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

ОТЧЕТ

по учебной практике

Тема: Новые языки программирования

Студент гр. 3342 _____ Иванов Д.М.

Студент гр. 3342 _____ Корниенко А.Е.

Студент гр. 3341 _____ Трофимов В.О.

Руководитель _____ Шестопалов Р.П.

Санкт-Петербург

2025

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Иванов Д.М. группы 3342

Студент Корниенко А.Е. группы 3342

Студент Трофимов В.О. группы 3341

Тема практики: Новые языки программирования

Задание на практику:

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

Алгоритм: Мосты графа.

Сроки прохождения практики: 25.06.2024 – 08.07.2024

Дата сдачи отчета: 04.07.2024

Дата защиты отчета: 04.07.2024

Студент	_____	Иванов Д.М.
Студент	_____	Корниенко А.Е.
Студент	_____	Трофимов В.О.
Руководитель	_____	Шестопалов Р.П.

АННОТАЦИЯ

Цель: изучить новый язык программирования Java и применить полученные знания на практике в виде реализации приложения с графическим интерфейсом с использования фреймворков данного языка программирования.

В содержание практики входит самостоятельное изучение языка Java с помощью онлайн-курса и дополнительных материалов, которые предоставил преподаватель. После этого полученные знания проверяются преподавателем. Дальше происходит разбивка на бригады по 3 человека, в которых студенты совместно разрабатывают графическое приложение и результат каждый итерации демонстрируют преподавателю.

SUMMARY

Objective: to learn a new programming language Java and apply the acquired knowledge in practice in the form of implementing an application with a graphical interface using the frameworks of this programming language. The content of the practice includes independent study of the Java language using an online course and additional materials provided by the teacher. After this, the acquired knowledge is checked by the teacher. Then there is a division into teams of 3 people, in which the students jointly develop a graphical application and demonstrate the result of each iteration to the teacher.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: Познакомится с языком программирования Java и с его фреймворком JavaFX. С их помощью реалтзовать графическое приложение.

Задачи: Изучить основы ЯП Java, изучить фреймворк JavaFX, с бригадой обсудить структуру проекта, роли и план разработки, налаживать взаимосвязь через систему git.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Предназначение программы:

Программа запускает графический интерфейс приложения по определению мостов в связном неориентированном графе, который задал сам пользователь. Задать граф можно через взаимодействие с графическим интерфесом или текстовым файлом. В результате выполнения алгоритма пользователь видит шаги обхода графа и итоговый набор ребер, которые являются мостами.

(Спецификация)

Подраздел «Исходные Требования к программе» следует разбить на подразделы 2-го уровня (1.1.1 – требования к вводу исходных данных, 1.1.2 – требования к визуализации и т. д.)

1.1.1 Требования к вводу исходных данных

Пользователю не нужно задавать название вершин. Их программа пронумерует автоматически от 1 до N (N – число вершин в графе). Способы задать граф:

1) По клику мыши добавляются вершины в графическое поле. Через взаимодействия с кнопками и текстовыми полями можно добавлять или удалять ребра между вершинами в зависимости от выбранного режима.

2) Граф представляется в виде матрицы смежности размера N*N в текстовом файле (*.txt). Он содержит N строчек, в каждой из которых через пробел стоит N чисел: 1 – есть связь между вершинами, 0 – ее нет.

1.1.2 Требования к визуализации

Описание элементов интерфейса:

1) Графическое поле: фон приложения, где находятся остальные компоненты.

2) Вершины в виде кругообразного элемента

3) Ребра в виде линий, соединящих вершины

4) Набор кнопок для функционала приложения:

- кнопка запуска алгоритма
- кнопка добавления ребра
- кнопка удаления ребра
- кнопка выбора режима добавления вершин
- кнопка выбора режима удаления вершин с выходящими из них ребер
- текстовые поля, в которые пользователь вводит смежные вершины: в каждое поле пишется номера вершин, которые уже добавлены в графическое поле и которые должны быть соединены ребром
- кнопка помощи (help): выводит дополнительное окно с инструкцией
- окно с интрукцией приложения
- кнопка очистки поля
- кнопка загрузки графа из текстового файла

1.1.2 Описание алгоритма поиска мостов

Начинаем обход DFS с любой вершины и во время него меняем направление ребер на обратное (то есть если мы прошли из вершины A в вершину B, то задаем ориентация B -> A).

После этого делаем 2-ой обход DFS с учетом заданной ориентации и красим вершины в цвета. Когда обход дальше невозможен, меняем цвет и начинаем DFS с еще не просмотренной вершины.

На последнем этапы находим те ребра, которые соединяют вершины разного цвета.

1.2. Уточнение требований после сдачи прототипа…

После сдачи прототипа следовало доделать функционал оставшихся кнопок: загрузки и сохранения графа. Также связать созданный интерфейс и алгоритм посика мостов. Добавить к нему визуализацию.

1.3. Уточнение требований после сдачи 1-ой версии

После сдачи 1-ой версии следовало добавить возможность сохранять граф в любое место файловой системы пользователя. Давать пользователю мгновенный результат мостов графа. И сохранять граф вместе с мостами.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Дата	Этап проекта	Реализованные возможности	Выполнено
27.06.2025	Согласование спецификации	Обсуждение с коллегами и наставником плана будущей разработки	
30.06.2025	Сдача прототипа	Реализация базового интерфейса отдельно от алгоритма	
02.07.2025	Сдача версии 1	Реализация связи алгоритма и интерфейса. Дописание функционала оставшихся кнопок.	
04.07.2025	Сдача версии 2	Прописание мгновенного результата алгоритма. Дать возможность пользователю сохранять граф вместе с мостами и в любое место файловой системы.	
04.07.2025	Сдача отчёта		
04.07.2025	Защита отчёта		

2.2. Распределение ролей в бригаде

Антон Корниенко — Прописание интерфейса приложения и визуализация выполнения алгоритма через создание потока. Реализация мгновенного результата алгоритма. Функционал очистки графа, добавление/удаления ребер и вершин. Подключение css стилей.

Иванов Данила — Прописание логики алгоритма и привязка его к графическому интерфейсу. Реализация класса загрузки и сохранения графа в текстовый файл и обработка исключения в случае неверного входного файла, реализация кнопки «Help». Очистка визуализации после окончания алгоритма или его прерывания.

Трофимов Владислав — Прописание архитектуры классов приложения. Реализация добавления/удаления ребер. Доработки по сохранению загрузки графа. Подключение css стилей.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

HashMap — для хранения графа в виде: вершины --> {набор вершин смежных с ней}.

List, ArrayList — для хранения набора вершин и ребер в графе. Динамическое их изменение.

File — для работы с файлами.

Массивы — для хранения набора возможных цветов для вершин.

Stack — для DFS в алгоритме мостов.

3.2. Основные методы

После создания всех кнопок пропишем их функционал через метод setOnAction. То есть какая функция будет выполняться при нажатии на ту или иною кнопку. Пропишем основные функции, которые работают в данном приложении.

graph.execute() - модель графа запускает работу алгоритма, куда передает в качестве данных информацию о связях между вершинами.

alg.findBridges(); alg.getResultFast() - метод, содержащий сам алгорим поиска мостов в графе. Выполняется либо в виде потока с задержкой, либо быстро, сразу выводя результат.

CleanState: graph.clear(); graphView.cleanSurface() - происходит при очистке графа. Очищаются поля из модели графа и убираются компоненты из основного поля приложения. Обнуляется счетчик вершин.

alert.showAndWait(); fileChooser.showOpenDialog(null) -запуск диалоговых окон для пользователя. Предназначены для инструкции, сообщения об ошибке или выбора файла из ФС.

LoadState — открывается диалоговое окно, откуда пользователь выбирает файл для загрузки графа. После информация идет в объект класса GraphFileReader.

SaveState - открывается диалоговое окно, откуда пользователь выбирает файл для сохранения графа. После этого информация идет в модель графа (graph.saveResult(filename)) и пользователь видит это отображение на панели.

handleClickOnPane — по нажатию на главное поле приложение и после проверки выбранного режима пользователь может добавить новую вершину.

handleClickOnVertex — происходит при нажатии на вершину. Если выбран режим удаления — она удалится и из модели, и из поля. Инача вершина покрасится в другой цвет, и станет возможным добавить ребро.

handleAddEdge — алгоритм добавления ребер.

handleDeleteEdge — алгоритм удаления ребер.

3.3. Основные пакеты

controller — прописана основная логика работа программы при нажатии на ту или иною кнопку. Как должна вести себя программа, как должен меняться алгоритм и графическое поле.

states — набор состояний, в которые может перейти программа. Реализована по принципу паттерна «Абстрактная фабрика». Каждое состояние наследуется от интерфейса EditorState со следующими методами: открытие состояние, закрытие состояния и обработка при том или ином нажатии мышкой.

model — хранит основные модели приложения. Для вершин хранится информация о состоянии на том или ином шаге алгоритма мостов. Также хранятся модели графа, вершин, класс работы с файлами, исключение при ошибки с файлами, сам класс алгоритма. То есть здесь используются данные для внутренней логики приложения.

view — хранит классы для визуализации приложения. В классах реализовано расположение кнопок и полей определенном образом. Координаты, размеры и стили каждого компонента.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

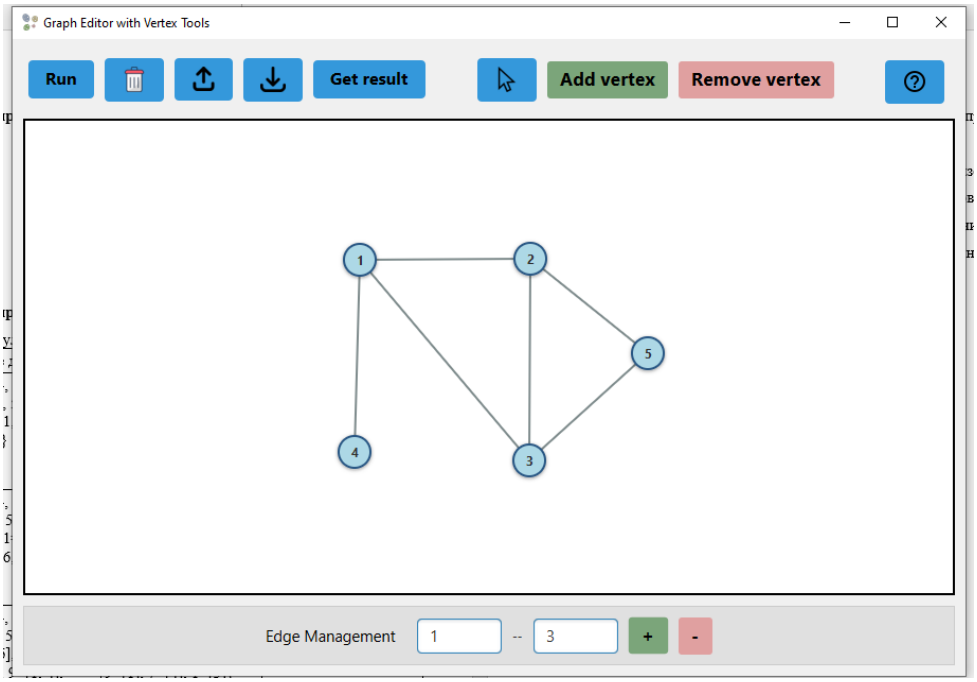

Рисунок 1 – Тестирование создания графа и удаления некоторых компонент

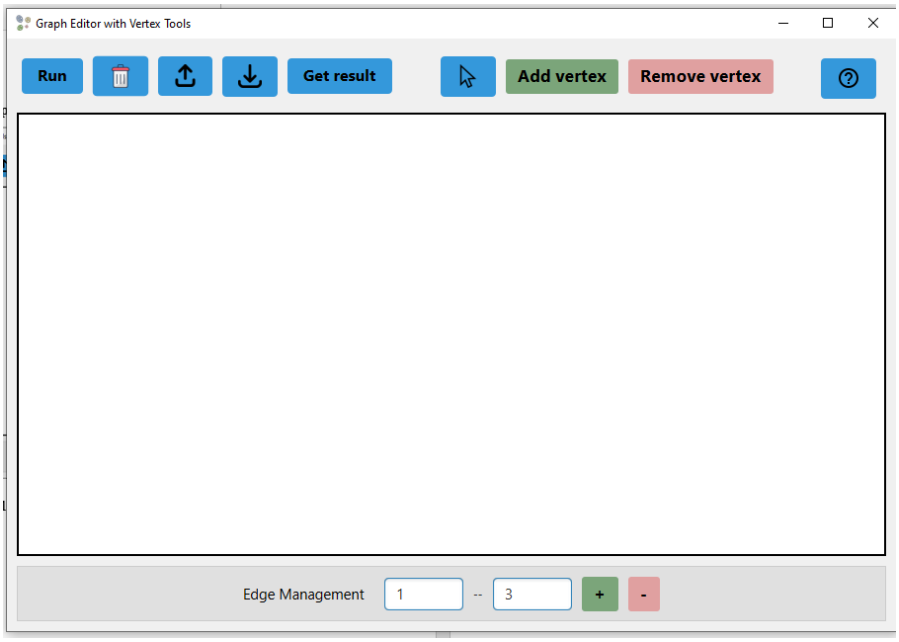

Рисунок 2 – Тестирование очистки графа

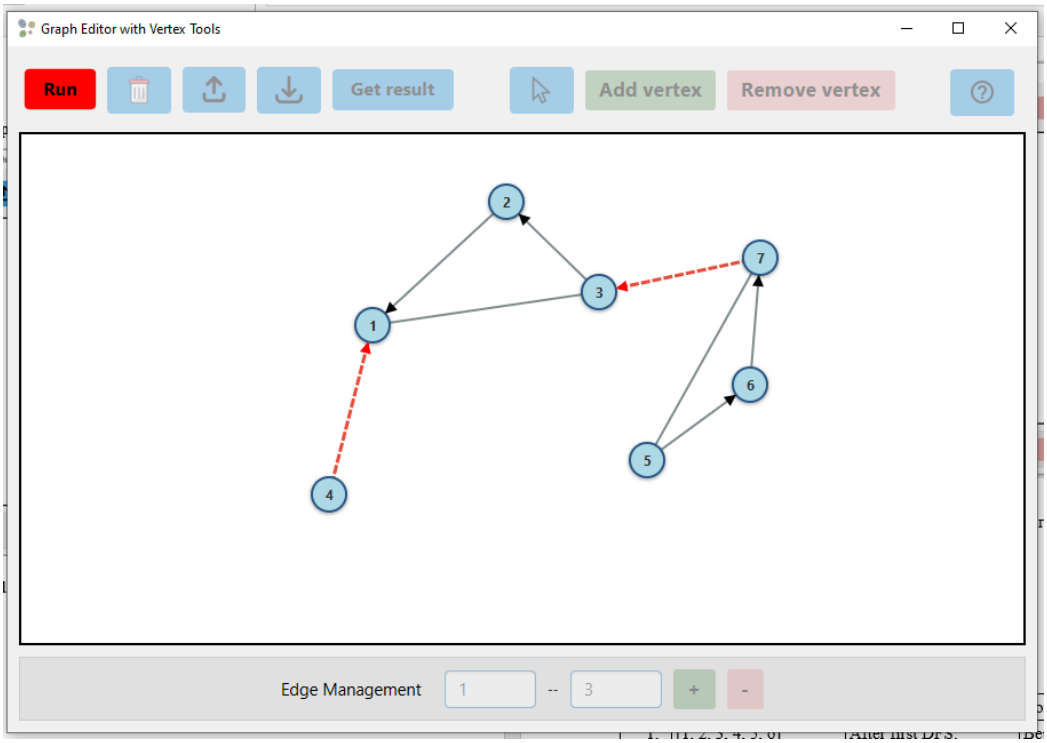

Рисунок 3 – Тестирование визуализации алгоритма

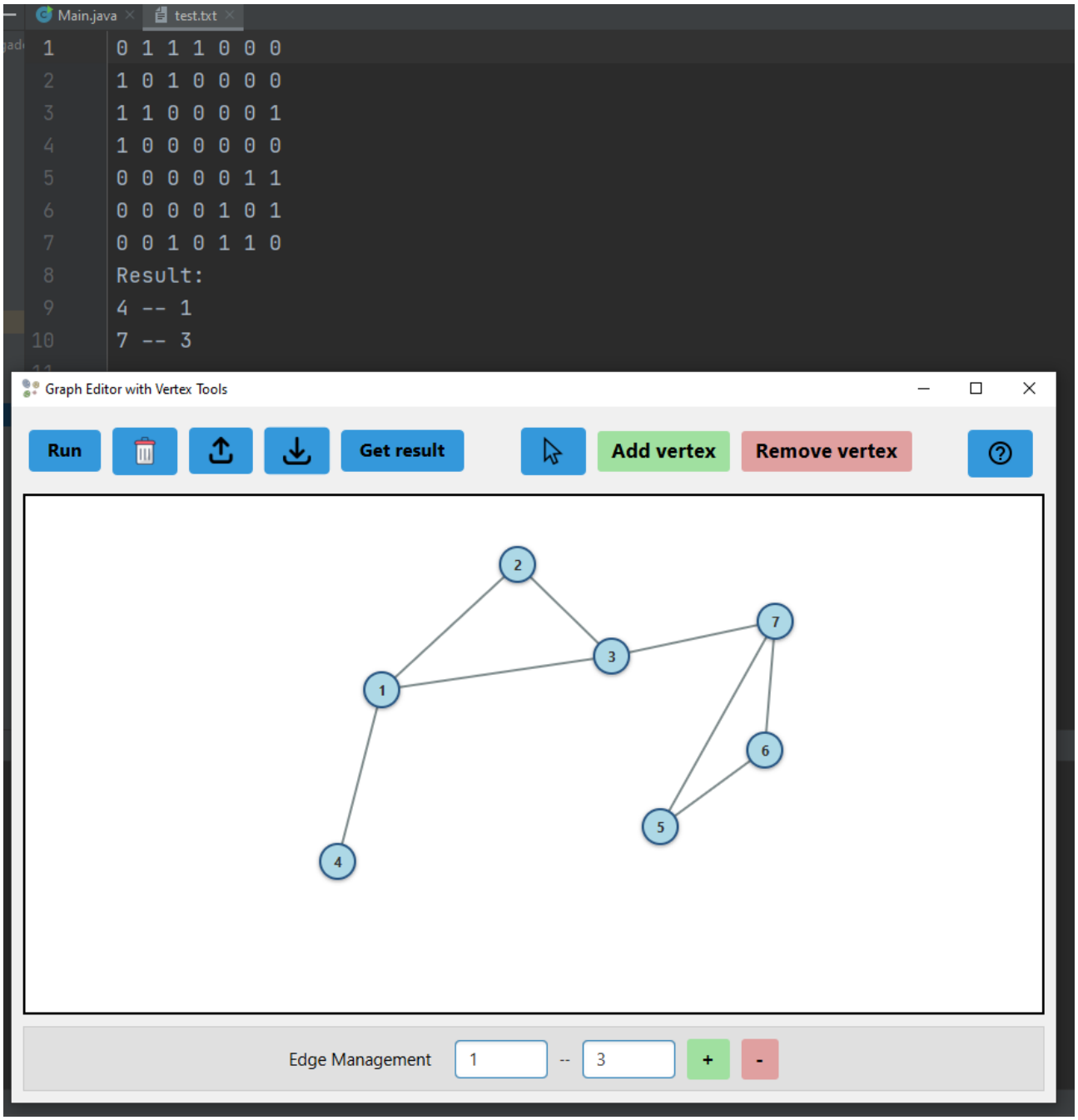

Рисунок 4 – Тестирование сохранения графа

4.2. Тестирование кода алгоритма

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	[1, 2, 3, 4, 5, 6] {2=[1, 4], 5=[1], 4=[2, 3, 6], 3=[1, 4], 1=[3, 2, 5], 6=[4]}	After first DFS: {2=[1, 4], 5=[1], 4=[3], 3=[1], 1=[2], 6=[4]} Result:	Верные мосты

		5--1 6--4	
2.	[1, 2, 3, 4, 5, 6] {3=[1, 2, 5], 2=[1, 3], 6=[4, 5], 1=[3, 2], 4=[6, 5], 5=[4, 6, 3]}	After first DFS: {3=[1], 2=[1, 3], 6=[4, 5], 1=[2], 4=[5], 5=[6, 3]} Result: 5--3	Верные мосты
3.	[1, 2, 3, 4, 5, 6, 7, 8, 9] {2=[8, 3, 5], 4=[9, 7], 3=[1, 2, 6], 6=[3], 8=[7, 2], 5=[2], 9=[1, 4], 7=[4, 8], 1=[9, 3]}	After first DFS: {2=[8], 4=[9], 3=[1, 2], 6=[3], 8=[7], 5=[2], 9=[1], 7=[4], 1=[3]} Result: 5--2 6--3	Верные мосты

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы был изучен и применен новый язык программирования Java и его фреймворк JavaFX. В результате бригадной работы с помощью этих технологий было реализовано приложение с графической визуализацией алгоритма по поиску мостов. После распределения ролей и совмещения написанных программ приложение было проверено на работоспособность. Были учтены все замечания и дополненения руководителя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

2. Основы и тонкости языка Java и ООП: https://javarush.com/

3. Базовые знания JavaFX: https://metanit.com/java/javafx/1.1.php?ysclid=mcnwkxbluk607088334

4. Учебное пособие по программированию на языке JAVA / Герасимова Т.В./ 2006.

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

Название файла: Main.java

```
package org.practice.application;

import javafx.application.Application;
import javafx.stage.Stage;
import org.practice.application.view.MainWindow;

public class Main extends  Application{
    @Override
    public void start(Stage primaryStage) {
        new MainWindow(primaryStage);
    }

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

Название файла: VertexView.java

```
package org.practice.application.view;

import javafx.scene.control.Label;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;

public class VertexView {
    private final int id;
    private final Circle circle;
    private final Label label;

    public VertexView(int id, double x, double y, double radius) {
        this.id = id;
        this.circle = new Circle(x, y, radius);
        circle.getStyleClass().add("vertex");
        this.circle.setUserData(id);
        this.label = new Label(String.valueOf(id));
        label.getStyleClass().add("vertex-label");

this.label.layoutXProperty().bind(this.circle.centerXProperty().subtract(this.la
bel.widthProperty().divide(2)));

this.label.layoutYProperty().bind(this.circle.centerYProperty().subtract(this.la
bel.heightProperty().divide(2)));
        this.label.setMouseTransparent(true);
        this.label.setPickOnBounds(false);
    }

    public void highlightCircle(Color color) {
        this.circle.setFill(color);
    }

    public int getId() {
        return id;
    }

    public Circle getCircle() {
        return circle;
    }
```

```
    public Label getLabel() {
        return label;
    }

    public void moveTo(double x, double y) {
        circle.setCenterX(x);
        circle.setCenterY(y);
    }

    public double getVertexPositionX() {
        return circle.getCenterX();
    }

    public double getVertexPositionY() {
        return circle.getCenterY();
    }

    public void clear(){
        circle.getStyleClass().add("vertex");
    }

    @Override
    public String toString() {
        return "VertexView{"
                + "x = " + circle.getCenterX()
                + ", y = " + circle.getCenterY()
                + "}";
    }
}
```

Название файла: ToolbarView.java

```
package org.practice.application.view;

import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.StackPane;

import java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.Map;

public class ToolbarView {
    private final Map<String, ButtonBase> buttons;
    private TextField firstVertexField;
    private TextField secondVertexField;
    private final HBox mainToolbar;
    private final HBox edgeToolbar;
    private final int iconSize;

    public ToolbarView() {
        this.buttons = new HashMap<>();
        this.mainToolbar = createToolbar();
        this.edgeToolbar = createEdgeToolbar();
        this.iconSize = 20;
    }
```

```
    private HBox createToolbar() {
        buttons.put("run", createToggleButton("Run", "run-button"));
        buttons.put("clean", createToggleButtonIcon("Clean", "clean-button"));
        buttons.put("load", createToggleButtonIcon("Load", "load-button"));
        buttons.put("save", createToggleButtonIcon ("Save","save-button"));
        buttons.put("help", createToggleButtonIcon("Help", "help-button"));
        buttons.put("addVertex",  createToggleButton("Add  vertex",  "add-vertex-
button"));
        buttons.put("deleteVertex", createToggleButton("Remove vertex", "delete-
vertex-button"));
        buttons.put("cursor",      createToggleButtonIcon("Cursor",      "cursor-
button"));
        buttons.put("result",    createToggleButton("Get    result",    "result-
button"));

        ToggleGroup vertexToolsGroup = new ToggleGroup();
        String[] keys = {"addVertex", "deleteVertex", "cursor"};
        for (String key : keys) {
            ButtonBase button = buttons.get(key);
            if (button instanceof ToggleButton) {
                ((ToggleButton) button).setToggleGroup(vertexToolsGroup);
            }
        }

        StackPane helpContainer = new StackPane(buttons.get("help"));
        helpContainer.setAlignment(Pos.TOP_RIGHT);
        StackPane.setMargin(buttons.get("help"), new Insets(5, 5, 0, 0));

        HBox leftSection = new HBox(10,
                buttons.get("run"),
                buttons.get("clean"),
                buttons.get("load"),
                buttons.get("save"),
                buttons.get("result"));
        leftSection.setAlignment(Pos.CENTER_LEFT);

        HBox centerSection = new HBox(10,
                buttons.get("cursor"),
                buttons.get("addVertex"),
                buttons.get("deleteVertex"));
        centerSection.setAlignment(Pos.CENTER);

        BorderPane container = new BorderPane();
        container.setLeft(leftSection);
        container.setCenter(centerSection);
        container.setRight(helpContainer);
        container.setPadding(new Insets(5));

        HBox toolbar = new HBox(container);
        toolbar.setAlignment(Pos.CENTER);
        HBox.setHgrow(container, Priority.ALWAYS);

        return toolbar;
    }

    private HBox createEdgeToolbar() {
        firstVertexField = createTextField("Vertex 1", 80);
        secondVertexField = createTextField("Vertex 2", 80);

        buttons.put("addEdge", createToggleButton("+", "add-edge-button"));
        buttons.put("deleteEdge",      createToggleButton("-",      "delete-edge-
button"));
```

```
        Label edgeManagement = new Label("Edge Management");
        edgeManagement.setId("edge-management-label");
        edgeManagement.getStyleClass().add("edge-management-label");

        HBox controls = new HBox(10,
                edgeManagement,
                firstVertexField,
                new Label("--"),
                secondVertexField,
                buttons.get("addEdge"),
                buttons.get("deleteEdge")
        );
        controls.setAlignment(Pos.CENTER);
        controls.setPadding(new javafx.geometry.Insets(10));
        controls.setStyle("-fx-background-color:    #e0e0e0;    -fx-border-color:
#ccc; -fx-border-width: 1;");
        return controls;
    }

    private ImageView loadIcon(String path, int size) throws  IOException{
        InputStream stream = getClass().getResourceAsStream(path);
        if (stream == null) {
            throw new IOException("Icon not found: " + path);
        }

        Image img = new Image(stream);
        ImageView imageView = new ImageView(img);
        imageView.setFitWidth(size);
        imageView.setFitHeight(size);
        return imageView;
    }

    private ToggleButton createToggleButton(String text, String id) {
        ToggleButton button = new ToggleButton(text);
        button.setId(id);
        button.getStyleClass().add(id);
        return button;
    }

    private ToggleButton createToggleButtonIcon(String text, String id) {
        ToggleButton button = new ToggleButton();
        button.setId(id);
        String name = text.toLowerCase() + "_icon.png";
        String path = "/images/" + name;
        try {
            ImageView iconButton = loadIcon(path, iconSize);
            button.setGraphic(iconButton);
            button.setTooltip(new Tooltip(text));
        } catch (IOException exception) {
            System.err.println("Icon load failed: " + exception.getMessage());
            button.setText(text);
        }
        return button;
    }

    private TextField createTextField(String text, int prefWidth) {
        TextField textField = new TextField();
        textField.getStyleClass().add("text-field");
        textField.setPromptText(text);
        textField.setPrefWidth(prefWidth);
        return textField;
    }
```

```
    public Map<String, ButtonBase> getButtons() {
        return buttons;
    }

    public ButtonBase getButton(String key) {
        return buttons.get(key);
    }

    public HBox getToolbar() {
        return mainToolbar;
    }

    public HBox getEdgeToolbar() {
        return edgeToolbar;
    }

    public TextField getFirstVertexField() {
        return firstVertexField;
    }

    public TextField getSecondVertexField() {
        return secondVertexField;
    }
}
```

Название файла: MainWindow.java

```
package org.practice.application.view;

import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.geometry.Insets;
import org.practice.application.controller.GraphEditorController;
import org.practice.application.model.Algorithm;
import org.practice.application.model.Graph;

public class MainWindow {
    private final Stage stage;
    private final Graph graph;
    private final ToolbarView toolbarView;
    private final GraphView graphView;
    private final GraphEditorController controller;
    private final Algorithm alg;

    public MainWindow(Stage primaryStage) {
        this.graph = new Graph();
        this.graphView = new GraphView();
        this.toolbarView = new ToolbarView();
        alg = new Algorithm(graphView);
        graph.setAlg(alg);
        this.controller    =    new    GraphEditorController(graph,    graphView,
toolbarView);
        this.stage = primaryStage;
        Image                    icon                    =                    new
Image(getClass().getResourceAsStream("/images/Bridge_1.png"));

        // Установка иконки
        primaryStage.getIcons().add(icon);
```

```
        VBox root = new VBox(10, toolbarView.getToolbar(),
graphView.getContainer(),toolbarView.getEdgeToolbar());
        root.setPadding(new Insets(10));
        root.setStyle("-fx-background-color: #f0f0f0;");
        VBox.setVgrow(graphView.getContainer(), Priority.ALWAYS);
        Scene scene = new Scene(root, 900, 600);
        scene.getStylesheets().add(
                getClass().getResource("/styles/graph-
style.css").toExternalForm()
        );
        stage.setTitle("Graph Editor with Vertex Tools");
        stage.setScene(scene);
        stage.show();
    }
}
```

Название файла: GraphView.java

```
package org.practice.application.view;

import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.Node;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javafx.scene.shape.Polygon;
import javafx.scene.transform.Rotate;


public class GraphView {
    private final Pane graphPane;
    private final StackPane container;
    private final Map<Integer, VertexView> vertexViewMap;
    private final List<EdgeView> edges;

    private static final Color[] PREDEFINED_COLORS = {
            Color.RED, Color.BLUE, Color.GREEN, Color.YELLOW,
            Color.ORANGE, Color.PURPLE, Color.CYAN, Color.MAGENTA
    };

    public GraphView() {
        graphPane = new Pane();
        graphPane.setPrefSize(600, 400);
        graphPane.setStyle("-fx-border-color: black; -fx-border-width: 2; -fx-
background-color: white;");

        container = new StackPane(graphPane);
        container.setPrefSize(600, 400);
        vertexViewMap = new HashMap<>();
        edges = new ArrayList<>();
    }

    public void addVertex(int id, double x, double y, double radius) {
        VertexView vertexView = new VertexView(id, x, y, radius);
        vertexViewMap.put(id, vertexView);
        graphPane.getChildren().addAll(vertexView.getCircle(),
vertexView.getLabel());
    }

    public void deleteVertex(int id) {
```

```
        VertexView removeVertexView = vertexViewMap.get(id);
        if (removeVertexView != null) {
            deleteAllEdgesByVertex(id);
            graphPane.getChildren().removeAll(removeVertexView.getCircle(),
removeVertexView.getLabel());
            vertexViewMap.remove(id);
        }
    }

    public void addEdge(int firstId, int secondId) {
        VertexView from = vertexViewMap.get(firstId);
        VertexView to = vertexViewMap.get(secondId);
        if (from == null || to == null) {
            return;
        }
        for (EdgeView edgeView : edges) {
            if ((edgeView.getFrom() == from && edgeView.getTo() == to) ||
                    (edgeView.getFrom() == to && edgeView.getTo() == from)) {
                return;
            }

        }
        EdgeView edgeView = new EdgeView(from, to);
        edges.add(edgeView);
        graphPane.getChildren().add(0, edgeView.getLine());
    }

    public void deleteEdge(int firstId, int secondId) {
        VertexView from = vertexViewMap.get(firstId);
        VertexView to = vertexViewMap.get(secondId);
        if (from == null || to == null) {
            return;
        }

        EdgeView edgeToRemove = null;
        for (EdgeView edge : edges) {
            if ((edge.getFrom() == from && edge.getTo() == to) ||
                    (edge.getFrom() == to && edge.getTo() == from)) {
                edgeToRemove = edge;
                break;
            }
        }
        if (edgeToRemove != null) {
            if (edgeToRemove.getLine().getUserData() != null) {
                Node arrow = (Node) edgeToRemove.getLine().getUserData();
                graphPane.getChildren().remove(arrow);  //  Удаляем  стрелку  с
панели
            }
            graphPane.getChildren().remove(edgeToRemove.getLine());
            edges.remove(edgeToRemove);
        }
    }

    public void deleteAllEdgesByVertex(int id) {
        VertexView removeVertexView = vertexViewMap.get(id);
        List<EdgeView> edgesToRemove = new ArrayList<>();
        for (EdgeView edge : edges) {
            if  (edge.getFrom()  ==  removeVertexView  ||  edge.getTo()  ==
removeVertexView) {
                edgesToRemove.add(edge);
            }
        }
        for (EdgeView edge : edgesToRemove) {
```

```
            if (edge.getLine().getUserData() != null) {
                Node arrow = (Node) edge.getLine().getUserData();
                graphPane.getChildren().remove(arrow);  //  Удаляем  стрелку  с
панели
            }
            graphPane.getChildren().remove(edge.getLine());
            edges.remove(edge);
        }
    }

    public void highlight(int id, Color color) {
        VertexView vertex = vertexViewMap.get(id);
        vertex.highlightCircle(color);
    }

    public void cleanSurface() {
        graphPane.getChildren().clear();
        vertexViewMap.clear();
        edges.clear();
    }

    public void showInfo() {
        vertexViewMap.forEach((id, view) ->
            System.out.println("id: " + id + ", vertexView: " + view)
        );
        edges.forEach(System.out::println);
    }
    public void moveToVertexView(int vertexViewId, double x, double y) {
        VertexView vertexView = vertexViewMap.get(vertexViewId);
        if (vertexView != null) {
            vertexView.moveTo(x, y);
        }
    }

    public Pair<Double, Double> getVertexPosition(int vertexViewId) {
        VertexView vertexView = vertexViewMap.get(vertexViewId);
        return            new            Pair<>(vertexView.getVertexPositionX(),
vertexView.getVertexPositionY());
    }

    public StackPane getContainer() {
        return container;
    }

    public Pane getGraphPane() {
        return graphPane;
    }

    private Pair<Integer,EdgeView> findEdge(int firstId, int secondId) {
        VertexView from = vertexViewMap.get(firstId);
        VertexView to = vertexViewMap.get(secondId);
        if (from == null || to == null) {
            return null;
        }

        EdgeView edgeToFind = null;
        for (EdgeView edge : edges) {
            if   ((edge.getFrom()   ==   from   &&   edge.getTo()   ==   to)   ||
(edge.getFrom() == to && edge.getTo() == from)) {
                edgeToFind = edge;
                break;
            }
        }
```

```
        if (edgeToFind.getFrom() == from && edgeToFind.getTo() == to) {
            return new Pair<>(1, edgeToFind);
        }
        if (edgeToFind.getFrom() == to && edgeToFind.getTo() == from) {
            return new Pair<>(-1, edgeToFind);
        }
        return null;
    }

    public void drawDirection(int firstId, int secondId) {
        VertexView from = vertexViewMap.get(firstId);

        Pair<Integer,EdgeView> edgeViewPair = findEdge(firstId, secondId);
        EdgeView edgeToDirection = edgeViewPair.getRight();
        int numberChooseDirection = edgeViewPair.getLeft();
        if (edgeToDirection != null) {
            addArrowToEdge(edgeToDirection, from, numberChooseDirection);
        }
    }


    private void addArrowToEdge(EdgeView edge, VertexView target, int number) {

        Polygon arrow = new Polygon();
        arrow.getPoints().addAll(
                0.0, 0.0,
                10.0, -5.0,
                10.0, 5.0
        );
        arrow.setFill(Color.BLACK);


        double dx = edge.getLine().getEndX() - edge.getLine().getStartX();
        double dy = edge.getLine().getEndY() - edge.getLine().getStartY();
        double length = Math.sqrt(dx*dx + dy*dy);


        dx /= length;
        dy /= length;


        double arrowX = target.getCircle().getCenterX() + number * dx *
target.getCircle().getRadius();
        double arrowY = target.getCircle().getCenterY() + number * dy *
target.getCircle().getRadius();

        arrow.setLayoutX(arrowX);
        arrow.setLayoutY(arrowY);



        double angle = Math.toDegrees(Math.atan2(dy, dx)) + (180 * ((number == -
1) ? 1 : 0));


        Rotate rotation = new Rotate(angle, 0, 0);
        arrow.getTransforms().add(rotation);


        graphPane.getChildren().add(arrow);


        edge.getLine().setUserData(arrow);
```

```
    }


    public void drawBridges(int firstId, int secondId) {

        EdgeView bridge = findEdge(firstId, secondId).getRight();
        if (bridge != null) {
            if (bridge.getLine().getUserData() != null) {
                Polygon arrow = (Polygon) bridge.getLine().getUserData();
                arrow.setFill(Color.RED);
            }
            bridge.getLine().getStyleClass().add("bridge");
        }
    }


    public void drawVertex(int id, int color) {
        VertexView vertex = vertexViewMap.get(id);
        vertex.getCircle().setFill(getColorByNumber(color));
    }

    public void clearAfterAlgorithm(){
        for (VertexView ver: vertexViewMap.values())
            ver.clear();
    }

    public static Color getColorByNumber(int number) {
        return PREDEFINED_COLORS[number % PREDEFINED_COLORS.length];
    }
}
```

Название файла: Vertex.java

```
package org.practice.application.model;

public class Vertex {
    private int id;
    private int color;
    private boolean isFirstDfs;
    private boolean isSecondDfs;

    public Vertex(int id) {
        this.id = id;
        this.color = -1;
        this.isFirstDfs = false;
        this.isSecondDfs = false;
    }

    public int getId() {
        return id;
    }

    public int getColor() {
        return this.color;
    }

    public boolean getStateFisrtDFS() {
        return this.isFirstDfs;
    }

    public boolean getStateSecondDFS() {
        return this.isSecondDfs;
    }
```

```
    public void firstDFS() {
        this.isFirstDfs = true;
    }

    public void secondDFS() {
        this.isSecondDfs = true;
    }

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

    public void clearVertex(){
        this.color = -1;
        this.isFirstDfs = false;
        this.isSecondDfs = false;
    }

    @Override
    public String toString() {
        return Integer.toString(id);
    }

}
```

Название файла: GraphFileReader.java

```
package org.practice.application.model;

import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;

public class GraphFileReader {
    private HashMap<Vertex, ArrayList<Vertex>> graph;
    private ArrayList<Vertex> vertex;
    private File file;

    public GraphFileReader(File file){
        int count_of_vertex = -1;
        this.file = file;
        this.vertex = new ArrayList<Vertex>();
        this.graph = new HashMap<Vertex, ArrayList<Vertex>>();

        if (!this.file.exists() || !this.file.isFile())
            throw new FileException("Incorrect filename or format!");

        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            int i = 0;
            boolean readingMatrix = true;
            while ((line = br.readLine()) != null) {
                if (line.trim().equals("Result:")) {
                    readingMatrix = false;
                    continue;
                }

                if (readingMatrix) {
                    String[] neighbours = line.split(" ");
                    if (count_of_vertex == -1) {
                        count_of_vertex = neighbours.length;
                        for (int j = 0; j < count_of_vertex; j++)
                            vertex.add(new Vertex(j + 1));
                    }
```

```
                    else{
                        if (count_of_vertex != neighbours.length)
                            throw new FileException("Incorrect matrix!");
                    }
                    graph.put(vertex.get(i), new ArrayList<Vertex>());
                    for (int j = 0; j < neighbours.length; j++){
                        if (neighbours[j].equals("1"))
                            graph.get(vertex.get(i)).add(vertex.get(j));
                    }
                    i++;
                }
            }

            if (count_of_vertex != i)
                throw new FileException("Incorrect matrix!");
        } catch (IOException e) {
            throw new FileException(e.getMessage());
        }
    }

    public ArrayList<Vertex> getVertex(){
        return this.vertex;
    }

    public HashMap<Vertex, ArrayList<Vertex>> getGraph(){
        return this.graph;
    }
}
```

Название файла: GraphEditorContext.java

```
package org.practice.application.model;

import org.practice.application.controller.states.EditorState;

public class GraphEditorContext {
    private EditorState currentState;
    private final EditorStateData stateData;

    public GraphEditorContext(EditorStateData stateData) {
        this.stateData = stateData;
    }

    public void transitToState(EditorState newState) {
        if (currentState != null) {
            currentState.exitState();
        }
        currentState = newState;
        currentState.openState();
    }

    public EditorState getState() {
        return currentState;
    }
}
```

Название файла: Graph.java

```
package org.practice.application.model;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Array;
```

```
import java.util.*;

public class Graph {
    private final Map<Integer, Vertex> vertices;
    private final List<Edge> edges;
    private int nextAvailableId;
    private Algorithm alg;
    public Graph() {
        this.vertices = new HashMap<>();
        this.edges = new ArrayList<>();
        this.nextAvailableId = 1;
    }

    public void setAlg(Algorithm alg) {
        this.alg = alg;
    }

    public void addVertex(int vertexId) {
        nextAvailableId = Math.max(vertexId + 1, nextAvailableId);
        vertices.put(vertexId, new Vertex(vertexId));
    }

    public void deleteVertex(int vertexId) {
        vertices.remove(vertexId);
        deleteAllEdgesByVertex(vertexId);
    }

    private void deleteAllEdgesByVertex(int vertexId) {
        List<Edge> edgesToRemove = new ArrayList<>();
        for (Edge edge : edges) {
            if (edge.getFromId() == vertexId || edge.getToId() == vertexId) {
                edgesToRemove.add(edge);
            }
        }
        edges.removeAll(edgesToRemove);
    }

    public void addEdge(int firstVertexId, int secondVertexId) {
        if (!hasVertex(firstVertexId) || !hasVertex(secondVertexId)) {
            System.out.println("first or second vertex doesn't exist");
            return;
        }
        if (hasEdge(firstVertexId, secondVertexId)) {
            return;
        }
        edges.add(new                          Edge(vertices.get(firstVertexId),
vertices.get(secondVertexId)));
    }

    public void deleteEdge(int firstVertexId, int secondVertexId) {
        if (!hasEdge(firstVertexId, secondVertexId)) {
            System.out.println("Edge doesn't exist");
            return;
        }
        edges.removeIf(edge    ->    edge.getToId()    ==    firstVertexId    &&
edge.getFromId() == secondVertexId);
        edges.removeIf(edge    ->    edge.getToId()    ==    secondVertexId    &&
edge.getFromId() == firstVertexId);
    }

    public int getNextAvailableVertexId() {
        return nextAvailableId++;
    }
```

```
    public boolean hasVertex(int vertexId) {
        return vertices.containsKey(vertexId);
    }

    public boolean hasEdge(int firstVertexId, int secondVertexId) {
        if (!hasVertex(firstVertexId) || !hasVertex(secondVertexId)) {
            return false;
        }
        Edge checkEdge = null;
        for (Edge edge : edges) {
            if  ((edge.getFromId()  ==  firstVertexId  &&    edge.getToId()  ==
secondVertexId)
                ||  (edge.getFromId()  ==  secondVertexId  &&  edge.getToId()  ==
firstVertexId)) {
                checkEdge = edge;
            }
        }
        return checkEdge != null;
    }

    public void clear() {
        vertices.clear();
        edges.clear();
        nextAvailableId = 1;
    }

    public void execute(){
        HashMap<Vertex, ArrayList<Vertex>> mapOfVertex = new HashMap<>();
        ArrayList<Vertex> arrayOfVertex = new ArrayList<>();
        for (Vertex vertex: this.vertices.values()) {
            arrayOfVertex.add(vertex);
            mapOfVertex.put(vertex, new ArrayList<>());
        }
        for (Edge edge: this.edges){
            mapOfVertex.get(edge.getFirstVertex()).add(edge.getSecondVertex());
            mapOfVertex.get(edge.getSecondVertex()).add(edge.getFirstVertex());
        }

        System.out.println(arrayOfVertex.toString());
        System.out.println(mapOfVertex.toString());

        alg.setGraph(mapOfVertex);
        alg.setVertex(arrayOfVertex);
        ArrayList<Vertex[]> result = alg.findBridges();
    }

    public void saveGraph(String filename){
        try(FileWriter writer = new FileWriter(filename, false))
        {
            HashMap<Vertex, ArrayList<Vertex>> mapOfVertex = new HashMap<>();
            ArrayList<Vertex> arrayOfVertex = new ArrayList<>();
            for (Vertex vertex: this.vertices.values()) {
                arrayOfVertex.add(vertex);
                mapOfVertex.put(vertex, new ArrayList<>());
            }
            for (Edge edge: this.edges){

mapOfVertex.get(edge.getFirstVertex()).add(edge.getSecondVertex());

mapOfVertex.get(edge.getSecondVertex()).add(edge.getFirstVertex());
            }
            System.out.println(arrayOfVertex);
```

```
        for (int i = 0; i < arrayOfVertex.size(); i++) {
            String rowStr = "";
            int[] rowInt = new int[arrayOfVertex.size()];
            for (Vertex ver : mapOfVertex.get(arrayOfVertex.get(i))) {
                rowInt[arrayOfVertex.indexOf(ver)] = 1;
            }
            for (int j = 0; j < rowInt.length; j++){
                if (j == rowInt.length - 1){
                    rowStr += rowInt[j];
                    break;
                }
                rowStr += rowInt[j];
                rowStr += " ";
            }
            System.out.println(rowStr);
            writer.write(rowStr);
            writer.write('\n');
        }
        writer.flush();
    }
    catch(IOException ex){

        System.out.println(ex.getMessage());
    }
}

public void saveResult(String filename) {
    saveGraph(filename);
    try (FileWriter writer = new FileWriter(filename, true)) {
        getRusult();
        ArrayList<Vertex[]> bridges = alg.getListBridges();
        writer.write("Result:\n");
        for (Vertex[] bridge : bridges) {
            writer.write(bridge[0] + " -- " + bridge[1] + "\n");
        }
        if (bridges.isEmpty()) {
            writer.write("No bridges");
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}

public void clearAlg(){
    alg.clear();
}

public ArrayList<Vertex> getArrayOfVertex(){
    ArrayList<Vertex> array = new ArrayList<>();
    array.addAll(this.vertices.values());
    return array;
}

public void stopAlg(){
    alg.stop();
}

public void getRusult() {
    HashMap<Vertex, ArrayList<Vertex>> mapOfVertex = new HashMap<>();
    ArrayList<Vertex> arrayOfVertex = new ArrayList<>();
    for (Vertex vertex: this.vertices.values()) {
        arrayOfVertex.add(vertex);
        mapOfVertex.put(vertex, new ArrayList<>());
```

```
        }
        for (Edge edge: this.edges){
            mapOfVertex.get(edge.getFirstVertex()).add(edge.getSecondVertex());
            mapOfVertex.get(edge.getSecondVertex()).add(edge.getFirstVertex());
        }

        alg.setGraph(mapOfVertex);
        alg.setVertex(arrayOfVertex);
        alg.getResultFast();
    }
}
```

Название файла: Algorithm.java

```
package org.practice.application.model;

import javafx.concurrent.Task;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Stack;
import javafx.application.Platform;
import org.practice.application.view.GraphView;

public class Algorithm {
    private HashMap<Vertex, ArrayList<Vertex>> graph;
    private ArrayList<Vertex> vertex;
    private ArrayList<Vertex[]> result;
    private ArrayList<Vertex> arrayAfterFirstDFS;
    private final GraphView view;
    private Thread taskOfAlg;


    public Algorithm(GraphView view){
        this.result = new ArrayList<Vertex[]>();
        this.arrayAfterFirstDFS = new ArrayList<Vertex>();
        this.view = view;
    }

    public void setGraph(HashMap<Vertex, ArrayList<Vertex>> graph) {
        this.graph = graph;
    }

    public void setVertex(ArrayList<Vertex> vertex) {
        this.vertex = vertex;
    }

    public ArrayList<Vertex[]> findBridges() {
        Task<Void> task = new Task<Void>() {
            @Override
            protected Void call() throws Exception {

                firstDFS(true);
                System.out.println(graph.toString());

                secondDFS(true);

                for (Vertex ver1 : vertex) {
                    for (Vertex ver2 : graph.get(ver1)) {
                        if (ver1.getColor() != ver2.getColor()) {

                            result.add(new Vertex[]{ver1, ver2});

                            Platform.runLater(() -> {
```

```
                                view.drawBridges(ver1.getId(), ver2.getId());
                            });

                        }
                    }
                }
                System.out.println("Result:");
                for (Vertex[] pair: result){
                    System.out.println(pair[0] + "--" + pair[1]);
                }
                return null;
            }
        };
        task.setOnSucceeded(event -> {
            view.clearAfterAlgorithm();
            for (Vertex ver: vertex)
                ver.clearVertex();
        });
        taskOfAlg = new Thread(task);
        taskOfAlg.start();
        return this.result;
    }

    private void firstDFS(boolean isStep){
        Stack<Vertex> stack = new Stack<Vertex>();
        Vertex nextVertex = this.findUnviewedVertexFisrtDFS();
        OUTER_LOOP:
        while (nextVertex != null){
            stack.push(nextVertex);
            arrayAfterFirstDFS.add(nextVertex);
            nextVertex.firstDFS();
            while (!stack.empty()) {
                for (int i = 0; i < graph.get(nextVertex).size(); i++) {
                    if (!graph.get(nextVertex).get(i).getStateFisrtDFS()) {
                        Vertex neighbour = graph.get(nextVertex).get(i);
                        graph.get(nextVertex).remove(i);

                        Vertex finalNextVertex = nextVertex;

                        if (isStep) {
                            Platform.runLater(() -> {
                                view.drawDirection(finalNextVertex.getId(),
neighbour.getId());
                            });


                            try {
                                Thread.sleep(1000);
                            } catch (InterruptedException e) {
                                Thread.currentThread().interrupt();
                                return;
                            }

                        }

                        nextVertex = neighbour;
                        continue OUTER_LOOP;
                    }
                }
                stack.pop();
                if (stack.empty())
                    break;
                nextVertex = stack.peek();
```

```
        }
        nextVertex = findUnviewedVertexFisrtDFS();
    }
}

private void secondDFS(boolean isStep){
    int color = 1;
    Stack<Vertex> stack = new Stack<Vertex>();
    Vertex nextVertex = this.findUnviewedVertexSecondDFS();
    OUTER_LOOP:
    while (nextVertex != null){
        stack.push(nextVertex);
        nextVertex.secondDFS();
        nextVertex.setColor(color);


        Vertex finalNextVertex = nextVertex;
        int finalColor = color;
        if (isStep) {
            Platform.runLater(() -> {
                view.drawVertex(finalNextVertex.getId(), finalColor);
            });

            // Пауза для визуализации
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                return;
            }
        }


        while (!stack.empty()) {
            for (int i = 0; i < graph.get(nextVertex).size(); i++) {
                if (!graph.get(nextVertex).get(i).getStateSecondDFS()) {
                    Vertex neighbour = graph.get(nextVertex).get(i);
                    graph.get(nextVertex).remove(i);
                    nextVertex = neighbour;
                    continue OUTER_LOOP;
                }
            }
            stack.pop();
            if (stack.empty())
                break;
            nextVertex = stack.peek();
        }
        color++;
        nextVertex = findUnviewedVertexSecondDFS();
    }
}

private Vertex findUnviewedVertexFisrtDFS(){
    for (int i = 0; i < vertex.size(); i++){
        if (!vertex.get(i).getStateFisrtDFS())
            return vertex.get(i);
    }
    return null;
}

private Vertex findUnviewedVertexSecondDFS(){
    for (int i = 0; i < arrayAfterFirstDFS.size(); i++){
        if (!arrayAfterFirstDFS.get(i).getStateSecondDFS())
```

```
                return arrayAfterFirstDFS.get(i);
            }
            return null;
        }

        public void clear(){
            graph.clear();
            vertex.clear();
            arrayAfterFirstDFS.clear();
            result.clear();
        }

        public void stop(){
            taskOfAlg.interrupt();
        }

        public void getResultFast() {


            firstDFS(false);
            secondDFS(false);

            for (Vertex ver1 : vertex) {
                for (Vertex ver2 : graph.get(ver1)) {
                    if (ver1.getColor() != ver2.getColor()) {
                        result.add(new Vertex[]{ver1, ver2});
                        view.drawBridges(ver2.getId(), ver1.getId());

                    }
                }
            }
            System.out.println("Result:");
            for (Vertex[] pair: result){
                System.out.println(pair[0] + "--" + pair[1]);
            }

            for (Vertex ver: vertex)
                ver.clearVertex();
        }

        public ArrayList<Vertex[]> getListBridges() {
            return result;
        }
    }
```

Название файла: GraphEditorController.java

```
package org.practice.application.controller;

import javafx.scene.Node;
import javafx.scene.control.ButtonBase;
import javafx.scene.control.ToggleButton;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import org.practice.application.controller.states.*;
import org.practice.application.model.*;
import org.practice.application.view.GraphView;
import org.practice.application.view.ToolbarView;

import java.util.Map;

public class GraphEditorController {
    private final GraphEditorContext context;
```

```
    private final VertexController vertexController;
    private final EdgeController edgeController;
    private final FileController fileController;

    private AddState addState;
    private DeleteState deleteState;
    private DragState dragState;
    private LoadState loadState;
    private SaveState saveState;
    private CleanState cleanState;
    private HelpState helpState;
    private RunState runState;
    private GetResultState getResultState;

    private static final Color[] PREDEFINED_COLORS = {
            Color.RED, Color.BLUE, Color.GREEN, Color.YELLOW,
            Color.ORANGE, Color.PURPLE, Color.CYAN, Color.MAGENTA
    };

    private final ToolbarView toolbarView;
    private final Graph graph;
    private final GraphView graphView;

    public GraphEditorController(Graph graph, GraphView graphView, ToolbarView
toolbarView) {
        this.toolbarView = toolbarView;
        this.graph = graph;
        this.graphView = graphView;

        EditorStateData stateData = new EditorStateData(graph, graphView, 15,
Color.ORANGE);
        this.vertexController = new VertexController(stateData);
        this.edgeController = new EdgeController(stateData);
        this.fileController = new FileController(stateData);

        this.context = new GraphEditorContext(stateData);

        this.addState   =   new   AddState(vertexController,   edgeController,
stateData);
        this.deleteState = new DeleteState(vertexController);
        this.dragState = new DragState(vertexController);
        this.saveState = new SaveState(fileController);
        this.loadState = new LoadState(fileController);
        this.cleanState = new CleanState(stateData);
        this.helpState = new HelpState(stateData);
        this.runState = new RunState(stateData);
        this.getResultState = new GetResultState(stateData);

        context.transitToState(dragState);
        setUpToolbarActions();
        setUpMouseHandlers();
    }

    private void setUpToolbarActions() {
        toolbarView.getButton("run").setOnAction(event -> toggleEditMode());
        toolbarView.getButton("clean").setOnAction(event                    ->
context.transitToState(cleanState));
        toolbarView.getButton("help").setOnAction(event                     ->
context.transitToState(helpState));
        toolbarView.getButton("load").setOnAction(event                     ->
context.transitToState(loadState));
        toolbarView.getButton("save").setOnAction(event                     ->
context.transitToState(saveState));
```

```
        toolbarView.getButton("addVertex").setOnAction(event                    ->
context.transitToState(addState));
        toolbarView.getButton("deleteVertex").setOnAction(event                 ->
context.transitToState(deleteState));
        toolbarView.getButton("cursor").setOnAction(event                       ->
context.transitToState(dragState));
        toolbarView.getButton("addEdge").setOnAction(event                      ->
handleAddEdge(toolbarView.getFirstVertexField().getText(),
toolbarView.getSecondVertexField().getText()));
        toolbarView.getButton("deleteEdge").setOnAction(event                   ->
handleDeleteEdge(toolbarView.getFirstVertexField().getText(),
toolbarView.getSecondVertexField().getText()));
        toolbarView.getButton("result").setOnAction(event -> resultEditMode());
    }

    private void resetControlsEnabled() {
        for (ButtonBase button: toolbarView.getButtons().values()) {
            button.setDisable(false);
            if (button instanceof ToggleButton) {
                ((ToggleButton) button).setSelected(false);
            }
        }
        toolbarView.getFirstVertexField().setDisable(false);
        toolbarView.getSecondVertexField().setDisable(false);
    }

    private void toggleEditMode() {
        boolean isInRunMode = context.getState() instanceof RunState;
        if (isInRunMode) {
            context.transitToState(dragState);
            resetControlsEnabled();
        } else {
            context.transitToState(runState);
            setEditModeControlsEnabled("run",false);
        }
    }

    private void resultEditMode() {
        boolean isGetResult = context.getState() instanceof GetResultState;
        if (isGetResult) {
            context.transitToState(dragState);
            resetControlsEnabled();
        } else {
            context.transitToState(getResultState);
            setEditModeControlsEnabled("result",false);
        }
    }

    private  void  setEditModeControlsEnabled(String  activeButtonKey,  boolean
enabled) {
        for         (Map.Entry<String,         ButtonBase>         entry         :
toolbarView.getButtons().entrySet()) {
            String key = entry.getKey();
            ButtonBase button = entry.getValue();
            button.setDisable(!key.equals(activeButtonKey));
        }
        toolbarView.getFirstVertexField().setDisable(!enabled);
        toolbarView.getSecondVertexField().setDisable(!enabled);
    }

    private void handleAddEdge(String firstVertex, String secondVertex) {
        edgeController.addEdge(firstVertex, secondVertex);
    }
```

```
    private void handleDeleteEdge(String firstVertex, String secondVertex) {
        edgeController.deleteEdge(firstVertex, secondVertex);
    }

    private void setUpMouseHandlers() {
        graphView.getGraphPane().setOnMouseClicked(event -> {
            if (event.getTarget() == graphView.getGraphPane()) {
                context.getState().handleClickOnPane(event.getX(),
event.getY());
            } else if (event.getTarget() instanceof Circle) {
                Circle circle = (Circle) event.getTarget();
                int vertexId = (Integer) circle.getUserData();
                context.getState().handleClickOnVertex(vertexId);
            }
        });

        graphView.getGraphPane().setOnMousePressed(event -> {
            context.getState().handleMousePressed(event.getX(),    event.getY(),
(Node) event.getTarget());
        });

        graphView.getGraphPane().setOnMouseDragged(event -> {
            if (context.getState() instanceof DragState) {
                context.getState().handleMouseDragged(event.getX(),
event.getY(), (Node) event.getTarget());
            }
        });

        graphView.getGraphPane().setOnMouseReleased(event -> {
            context.getState().handleMouseReleased(event.getX(),   event.getY(),
(Node) event.getTarget());
        });
    }
}
```

Название файла: EditorState.java

```
package org.practice.application.controller.states;

import javafx.scene.Node;

public interface EditorState {
    default void openState() {}
    default void exitState() {}
    default void handleClickOnVertex(int vertexId) {}
    default void handleClickOnPane(double x, double y) {}
    default void handleMouseDragged(double x, double y, Node target) {}
    default void handleMouseReleased(double x, double y, Node target) {}
    default void handleMousePressed(double x, double y, Node target) {}
}
```