPane.rightAnchor="14.0" />
      <Button fx:id="butt_backScene" layoutX="14.0" layoutY="361.0"
mnemonicParsing="false" onAction="#backScene" prefHeight="24.0"
prefWidth="100.0" text="Назад" AnchorPane.bottomAnchor="15.0"
AnchorPane.leftAnchor="14.0" />
      <Text layoutX="241.0" layoutY="378.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Окно 2. Ввод основных данных"
textAlignment="CENTER" wrappingWidth="218.7294921875"
AnchorPane.bottomAnchor="14.0" AnchorPane.leftAnchor="241.0"
AnchorPane.rightAnchor="240.2705078125" />
      <TableView fx:id="table" layoutX="373.0" layoutY="37.0"
prefHeight="344.0" prefWidth="381.0" AnchorPane.bottomAnchor="47.0"
AnchorPane.leftAnchor="305.0" AnchorPane.rightAnchor="14.0"
AnchorPane.topAnchor="9.0">
        <columns>
          <TableColumn fx:id="col_fNode"
maxWidth="1.7976931348623157E308" minWidth="127.0" prefWidth="127.0"
text="Первая" />
          <TableColumn fx:id="col_sNode"
maxWidth="1.7976931348623157E308" minWidth="127.0" prefWidth="127.0"
text="Вторая" />
            <TableColumn fx:id="col_price"
maxWidth="1.7976931348623157E308" minWidth="127.0" prefWidth="127.0"
text="Цена" />
        </columns>
      </TableView>
      <TextField fx:id="fie_fNode" layoutX="57.0" layoutY="81.0"
promptText="Первая вершина" />
      <TextField fx:id="fie_sNode" layoutX="57.0" layoutY="115.0"
promptText="Вторая вершина" />
      <TextField fx:id="fie_price" layoutX="57.0" layoutY="148.0"
promptText="Стоимость перехода" />
      <CheckBox fx:id="che_randomNode" layoutX="23.0" layoutY="216.0"
mnemonicParsing="false" onAction="#selectRandomNode" selected="true"
text="Начать с произвольной вершины." />
      <CheckBox fx:id="che_startNode" layoutX="23.0" layoutY="243.0"
mnemonicParsing="false" onAction="#selectStartNode" text="Указать
стартовую вершину:" />
      <TextField fx:id="fie_forNode" disable="true" layoutX="50.0"
layoutY="271.0" />
      <Text layoutX="30.0" layoutY="29.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Введите начальные данные:">
        <font>
```

```
            <Font size="16.0" />
         </font>
      </Text>
      <Button fx:id="butt_addNode" layoutX="100.0" layoutY="181.0"
mnemonicParsing="false" onAction="#addNode" text="Добавить" />
      <Button fx:id="butt_loadFromFile" layoutX="57.0" layoutY="47.0"
mnemonicParsing="false" onAction="#loadFromFile" prefWidth="171.0"
text="Загрузить из файла" />
      <CheckBox fx:id="che_debugInfo" layoutX="23.0" layoutY="313.0"
mnemonicParsing="false" onAction="#debugInfo" text="Выводить отладочную
информацию." />
   </children>
</AnchorPane>
```

Название файла: ***page3.fxml***

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane maxHeight="400.0" maxWidth="700.0" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="400.0" prefWidth="700.0"
xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="sample.Controllers.page3Controller">
   <children>
      <Button fx:id="butt_stepForward" layoutX="479.0" layoutY="264.0"
mnemonicParsing="false" onAction="#stepF" prefHeight="24.0"
prefWidth="100.0" text="Вперед" AnchorPane.bottomAnchor="113.0"
AnchorPane.rightAnchor="123.0" />
      <Button fx:id="butt_finish" layoutX="479.0" layoutY="330.0"
mnemonicParsing="false" onAction="#onFinish" prefHeight="24.0"
prefWidth="210.0" text="До конца" AnchorPane.bottomAnchor="80.0"
AnchorPane.rightAnchor="14.0" />
      <Button fx:id="butt_stepBack" layoutX="585.0" layoutY="264.0"
mnemonicParsing="false" onAction="#stepB" prefHeight="24.0"
prefWidth="100.0" text="Назад" AnchorPane.bottomAnchor="113.0"
AnchorPane.rightAnchor="14.0" />
      <Button fx:id="butt_refresh" layoutX="476.0" layoutY="326.0"
mnemonicParsing="false" onAction="#refresh" prefHeight="24.0"
prefWidth="210.0" text="К изначальному состоянию"
AnchorPane.bottomAnchor="47.0" AnchorPane.rightAnchor="14.0" />
      <TableView fx:id="logTable" layoutX="476.0" layoutY="14.0"
prefHeight="241.0" prefWidth="210.0" AnchorPane.bottomAnchor="145.0"
AnchorPane.rightAnchor="14.0" AnchorPane.topAnchor="14.0">
        <columns>
          <TableColumn fx:id="col_log" maxWidth="1.7976931348623157E308"
prefWidth="210.0" text="Логи" />
        </columns>
      </TableView>
      <Button fx:id="butt_newData" layoutX="480.0" layoutY="362.0"
mnemonicParsing="false" onAction="#backScene" prefHeight="24.0"
prefWidth="210.0" text="Ввести новые данные"
AnchorPane.bottomAnchor="14.0" AnchorPane.rightAnchor="14.0" />
```

```
      <ScrollPane layoutX="14.0" layoutY="14.0" prefHeight="373.0"
prefWidth="450.0" AnchorPane.bottomAnchor="14.0"
AnchorPane.leftAnchor="14.0" AnchorPane.rightAnchor="233.0"
AnchorPane.topAnchor="14.0">
         <content>
            <ImageView fx:id="imageArea" fitHeight="500.0"
fitWidth="500.0" pickOnBounds="true" preserveRatio="true" />
         </content></ScrollPane>
   </children>
</AnchorPane>
```