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. 9381 Прашутинский К.И.

Студент гр. 9381 Николаев А.А.

Руководитель Фирсов М.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Прашутинский К.И. группы 9381

Студент Николаев А.А. группы 9381

Тема практики: Топологическая сортировка.

Задание на практику:

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

Алгоритм: Топологическая сортировка.

Сроки прохождения практики: 1.07.2021 – 12.07.2021

Дата сдачи отчета: 12.07.2021

Дата защиты отчета: 14.07.2021

Студент		Прашутинский К.И.
Студент		Николаев А.А.
Руководитель		Фирсов М.А.

АННОТАЦИЯ

Целью текущей учебной практики является разработка GUI приложения для топологической сортировки для заданного графа.

Программа разрабатывается на языке Java командой из двух человек, каждый из которых имеет определенную специализацию. Сдача и показ проекта на определенном этапе выполнения осуществляется согласно плану разработки.

SUMMARY

The goal of the current training practice is to develop a GUI application for topological sorting for a given graph.

The program is developed in Java by a team of two people, each of whom has a specific specialization. The delivery and display of the project at a certain stage of implementation is carried out according to the development plan.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью текущей учебной практики является разработка GUI приложения для топологической сортировки для заданного графа.

Программа предоставляет пользователю структурированный интерфейс для создания и изменения графа. Во время визуализации работы алгоритма Топологической сортировки пользователь может либо сразу вывести результат, либо пошагово идти по алгоритму. Каждый шаг алгоритма имеет графическое отображение.

Алгоритм Топологической сортировки представляет собой

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Программа представляет собой визуализацию алгоритма Топологической сортировки для ориентированного графа без циклов.

1.1.1. Требования к вводу исходных данных

Программа предоставляет пользователю следующий интерфейс для создания графа:

- Чтения графа из файла
- Изменения текущего графа

1.1.2. Требования к визуализации

Графический интерфейс предоставляет пользователю следующие методы для изменения графа:

- Добавить вершину
- Удалить вершину
- Переместить вершину
- Удалить вершину

Пользователь имеет возможность применить алгоритм Топологической сортировки для построенного графа, если он удовлетворяет условию отсутствия циклов. Алгоритм Топологической сортировки нахождения для заданного графа имеет два способа реализации:

- Вывод результата алгоритма
- Пошаговая работа алгоритма

Если выбран вариант пошаговой визуализации алгоритма, пользователю предоставляется возможность самому управлять последовательность выполнения алгоритма: он может либо «откатиться» к предыдущей итерации, либо пройти к следующей.

Таким образом, программа предлагает следующий интерфейс для прохода по алгоритму:

- Вперед (не доступен на последнем шаге)
- Назад (не доступен на первом шаге)

1.1.3. Внешний вид приложения

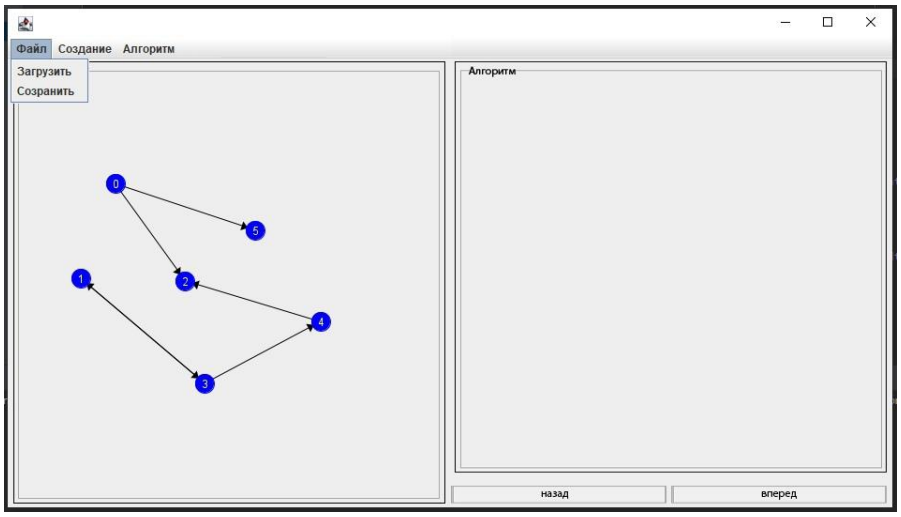

Рис. 1 – Внешний вид приложения (1)

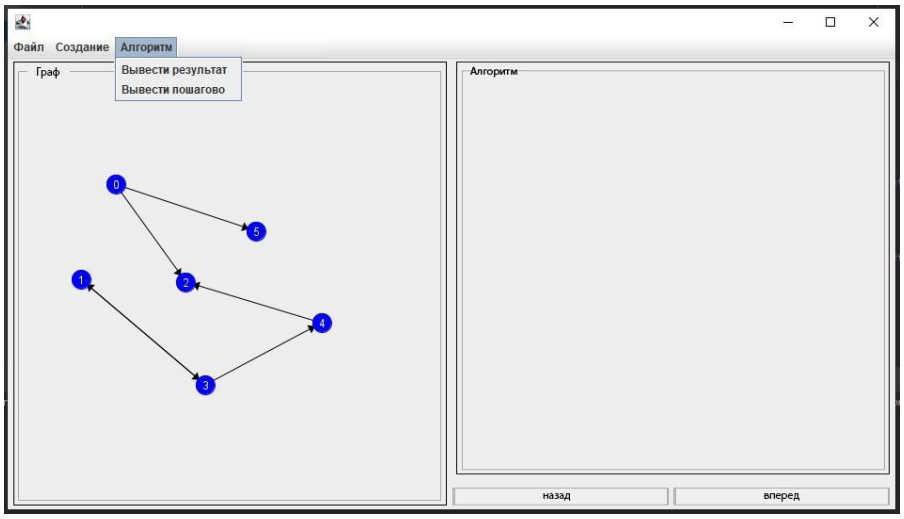

Рис. 2 – Внешний вид приложения (2)

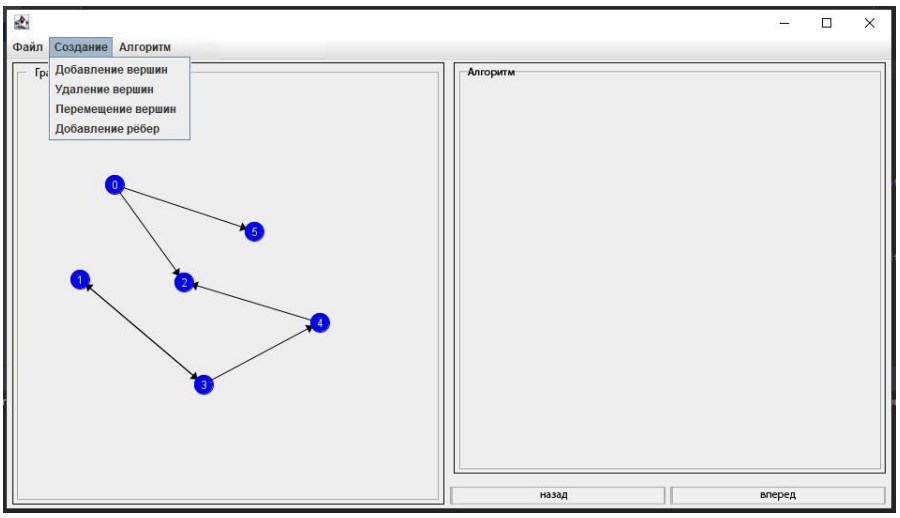

Рис. 3 – Внешний вид приложения (3)

1.2. Уточнение требований после проверки.

1.2.1. После загрузки прототипа.

- Заголовок окна - название алгоритма. (Сделано)
- Не "Пояснение", а "Пояснения". В поле должно помещаться несколько строк. (Сделано)
- Уменьшить начальный размер окна: сейчас после запуска нижняя часть окна оказалась скрыта под панелью задач (а если она слева, а не снизу, то кусок окна справа оказывается за пределами экрана). (Сделано)
- Над окном создания графа добавьте 4 радиокнопки, соответствующие 4 пунктам меню "Создание". (Сделано)

1.2.2. После загрузки 1-й версии.

- Уточнения отсутствуют

1.2.3. После загрузки 2-й версии.

- Перемещение вершин работает, только если перемещать их медленно. (Сделано)
- Кнопки "Назад" и "Вперёд" должны быть неактивны в том случае, когда их нажатие не имеет эффекта. (Сделано)

- Пояснения должны быть выделяемыми и копируемыми. (Сделано)
- По ходу алгоритма удаляемая вершина вместе с инцидентными ей рёбрами должна перекрашиваться. (Не сделано)
- При наведении курсора на вершину в ходе выполнения алгоритма должна всплывать подсказка с текущим количеством вхождений. (Сделано)
- После нажатия "Вывести результат" должна быть возможность делать шаги назад. (Сделано)
- В меню Алгоритм добавьте пункт "Заново", возвращающий состояние программы в начало. (Сделано)

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

- Согласование
    - Распределение ролей
    - Составление спецификации
- Прототип (06.07.2021)
    - Создание прототипа пользовательского интерфейса
    - Разработка интерфейса позволяющего построить граф
- 1-я Итерация (07.07.2021)
    - Реализация алгоритма и вывод результата работы алгоритма в приложение
- 2-я Итерация (09.07.2021)
    - Добавление возможность пошагового исполнения алгоритма
    - Добавление возможности возврата исполнения алгоритма к предыдущим шагам
    - Добавление возможности записи/чтения графа из файла
    - Тестирование приложения

2.2. Распределение ролей в бригаде

1. Николаев Александр (9381) - Разработка пользовательского интерфейса отвечающего за создание (Кнопки создания графа, редактирования, работы алгоритма, привязка к этим компонентам функционал). Визуализация графа.
2. Прашутинский Кирилл (9381) - Создание графа, как некоторой структуры данных и реализация самого алгоритма. Визуализация работы алгоритма.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Пакет resources

Алгоритм топологической сортировки, предназначен для обработки графа. Хранение графа осуществляется при помощи класса Graph. Граф содержит коллекцию вершин, которые представляют собой объекты класса Node. Каждая вершина хранит список указателей на инцидентные этой вершине вершины, свой ID, который также является и её именем и положение на экране.

Класс UserMeta отвечает за хранение настроек программы и отслеживание действий пользователя. Она хранит поле editMode, типа перечисления EditMode, которое содержит текущий выбранный пользователем режим редактирования (Создать вершину, удалить вершину, переместить вершину, создать ребро, ничего не выбрано).

Пакет UI

Класс MainWindow является главным классом в программе. Он отвечает за GUI и выводит на экран пользователя все компоненты программы.

Класс DrawPanel extends JPanel - реализует рисование графа, а AlgorithmPanel extends JPanel - реализует вывод всей информации об алгоритме пользователю (пояснение к каждому шагу итерации, таблица, содержащая информацию о вершинах, необходимую для выполнения алгоритма и отрисовка полученного графа на каждом шаге итерации). Класс UIStyleSetting отвечает за хранение некоторых констант программы, таких как цвет и радиус вершины, цвет и размер имени вершины, цвет и размер ребра, а также настройки сглаживания и т.п.

3.2. Основные методы

В Graph метод saveGraph() записывает граф в файл для того, чтобы его можно было потом заново загрузить использовать при последующих запусках программы.

В Graph метод loadGraph() создает граф на основе сохраненного файла.

В Graph метод Algorithm() - алгоритм топологической сортировки.

В Graph метод addNode() - позволяет пользователю добавить вершину в граф.

В Graph метод deleteNode() - позволяет пользователю удалить вершину из графа.

В Graph метод move() - позволяет пользователю переместить вершину в графе.

В Graph метод addEdge() - позволяет пользователю добавить ребро в графе.

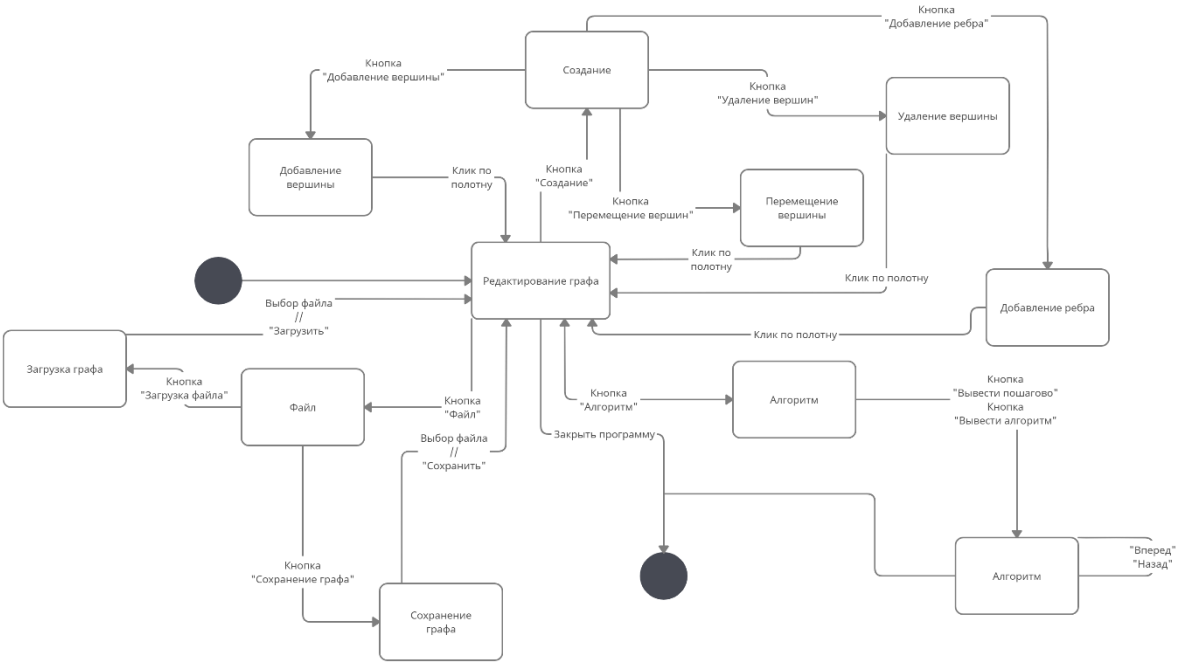

Рис. 4 – Диаграмма состояний программы

4. ТЕСТИРОВАНИЕ

4.1. План тестирования программы.

Проведение тестирования программы планируется с использованием библиотеки для модульного тестирования Junit. Объектами тестирования станут ключевые методы, используемые в алгоритме Топологической сортировки.

Объект тестирования	Данные	Результаты	Ожидаемые тестовые данные	Актуальные тестовые данные

Метод addEdge()	graph.addEdge(graph.nodes.get(0), graph.nodes.get(0));
	graph.addEdge(graph.nodes.get(0), graph.nodes.get(1));
	graph.addEdge(graph.nodes.get(0), graph.nodes.get(2));
	graph.addEdge(graph.nodes.get(0), graph.nodes.get(3));
	graph.addEdge(graph.nodes.get(0), graph.nodes.get(4));
	graph.addEdge(graph.nodes.get(0), graph.nodes.get(5));
	graph.addEdge(graph.nodes.get(2), graph.nodes.get(0));
	graph.addEdge(graph.nodes.get(0), graph.nodes.get(1));
	graph.addEdge(graph.nodes.get(4), graph.nodes.get(2));
	graph.addEdge(graph.nodes.get(5),
		Проверка добавления ребра в граф. Количество добавленных ребер.	CountEdge = 11
			Edges:
			0 -> 0 1 2 3 4 5
			1 -> 5
			2 -> 0
			3 ->
			4 -> 2
			5 -> 3
			6 -> 4
				CountEdge = 11
				Edges:
				0 -> 0 1 2 3 4 5
				1 -> 5
				2 -> 0
				3 ->
				4 -> 2
				5 -> 3
				6 -> 4

	graph.nodes.get(3));
	graph.addEdge(graph.nodes.get(6),
	graph.nodes.get(4));
	graph.addEdge(graph.nodes.get(1),
	graph.nodes.get(5));			

Метод addNode()	graph.addNode(0,0);
	graph.addNode(-50, -20);
	graph.addNode(100, 110);
	graph.addNode(132165156,16518163);
	graph.addNode(0,654);
	graph.addNode(6562,0);
	graph.addNode(-27,527);
		Проверка добавления вершины в граф.	CountNode = 7
			0 (0,0)
			1 (-50,-20)
			2 (100,110)
			3 (132165156, 16518163)
			4 (0, 654)
			5 (6562, 0)
			6 (-27,527)
				CountNode = 7
				0 (0,0)
				1 (-50,-20)
				2 (100,110)
				3 (132165156, 16518163)
				4 (0, 654)
				5 (6562, 0)
				6 (-27,527)

Метод saveGraph ()	graph.addNode(0,0);
	graph.addNode(50, 20);
	graph.addNode(100, 110);
	graph.addNode(132165156,16518163);
	graph.addNode(0,654);
	graph.addNode(6562,0);
	graph.addNode(27,527);
	graph.saveGraph(“save.txt”)
	graph1.loadGraph(“save.txt”)
		Запись графа в файл, а затем загрузка этого же файла для проверки правильности записи графа.	SaveText=
			“BEGIN 7
			0 0 0
			1 50 20
			2 100 110
			3 132165156
			16518163
			4 0 654
			5 6562 0
			6 27 527
			////////////////
			0 : 0 -
			1 : 0 -
			2 : 0 -
			3 : 0 -
			4 : 0 -
			5 : 0 -
			6 : 0 -
			END”
			graph == graph1
				SaveText=
				“BEGIN 7
				0 0 0
				1 50 20
				2 100 110
				3 132165156
				16518163
				4 0 654
				5 6562 0
				6 27 527
				////////////////
				0 : 0 -
				1 : 0 -
				2 : 0 -
				3 : 0 -
				4 : 0 -
				5 : 0 -
				6 : 0 -
				END”
				graph == graph1

Метод loadGraph ()	String test1 = "BEGIN 7\n" + "0 0 0\n" + "1 50 20\n" + "2 100 110\n" + "3 132165156 16518163\n" + "4 0 654\n" + "5 6562 0\n" +	Проверка данных в файле для создания графа	test1=true test2=true text3=false	test1=true test2=true text3=false

	"6 27 527\n" + "///////////////\n" + "0 : 0 -\n" + "1 : 0 -\n" + "2 : 0 -\n" + "3 : 0 -\n" + "4 : 0 -\n" + "5 : 0 -\n" + "6 : 0 -\n" + "END"; String test2 = "BEGIN 5\n" + "0 263 128\n" + "1 367 262\n" + "2 280 384\n" + "3 340 232\n" + "4 398 170\n" + "///////////////\n" + "0 : 0 -\n" + "1 : 0 -\n" + "2 : 0 -\n" + "3 : 0 -\n" + "4 : 0 -\n" + "END"; String test3 = "BEGIN 5\n" + "1 304 151\n" + "2 107 289\n" + "4 336 479\n" + "5 171 477\n" + "6 386 294\n" +			

	"////////////////\n" + "1 : 0 -\n" + "2 : 0 -\n" + "4 : 1 - 6\n" + "5 : 3 - 1 2 4\n" + "6 : 1 - 1\n" + "END";			

Метод deleteNode()	graph.addNode(0,0);
	graph.addNode(-50, -20);
	graph.addNode(100, 110);
	graph.addNode(132165156,16518163);
	graph.addNode(0,654);
	graph.addNode(6562,0);
	graph.addNode(-27,527);
	graph.deleteNode(graph.nodes.get(0));
	graph.deleteNode(graph.nodes.get(1));
		Проверка удаления вершины из графа.	CountNode = 5
			2 (100,110)
			3 (132165156, 16518163)
			4 (0, 654)
			5 (6562, 0)
			6 (-27,527)
				CountNode = 5
				2 (100,110)
				3 (132165156, 16518163)
				4 (0, 654)
				5 (6562, 0)
				6 (-27,527)

4.2. Результаты тестирования.

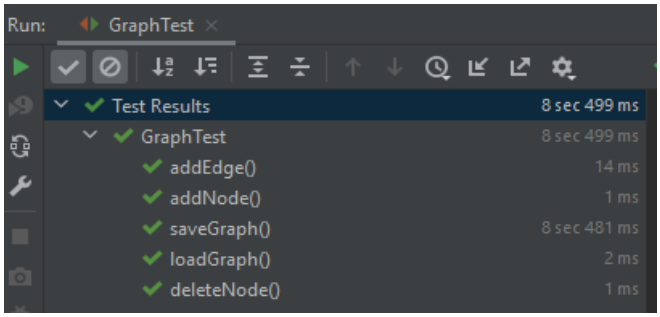

Рис. 5 – результаты тестирования

ЗАКЛЮЧЕНИЕ

Таким образом, командой была реализована работа и визуализация алгоритма Топологической сортировки. Пользователь имеет возможность задать входные данные двумя способами – вручную и с помощью загрузки из файла. Есть возможность пошагового выполнения алгоритма.

В ходе совместной работы были получены практические знания о ЯП Java, системе тестирования Junit и навыках командной разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Описание работы с JUnit // JUnit 5 User Guide. URL: https://junit.org/junit5/docs/current/user-guide/#writing-tests-classes-and-methods (дата обращения: 11.07.2021).
2. Описание алгоритма топологической сортировки // Wikipedia. URL: https://ru.wikipedia.org/wiki/Топологическая_сортировка (дата обращения: 3.07.2021).
3. Описание ЯП Java // Java Platform, Standard Edition 7 API Specification. URL: https://docs.oracle.com/javase/7/docs/api/overview-summary.html (дата обращения: 3.07.2021-11.07.2021).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД АЛГОРИТМА.

Имя файла: MainWindow.java

```
package JavaCode.UI;

import JavaCode.resources.DataAlgorithm;
import JavaCode.resources.Graph;
import JavaCode.resources.UserMeta;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.Point2D;
import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;
import javax.swing.*;
import javax.swing.border.TitledBorder;
import javax.swing.filechooser.FileFilter;

import static java.awt.Cursor.*;

public class MainWindow extends JFrame {
   public Graph graph = new Graph();
   public static UserMeta userMeta = new UserMeta();

   public DrawPanel graphCreatePanel = addGraphCreatePanel();
   public AlgorithmPanel algorithmCreatePanel = addAlgorithmCreatePanel();




////////////////////////////////////////////////////////////////////////////////////////////////////////////
/
   public MainWindow() {
      initUI();
```

```
    }
    private void initUI() {

        // Устанавливаем настройки окна приложения.
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);

        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension appSize = new Dimension((int)(toolkit.getScreenSize().width / 1.5d),
(int)(toolkit.getScreenSize().height / 1.5d));
        Dimension screenSize = toolkit.getScreenSize();

        // Устанавливаем размеры окна.
        setBounds(screenSize.width / 2 - appSize.width / 2, screenSize.height / 2 - appSize.height / 2,
appSize.width, appSize.height);

        getContentPane().setLayout(new GridLayout());
        // Создаём меню.
        addMenuToPane();
        // Создание панели для отображения графа.
        addPanelsToPane();

        revalidate();
    }

    private void addPanelsToPane() {
        JPanel leftPanel = new JPanel();
        JPanel rightPanel = new JPanel();

        leftPanel.setLayout(new BorderLayout());
        rightPanel.setLayout(new BorderLayout());

        JRadioButton creating = new JRadioButton("Создать вершину");
        JRadioButton deleting = new JRadioButton("Удалить вершину");
        JRadioButton moving = new JRadioButton("Переместить вершину");
        JRadioButton linking = new JRadioButton("Добавить ребро");
```

```
        ButtonGroup editing = new ButtonGroup();
        editing.add(creating);
        editing.add(deleting);
        editing.add(moving);
        editing.add(linking);

        creating.addActionListener(e -> userMeta.editMode = UserMeta.EditMode.Creating);
        deleting.addActionListener(e -> userMeta.editMode = UserMeta.EditMode.Deleting);
        moving.addActionListener(e -> userMeta.editMode = UserMeta.EditMode.Moving);
        linking.addActionListener(e -> userMeta.editMode = UserMeta.EditMode.Linking);

        JPanel editingPanel = new JPanel();
        editingPanel.setLayout(new GridLayout(1, 3));

        editingPanel.add(creating);
        editingPanel.add(deleting);
        editingPanel.add(moving);
        editingPanel.add(linking);

        leftPanel.add(editingPanel, BorderLayout.NORTH);

        leftPanel.setBorder(new TitledBorder(null, "Окно создания графа", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
        leftPanel.add(graphCreatePanel, BorderLayout.CENTER);

        rightPanel.setBorder(new TitledBorder(null, "Окно просмотра алгоритма", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
        rightPanel.add(algorithmCreatePanel, BorderLayout.CENTER);

        getContentPane().add(leftPanel);
        getContentPane().add(rightPanel);
    }

    private DrawPanel addGraphCreatePanel() {
        DrawPanel drawPanel = new DrawPanel();
```

```
drawPanel.setGraph(graph);
drawPanel.setManaged(true);
drawPanel.setCurved(false);

drawPanel.addMouseListener(new MouseAdapter() {
  @Override
  public void mouseReleased(MouseEvent e) {
    switch (userMeta.editMode) {
      case None:
        break;
      case Creating:
        if (graphCreatePanel.contains(e.getX(), e.getY())) {
          graph.addNode(e.getX(), e.getY());
        }
        break;
      case Deleting:
        Graph.Node node = graph.getNodeOnFocus(e.getX(), e.getY());
        graph.deleteNode(node);
        break;
      case Linking:
        userMeta.finishPoint = new Point2D.Double(e.getX(), e.getY());
        Graph.Node first = graph.getNodeOnFocus(userMeta.startPoint.x, userMeta.startPoint.y);
        Graph.Node second = graph.getNodeOnFocus(userMeta.finishPoint.x, userMeta.finishPoint.y);
        graph.addEdge(first, second);
        userMeta.startPoint = null;
        userMeta.finishPoint = null;
        break;
      case Moving:
        userMeta.buffer = null;
    }
  }
});

drawPanel.addMouseListener(new MouseAdapter() {
  @Override
  public void mousePressed(MouseEvent e) {
```

```
        System.out.println(e.getX() + " " + e.getY());
        switch (userMeta.editMode) {
            case Creating, Deleting, None -> {
            }
            case Linking -> userMeta.startPoint = new Point2D.Double(e.getX(), e.getY());
            case Moving -> userMeta.buffer = graph.getNodeOnFocus(e.getX(), e.getY());
        }
    }
});

drawPanel.addMouseMotionListener(new MouseMotionAdapter() {
    @Override
    public void mouseDragged(MouseEvent e) {
        switch (userMeta.editMode) {
            case Moving -> {
                //Graph.Node node = graph.getNodeOnFocus(e.getX(), e.getY());
                //if (node != null && graphCreatePanel.contains(e.getX(), e.getY())) {
                //    node.move(e.getX(), e.getY());
                //}
                if (userMeta.buffer != null && graphCreatePanel.contains(e.getX(), e.getY())) {
                    userMeta.buffer.move(e.getX(), e.getY());
                }
            }
            case Linking -> userMeta.finishPoint = new Point2D.Double(e.getX(), e.getY());
        }
    }
});

drawPanel.addMouseMotionListener(new MouseMotionAdapter() {
    @Override
    public void mouseMoved(MouseEvent e) {
        if (userMeta.editMode == UserMeta.EditMode.Moving) {
            Graph.Node node = graph.getNodeOnFocus(e.getX(), e.getY());
            setCursor(getPredefinedCursor(Cursor.DEFAULT_CURSOR));
            if (node != null) {
                setCursor(getPredefinedCursor(Cursor.MOVE_CURSOR));
```

```
            }
        }
    }
});

return drawPanel;
}

private AlgorithmPanel addAlgorithmCreatePanel() {
    AlgorithmPanel algorithmPanel = new AlgorithmPanel();
    return algorithmPanel;
}

public void addMenuToPane() {
    JMenuBar jMenuBar = new JMenuBar();

    JMenu file = new JMenu("Файл");
    jMenuBar.add(file);

    JMenuItem load = new JMenuItem("Загрузить");

    load.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            StringBuilder input = new StringBuilder();

            JFileChooser fileChooser = new JFileChooser();

            fileChooser.setDialogTitle("Выбор директории");
            // Определение режима - только каталог
            fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
            int result = fileChooser.showOpenDialog(MainWindow.this);

            algorithmCreatePanel.setTextjLabel(graph.loadGraph(fileChooser, result));
        }
    });
```

```
JMenuItem save = new JMenuItem("Сохранить");

save.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileChooser = new JFileChooser();

        fileChooser.setDialogTitle("Сохранение файла");

        // Определение режима - только файл
        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        int result = fileChooser.showSaveDialog(MainWindow.this);

        algorithmCreatePanel.setTextjLabel(graph.saveGraph(fileChooser, result));
    }
});

file.add(load);
file.add(save);

JMenu algorithm = new JMenu("Алгоритм");
jMenuBar.add(algorithm);

JMenuItem run = new JMenuItem("Вывести результат");
JMenuItem debug = new JMenuItem("Вывести пошагово");
JMenuItem restart = new JMenuItem("Заново");

run.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        ArrayList<DataAlgorithm> dataAlgorithm = graph.Algorithm();
        algorithmCreatePanel.setStepIndex(dataAlgorithm.size() - 1);
        algorithmCreatePanel.setDataAlgorithms(dataAlgorithm);
    }
});
```

```
        debug.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                ArrayList<DataAlgorithm> dataAlgorithm = graph.Algorithm();
                algorithmCreatePanel.setStepIndex(0);
                algorithmCreatePanel.setDataAlgorithms(dataAlgorithm);
            }
        });

        restart.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                getContentPane().removeAll();
                graph = new Graph();
                userMeta = new UserMeta();

                graphCreatePanel = addGraphCreatePanel();
                algorithmCreatePanel = addAlgorithmCreatePanel();
                addPanelsToPane();
                revalidate();
            }
        });

        algorithm.add(run);
        algorithm.add(debug);
        algorithm.addSeparator();
        algorithm.add(restart);

        setJMenuBar(jMenuBar);
    }


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

```
        MainWindow mainWindow = new MainWindow();
    }
}
```

Имя файла: AlgorithmPanel.java

```
package JavaCode.UI;

import JavaCode.resources.DataAlgorithm;
import JavaCode.resources.Graph;
import JavaCode.resources.UserMeta;

import javax.swing.*;
import javax.swing.border.TitledBorder;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;

import static java.awt.Cursor.getPredefinedCursor;

public class AlgorithmPanel extends JPanel {

    private final JWindow toolTipWindow = new JWindow();
    private final JLabel toolTipLabel = new JLabel("", SwingConstants.CENTER);

    private final JTextArea jTextArea = new JTextArea();
    private final DrawPanel drawPanel = new DrawPanel();
    private ArrayList<DataAlgorithm> dataAlgorithms = new ArrayList<>();

    private final JButton stepBack = new JButton("Назад");
    private final JButton stepForward = new JButton("Вперёд");

    private final DefaultTableModel model = new DefaultTableModel() {
```

```
    @Override
    public boolean isCellEditable(int row, int column) {
      return false;
    }
  };

  private final JTable jTable = new JTable(model);

  private int stepIndex = 0;

  public AlgorithmPanel() {
    initUI();
  }

  private void initUI() {
    toolTipLabel.setOpaque(false);
    toolTipLabel.setBackground(UIManager.getColor("ToolTip.background"));
    toolTipLabel.setBorder(UIManager.getBorder("ToolTip.border"));
    toolTipWindow.add(toolTipLabel);
    toolTipWindow.setSize(30, 20);
    toolTipWindow.setVisible(false);

    drawPanel.addMouseMotionListener(new MouseMotionAdapter() {
      @Override
      public void mouseMoved(MouseEvent e) {
        if (drawPanel.getGraph() != null) {
          Graph.Node node = drawPanel.getGraph().getNodeOnFocus(e.getX(), e.getY());
          if (node != null) {
            toolTipLabel.setText(Integer.toString(drawPanel.getGraph().getCountParents(node)));
            Point point = e.getPoint();
            SwingUtilities.convertPointToScreen(point, e.getComponent());
            point.translate(-toolTipWindow.getWidth() / 2, (int) (-(toolTipWindow.getHeight() +
MainWindow.userMeta.settings.UIStyleSetting_NodeSize)));
            toolTipWindow.setLocation(point);
            toolTipWindow.setVisible(true);
          }
```

```
            else {
                toolTipWindow.setVisible(false);
            }
        }
    }
});


jTextArea.setBackground(new Color(239, 239, 239));
jTextArea.setEditable(false);

JScrollPane jTableScrollPane = new JScrollPane(jTable);
jTable.setBackground(new Color(239, 239, 239));

drawPanel.setCurved(true);
drawPanel.setManaged(false);

setLayout(new BorderLayout());

JPanel northPanel = new JPanel();
JPanel midPanel = new JPanel();
JPanel downPanel = new JPanel();

northPanel.setLayout(new BorderLayout());
midPanel.setLayout(new BorderLayout());
downPanel.setLayout(new BorderLayout());

northPanel.setBorder(new TitledBorder(null, "Пояснения", TitledBorder.LEADING, TitledBorder.TOP,
null, null));
northPanel.add(jTextArea, BorderLayout.CENTER);

JPanel centralPanel = new JPanel();

centralPanel.setLayout(new GridLayout(2, 1));
```

```
        midPanel.setBorder(new TitledBorder(null, "Таблица", TitledBorder.LEADING, TitledBorder.TOP, null, null));
        midPanel.add(jTableScrollPane, BorderLayout.CENTER);

        downPanel.setBorder(new TitledBorder(null, "Результат", TitledBorder.LEADING, TitledBorder.TOP, null, null));
        downPanel.add(drawPanel, BorderLayout.CENTER);

        centralPanel.add(midPanel);
        centralPanel.add(downPanel);

        JPanel southPanel = new JPanel();

        stepBack.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (stepIndex != 0) {
                    stepIndex -= stepIndex == 0 ? 0 : 1;
                    jTextArea.setText(dataAlgorithms.get(stepIndex).explanation);
                    drawPanel.setGraph(dataAlgorithms.get(stepIndex).graph);
                    model.removeRow(stepIndex + 1);

                    int counter = 0;
                    int dx = drawPanel.getWidth() / dataAlgorithms.get(stepIndex).graph.nodes.size();
                    for (int nodeID : dataAlgorithms.get(stepIndex).getQueue()) {

dataAlgorithms.get(stepIndex).graph.nodes.get(nodeID).move(MainWindow.userMeta.settings.UIStyleSetting_NodeSize / 2 + counter++ * dx, drawPanel.getHeight() / 2);
                    }
                    drawPanel.setGraph(dataAlgorithms.get(stepIndex).graph);

                    setEnabled(stepIndex != 0);
                }

                stepBack.setEnabled(stepIndex != 0);
                stepForward.setEnabled(stepIndex + 1 < dataAlgorithms.size());
```

```
        }
    });

    stepForward.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (stepIndex < dataAlgorithms.size() - 1) {
                stepIndex += stepIndex == dataAlgorithms.size() - 1 ? 0 : 1;
                jTextArea.setText(dataAlgorithms.get(stepIndex).explanation);
                drawPanel.setGraph(dataAlgorithms.get(stepIndex).graph);
                model.addRow(dataAlgorithms.get(stepIndex).values);

                int counter = 0;
                int dx = drawPanel.getWidth() / dataAlgorithms.get(stepIndex).graph.nodes.size();
                for (int nodeID : dataAlgorithms.get(stepIndex).getQueue()) {

dataAlgorithms.get(stepIndex).graph.nodes.get(nodeID).move(MainWindow.userMeta.settings.UIStyleSetti
ng_NodeSize / 2 + counter++ * dx, drawPanel.getHeight() / 2);
                }
                drawPanel.setGraph(dataAlgorithms.get(stepIndex).graph);


            }

            stepBack.setEnabled(stepIndex != 0);
            stepForward.setEnabled(stepIndex + 1 < dataAlgorithms.size());
        }
    });

    stepBack.setEnabled(stepIndex != 0);
    stepForward.setEnabled(stepIndex + 1 < dataAlgorithms.size());

    southPanel.add(stepBack);
    southPanel.add(stepForward);

    add(northPanel, BorderLayout.NORTH);
```

```
    add(centralPanel, BorderLayout.CENTER);
    add(southPanel, BorderLayout.SOUTH);

    revalidate();
}

public void setTextjLabel(String string){
    jTextArea.setText(string);
}

public void setDataAlgorithms(ArrayList<DataAlgorithm> dataAlgorithms) {
    clear();
    if (this.dataAlgorithms != null) {
        this.dataAlgorithms = dataAlgorithms;

        this.jTextArea.setText(this.dataAlgorithms.get(stepIndex).explanation);

        if (this.dataAlgorithms.get(stepIndex).names != null) {
            for (String name : this.dataAlgorithms.get(stepIndex).names.split(" ")) {
                this.model.addColumn(name);
            }
        }

        if (this.dataAlgorithms.get(stepIndex).values != null) {
            for (int index = 0; index < stepIndex + 1; ++index) {
                this.model.addRow(this.dataAlgorithms.get(index).values);
            }
        }

            for (DataAlgorithm date : this.dataAlgorithms) {

                if (date.graph != null) {
                    int counter = 0;
                    int dx = drawPanel.getWidth() / date.graph.nodes.size();

                    for (int nodeID : date.getQueue()) {
```

```
date.graph.nodes.get(nodeID).move(MainWindow.userMeta.settings.UIStyleSetting_NodeSize / 2 +
counter++ * dx, drawPanel.getHeight() / 2);
                }
              }
            }

        this.drawPanel.setGraph(dataAlgorithms.get(stepIndex).graph);

        stepBack.setEnabled(stepIndex != 0);
        stepForward.setEnabled(stepIndex + 1 < dataAlgorithms.size());
    }
  }

  public void clear() {
    jTextArea.setText(null);
    this.drawPanel.setGraph(null);

    model.setColumnCount(0);
    model.getDataVector().removeAllElements();

    stepBack.setEnabled(stepIndex != 0);
    stepForward.setEnabled(stepIndex + 1 < dataAlgorithms.size());
    dataAlgorithms = new ArrayList<>();
  }

  public DrawPanel getDrawPanel() {
    return drawPanel;
  }

  public ArrayList<DataAlgorithm> getDataAlgorithms() {
    return dataAlgorithms;
  }

  public void setStepIndex(int stepIndex) {
    this.stepIndex = stepIndex;
```

```
    }
}
```

Имя файла: DrawPanel.java

```
package JavaCode.UI;

import JavaCode.resources.Graph;
import JavaCode.resources.UserMeta;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.util.Map;

import static JavaCode.UI.MainWindow.userMeta;

// Панель для отображения графа.
public class DrawPanel extends JPanel {

    private boolean defaultCurved = false;
    // TODO: Придумать нормальное название. Она отвечает за отрисовку курсора пользователя.
    private boolean isManaged = true;
    private Graph graph;



    private void drawNodes(Graphics2D g2d) {
        // Выводим круги.
        g2d.setColor(userMeta.settings.colors.UIStyleColor_Node);
        // Установил сглаживание фигур.
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
userMeta.settings.UIStyleSetting_Node_KEY_ANTIALIASING);

        for (Map.Entry<Integer, Graph.Node> node : graph.nodes.entrySet()) {
            Ellipse2D ellipse2D = new Ellipse2D.Double(
                    node.getValue().x - userMeta.settings.UIStyleSetting_NodeSize / 2d,
                    node.getValue().y - userMeta.settings.UIStyleSetting_NodeSize / 2d,
```

```
                userMeta.settings.UIStyleSetting_NodeSize,
                userMeta.settings.UIStyleSetting_NodeSize);
            g2d.fill(ellipse2D);
            g2d.draw(ellipse2D);
        }
    }

    private void drawNames(Graphics2D g2d) {
        // Выводим имена.
        g2d.setColor(userMeta.settings.colors.UIStyleColor_ID);
        // Установил сглаживание текста.
        g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
userMeta.settings.UIStyleSetting_ID_KEY_TEXT_ANTIALIASING);

        for (Map.Entry<Integer, Graph.Node> node : graph.nodes.entrySet()) {
            g2d.drawString(Integer.toString(node.getValue().ID), (float) (node.getValue().x -
Integer.toString(node.getValue().ID).length() * 3.5f), (float) (node.getValue().y + 5f));
        }
    }

    private void drawArrow(Graphics2D g2d, Point2D.Double start, Point2D.Double finish, double radius,
boolean isCurved) {

        // Выводим рёбра.
        g2d.setColor(userMeta.settings.colors.UIStyleColor_Edge);
        // Установил сглаживание рёбер.
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING ,
userMeta.settings.UIStyleSetting_Edge_KEY_ANTIALIASING);

        double angle1 = Math.atan2(finish.getY() - start.getY(), finish.getX() - start.getX() + ((isCurved) ? radius *
2 : 0));
        double cosx1 = Math.cos(angle1);
        double siny1 = Math.sin(angle1);
        double cosx2 = Math.cos(angle1 + Math.PI);
        double siny2 = Math.sin(angle1 + Math.PI);
```

```
        double fin1x = cosx1 * radius + start.getX();
        double fin1y = siny1 * radius + start.getY();
        double fin2x = cosx2 * radius + finish.getX();
        double fin2y = siny2 * radius + finish.getY();

        // Линия стрелки.
        QuadCurve2D QC2D = new QuadCurve2D.Double(fin1x, fin1y,(fin1x + fin2x) / 2, ((isCurved) ? ((fin2y + fin1y) / 2 + ( start.x == finish.x ? -80 : (fin1x - fin2x) / 2)) : (fin2y + fin1y) / 2), fin2x, fin2y);
        // Стрелка
        Polygon polygon = new Polygon(new int [] {(int)fin2x, (int)fin2x - 5, (int)fin2x - 5}, new int [] {(int)fin2y, (int)fin2y - 5, (int)fin2y + 5}, 3);

        double angle = Math.atan2(QC2D.getY2() - ((isCurved) ? QC2D.getCtrlY() : QC2D.getY1()), QC2D.getX2() - ((isCurved) ? QC2D.getCtrlX() : QC2D.getX1()));
        AffineTransform savedTransform = g2d.getTransform();
        g2d.rotate(angle, fin2x, fin2y);
        g2d.fill(polygon);
        g2d.setTransform(savedTransform);

        g2d.draw(QC2D);
    }

    private void drawEdges(Graphics2D g2d) {
        // Выводим рёбра.
        g2d.setColor(userMeta.settings.colors.UIStyleColor_Edge);
        // Установил сглаживание рёбер.
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING                                                    ,
userMeta.settings.UIStyleSetting_Edge_KEY_ANTIALIASING);

        for (Map.Entry<Integer, Graph.Node> node : graph.nodes.entrySet()) {
            for(Graph.Node child : node.getValue().childs) {
                drawArrow(
                        g2d,
                        new Point2D.Double(node.getValue().x, node.getValue().y),
                        new Point2D.Double(child.x, child.y),
                        userMeta.settings.UIStyleSetting_NodeSize / 2d,
```

```
                node.getValue().ID == child.ID || defaultCurved);
        }
    }
}

private void drawUserMeta(Graphics2D g2d) {
    if (userMeta.startPoint != null && userMeta.finishPoint != null && isManaged) {
        drawArrow(g2d, new Point2D.Double(
                userMeta.startPoint.x,
                userMeta.startPoint.y),
            new Point2D.Double(userMeta.finishPoint.x, userMeta.finishPoint.y),
            0d,
            false);
    }
}

private void render(Graphics g) {
    if (graph != null) {
        Graphics2D g2d = (Graphics2D) g;

        drawUserMeta(g2d);
        drawEdges(g2d);
        drawNodes(g2d);
        drawNames(g2d);

        repaint();
    }
}

@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    render(g);
}

public void setGraph(Graph graph) {
```

```
        this.graph = graph;
    }

    public void setCurved(boolean curved) {
        defaultCurved = curved;
    }

    public void setManaged(boolean managed) {
        isManaged = managed;
    }

    public Graph getGraph() {
        return this.graph;
    }
}
```

Имя файла: UIStyleSetting.java

```
package JavaCode.UI;

import java.awt.*;

public class UIStyleSetting {

    public double UIStyleSetting_NodeSize   = 20d;
    public double UIStyleSetting_EdgeSize   = 1d;
    public double UIStyleSetting_IDSize     = 12d;

    public Object UIStyleSetting_Node_KEY_ANTIALIASING     = RenderingHints.VALUE_ANTIALIAS_ON;
    public Object UIStyleSetting_ID_KEY_TEXT_ANTIALIASING  = RenderingHints.VALUE_TEXT_ANTIALIAS_ON;
    public Object UIStyleSetting_Edge_KEY_ANTIALIASING     = RenderingHints.VALUE_ANTIALIAS_ON;

    public UIStyleColor colors = new UIStyleColor();

    public UIStyleSetting() {
    }
```

```
    class UIStyleColor {
        Color UIStyleColor_Node = Color.blue;
        Color UIStyleColor_ID = Color.white;
        Color UIStyleColor_Edge = Color.black;

        UIStyleColor() {
        }
    }
}
```

Имя файла: DataAlgorithm.java

```
package JavaCode.resources;

import java.util.ArrayList;

public class DataAlgorithm {
    public Graph graph;
    public String explanation;
    public String names;
    public String[] values;
    private ArrayList<Integer> queue;

    public DataAlgorithm(Graph graph, String explanation, String names, String[] values, ArrayList<Integer>
queue) {
        this.graph = graph;
        this.explanation = explanation;
        this.names = names;
        this.values = values;
        setQueue(queue);
    }

    public void setQueue(ArrayList<Integer> queue) {
        if (queue != null) {
            this.queue = new ArrayList<>(queue);
        }
    }
```

```
    public ArrayList<Integer> getQueue(){
        if (this.queue != null) {
            System.out.println(this.queue);
            return queue;
        }

        return null;
    }
}
```

Имя файла: Graph.java

```
package JavaCode.resources;

import javax.swing.*;

import static JavaCode.UI.MainWindow.userMeta;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

public class Graph {

    private int nodeIDcounter = 0;
    public HashMap<Integer, Node> nodes = new HashMap<>();

    public Graph() {
        nodeIDcounter = 0;
    }

    // Конструктор копирования.
    public Graph(Graph other) {
        for(Map.Entry<Integer, Node> currentNode : other.nodes.entrySet()) {
```

```
            this.addNode(currentNode.getValue().ID, currentNode.getValue().x, currentNode.getValue().y);
            for (Node child : currentNode.getValue().childs) {
                this.nodes.get(currentNode.getKey()).addEdge(this.nodes.get(child.ID));
            }
        }
    }

    public String saveGraph(JFileChooser fileChooser, int result){
        String ret = "";

        StringBuilder output = new StringBuilder("BEGIN " + nodes.size() + "\n");
        for (Graph.Node node : nodes.values()){
            output.append(node.ID).append("                                    ").append((int)node.x).append("
").append((int)node.y).append("\n");
        }

        output.append("/////////////////\n");
        for(Graph.Node node : nodes.values()){
            output.append(node.ID).append(" : " + node.childs.size() + " -");
            for (Graph.Node child : node.childs){
                output.append(" ").append(child.ID);
            }
            output.append("\n");
        }
        output.append("END");

        System.out.println(output);

        File file1 = fileChooser.getSelectedFile();
        if(result == JFileChooser.APPROVE_OPTION){

            String fileName = fileChooser.getSelectedFile().getName().matches(".*\\.txt$") ?
                    fileChooser.getSelectedFile().getParent() + "\\" + fileChooser.getSelectedFile().getName() :
                    fileChooser.getSelectedFile().getParent() + "\\" + fileChooser.getSelectedFile().getName() + ".txt";
            try (FileWriter writer = new FileWriter(fileName, false)){
                writer.write(output.toString());
```

```
            writer.flush();
            ret = "Граф сохранен в файле " + fileName;
        } catch (IOException ioException) {
            ret = "Граф не удалость сохранить!";
            ioException.printStackTrace();
        }
    }
    return ret;
  }

  public String loadGraph(JFileChooser fileChooser, int result){

    boolean isLoad = false;

    fileChooser.setDialogTitle("Загрузка файла");
    // Если директория выбрана, покажем ее в сообщении
    StringBuilder input = new StringBuilder();
    if (result == JFileChooser.APPROVE_OPTION ){
//              JOptionPane.showMessageDialog(MainWindow.this,
//                      fileChooser.getSelectedFile());
        try {
            FileReader reader = new FileReader(fileChooser.getSelectedFile());
            int c;
            while((c=reader.read())!=-1){
               input.append((char) c);
            }
            //System.out.println(input);
        } catch (IOException fileNotFoundException) {
            fileNotFoundException.printStackTrace();
        }
    }
//
if(input.toString().matches("^BEGIN\\s\\d+[\\n](\\d+\\s\\d+\\s\\d+[\\n])+[/]+[\\n](\\d+\\s:\\s\\d+\\s-
\\s\\d?\\n)+[^$]+END$")){
//              System.out.println("Сохранение некорректно!");
//              return;
```

```
//            }


        if(!input.isEmpty()){
            Scanner scanner = new Scanner( input.toString() );
            scanner.useDelimiter( "\\D+" );

            int N = scanner.nextInt(); // get int
            String regex = "^BEGIN \\d+\n(\\d+ \\d+ \\d+\n){"+N+"}[/]{16}\n(\\d+ : \\d+ -( \\d+)*\n){"+N+"}END$";

            if(!input.toString().matches(regex)){
                System.out.println("Сохранение некорректно!");
                return "Сохранение некорректно!";
            }
            isLoad = true;

            nodes.clear();
            for(int i = 0; i < N; i++){

                String ID = scanner.next();
                System.out.println(ID);
                addNode(Integer.parseInt(ID), scanner.nextInt(),scanner.nextInt());
            }
            System.out.println();
            if(true){
                for(int i = 0; i < N; i++){
                    Integer ID = scanner.nextInt();
                    int K = scanner.nextInt();
                    for(int k = 0; k < K; k++){
                        Integer buff = scanner.nextInt();
                        System.out.println(buff);
                        addEdge(nodes.get(ID), nodes.get(buff));
                    }
                }
            }
            else {
```

```
            System.out.println("Некорректный файл.");
        }
    }

    if(!nodes.isEmpty())
        nodeIDcounter = Collections.max(nodes.keySet()) + 1;
    System.out.println(nodeIDcounter);



    if(isLoad)
        return "Граф успешно загружен!";
    return "";
}

public void addNode(double x, double y) {
    nodes.put(nodeIDcounter, new Node(nodeIDcounter, x, y));
    nodeIDcounter++;

    System.out.println(nodes);
}

public void addNode(int ID, double x, double y) {
    nodes.put(ID, new Node(ID, x, y));
}

public void addNode(int ID, int name, double x, double y) {
    nodes.put(ID, new Node(name, x, y));
}

public void deleteNode(Node node) {
    if (node != null) {
        for (Map.Entry<Integer, Node> currentNode : nodes.entrySet()) {
            currentNode.getValue().deleteNode(node);
        }
        nodes.remove(node.ID);
    }
```

```
  }

  private void _Algorithm(HashMap<Integer, Integer> GraphCycle, ArrayList<DataAlgorithm>
dataAlgorithms){
    int breakCounter = 0;
    ArrayList<Integer> deletedNodes = new ArrayList<>();

    while (true) {
      for(Integer str : GraphCycle.keySet()) {
        if(GraphCycle.get(str) == 0){
          deletedNodes.add(str);
          Graph buff = new Graph();
          for(int names : deletedNodes){
            buff.addNode(names, 0, 0);
          }
          for(Integer first : deletedNodes){
            for(Integer second : deletedNodes){
              if(this.nodes.get(first).childs.contains(this.nodes.get(second))){
                buff.addEdge(buff.nodes.get(first),buff.nodes.get(second));
              }
            }
          }

          dataAlgorithms.add(new DataAlgorithm(buff, "Удаляем вершину ( " + str + " ) т.к. у неё " +
GraphCycle.get(str)
              + " (минимальное) число вхождений. \nУменьшаем значения количества вхождений у
вершин ( "
              , GraphCycle.keySet().toString().replaceAll("[\\]\\[\\,]", "")
              , GraphCycle.values().toString().replaceAll("[\\]\\[\\,]", "").replaceAll("-1", "-").split(" "),
deletedNodes));

          for(Node node : nodes.get(str).childs){
            dataAlgorithms.get(dataAlgorithms.size() - 1).explanation += node.ID + " ";
            GraphCycle.put(node.ID, GraphCycle.get(node.ID) - 1);
          }
          dataAlgorithms.get(dataAlgorithms.size() - 1).explanation += ").";
```

```
            GraphCycle.put(str, GraphCycle.get(str) - 1);
          }

        }
        for(Integer integer : GraphCycle.values()){
          if(integer == -1){
            breakCounter++;
          }
        }

        if(breakCounter == GraphCycle.size()) break;
        breakCounter = 0;
      }
    }

    public Integer getCountParents(Node node) {
      HashMap<Integer,Integer> GraphCycle = new HashMap<>();
      for( Integer name : nodes.keySet()) {
        if(!GraphCycle.containsKey(name)) {
          GraphCycle.put(name, 0);
        }
        if(!nodes.get(name).childs.isEmpty()) {
          for (Node child: nodes.get(name).childs) {
            if (GraphCycle.containsKey(child.ID)) {
              GraphCycle.put(child.ID, GraphCycle.get(child.ID) + 1);
            }
            else {
              GraphCycle.put(child.ID, 1);
            }
          }
        }
      }

      if(GraphCycle.containsKey(node.ID)){
        return GraphCycle.get(node.ID);
```

```
    }

    return 0;
  }
  public ArrayList<DataAlgorithm> Algorithm() {
    ArrayList<DataAlgorithm> dataAlgorithms = new ArrayList<>();

    if(this.nodes.isEmpty()){
      System.out.println("Граф не построен. Сортировка невозможна!");
      dataAlgorithms.add(new DataAlgorithm(null,"Граф не построен. Сортировка невозможна!",
null,null, null));
      return dataAlgorithms;
    }

    //Храним ключ - название вершины, значение - количество ребер, входящих в эту вершину

    System.out.println("Алгоритм");



    //проверка условия работы алгоритма
    System.out.println("Проверка графа на цикл!");
    for(Node node : nodes.values()){
      ArrayList<Integer> visited = new ArrayList<>();

      if (node.isCycle(visited)) {
        dataAlgorithms.add(new DataAlgorithm(null,"Граф содержит цикл. Сортировка невозможна!",
null,null, null));
        System.out.println("Граф содержит цикл. Сортировка невозможна!");
        return dataAlgorithms;
      }

      visited.clear();
    }
    System.out.println("Граф не содержит циклов!");
```

```
        HashMap<Integer,Integer> GraphCycle = new HashMap<>();

        for( Node node : nodes.values()){
            GraphCycle.put(node.ID,getCountParents(node));
        }



//      for(Node node : nodes.values()){
//          if(GraphCycle.get(node.ID) == 0 && node.childs.isEmpty()){
//              dataAlgorithms.add(new DataAlgorithm(null,"Вершины не соединены!", null,null));
//              System.out.println("Вершины не соединены");
//              return dataAlgorithms;
//          }
//      }

        _Algorithm(GraphCycle, dataAlgorithms);

        return dataAlgorithms;
    }

    public void addEdge(Node first, Node second) {
        if (first != null && second != null) {
            nodes.get(first.ID).addEdge(second);
        }
    }

    public Node getNodeOnFocus(double x, double y) {
        for (Map.Entry<Integer, Node> node : nodes.entrySet()) {
            double currentX = (x - node.getValue().x);
            double currentY = (y - node.getValue().y);
            double currentR = (userMeta.settings.UIStyleSetting_NodeSize / 2);

            if ((Math.pow(currentX, 2f)) + (Math.pow(currentY, 2f)) <=  Math.pow(currentR, 2f)) {
                return node.getValue();
            }
```

```
        }

        return null;
    }

    public static class Node {

        public double x;
        public double y;
        public int ID;

        public LinkedList<Node> childs = new LinkedList<>();

        Node (int ID, double x, double y) {
            this.ID = ID;
            this.x = x;
            this.y = y;
        }

        public boolean isCycle(ArrayList<Integer> visited) {
            if(visited.contains(this.ID)) return true;
            visited.add(this.ID);
            for (Node child : childs){
                if(visited.size() - 1 != 0 && visited.get(0).equals(visited.get(visited.size() - 1)) ||
child.isCycle(visited)){
                    System.out.println("Cycle: "+visited);
                    return true;
                }
                for(int index = visited.indexOf(child.ID); index < visited.size(); index++){
                    visited.remove(visited.size()-1);
                }
            }
            return false;
        }

        public void addEdge(Node node) {
```

```
            if (node != null) {
                if (!childs.contains(node)) {
                    childs.add(node);
                }
            }
        }

        public void move(double x, double y) {
            this.x = x;
            this.y = y;
        }

        public void deleteNode(Node node) {
            if (node != null) {
                childs.remove(node);
            }
        }
    }
}
```

Имя файла: UserMeta.java

```
package JavaCode.resources;

import JavaCode.UI.UIStyleSetting;

import java.awt.geom.Point2D;

public class UserMeta {
    public UIStyleSetting settings = new UIStyleSetting();

    public EditMode editMode = EditMode.None;

    public Graph.Node buffer = null;

    public enum EditMode {
        Creating,
```

```
        Deleting,
        Moving,
        Linking,
        None
    }

    public Point2D.Double startPoint = null;
    public Point2D.Double finishPoint = null;

    public UserMeta () {
    }
}
```