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

ОТЧЕТ

по учебной практике

Тема: Топологическая сортировка

Студент гр. 3342 _____ Иванов С.С.

Студент гр. 3342 _____ Гончаров С.А.

Студент гр. 3342 _____ Мохамед М.Х.

Студент гр. 3388 _____ Березовский М.А.

Руководитель _____ Шестопалов Р.П.

Санкт-Петербург

2025

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Иванов С.С. группы 3342

Студент Гончаров С.А. группы 3342

Студент Мохамед М.Х. группы 3342

Студент Березовский М.А. группы 3388

Тема практики: топологическая сортировка, реализация на языке программирования Java

Задание на практику:

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

Алгоритм: Топологическая сортировка.

Сроки прохождения практики: 25.06.2024 – 08.07.2024

Дата сдачи отчета: 05.07.2024

Дата защиты отчета: 05.07.2024

Студент		Иванов С.С.
Студент		Гончаров С.А.
Студент		Мохамед М.Х.
Студент		Березовский М.А.
Руководитель		Шестопалов Р.П.

АННОТАЦИЯ

Проект представляет собой интерактивную программу, реализующую алгоритм топологической сортировки ориентированного ациклического графа (DAG). Пользователь может загрузить его из файла. Интерфейс предусматривает кнопку «Загрузить из файла», позволяющую автоматически считать граф в виде списка ребер. Для визуализации выполнения алгоритма пользователь может воспользоваться кнопками “Вперед” и “Назад”, при этом на каждом этапе будет отображаться текущая вершина и обновленный порядок. Кнопка «Выполнить сразу» позволяет запустить сортировку целиком и получить результат мгновенно.

Граф отображается в виде визуальной схемы с пронумерованными вершинами. По завершении выполнения сортировки отображается финальный порядок обхода и граф, аннотированный результатом. Кнопка «Завершить» закрывает программу. Взаимодействие с программой осуществляется через графический интерфейс, обеспечивая наглядность и удобство изучения алгоритма. Визуальная часть программы будет выполнена с использованием библиотеки JavaFX.

SUMMARY

The project is an interactive program implementing the algorithm of topological sorting for a directed acyclic graph (DAG). The user can input the graph manually through the interface or load it from a file. The interface includes a "Load from File" button, which allows the graph to be automatically read in the form of an edge list. For visualizing the execution of the algorithm, the user can use the "Step-by-Step Execution" button, which displays the current vertex and the updated order at each stage. The "Execute Immediately" button allows the algorithm to run entirely and display the result instantly.

The graph is presented as a visual diagram with vertices numbered according to the sorting order. Upon completion of the sorting, the final traversal order and the graph annotated with the result are displayed. The "Exit" button closes the program. Interaction with the program is carried out through a graphical interface, providing clarity and convenience for studying the algorithm. The visual part of the program is implemented using the JavaFX library.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Изучить и реализовать алгоритм **топологической сортировки** ориентированного ациклического графа (DAG), а также понять его практическое применение в задачах планирования и управления зависимостями.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

(Спецификация)

Проект представляет собой интерактивную программу, реализующую алгоритм топологической сортировки ориентированного ациклического графа (DAG). Взаимодействие с программой осуществляется через графический интерфейс, обеспечивая наглядность и удобство изучения алгоритма. Визуальная часть программы будет выполнена с использованием библиотеки JavaFX.

1.1.1 Требования к вводу исходных данных

Программа должна обеспечивать возможность загрузки графа из файла. Формат файла должен быть понятен пользователю и поддерживать описание ориентированных графов.

1.1.2 Требования к визуализации

Визуализация графа должна осуществляться с использованием библиотеки JavaFX. Граф должен отображаться на экране в наглядной форме, позволяющей пользователю видеть структуру вершин и рёбер.

1.1.3 Требования к выполнению алгоритма

Программа должна реализовывать алгоритм топологической сортировки для ориентированного графа. Должна быть реализована возможность пошагового выполнения алгоритма с отображением текущего состояния сортировки на каждом шаге. Должна быть реализована возможность мгновенного выполнения алгоритма до конца по нажатию соответствующей кнопки.

1.2. Уточнение требований после сдачи прототипа

В процессе пошагового выполнения алгоритма топологической сортировки в отдельной области интерфейса должно отображаться текущее состояние сортировки в текстовом виде (например: 1 → 2 → 3 → 4). Это позволяет пользователю отслеживать ход выполнения алгоритма не только визуально, но и в письменной форме.

В программе должна быть реализована возможность масштабирования окна с визуализацией графа. Пользователь должен иметь возможность увеличивать и уменьшать масштаб отображения графа для удобства просмотра структуры и деталей.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Дата	Этап проекта	Реализованные возможности	Выполнено
27.06.25	Согласование спецификации	План разработки Роли в бригаде	Написали план разработки, распределились по ролям в команде
02.07.25	Сдача прототипа	Первый прототип программы	Показали работоспособность программы, получили замечания по визуальной части.
02.07.25	Сдача версии 1	Рабочая программа Исправили замечания с прошлой защиты	Получили замечания по масштабированию окна с визуализацией графа, добавить текстовое представление сортировки
05.07.25	Сдача финальной версии + отчёт	Рабочая программа Отчёт к проделанной работе	Создан отчёт

2.2. Распределение ролей в бригаде

Студент	Роль	Что делал	Что выполнено
Иванов С.С.	Разработчик	Занимался разработкой логической части программы, разработал установщик, оформление репозитория на Github, а также совместно с Березовским М.А. реализовал основную логику приложения.	Разработана логика программы топологической сортировки, оформлен Github репозиторий в соответствии с требованиями, помогал другим участникам бригады
Березовский М.А.	Разработчик	Работал в паре с Ивановым С.С. над созданием логики программы, а также активно помогал в создании графического интерфейса.	Разработана логика программы, помогал всем участникам бригады.
Гончаров С.А.	Разработчик	Разработал графический интерфейс пользователя GUI программы в паре с Мохамедом М.Х. занимался редактированием и оформлением отчета к проекту.	Разработан GUI интерфейс программы, составлен отчёт к проделанной работе.
Мохамед М.Х.	Разработчик	Был в паре с Гончаровым С.А. помогал реализовывать GUI интерфейс программы.	Помогал другим участникам бригады, помогал в разработке GUI интерфейса, написании отчёта.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

В программе реализованы следующие основные классы:

Graph — класс для представления ориентированного графа, хранения вершин и рёбер, а также выполнения операций над графом.

Vertex — класс, описывающий вершину графа, её идентификатор и связанные свойства.

TopologicalSorter — класс, реализующий алгоритм топологической сортировки и управляющий процессом пошагового выполнения.

GraphParser — класс для загрузки и парсинга графа из файла.

Controller — класс-контроллер для взаимодействия между пользовательским интерфейсом и логикой приложения.

ExtendedIterator — вспомогательный класс-итератор для пошагового выполнения алгоритма.

IConstGraph, IGraph — интерфейсы для абстракции работы с графом.

3.2. Основные методы

Graph.addVertex(Vertex v) — добавляет вершину в граф.

Graph.addEdge(Vertex from, Vertex to) — добавляет ребро между двумя вершинами.

Graph.getVertices() — возвращает список всех вершин графа.

Graph.getEdges() — возвращает список всех рёбер графа.

GraphParser.parse(File file) — загружает и парсит граф из файла.

TopologicalSorter.sort() — выполняет топологическую сортировку графа.

TopologicalSorter.nextStep() — выполняет следующий шаг сортировки (для пошагового режима).

Controller.onLoadGraph() — обработчик загрузки графа из файла.

Controller.onStep() — обработчик пошагового выполнения сортировки.

Controller.onRunAll() — обработчик полного выполнения сортировки.

Controller.onZoomIn()/onZoomOut() — обработчики масштабирования окна с графом.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование приложения

Мы протестировали приложение на графе с большим количеством вершин, программа успешно выполняет сортировку (Рис. 1)

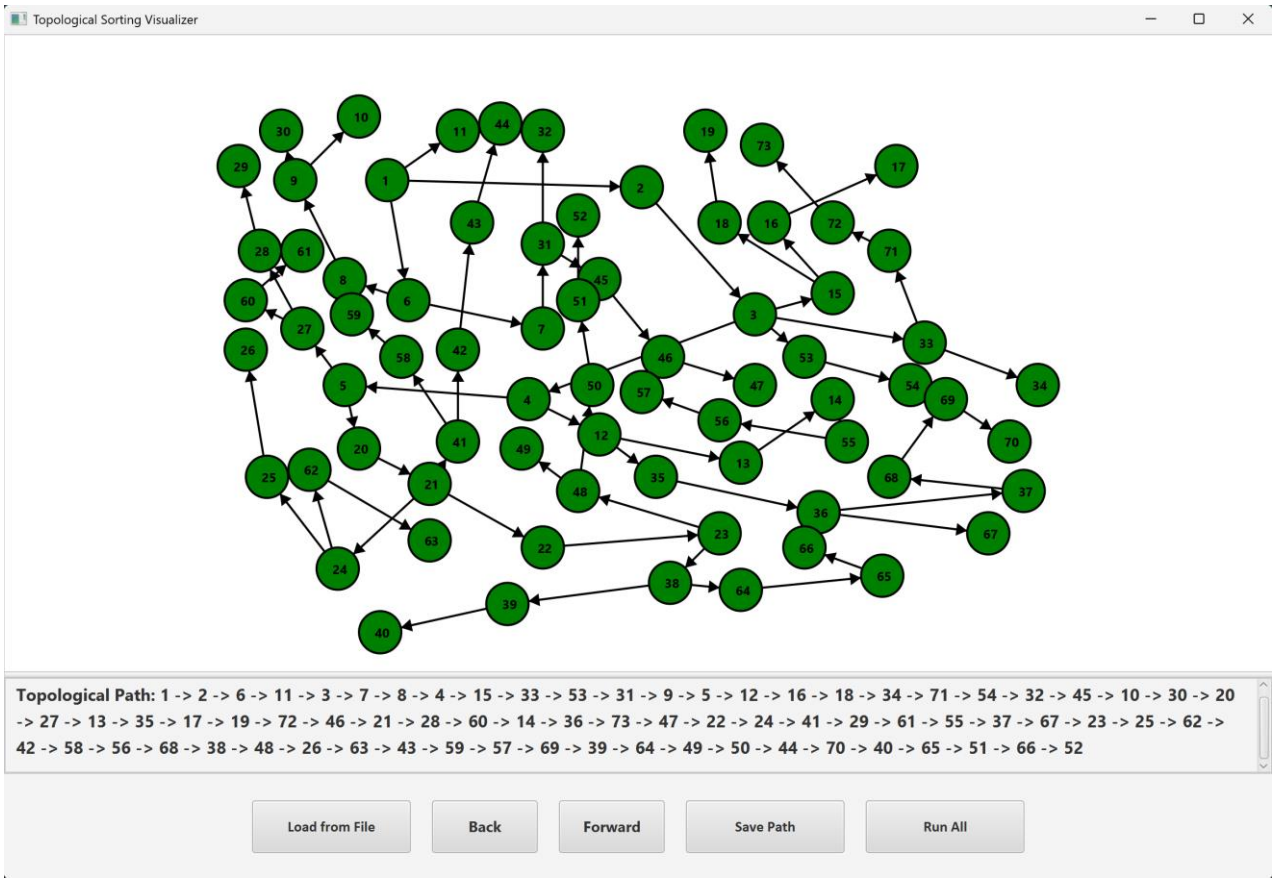

Рисунок 1 - граф с большим количеством вершин

4.2. Тестирование кода алгоритма

В качестве тестирования был использован неправильно построенный граф (Рис. 2)

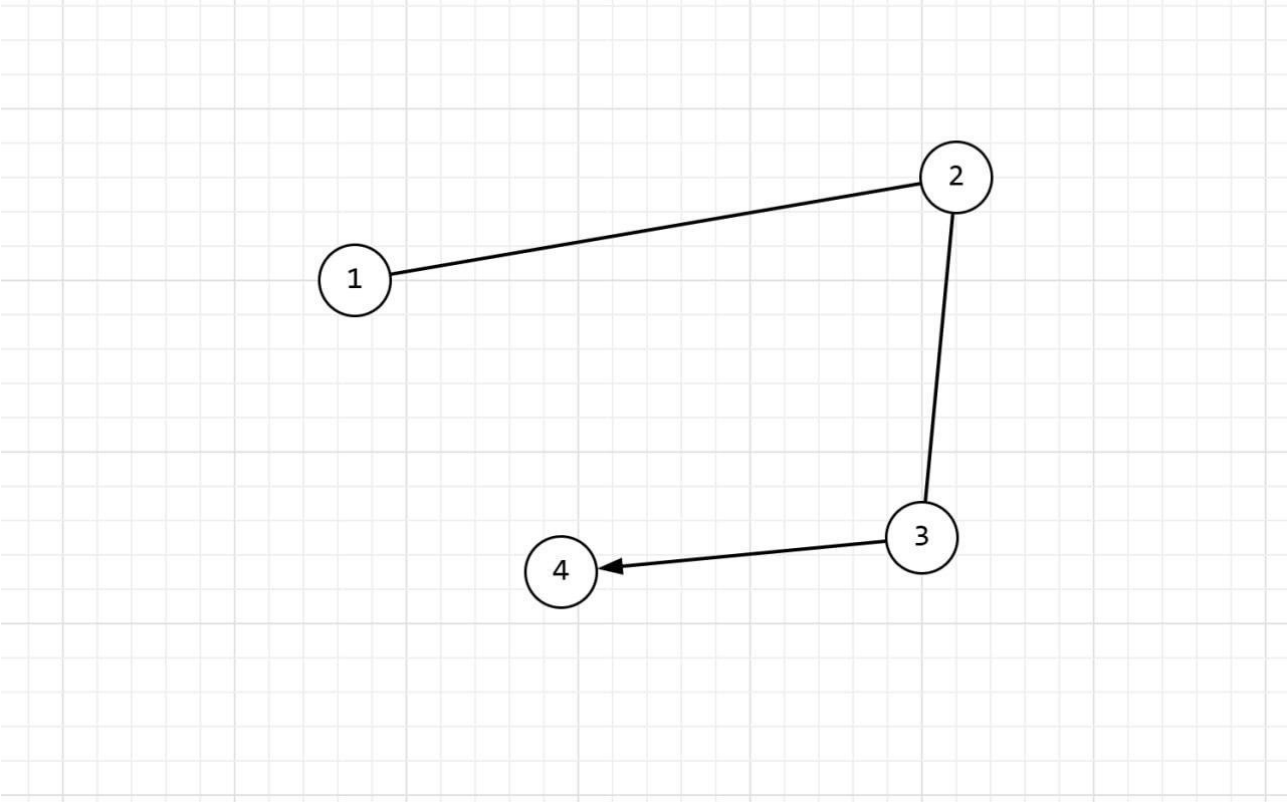

Рисунок 2 - неправильный граф

Если попытаться выполнить сортировку, то программа выведет сообщение об ошибке (Рис. 3)

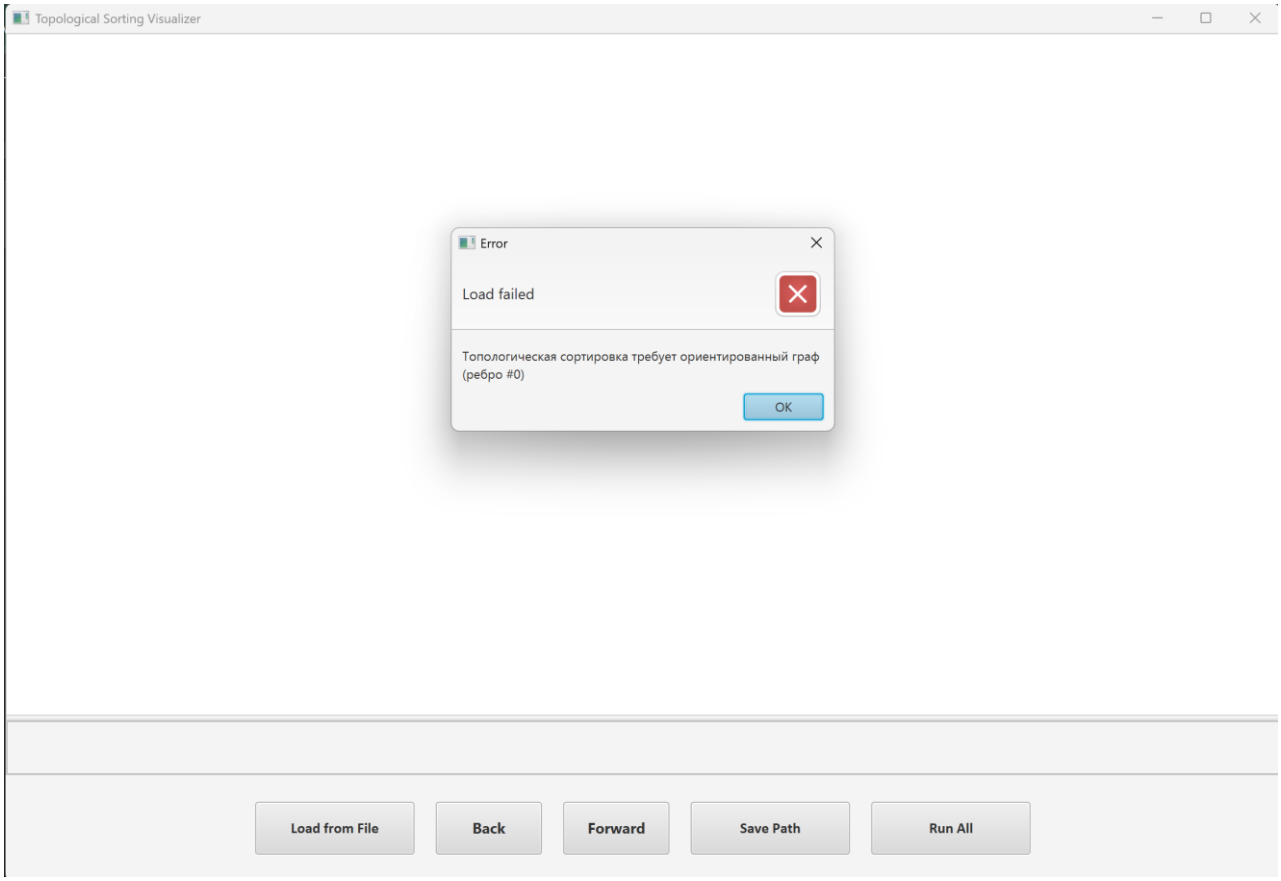

Рисунок 3 - сообщение об ошибке

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была разработана программа для визуализации топологической сортировки ориентированного графа с использованием библиотеки JavaFX. Реализованы функции загрузки графа из файла, пошагового и полного выполнения алгоритма, а также отображение текущего состояния сортировки в текстовом виде. Добавлена возможность масштабирования окна с графом для удобства пользователя.

Поставленные цели и требования были полностью выполнены. Программа соответствует заявленным исходным и уточненным требованиям, обеспечивает наглядную визуализацию процесса топологической сортировки и удобный пользовательский интерфейс.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация по Java. [Электронный ресурс]. URL: https://docs.oracle.com/javase/ (дата обращения: 27.06.2025).
2. Официальная документация по JavaFX. [Электронный ресурс]. URL: https://openjfx.io/ (дата обращения: 27.06.2025).
3. Gson — библиотека для работы с JSON / под ред. Google Inc. [Электронный ресурс]. URL: https://github.com/google/gson (дата обращения: 30.06.2025).
4. Stack Overflow — вопросы и ответы по программированию. [Электронный ресурс]. URL: https://stackoverflow.com/ (дата обращения: 01.07.2025).
5. Хабр — публикации по программированию и Java. [Электронный ресурс]. URL: https://habr.com/ru/hub/java/ (дата обращения: 01.07.2025).
6. JavaFX: библиотека для создания графических интерфейсов / И. И. Иванов, П. П. Петров, С. С. Сидоров и др. СПб.: БХВ-Петербург, 2019. 480 с.

ПРИЛОЖЕНИЕ А
TOPOLOGICAL - SORTING

MainApplication.fxml:

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

<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.SplitPane?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.text.Font?>
<?import javafx.geometry.Insets?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
        prefHeight="800.0" prefWidth="1200.0"
        xmlns="http://javafx.com/javafx/21"
        xmlns:fx="http://javafx.com/fxml/1"
        fx:controller="Controller">

  <children>
    <VBox fx:id="mainContainer" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
      <children>
        <!-- Область для рисования графа и контейнера пути-->
        <SplitPane fx:id="mainSplitPane" orientation="VERTICAL" dividerPositions="0.95"
                VBox.vgrow="ALWAYS">
          <items>
            <!-- Верхняя часть - граф -->
            <ScrollPane fitToWidth="true" fitToHeight="true"
                    hbarPolicy="AS_NEEDED" vbarPolicy="AS_NEEDED"
                    minViewportWidth="600" minViewportHeight="400"
                    style="-fx-background-color: #f0f0f0;">
              <Pane fx:id="graphCanvas" minWidth="800" minHeight="600"
                  style="-fx-background-color: white;" />
            </ScrollPane>

            <!-- Нижняя часть - контейнер пути -->
            <VBox minHeight="50" >
              <HBox fx:id="pathContainer" style="-fx-background-color: #f0f0f0;"
                  VBox.vgrow="ALWAYS">
                <ScrollPane fitToWidth="true" hbarPolicy="NEVER" vbarPolicy="AS_NEEDED"
                        HBox.hgrow="ALWAYS">
                  <Label fx:id="pathLabel" wrapText="true"
                        style="-fx-font-size: 16px; -fx-font-weight: bold; -fx-padding: 10px;"/>
                </ScrollPane>
              </HBox>
            </VBox>
          </items>
        </SplitPane>

        <!-- Панель кнопок -->
        <HBox alignment="CENTER" prefHeight="100.0" prefWidth="1200.0"
            spacing="20.0" style="-fx-padding: 10;">
          <children>
            <Button fx:id="OnLoadFromFile" mnemonicParsing="false"
                  onAction="#onLoadFromFileClick" prefHeight="50.0" prefWidth="150.0"
```

```
                text="Load from File">
              <font>
                <Font name="Calibri Bold" size="14.0" />
              </font>
            </Button>

            <Button fx:id="BackButton" mnemonicParsing="false"
                onAction="#onBackButtonClick" prefHeight="50.0" prefWidth="100.0"
                text="Back">
              <font>
                <Font name="System Bold" size="14.0" />
              </font>
            </Button>

            <Button fx:id="ForwardButton" mnemonicParsing="false"
                onAction="#onForwardButtonClick" prefHeight="50.0" prefWidth="100.0"
                text="Forward">
              <font>
                <Font name="System Bold" size="14.0" />
              </font>
            </Button>

            <!-- Кнопка для сохранения пути -->
            <Button fx:id="SavePathButton" mnemonicParsing="false"
                onAction="#onSavePathClick" prefHeight="50.0" prefWidth="150.0"
                text="Save Path">
              <font>
                <Font name="Calibri Bold" size="14.0" />
              </font>
            </Button>

            <Button fx:id="RunImmediately" mnemonicParsing="false"
                onAction="#onRunImmediatelyClick" prefHeight="50.0" prefWidth="150.0"
                text="Run All">
              <font>
                <Font name="Calibri Bold" size="14.0" />
              </font>
            </Button>
          </children>
        </HBox>
      </children>
    </VBox>
  </children>
</AnchorPane>
```

Controller.java:

```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.SplitPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.shape.Polygon;
import javafx.scene.text.Text;
```

```
import javafx.stage.FileChooser;
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.HashMap;
import java.util.Map;

public class Controller {

    @FXML private VBox mainContainer;
    @FXML private SplitPane mainSplitPane;
    @FXML private HBox graphContainer;
    @FXML private Pane graphCanvas;
    @FXML private HBox pathContainer;

    // Кнопки
    @FXML private Button BackButton;
    @FXML private Button ForwardButton;
    @FXML private Button OnLoadFromFile;
    @FXML private Button RunImmediately;
    @FXML private Button SavePathButton;

    @FXML private Label pathLabel;

    private Graph graph;
    private TopologicalSorter sorter;
    private final Map<Vertex, Circle> vertexNodes = new HashMap<>();
    private double scale = 1.0;
    private double offsetX = 0;
    private double offsetY = 0;

    @FXML
    void onLoadFromFileClick(ActionEvent event) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open Graph JSON");
        fileChooser.getExtensionFilters().addAll(
            new FileChooser.ExtensionFilter("Graph Files (*.json, *.graph)", "*.json", "*.graph"),
            new FileChooser.ExtensionFilter("JSON Files", "*.json"),
            new FileChooser.ExtensionFilter("Graph Files", "*.graph"),
            new FileChooser.ExtensionFilter("All Files", "*.*")
        );
        File file = fileChooser.showOpenDialog(null);

        if (file != null) {
            try {
                Path path = file.toPath();
                graph = GraphParser.parseFromFile(path);
                sorter = new TopologicalSorter(graph);
                calculateScaling();
                drawGraph();
                updatePathLabel();
            } catch (Exception e) {
                showAlert(AlertType.ERROR, "Error", "Load failed", e.getMessage());
            }
        }
    }

    // Метод для сохранения пути в файл
    @FXML
    void onSavePathClick(ActionEvent event) {
```

```
    if (sorter == null || sorter.getSortedSoFar().isEmpty()) {
        showAlert(AlertType.WARNING, "Warning", "No path", "Nothing to save - sort the graph first");
        return;
    }

    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Save Path to File");
    fileChooser.getExtensionFilters().add(
        new FileChooser.ExtensionFilter("Text Files", "*.txt")
    );
    File file = fileChooser.showSaveDialog(null);

    if (file != null) {
        try {
            // Формируем содержимое файла в формате "1 -> 2 -> 3"
            StringBuilder content = new StringBuilder();
            for (int i = 0; i < sorter.getSortedSoFar().size(); i++) {
                Vertex v = sorter.getSortedSoFar().get(i);
                content.append(v.getName());
                if (i < sorter.getSortedSoFar().size() - 1) {
                    content.append(" -> ");
                }
            }

            // Записываем в файл
            Path path = file.toPath();
            Files.write(path, content.toString().getBytes());

            showAlert(AlertType.INFORMATION, "Success", "Path saved",
                    "Path was successfully saved to: " + path);
        } catch (Exception e) {
            showAlert(AlertType.ERROR, "Error", "Save failed", e.getMessage());
        }
    }
}

private void calculateScaling() {
    if (graph == null || graph.getVertices().isEmpty()) return;

    double minX = Double.MAX_VALUE;
    double maxX = Double.MIN_VALUE;
    double minY = Double.MAX_VALUE;
    double maxY = Double.MIN_VALUE;

    for (Vertex v : graph.getVertices()) {
        minX = Math.min(minX, v.getX());
        maxX = Math.max(maxX, v.getX());
        minY = Math.min(minY, v.getY());
        maxY = Math.max(maxY, v.getY());
    }

    double graphWidth = maxX - minX;
    double graphHeight = maxY - minY;

    // Получаем реальные размеры с учетом ScrollPane
    double canvasWidth = graphCanvas.getWidth();
    double canvasHeight = graphCanvas.getHeight();

    // Добавляем 20% отступ для раннего появления скролла
```

```
    double paddingFactor = 0.2;
    double scaleX = (canvasWidth > 0 && graphWidth > 0)
            ? (canvasWidth * (1 - paddingFactor)) / graphWidth : 1;
    double scaleY = (canvasHeight > 0 && graphHeight > 0)
            ? (canvasHeight * (1 - paddingFactor)) / graphHeight : 1;

    scale = Math.min(scaleX, scaleY) * 0.95; // Добавляем небольшой отступ

    // Центрирование графа
    offsetX = (canvasWidth - graphWidth * scale) / 2 - minX * scale;
    offsetY = (canvasHeight - graphHeight * scale) / 2 - minY * scale;
}

private double transformX(double x) {
    return offsetX + x * scale;
}

private double transformY(double y) {
    return offsetY + y * scale;
}

private void drawGraph() {
    graphCanvas.getChildren().clear();
    vertexNodes.clear();

    // Рисуем рёбра со стрелками
    for (Vertex vertex : graph.getVertices()) {
        double x1 = transformX(vertex.getX());
        double y1 = transformY(vertex.getY());

        for (Vertex neighbor : vertex.getNeighbors()) {
            double x2 = transformX(neighbor.getX());
            double y2 = transformY(neighbor.getY());

            // Рассчитываем вектор направления
            double dx = x2 - x1;
            double dy = y2 - y1;
            double length = Math.sqrt(dx * dx + dy * dy);

            if (length < 1e-6) continue;

            // Нормализуем вектор
            dx /= length;
            dy /= length;

            // Отступаем от конечной точки
            double endX = x2 - 20 * dx;
            double endY = y2 - 20 * dy;

            // Отрисовка линию
            Line edgeLine = new Line(x1, y1, endX, endY);
            edgeLine.setStrokeWidth(2);
            graphCanvas.getChildren().add(edgeLine);

            // Отрисовка стрелку
            drawArrow(endX, endY, dx, dy);
        }
    }
```

```
    // Рисуем вершины
    for (Vertex vertex : graph.getVertices()) {
        double x = transformX(vertex.getX());
        double y = transformY(vertex.getY());

        Circle node = new Circle(x, y, 20);
        node.setStyle("-fx-fill: lightgray; -fx-stroke: black; -fx-stroke-width: 2;");

        Text label = new Text(x - 5, y + 5, vertex.getName());
        label.setStyle("-fx-font-weight: bold;");

        graphCanvas.getChildren().addAll(node, label);
        vertexNodes.put(vertex, node);
    }
}

// Метод для рисования стрелки
private void drawArrow(double endX, double endY, double dx, double dy) {
    double arrowLength = 12;
    double arrowAngle = Math.toRadians(30);

    double x1 = endX - arrowLength * (dx * Math.cos(arrowAngle) + dy * Math.sin(arrowAngle));
    double y1 = endY - arrowLength * (dy * Math.cos(arrowAngle) - dx * Math.sin(arrowAngle));

    double x2 = endX - arrowLength * (dx * Math.cos(-arrowAngle) + dy * Math.sin(-arrowAngle));
    double y2 = endY - arrowLength * (dy * Math.cos(-arrowAngle) - dx * Math.sin(-arrowAngle));

    Polygon arrowHead = new Polygon();
    arrowHead.getPoints().addAll(endX, endY, x1, y1, x2, y2);
    arrowHead.setStyle("-fx-fill: black;");

    graphCanvas.getChildren().add(arrowHead);
}

@FXML
void onForwardButtonClick(ActionEvent event) {
    if (sorter == null) {
        showAlert(AlertType.WARNING, "Warning", "No graph", "Load a graph first");
        return;
    }

    if (sorter.hasNext()) {
        Vertex current = sorter.next();
        highlightVertex(current, "green");
        updatePathLabel();
    } else {
        showAlert(AlertType.INFORMATION, "Complete", "Sorting finished", "All vertices processed");
    }
}

@FXML
void onBackButtonClick(ActionEvent event) {
    if (sorter == null) {
        showAlert(AlertType.WARNING, "Warning", "No graph", "Load a graph first");
        return;
    }

    if (!sorter.getSortedSoFar().isEmpty()) {
        Vertex current = sorter.prev();
```

```
            highlightVertex(current, "lightgray");
            updatePathLabel();
        }
    }

    @FXML
    void onRunImmediatelyClick(ActionEvent event) {
        if (sorter == null) {
            showAlert(AlertType.WARNING, "Warning", "No graph", "Load a graph first");
            return;
        }

        while (sorter.hasNext()) {
            Vertex current = sorter.next();
            highlightVertex(current, "green");
        }
        updatePathLabel();
    }

    private void highlightVertex(Vertex vertex, String color) {
        Circle node = vertexNodes.get(vertex);
        if (node != null) {
            node.setStyle("-fx-fill: " + color + "; -fx-stroke: black; -fx-stroke-width: 2;");
        }
    }

    // Обновление отображения пути
    private void updatePathLabel() {
        if (sorter == null || sorter.getSortedSoFar().isEmpty()) {
            pathLabel.setText("Topological Path: ");
            return;
        }

        StringBuilder path = new StringBuilder("Topological Path: ");
        for (Vertex v : sorter.getSortedSoFar()) {
            path.append(v.getName()).append(" -> ");
        }
        path.setLength(path.length() - 4); // Удаляем последний " -> " (4 символа)
        pathLabel.setText(path.toString());
    }

    private void showAlert(AlertType type, String title, String header, String content) {
        Alert alert = new Alert(type);
        alert.setTitle(title);
        alert.setHeaderText(header);
        alert.setContentText(content);
        alert.showAndWait();
    }
}


ExtendedIterator.java:
import java.util.Iterator;

public interface ExtendedIterator<T> extends Iterator<T> {
    T prev();
}

Graph.java:
import java.util.*;
```

```
/**
 * Реализация направленного графа с возможностью добавления вершин и рёбер
 */
public class Graph implements IGraph {
    private List<Vertex> vertices;

    /**
     * Создает пустой граф
     */
    public Graph() {
        this(new ArrayList<>());
    }

    /**
     * Создает граф с заданным списком вершин
     *
     * @param vertices список вершин графа
     */
    public Graph(List<Vertex> vertices) {
        this.vertices = vertices;
    }

    /**
     * Добавляет вершину в граф
     *
     * @param vertex вершина для добавления
     * @throws IllegalArgumentException если вершина уже существует в графе
     */
    @Override
    public void addVertex (Vertex vertex) {
        if (exists(vertex)) {
            throw new IllegalArgumentException("Вершина в графе уже существует");
        }

        vertices.add(vertex);
    }

    /**
     * Добавляет ориентированное ребро от вершины _s к вершине _e
     *
     * @param _s начальная вершина
     * @param _e конечная вершина
     * @throws IllegalArgumentException если хотя бы одна из вершин отсутствует в графе
     */
    @Override
    public void addEdge(Vertex _s, Vertex _e) {
        if (!exists(_s) || !exists(_e)) {
            throw new IllegalArgumentException("Вершины добавляемых рёбер должны существовать в графе");
        }

        // Проверка на существующее ребро
        if (_s.getNeighbors().contains(_e)) {
            throw new IllegalArgumentException(
                "Ребро уже существует: " + _s.getName() + " -> " + _e.getName()
            );
        }

        _s.addNeighbor(_e);
    }

    /**
     * Возвращает вершину по имени
     *
     * @param name имя вершины
```

```
     * @return вершина с заданным именем или null, если такой вершины нет
     */
    @Override
    public Vertex getVertexByName (String name) {
        for (Vertex v : vertices) {
            if (v.getName().equals(name)) {
                return v;
            }
        }
        return null;
    }

    /**
     * Возвращает список всех вершин графа
     *
     * @return немодифицируемый список вершин
     */
    @Override
    public List<Vertex> getVertices() {
        return Collections.unmodifiableList(vertices);
    }

    /**
     * Проверяет, существует ли вершина в графе
     *
     * @param vertex вершина для проверки
     * @return true, если вершина есть в графе, иначе false
     */
    @Override
    public boolean exists(Vertex vertex) {
        for (Vertex v : vertices) {
            if (vertex.equals(v)) {
                return true;
            }
        }
        return false;
    }

    /**
     * Возвращает строковое представление графа
     *
     * @return строка, описывающая граф
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("Graph:\n");
        for (Vertex v : vertices) {
            sb.append(v.toString()).append("\n");
        }
        return sb.toString();
    }

}
```

GraphParser.java:

```
import com.google.gson.Gson;
import com.google.gson.JsonSyntaxException;
import java.io.Reader;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;

public final class GraphParser {
```

```
// Приватный конструктор - утилитарный класс
private GraphParser() {}

/**
 * Парсит граф из JSON-файла
 *
 * @param filePath путь к JSON-файлу
 * @return объект Graph
 * @throws Exception при ошибках чтения/парсинга
 */
public static Graph parseFromFile(Path filePath) throws Exception {
    try (Reader reader = Files.newBufferedReader(filePath)) {
        return parse(reader);
    }
}

/**
 * Парсит граф из Reader
 *
 * @param reader источник JSON-данных
 * @return объект Graph
 * @throws JsonSyntaxException при ошибке синтаксиса JSON
 * @throws IllegalArgumentException при невалидных данных
 */
public static Graph parse(Reader reader) {
    Gson gson = new Gson();
    GraphJsonData graphData = gson.fromJson(reader, GraphJsonData.class);

    // Валидация структуры JSON
    validateGraphData(graphData);

    // Создаем вершины
    List<Vertex> vertices = createVertices(graphData.vertices);
    Graph graph = new Graph(vertices);

    // Добавляем ребра
    addEdges(graph, vertices, graphData.edges);

    return graph;
}

private static void validateGraphData(GraphJsonData graphData) {
    if (graphData.vertices == null) {
        throw new IllegalArgumentException("Отсутствует массив vertices в JSON");
    }
    if (graphData.edges == null) {
        graphData.edges = new ArrayList<>(); // Разрешаем пустые графы
    }
}

private static List<Vertex> createVertices(List<VertexJsonData> verticesData) {
    List<Vertex> vertices = new ArrayList<>(verticesData.size());
    for (int i = 0; i < verticesData.size(); i++) {
        VertexJsonData vData = verticesData.get(i);
        validateVertexData(vData, i);
        vertices.add(new Vertex(vData.name, vData.x, vData.y));
    }
    return vertices;
}

private static void validateVertexData(VertexJsonData vData, int index) {
    if (vData.name == null || vData.name.isEmpty()) {
        throw new IllegalArgumentException(
            "Вершина #" + index + " имеет пустое имя"
        );
```

```
        }
    }

    private static void addEdges(Graph graph, List<Vertex> vertices, List<EdgeJsonData> edgesData) {
        for (int i = 0; i < edgesData.size(); i++) {
            EdgeJsonData eData = edgesData.get(i);
            validateEdgeData(eData, i, vertices.size());

            Vertex source = vertices.get(eData.vertex1);
            Vertex target = vertices.get(eData.vertex2);
            graph.addEdge(source, target);
        }
    }

    private static void validateEdgeData(EdgeJsonData eData, int index, int vertexCount) {

        if (!eData.isDirected) {
            throw new IllegalArgumentException(
                "Топологическая сортировка требует ориентированный граф (ребро #" + index + ")"
            );
        }
        validateVertexIndex(eData.vertex1, index, vertexCount, "vertex1");
        validateVertexIndex(eData.vertex2, index, vertexCount, "vertex2");
    }

    private static void validateVertexIndex(int index, int edgeIndex, int max, String field) {
        if (index < 0 || index >= max) {
            throw new IllegalArgumentException(
                String.format("Неверный индекс вершины %s: %d (допустимо 0-%d) в ребре #%d",
                field, index, max - 1, edgeIndex)
            );
        }
    }

    // Внутренние классы для структуры JSON
    private static class GraphJsonData {
        List<VertexJsonData> vertices;
        List<EdgeJsonData> edges;
    }

    private static class VertexJsonData {
        int x;
        int y;
        String name;
    }

    private static class EdgeJsonData {
        int vertex1;
        int vertex2;
        boolean isDirected;
    }
}
```

IConstGraph.java:

```
import java.util.List;

public interface IConstGraph {
    Vertex getVertexByName (String name);
    List<Vertex> getVertices();
    boolean exists(Vertex vertex);

}
```

IGraph.java:

```
public interface IGraph extends IConstGraph {
```

```
    void addVertex (Vertex vertex);
    void addEdge(Vertex _s, Vertex _e);

}
```

Main.java:

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

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("/MainApplication.fxml"));
        primaryStage.setTitle("Topological Sorting Visualizer");

        // Увеличиваем размеры основного окна
        Scene scene = new Scene(root, 1200, 800);
        primaryStage.setScene(scene);

        primaryStage.setMinWidth(1000);
        primaryStage.setMinHeight(700);
        primaryStage.show();
    }

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

TopologicalSorter.java:

```
import java.util.*;

/**
 * Класс, выполняющий топологическую сортировку графа с возможностью
 * пошагового прохождения вперед и назад.
 */
public class TopologicalSorter implements ExtendedIterator<Vertex> {

    private final Graph graph;
    private final Map<Vertex, Integer> inDegree;
    private final Deque<Vertex> queue;
    private final List<Vertex> result;
    private final Deque<List<Vertex>> addedToQueueHistory;

    /**
     * Создает объект сортировщика для заданного графа.
     * При создании вычисляет степени входа всех вершин
     * и формирует очередь вершин с нулевой степенью.
     *
     * @param graph граф, который будет отсортирован
     */
    public TopologicalSorter(Graph graph) {
        this.graph = graph;
        this.inDegree = new HashMap<>();
        this.queue = new ArrayDeque<>();
        this.result = new ArrayList<>();
        this.addedToQueueHistory = new ArrayDeque<>();

        for (Vertex v : graph.getVertices()) {
            inDegree.put(v, 0);
        }
```

```
    for (Vertex v : graph.getVertices()) {
        for (Vertex u : v.getNeighbors()) {
            inDegree.put(u, inDegree.get(u) + 1);
        }
    }

    for (Map.Entry<Vertex, Integer> entry : inDegree.entrySet()) {
        if (entry.getValue() == 0) {
            queue.addLast(entry.getKey());
        }
    }

}

/**
 * Возвращает неизменяемый доступ к исходному графу.
 *
 * @return интерфейс графа
 */
public IConstGraph getGraph() {
    return graph;
}

/**
 * Проверяет, остались ли ещё вершины, которые можно извлечь из очереди.
 *
 * @return true, если очередная вершина доступна
 */
@Override
public boolean hasNext() {
    return !queue.isEmpty();
}

/**
 * Выполняет следующий шаг сортировки.
 * Удаляет вершину из очереди, уменьшает степени входа её соседей,
 * добавляет их в очередь, если их степень стала нулевой.
 *
 * @return вершина, обработанная на этом шаге
 * @throws NoSuchElementException если больше нет элементов
 */
@Override
public Vertex next() throws NoSuchElementException {
    if (!hasNext()) {
        throw new NoSuchElementException("No more elements");
    }
    return iterateNextStep();
}

/**
 * Реализует шаг вперед в сортировке.
 * Обновляет очередь и историю.
 *
 * @return вершина, извлечённая на этом шаге
 */
private Vertex iterateNextStep() {
    Vertex v = queue.removeFirst();
    result.add(v);

    List<Vertex> addedNow = new ArrayList<>();
    for (Vertex neigh : v.getNeighbors()) {
        int deg = inDegree.get(neigh) - 1;
        inDegree.put(neigh, deg);
        if (deg == 0) {
            queue.addLast(neigh);
```

```
                addedNow.add(neigh);
            }
        }
        addedToQueueHistory.push(addedNow);
        return v;
    }

    /**
     * Выполняет шаг назад.
     * Восстанавливает вершину обратно в очередь,
     * возвращает степень входа её соседей.
     *
     * @return вершина, возвращённая на шаг назад
     * @throws NoSuchElementException если нечего откатывать
     */
    @Override
    public Vertex prev() throws NoSuchElementException {
        if (result.isEmpty()) {
            throw new NoSuchElementException("No previous element to revert");
        }

        return iteratePrevStep();
    }

    /**
     * Реализует логику отката шага сортировки.
     * Обновляет очередь и историю.
     *
     * @return вершина, возвращённая на шаг назад
     */
    private Vertex iteratePrevStep() {
        Vertex v = result.remove(result.size() - 1);
        List<Vertex> addedNow = addedToQueueHistory.pop();
        for (Vertex neighbor : v.getNeighbors()) {
            inDegree.put(neighbor, inDegree.get(neighbor) + 1);
        }

        for (Vertex neighbor : addedNow) {
            queue.remove(neighbor);
        }

        queue.addFirst(v);
        return v;
    }

    /**
     * Возвращает неизменяемый список вершин,
     * которые уже были обработаны (отсортированы).
     *
     * @return список отсортированных вершин
     */
    public List<Vertex> getSortedSoFar() {
        return Collections.unmodifiableList(result);
    }

}
```

Vertex.java:

```
import java.util.*;

/**
 * Представляет вершину графа с координатами и списком смежных вершин
 */
public class  Vertex {
    private final int x, y;
```

```
private final String name;
private final int _hash_code;
private List<Vertex> neighbors;

/**
 * Создает вершину только с именем
 * Координаты устанавливаются в (0,0)
 *
 * @param name имя вершины
 */

public Vertex(String name) {
   this(name, 0, 0);
}

/**
 * Создает вершину с указанным именем и координатами
 *
 * @param name имя вершины
 * @param x координата X
 * @param y координата Y
 */
public Vertex (String name, int x, int y) {
   this.name = name;
   this.x = x;
   this.y = y;
   this._hash_code = _calc_hash_code();
   this.neighbors = new ArrayList<>();
}

/**
 * @return координата X вершины
 */
public int getX() { return x; }

/**
 * @return координата Y вершины
 */
public int getY() { return y; }

/**
 * @return имя вершины
 */
public String getName() { return name; }

/**
 * Возвращает список соседних вершин, в которые есть исходящие рёбра.
 * Список недоступен для изменения извне.
 *
 * @return неизменяемый список соседей
 */
public List<Vertex> getNeighbors() {
   return Collections.unmodifiableList(neighbors);
}

/**
 * Добавляет смежную вершину (рёберное соединение).
 *
 * @param neighbor вершина-сосед
 */
public void addNeighbor(Vertex neighbor) {
   if (neighbors.contains(neighbor)) {
      throw new IllegalArgumentException(
         "Сосед уже существует: " + neighbor.getName() + " для вершины " + this.name
      );
```

```
        }
        neighbors.add(neighbor);
    }

    /**
     * Удаляет указанного соседа из списка смежных вершин.
     *
     * @param neighbor вершина-сосед, которую нужно удалить
     */
    public void removeNeighbor(Vertex neighbor) {
        neighbors.remove(neighbor);
    }

    /**
     * Проверяет равенство вершин по предрассчитанному хеш-коду.
     *
     * @param obj объект для сравнения
     * @return true, если вершины считаются равными
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }

        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }

        Vertex vertex = (Vertex) obj;
        return _hash_code == vertex._hash_code;
    }

    /**
     * @return хеш-код вершины
     */
    @Override
    public int hashCode() {
        return _hash_code;
    }

    /**
     * Вычисляет хеш-код для вершины на основе имени и координат.
     *
     * @return целочисленный хеш-код
     */
    private int _calc_hash_code() {
        int hash = name != null ? name.hashCode() : 0;
        hash = (hash << 16) | (hash >>> 16);
        hash ^= (x << 16) | (x >>> 16);
        hash ^= y;
        return hash;
    }

    /**
     * @return строковое представление вершины и её соседей
     */
    @Override
    public String toString() {
        return String.format("Vertex(%s (%d, %d) -> %s)", name, x, y,
            neighbors.stream().map(Vertex::getName).toList());
    }

}
```