**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по учебной практике**

**“Визуализация алгоритма поиска в глубину в ориентированном графе на языке Java.”**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 5381 |  | Боровикова О.В. |
| Студентка гр. 5381 |  | Буздина М.А |
| Студентка гр. 5381 |  | Кочнева О.Р. |
| Руководитель |  | Фирсов М.А. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студентка группы 5381 Боровикова О.В. | | |
| Студентка группы 5381 Буздина М.А. | | |
| Студентка группы 5381 Кочнева О.Р. | | |
| Тема практики: визуализация алгоритмов на языке Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: обход ориентированного графа в глубину. | | |
| Сроки прохождения практики: 21.06.2017 – 04.07.2017 | | |
| Дата сдачи отчета: 28.06.2017 | | |
| Дата защиты отчета: 28.06.2017 | | |
|  | | |
| Студентка гр.5381 |  | Боровикова О.В. |
| Студентка гр.5381 |  | Буздина М.А. |
| Студентка гр.5381 |  | Кочнева О.Р. |
| Руководитель |  | Фирсов М.А. |

# Аннотация

Темой данной учебной практики является командная итеративная разработка визуализатора алгоритма на языке программирования Java. Цель учебной практики – получить практические навыки в визуализации алгоритмов, изучить и получить навыки использования языка программирования Java, получить навыки работы в команде. В работе представлена визуализация алгоритма обхода ориентированного графа в глубину с пользовательским интерфейсом.

# Summary

## The subject of this training practice is the command iterative development of the algorithm visualizer in the Java programming language. The goal of the training practice is to gain practical skills in visualizing Java algorithms and team skills. The work shows the visualization of the algorithm DFS in an oriented graph with the user interface.

## **Содержание**

[ВВЕДЕНИЕ 5](#_Toc486433905)

[1. ТРЕБОВАНИЯ К ПРОГРАММЕ 6](#_Toc486433906)

[1.1 Исходные Требования к программе 6](#_Toc486433907)

[1.2 Изменения к спецификации 7](#_Toc486433908)

[1.2.1 Изменения к спецификации после сдачи прототипа 7](#_Toc486433909)

[1.2.2 Изменения к спецификации после сдачи первой версии 7](#_Toc486433910)

[1.3 Описание входных и выходных данных 8](#_Toc486433911)

[1.4 Примерная работа программы 8](#_Toc486433912)

[2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ 12](#_Toc486433913)

[2.1. План разработки 12](#_Toc486433914)

[2.2. Распределение ролей в бригаде 13](#_Toc486433915)

[3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ 14](#_Toc486433916)

[3.1 Используемые структуры данных 14](#_Toc486433917)

[3.2 Использованные возможности библиотеки 19](#_Toc486433918)

[3.3 UML – диаграмма классов 20](#_Toc486433919)

[4. ТЕСТИРОВАНИЕ 20](#_Toc486433920)

[4.1 Тестирование интерфейса 20](#_Toc486433921)

[4.2 Тестирование алгоритма 24](#_Toc486433922)

[5. ЗАКЛЮЧЕНИЕ 26](#_Toc486433923)

[Обход орграфа в глубину 27](#_Toc486433924)

ВЕДЕНИЕ

Поиск в глубину – один из методов обхода графа. Стратегия поиска в глубину, как и следует из названия, состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину. Если после завершения алгоритма не все вершины были рассмотрены, то необходимо запустить алгоритм от одной из нерассмотренных вершин. В нашей работе будет реализован визуализатор данного алгоритма, с использованием графического интерфейса.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1 Исходные Требования к программе

При запуске проекта будет создаваться отдельное окно. Данное окно будет иметь 5 полей:

1. Поле графического представления графа.

Граф строится по данным из файла автоматически при запуске программы, вершины размещаются по окружности; если файл пуст или отсутствует, то поле представление графа при запуске будет пустым.

1. Поле с выводом результата.
2. Поле с выводом промежуточных результатов.
3. Поле входных данных, содержащее представление исходного графа.
4. Поле интерфейса создания графа:

* добавить вершину (вершина добавляется в центр графического поля, и может перемещаться пользователем при помощи мыши.)
* добавить ребро
* старт
* следующий шаг
* результат
* очистить поля

При нажатии кнопки следующий шаг происходят действия: в поле промежуточных выводов указывается состояние вершины; на графе состояние вершины отображается соответствующим цветом (1 цвет – посещённая вершина, 2 цвет – использованная вершина) и номером порядка посещения и использования. При переходе к следующей вершине ребро, соединяющее вершины, будет подсвечено в цвет 3. При возврате ребро будет подсвечено в цвет 4.

Примерный вид интерфейса представлен на рис.1



Рисунок 1. Интерфейс программы. (Построен на сайте draw.io)

1.2 Изменения к спецификации

1.2.1 Изменения к спецификации после сдачи прототипа

Для удобства пользователя было решено ввести кнопку «Выбрать файл», для считывания информации из файла.

Также в окне начальных данных граф представляется не в виде матрицы смежности, а в виде списков смежности.

При ручном введении данных добавление вершин происходит по окружности, а не в центр.

При пошаговом выведении номер посещения и номер использования пишутся в скобках под номером вершины через запятую.

1.2.2 Изменения к спецификации после сдачи первой версии

При пошаговой визуализации алгоритма подсвечиваются стрелочки, которые существуют из рассматриваемой вершины в уже использованные.

1.3 Описание входных и выходных данных

Входные данные:

Граф, записанный в виде матрицы смежности.

Выходные данные:

Графическое представление графа с подсветкой всех посещенных вершин и соответствующих пройденных ребер. Вывод пошаговой работы алгоритма в текстовое окно. Вывод порядка обхода в глубину.

1.4 Примерная работа программы

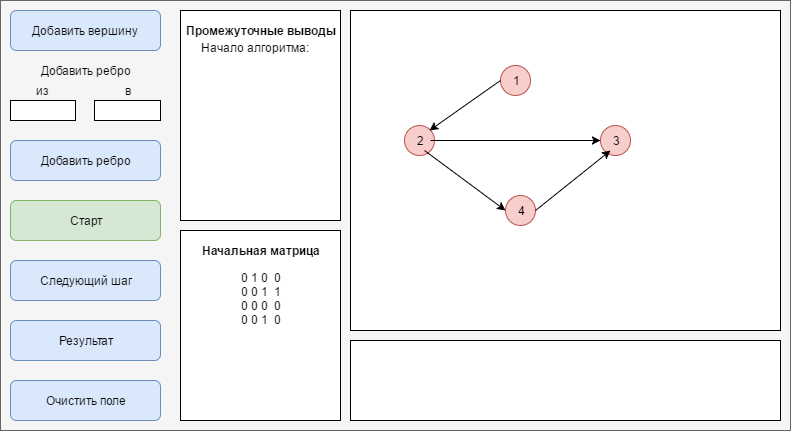


Рисунок 2. (Построен на сайте draw.io)

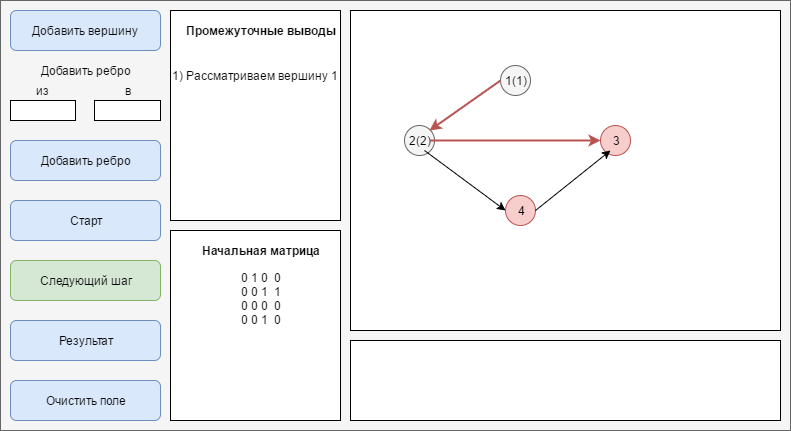


Рисунок 3. (Построен на сайте draw.io)

## 

Рисунок 4. (Построен на сайте draw.io)

## 

Рисунок 5. (Построен на сайте draw.io)

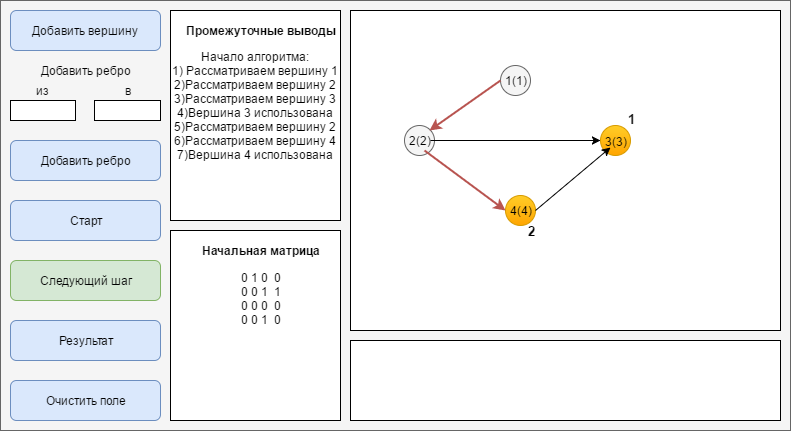


Рисунок 6. (Построен на сайте draw.io)

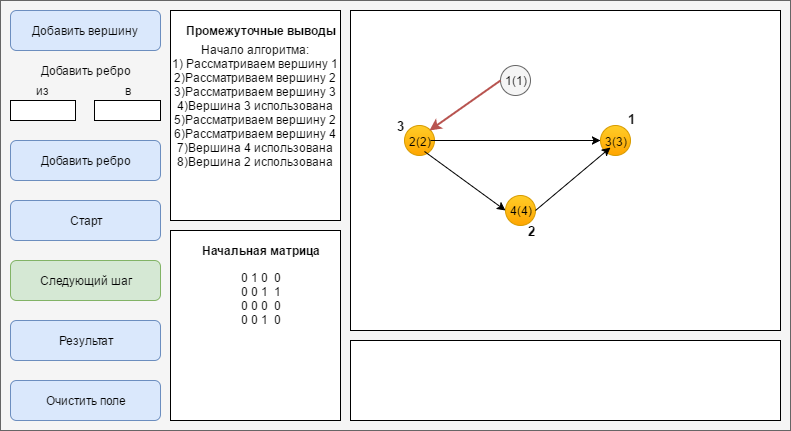


Рисунок 7. (Построен на сайте draw.io)

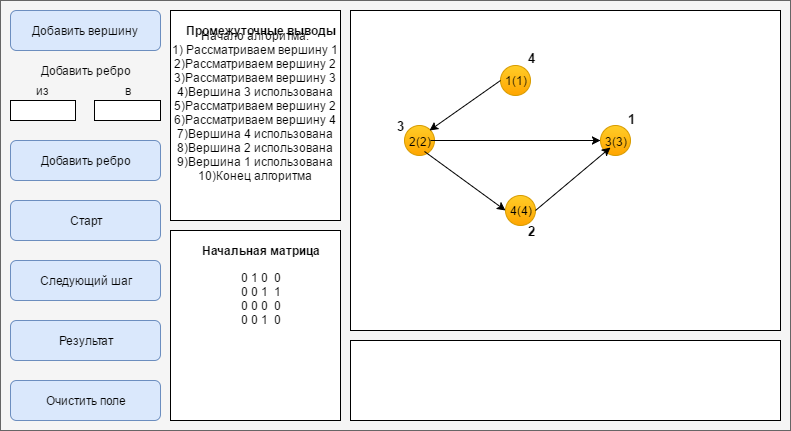


Рисунок 8. (Построен на сайте draw.io)

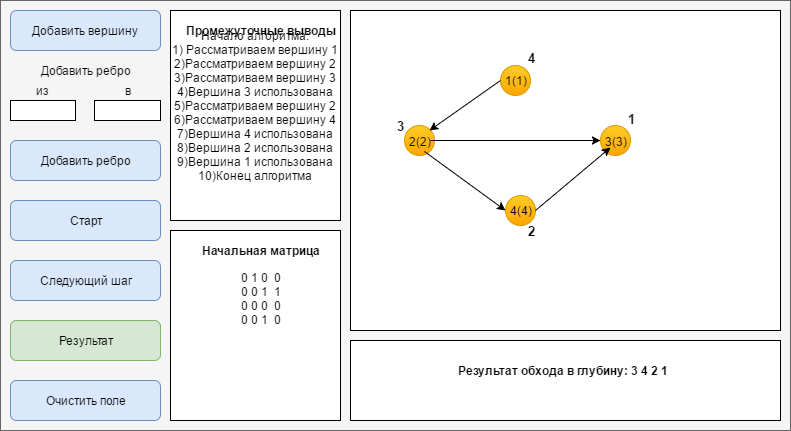


Рисунок 9. (Построен на сайте draw.io)

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

22.06.2017 (четверг) – разработка спецификации

23.06.2017 – согласование спецификации с руководителем

(24.06 -25.06).2017 – разработка интерфейса

26.06.2017 (понедельник) - предоставление планируемого интерфейса (прототип), с отсутствующей функциональностью.

27.06.2017 – реализация графического представления графа, на основе входных данных с файла и осуществление работы алгоритма с выводом результата в соответствующее окно(частичная функциональность)

28.06.2017 (среда) - сдача 1-ой версии с добавлением частичной функциональности к пользовательскому интерфейсу;

29.06.2017 – реализация пошаговой работы алгоритма, реализация графического представления графа, на основе входных данных с использованием пользовательского интерфейса, тестирование программы. Оформление отчета.

30 июня (пятница) - сдача финальной версии. Представление проекта с полной функциональностью.

2.2. Распределение ролей в бригаде

В таблице 1 представлен состав бригады и распределение обязанностей.

Таблица 1.

|  |  |
| --- | --- |
| Имя студентки | Обязанность |
| Боровикова Ольга | Осуществление пошаговой реализации алгоритма на графической части интерфейса. Тестирование программы. |
| Буздина Мария | Реализация алгоритма поиска в глубину. Осуществление пошаговой работы алгоритма и выведения результатов на текстовой панели графического интерфейса. |
| Кочнева Ольга | Разработка прототипа. Реализация интерфейса и необходимого функционала для обеспечения добавления ребер и вершин на изображении графа. |

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1 Используемые структуры данных

Граф для реализации алгоритма представляется в виде списка смежности. Для визуализации алгоритма граф представляется в виде четырех списков: список вершин прямого обхода, список вершин обратного обхода, список ребер использованных при обходе, список ребер неиспользованных при обходе.

Для реализации графического интерфейса разрабатываемого приложения используется библиотека Swing, для изображения исходного графа и порядка его обхода – библиотека JGraphX.

Программа состоит из трех классов: Main, Algo,Visualisator.

***Класс Main***

Класс Main отвечает за изображение графического интерфейса и выполнение программы в целом.

В основе работы класса лежат методы класса JFrame, запускаемые в отдельном потоке. Данные методы реализуют весь графический интерфейс, кроме изображения графа.

Кнопки, используемые в программе представлены в таблице 2:

Таблица 2

|  |  |
| --- | --- |
| Название кнопки | Функциональность |
| btnEdge | Добавление вершины |
| btnDepht | Добавление ребра |
| btnStart | Начало алгоритма |
| btnNext | Следующий шаг алгоритма |
| BtnResult | Результат |
| btnClear\_fields | Очистить поля |
| btnData | Считать информацию из файла |

Метод addActionListener(new ActionListener) вызывается для добавления кнопкам функциональности.

Для изображения графа создан объект panel класса Algo.

***Класс Algo***

Класс Algo содержит в себе поле для хранения промежуточных данных при реализации алгоритма обхода и методы для реализации алгоритма обхода.

Поля класса:

* protected ArrayList<String> output – массив для вывода промежуточных данных

Методы класса:

Таблица 3.

|  |  |
| --- | --- |
| Метод | Предназначение |
| public void dfs(int v) | Функция обхода графа в глубину  Параметры: рассматриваемая вершина |
| public String printresult() | Функция для вывода результатов, возвращает строку с обходом графа |
| public void changeDataEdge(int v, int w) | Функция добавления ребра  Параметры: начало и конец ребра |
| public void run() | Функция запуска алгоритма обхода |
| public void changeData\_adj() | Функция добавления новой вершины |
| public void readData(File file) | Функция считывания с файла  Параметры: файл, с которого считывается информация |
| public int getN() | Функция, возвращаемая количество количество пройденных вершин |
| public String print\_start\_data(int i) | Функция вывода начальных даны, возвращает список смежности для одной вершины.  Параметры: номер вершины |
| public boolean search\_for(int v, int w) | Функция проверки добавления ребер, возвращает false, если ребро есть, true, не было ребра.  Параметры: начало и конец ребра |

***Класс Visualisator***

Класс Visualisator основанный на использовании методов библиотеки JGraphX, отвечает за изображение графов.

Поля класса:

* protected static int n - количество вершин;
* protected boolean provv – проверяет наличие ребра между вершинами;
* protected static int m – количество ребер;
* protected ArrayList<ArrayList<Integer>> adj – списки смежности графа
* protected boolean used[] – массив, в котором проверяется посещена ли ершина
* protected ArrayList<Integer> listDepht – список вершин;
* protected ArrayList<Pair<Integer, Integer>> listEdges – списки пар ребер;
* protected ArrayList<Object> points – массив объектов – вершина, для отрисовки графа;
* protected mxIGraphModel model – модель графа;
* protected ArrayList<Integer> result – массив, в котором записаны вершины в порядке обхода в глубину

Методы класса:

Таблица 4.

|  |  |
| --- | --- |
| Метод | Предназанчение |
| public int get\_N() | Функция возвращающая количество вершин |
| public void inc\_N() | Функция увеличения количества вершин |
| public void inc\_M() | Функция увеличения количества ребер |
| public void setnull() | Функция обнуляющая количество вершин и ребер |
| public Visualisator() | Конструктор |
| public boolean dfsVisual(int step, int stepp, int kol) | Функция, которая проходит граф в прямом порядке (помечает пройденные вершины), возращает возвращает false, если вершины не связаны, true, если связаны.  Параметры:  step - текущая вершина;  stepp – предыдущая вершина;  kol – номер просматриваемой вершины |
| public boolean dfsVisual\_1(int step, int prov, int kol) | Функция, которая проходит граф в обратном порядке, возвращает false, если ребро и вершина перекрашены, true, если нет.  step - предыдущая вершина;  prov – текущая вершина;  kol – номер просматриваемой вершины |
| public int dfsVisual\_2(int step, int kol) | Функция, которая отмечает неиспользованные ребра, возвращает номер использованной вершины, в которую идет ребро  step - предыдущая вершина;  kol – номер просматриваемой вершины |
| public void functionVisual() | Функция отрисовки графа |
| Public int used(int step) | Функция возвращает номер вершины в порядке использования, возвращает номер вершины в порядке использования  Параметры: индекс вершины в списке |

3.2 Использованные возможности библиотеки

В данной работе для визуализации графа была использована сторонняя библиотека JGraphX.

*Использованные поля:*

mxIGraphModel -определение модели графа для установки ее в граф

mxGraph –создание объекта графа

*Использованные методы для графа:*

setModel - установка модели графа

setCellStyles - покраска вершины

getModel().setValue - установка текста внутрь вершины

getModel().setStyle - покраска ребра

graph.insertEdge - создание ребер

*Использованные методы для модели графа:*

beginUpdate - фиксирует начало изменения и конец изменен

endUpdate - фиксирует конец изменения

3.3 UML – диаграмма классов

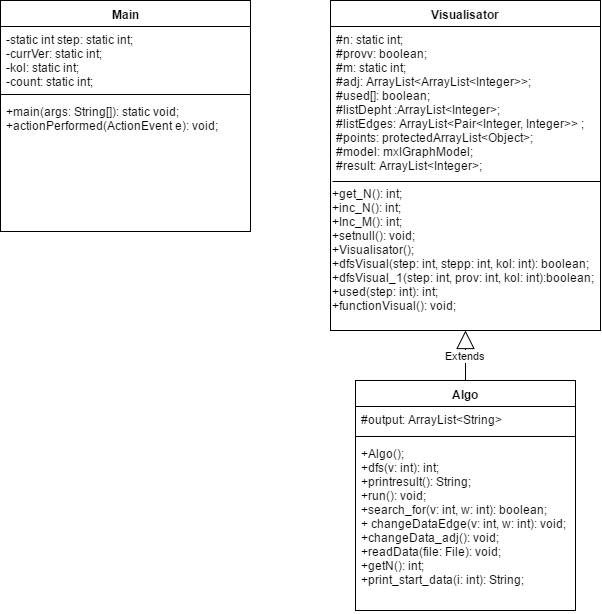


Рисунок 10. (Построен на сайте draw.io)

4. ТЕСТИРОВАНИЕ

4.1 Тестирование интерфейса

1) Запускаем jar файл

2) Добавляем 4 вершины, нажатием на кнопку “Добавить вершину”. На экране по порядку нажатия появляются 4 вершины по окружности.

3) Добавляем ребра: 1 - 4; 2 - 3, 2 - 3, 3 - 4;

4) Нажимаем кнопку “Старт” который запускает алгоритм и в «Поле 4» выводится списки смежности:

***1 вершина соединена с : 4 3 2***

***2 вершина соединена с: 3***

***3 вершина соединена с: 4***

***4 не соединена ни с одной вершиной***

5) Нажатием на кнопку “Следующий шаг” на графическом поле подсвечиваются вершина 1 в розовый цвет, в текстовом “Поле 3” комментарий: «Рассматриваем вершину 1».

6) Нажали на кнопку “Следующий шаг”

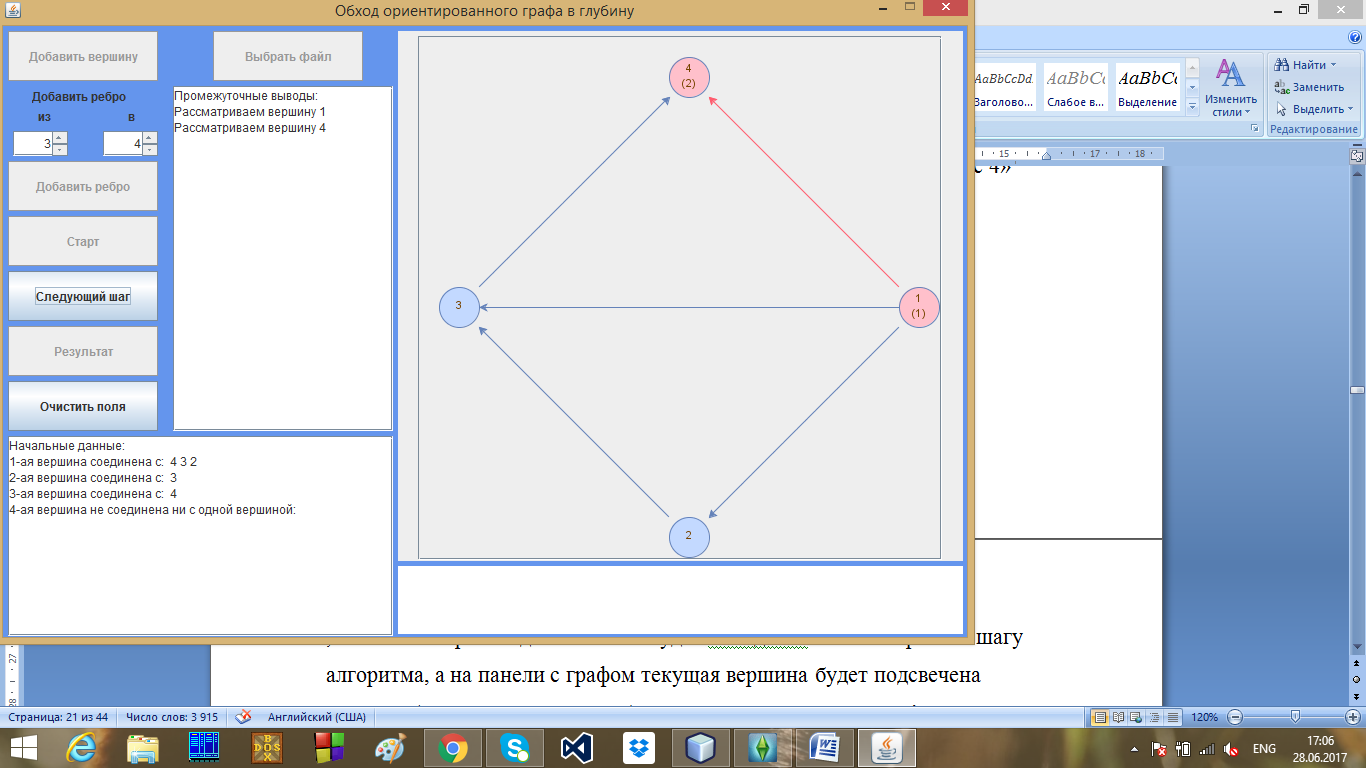


Рисунок 11.

Вершина смежная с текущей и ребро, соединяющее их, перекрашивается розовый цвет.

7) Нажали на кнопку “Следующий шаг”

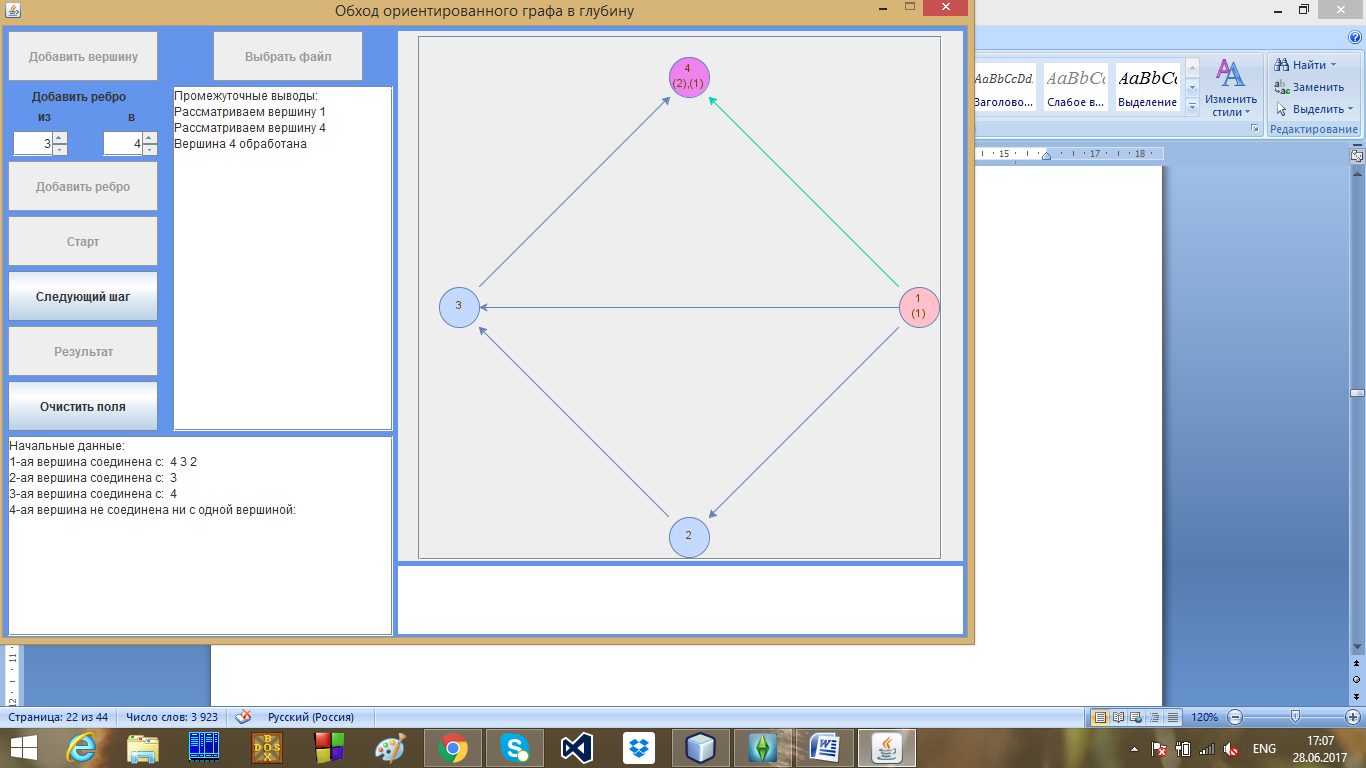


Рисунок 12.

Вершина 4 – обработана и она перекрашивается в фиолетовый цвет, а ребро – в зеленый.

8) Нажали на кнопку “Следующий шаг”

Ребро 1 – 3 перекрасится в розовый и вершина 3 тоже.

Рисунок 13.

9) Нажали на кнопку “Следующий шаг”

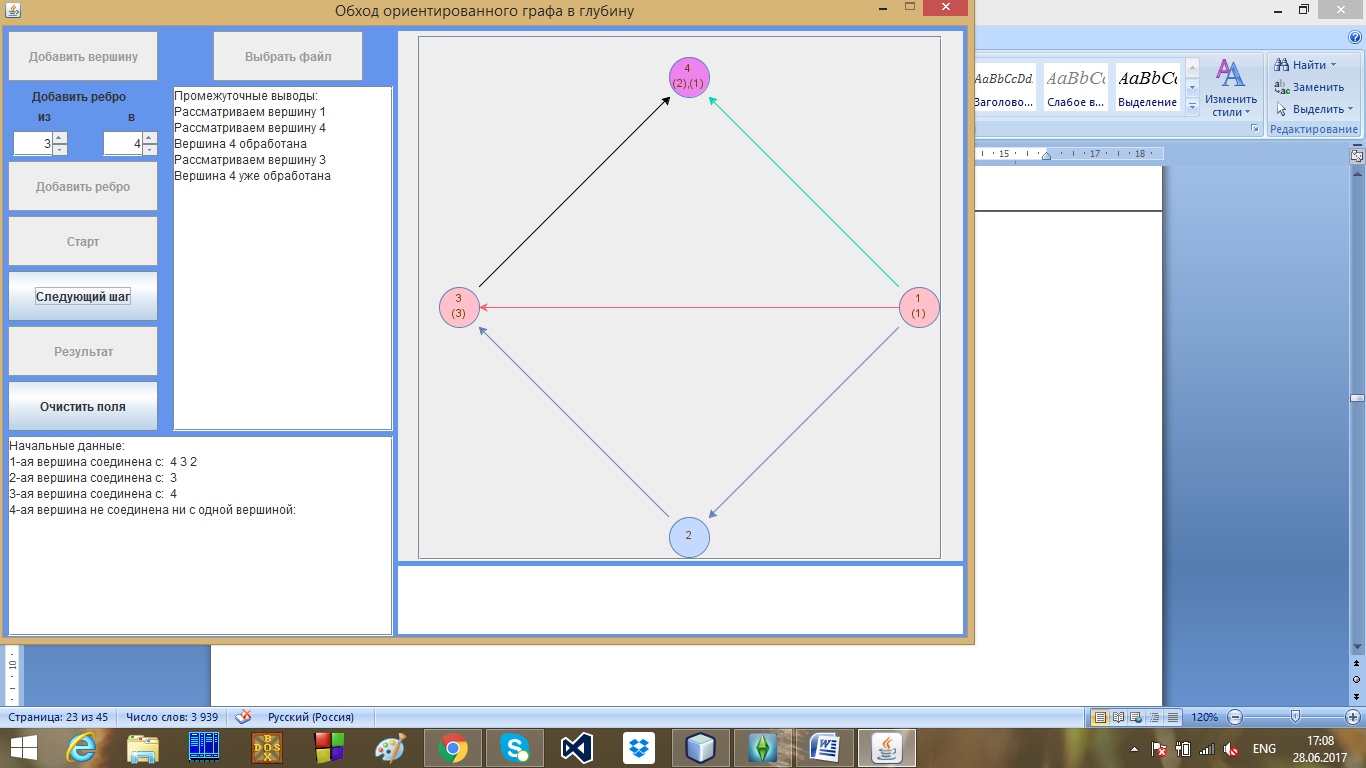


Рисунок 14.

У вершины 3 единственной смежной является вершина 4, поэтому ребро, соединяющее их, перекрашивается в черный цвет.

10) Нажали на кнопку “Следующий шаг”

Вершина 3 перекрашивается как использованная в фиолетовый цвет

11) Нажали на кнопку “Следующий шаг”

Ребро 1 – 3 перекрашивается в зеленый, т.е. оно рассмотрено.

12) Нажали на кнопку “Следующий шаг”

Рассматривается вершина 2 и ребро 1 – 2, они перекрашиваются в розовый цвет.

13) Нажали на кнопку “Следующий шаг”

Ребро 2 – 3 перекрашивается в черное, так как вершина 3 рассмотрена.

14) Нажали на кнопку “Следующий шаг”

Ребро 1 – 2 перекрашивается в зеленое, как рассмотренное.

15) Нажали на кнопку “Результат”

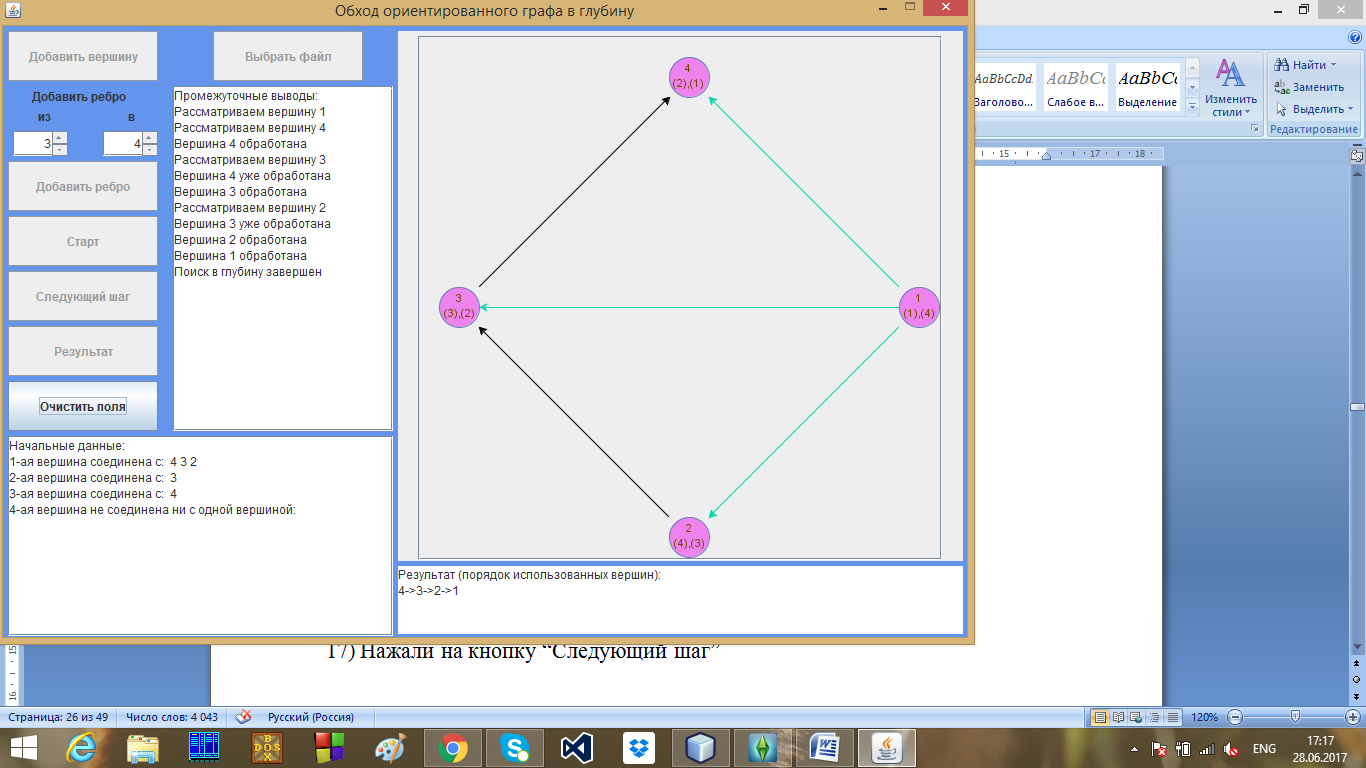


Рисунок 20.

16) Нажали на кнопку “Очистить поля”

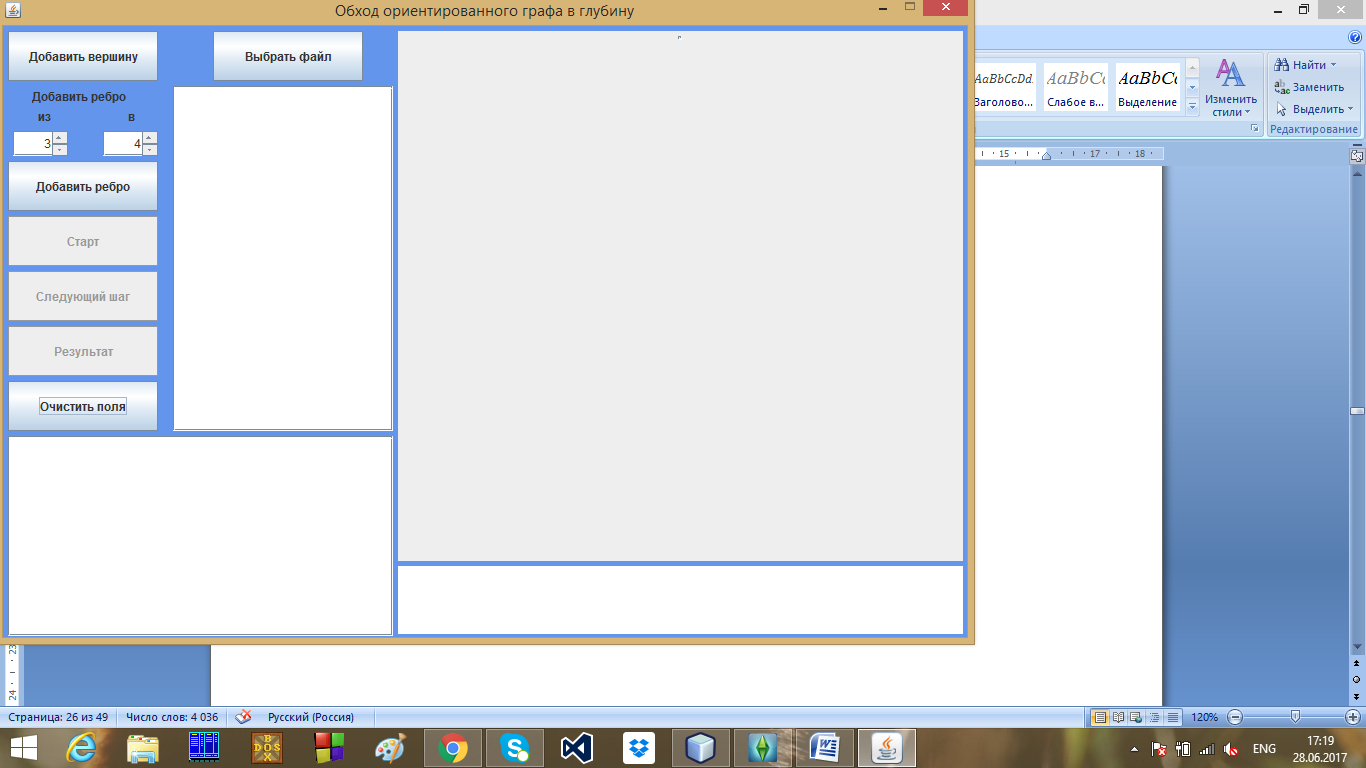


Рисунок 21.

4.2 Тестирование алгоритма

Таблица 5.

|  |  |  |
| --- | --- | --- |
| № | Начальные данные | Результат (порядок использования) |
| 1 | Начальные данные:  1-ая вершина соединена с: 2 3 4 5 6  2-ая вершина не соединена ни с одной вершиной:  3-ая вершина не соединена ни с одной вершиной:  4-ая вершина соединена с: 2  5-ая вершина соединена с: 3  6-ая вершина соединена с: 2 4 5 | Результат:  2->3->4->5->6->1 |
| 2 | Начальные данные:  1-ая вершина соединена с: 7 6 4 3 3  2-ая вершина соединена с: 7 3  3-ая вершина не соединена ни с одной вершиной:  4-ая вершина соединена с: 6 5 7 2  5-ая вершина не соединена ни с одной вершиной:  6-ая вершина соединена с: 5 2  7-ая вершина соединена с: 3 | Результат:  3->7->5->2->6->4->1 |
| 3 | Начальные данные:  1-ая вершина соединена с: 2  2-ая вершина соединена с: 3  3-ая вершина соединена с: 1 | Результат:  3->2->1 |
| 4 | Начальные данные:  1-ая вершина соединена с: 5  2-ая вершина соединена с: 3  3-ая вершина не соединена ни с одной вершиной:  4-ая вершина не соединена ни с одной вершиной:  5-ая вершина соединена с: 4 | Результат:  4->5->1->3->2 |
| 5 | Начальные данные:  1-ая вершина соединена с: 2 3 4  2-ая вершина не соединена ни с одной вершиной:  3-ая вершина соединена с: 4  4-ая вершина соединена с: 2 | Результат:  2->4->3->1 |
| 6 | Начальные данные:  1-ая вершина соединена с: 7 2 6  2-ая вершина соединена с: 4 8 7  3-ая вершина соединена с: 4  4-ая вершина не соединена ни с одной вершиной:  5-ая вершина соединена с: 7  6-ая вершина не соединена ни с одной вершиной:  7-ая вершина соединена с: 3 4  8-ая вершина соединена с: 4  9-ая вершина соединена с: 2 1 6  10-ая вершина соединена с: 8 | Результат:  4->3->7->8->2->6->1->5->9->10 |
| 7 | Начальные данные:  1-ая вершина соединена с: 10 5  2-ая вершина соединена с: 6 4  3-ая вершина соединена с: 8 9  4-ая вершина соединена с: 3 10 9 5  5-ая вершина не соединена ни с одной вершиной:  6-ая вершина не соединена ни с одной вершиной:  7-ая вершина соединена с: 5 10 6 9  8-ая вершина соединена с: 5 1  9-ая вершина соединена с: 1 5 6  10-ая вершина соединена с: 6 | Результат:  6->10->5->1->8->9->3->4->2->7 |
| 8 | В файле некорректные данные:  З  Р  0 0 | Error: «Некорректные данные» |
| 9 | Ввод вручную  5 вершин  4 2  2 5  8 7 | Error: «Нельзя соединить данные вершины» |
| 10 | Ввод из файла  4  7  1 3  2 4  4 1  2 3  2 1  7 7  3 4 | Error: «Попытка связать несуществующие вершины» |

5. ЗАКЛЮЧЕНИЕ

Таким образом, был реализован алгоритм обхода орграфа в глубину с визуализацией. Был разработан интерфейс удобный и понятный для пользователя. Так же в ходе учебной практики были получены знания по языку программирования Java, получены навыки в визуализации алгоритмов и работе в команде.

ПРИЛОЖЕНИЕ А

Обход орграфа в глубину

**Main.java**

package newpackege;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.\*;

import java.util.ArrayList;

import javax.swing.JFrame;

import javax.swing.JSpinner;

import javax.swing.SpinnerModel;

import javax.swing.SpinnerNumberModel;

public class Main {

static int step = 0;//счетчики

static int currVer = 0;//

static int kol = 0;//счетчик вершин

static int count = 0;//номер шага

public static void main(String[] args) throws IOException {

JFrame frame = new JFrame("Обход ориентированного графа в глубину");

Color c1 = new Color(0x6495ED);

frame.getContentPane().setBackground(c1);

Visualisator visio = new Visualisator();

frame.setSize(new Dimension(970, 640));//устанавливаем размер

frame.setResizable(false);//не двигаем

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);//вырубаем все

frame.setLayout(null);

//левое поле

JButton btnDepht = new JButton("Добавить вершину");

btnDepht.setSize(new Dimension(150, 50));

btnDepht.setLocation(5, 5);

JButton data = new JButton("Выбрать файл");

data.setSize(new Dimension(150, 50));

data.setLocation(210, 5);

JLabel lb1 = new JLabel("Добавить ребро");

lb1.setSize(new Dimension(150, 50));

lb1.setLocation(29, 45);

JLabel lb2 = new JLabel("из");

lb2.setSize(new Dimension(75, 50));

lb2.setLocation(35, 65);

JLabel lb3 = new JLabel("в");

lb3.setSize(new Dimension(75, 50));

lb3.setLocation(125, 65);

SpinnerModel model = new SpinnerNumberModel(1, 1, 50, 1);

JSpinner from = new JSpinner(model);

from.setSize(new Dimension(55, 25));

from.setLocation(10, 105);

SpinnerModel model2 = new SpinnerNumberModel(1, 1, 50, 1);

JSpinner to = new JSpinner(model2);

to.setSize(new Dimension(55, 25));

to.setLocation(100, 105);

JButton btnEdge = new JButton("Добавить ребро");

btnEdge.setSize(new Dimension(150, 50));

btnEdge.setLocation(5, 135);

JButton btnStart = new JButton("Старт");

btnStart.setSize(new Dimension(150, 50));

btnStart.setLocation(5, 190);

JButton btnNext = new JButton("Следующий шаг");

btnNext.setSize(new Dimension(150, 50));

btnNext.setLocation(5, 245);

JButton btnResult = new JButton("Результат");

btnResult.setSize(new Dimension(150, 50));

btnResult.setLocation(5, 300);

JButton btnClear\_fields = new JButton("Очистить поля");

btnClear\_fields.setSize(new Dimension(150, 50));

btnClear\_fields.setLocation(5, 355);

//центральное поле

JTextArea out = new JTextArea(); //промежуточные результаты

JScrollPane my\_pane2 = new JScrollPane(out);

my\_pane2.setBounds(170, 60, 220, 345);

out.setEditable(false);

JTextArea start\_data = new JTextArea(); //начальная матрица

JScrollPane my\_pane = new JScrollPane(start\_data);

my\_pane.setBounds(5, 410, 385, 200);

start\_data.setEditable(false);

//правое поле

Algo panel = new Algo();

panel.adj = new ArrayList<ArrayList<Integer>>();//создаем пустой список вершин

panel.setSize(565, 530);

panel.setLocation(395, 5);

JTextArea textresult = new JTextArea();

textresult.setSize(565, 68);

textresult.setLocation(395, 540);

textresult.setEditable(false);

//размещаем

frame.add(btnDepht);

frame.add(lb1);

frame.add(lb2);

frame.add(lb3);

frame.add(from);

frame.add(to);

frame.add(btnEdge);

frame.add(btnStart);

frame.add(btnNext);

frame.add(data);

frame.add(panel);

frame.add(textresult);

frame.add(btnResult);

frame.add(btnClear\_fields);

frame.add(my\_pane2);

frame.add(my\_pane);

frame.setVisible(true);

btnStart.setEnabled(false);

btnClear\_fields.setEnabled(true);

btnDepht.setEnabled(true);

btnEdge.setEnabled(true);

btnNext.setEnabled(false);

btnResult.setEnabled(false);

btnEdge.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

panel.inc\_M();

int v = (Integer) from.getValue();

int w = (Integer) to.getValue();

if (panel.search\_for(v - 1, w - 1)) {

panel.changeDataEdge(v - 1, w - 1);

panel.functionVisual();

panel.repaint();

} else {

JOptionPane.showMessageDialog(null, "Нельзя соединить данные вершины!", "ERROR", JOptionPane.ERROR\_MESSAGE);

}

}

});

btnStart.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (!panel.adj.isEmpty()) {

start\_data.append("Начальные данные: " + "\n");

out.append("Промежуточные выводы: " + "\n");

panel.used = new boolean[visio.get\_N()];

panel.run();

data.setEnabled(false);

btnDepht.setEnabled(false);

btnEdge.setEnabled(false);

btnStart.setEnabled(false);

btnNext.setEnabled(true);

for (int i = 0; i < visio.get\_N(); i++) {

start\_data.append(panel.print\_start\_data(i) + "\n");

}

} else {

btnStart.setEnabled(false);

JOptionPane.showMessageDialog(null, "Данные отсутствуют!", "ERROR", JOptionPane.ERROR\_MESSAGE);

}

}

});

btnDepht.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (panel.get\_N() == 50) {

JOptionPane.showMessageDialog(null, "Запрещено работать более чем с 50 вершинами!", "ERROR", JOptionPane.ERROR\_MESSAGE);

} else {

btnStart.setEnabled(true);

panel.inc\_N();

panel.changeData\_adj();

panel.functionVisual();

panel.repaint();

}

}

});

btnNext.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.out.println(step);

System.out.println(panel.listEdges\_1.size());

if (panel.dfsVisual(step, currVer, kol)) {

kol++;

step++;

currVer = step;

out.append(panel.output.get(count) + "\n");

count++;

System.out.println("Next");

} else {

int st=panel.dfsVisual\_2(step-1, kol);

if(st==-1){

if (panel.dfsVisual\_1(currVer - 1, step, kol)) {

currVer = step;

System.out.println("Exitttt");

} else {

System.out.println("Exit");

currVer--;

}

out.append(panel.output.get(count) + "\n");

step--;

count++;

}else{

out.append("Вершина " + (st+1) + " уже обработана"+"\n");

// coun++;

}

}

if (panel.listDepht.size() == 0) {

out.append("Поиск в глубину завершен\n");

btnNext.setEnabled(false);

btnResult.setEnabled(true);

}

}

});

btnResult.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

textresult.setText(panel.printresult());

btnNext.setEnabled(false);

btnResult.setEnabled(false);

}

});

btnClear\_fields.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

panel.setnull();

panel.adj.clear();

panel.listDepht.clear();

panel.listEdges.clear();

panel.model.remove(this);

panel.points.clear();

panel.result.clear();

panel.output.clear();

panel.listEdges\_1.clear();

panel.repaint();

panel.functionVisual();

start\_data.selectAll();

start\_data.replaceSelection("");

out.selectAll();

out.replaceSelection("");

textresult.selectAll();

textresult.replaceSelection("");

btnDepht.setEnabled(true);

btnEdge.setEnabled(true);

btnResult.setEnabled(false);

data.setEnabled(true);

step = 0;

currVer = 0;

kol = 0;

count = 0;

}

});

data.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

int ret = fileChooser.showDialog(null, "Open file");

if (ret == JFileChooser.APPROVE\_OPTION) {

try {

panel.readData(fileChooser.getSelectedFile());

panel.functionVisual();

btnStart.setEnabled(true);

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(null, "Некорректный данные!", "ERROR", JOptionPane.ERROR\_MESSAGE);

} catch (NullPointerException ex) {

//число пар меньше чем задано во 2 поле

JOptionPane.showMessageDialog(null, "Несоответствие заданного и фактического количества ребер!", "ERROR", JOptionPane.ERROR\_MESSAGE);

} catch (IndexOutOfBoundsException ex) {

//связываются не существующие вершины

JOptionPane.showMessageDialog(null, "Попытка связать несуществующие вершины!", "ERROR", JOptionPane.ERROR\_MESSAGE);

}

}

}

});

}

}

**Algo.java**

package newpackege;

import javafx.util.Pair;

import java.io.\*;

import java.util.ArrayList;

public class Algo extends Visualisator {

protected ArrayList<String> output;//массив для хранения промежуточных выводов

public Algo() throws IOException {//конструктор

this.output = new ArrayList<String>();

}

public void dfs(int v) {//обход в глуюину с записью порядка обхода и промежуточных шагов

used[v] = true;//отмечаем, что вершина просмотрена

output.add("Рассматриваем вершину " + (v + 1));

listDepht.add(v + 1);//добавляем в список в вершин

for (int i = 0; i < adj.get(v).size(); ++i) {//для всех смежных вершин

int w = adj.get(v).get(i);

if (!used[w]) {//если вершина не просмотрена

listEdges.add(new Pair(v, w));

dfs(w);//вызываем dfs

}

else{

listEdges\_1.add(new Pair(v, w));

}

}

result.add(v + 1);//запись результатов

output.add("Вершина " + (v+1) + " обработана");//запись промежуточных шагов

}

public String printresult() {//функция вывода результатов

int k = 0;

String str\_res = "Результат (порядок использованных вершин): ";

for (int i = 0; i < n; i++) {

if (i == k \* 22) {

str\_res += "\n";

k++;

}

if (i == (n - 1)) {

str\_res = str\_res + result.get(i);

} else {

str\_res = str\_res + result.get(i) + "->";

}

}

return str\_res;

}

public void run() {//запуск обхода

for (int v = 0; v < n; ++v) {

if (!used[v]) {

dfs(v);

}

}

}

public boolean search\_for(int v, int w) {//функция проверки добавления ребер

if (w < n && v < n) {

for (int i = 0; i < adj.get(v).size(); i++) {

if (adj.get(v).get(i) == w) {

return false;

}

}

} else {

return false;

}

return true;

}

public void changeDataEdge(int v, int w) {//добавление ребер

adj.get(v).add(w);

}

public void changeData\_adj() {//добавление вершин

adj.add(new ArrayList<Integer>());

}

public void readData(File file) {//считывание с файла

listDepht = new ArrayList<>();

listEdges = new ArrayList<>();

try {

BufferedReader fin = new BufferedReader(new FileReader(file));

n = Integer.parseInt(fin.readLine());

if (n > 50) {

n = 50;

}

m = Integer.parseInt(fin.readLine());

adj = new ArrayList<ArrayList<Integer>>(n);

for (int i = 0; i < n; i++) {

adj.add(new ArrayList<Integer>());

}

for (int i = 0; i < m; i++) {

String[] s = fin.readLine().split(" ");

int v = Integer.parseInt(s[0]);

int w = Integer.parseInt(s[1]);

v--;

w--;

if(search\_for(v,w)){

adj.get(v).add(w);

}

}

} catch (IOException ex) {

}

}

public int getN() {//возвращает количество вершин

return this.listDepht.size();

}

public String print\_start\_data(int i) {//вывод начальных данных(матрицы смежности)

String str\_data = "";

if (adj.get(i).size() != 0) {

str\_data = (i + 1) + "-ая вершина соединена с: ";

for (int j = 0; j < adj.get(i).size(); j++) {

str\_data = str\_data + (adj.get(i).get(j) + 1) + " ";

}

} else {

str\_data = (i + 1) + "-ая вершина не соединена ни с одной вершиной: ";

}

return (str\_data);

}

}

**Visualisator.java**

package newpackege;

import javax.swing.\*;

import com.mxgraph.util.mxConstants;

import com.mxgraph.model.mxIGraphModel;

import com.mxgraph.swing.mxGraphComponent;

import com.mxgraph.view.mxGraph;

import javafx.util.Pair;

import java.awt.\*;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.Objects;

public class Visualisator extends JPanel {

protected static int n;

public int get\_N() {

return n;

};

public void inc\_N() {

n++;

};

public void inc\_M() {

m++;

};

protected boolean provv; // проверка на наличие ребра

protected static int m;

protected ArrayList<ArrayList<Integer>> adj; // список смежности

protected boolean used[]; // массив использованных вершин

protected ArrayList<Integer> listDepht; // список вершин в порядке прямого обхода

protected ArrayList<Pair<Integer, Integer>> listEdges; // список рёбер

protected ArrayList<Pair<Integer, Integer>> listEdges\_1; // список неиспользованных рёбер

protected ArrayList<Object> points; // вершины графа

protected mxIGraphModel model;

protected ArrayList<Integer> result; // список вершин в порядке обратного обхода

public PrintWriter cout;

public void setnull() {

n = 0;

m = 0;

}

@SuppressWarnings("unused")

public Visualisator() throws IOException {

this.setSize(700, 700);

this.setPreferredSize(new Dimension(700, 700));

// создание основных списков

listDepht = new ArrayList<>();

listEdges = new ArrayList<>();

listEdges\_1 = new ArrayList<>();

result = new ArrayList<>(n);

points = new ArrayList<>();

}

public boolean dfsVisual(int step, int stepp, int kol) { // функция визуализации пошагового прямого обхода графа

mxGraph graph = new mxGraph();

graph.setModel(model);

model.beginUpdate();

if (kol == 0) { // если первый шаг

// покраска вершины

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, "pink", new Object[]{points.get(listDepht.get(0) - 1)});

// указание порядкового номера при обходе

graph.getModel().setValue(points.get(listDepht.get(0) - 1), graph.getModel().getValue(points.get(listDepht.get(0) - 1)).toString() + "\n(1)");

} else {

if (step + 1 > listDepht.size()) { // если индекс требуемой вершины находится за пределами списка вершин

System.out.println("первый false");

model.endUpdate();

this.revalidate();

return false;

}

if (step != 0) {

// проверка на связность рядом стоящих верин при прямом обходе

if ((listDepht.size() == 1) || (!listEdges.contains(new Pair<>(listDepht.get(stepp - 1) - 1, listDepht.get(step) - 1)))) {

System.out.println("второй false");

model.endUpdate();

this.revalidate();

return false;

}

}

for (int i = 0; i < listDepht.size(); i++) {

provv = false;

// поиск ребра для указанной вершины

if (listEdges.contains(new Pair<>(listDepht.get(i) - 1, listDepht.get(step) - 1))) {

provv = false;

//покраска пройденного ребра

graph.getModel().setStyle(graph.getEdgesBetween(points.get(listDepht.get(i) - 1), points.get(listDepht.get(step) - 1))[0], "strokeColor=#FF5C6E");

provv = true;

break;

}

}

// покраска вершины

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, "pink", new Object[]{points.get(listDepht.get(step) - 1)});

// установка порядкового номера

graph.getModel().setValue(points.get(listDepht.get(step) - 1), graph.getModel().getValue(points.get(listDepht.get(step) - 1)).toString() + "\n(" + (kol + 1) + ")");

}

System.out.println("первый true");

model.endUpdate();

this.revalidate();

return true;

}

public int used(int step) { // функция поиска порядкового номера при обратном обходе

int st = 0;

for (int i = 0; i < result.size(); i++) {

if (Objects.equals(listDepht.get(step), result.get(i))) {

st = i;

}

}

return st;

}

public boolean dfsVisual\_1(int step, int prov, int kol) { // функция визуализации обратного пошагового обхода графа

mxGraph graph = new mxGraph();

graph.setModel(model);

model.beginUpdate();

provv = false;

if ((listDepht.size() == 1) && (step == -1)) { // если остался один элемен, то необходимо исползовать его( удалить)

// покраска вершины в использованный цвет

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, "violet", new Object[]{points.get(listDepht.get(prov) - 1)});

// назначение номера в порядке обхода(использования)

graph.getModel().setValue(points.get(listDepht.get(step) - 1), graph.getModel().getValue(points.get(listDepht.get(step) - 1)).toString() + ",(" + (used(step) + 1) + ")");

listDepht.remove(prov); // удаление вершины из списка

System.out.println("удаление\_1 true");

model.endUpdate();

this.revalidate();

return true;

}

for (int i = 0; i < listDepht.size(); i++) { // поиск ребра для вершины с индексом step

if (listEdges.contains(new Pair<>(listDepht.get(i) - 1, listDepht.get(step) - 1))) {

// покраска вершины в использованный цвет

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, "violet", new Object[]{points.get(listDepht.get(step) - 1)});

// назначение номера в порядке обхода(использования)

graph.getModel().setValue(points.get(listDepht.get(step) - 1), graph.getModel().getValue(points.get(listDepht.get(step) - 1)).toString() + ",(" + (used(step) + 1) + ")");

// покраска ребра, как пройденного(использованного)

graph.getModel().setStyle(graph.getEdgesBetween(points.get(listDepht.get(i) - 1), points.get(listDepht.get(step) - 1))[0], "strokeColor=#00D6AC");

listDepht.remove(step);// удаление вершины из списка

provv = true;

break;

}

}

if (provv == false) { // если ребра для указанной вершины нет, то перекрашиваем только вершину

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, "violet", new Object[]{points.get(listDepht.get(step) - 1)});

// назначение номера в порядке обхода(использования)

graph.getModel().setValue(points.get(listDepht.get(step) - 1), graph.getModel().getValue(points.get(listDepht.get(step) - 1)).toString() + ",(" + (used(step) + 1) + ")");

listDepht.remove(step);// удаление вершины из списка

} else { // если вершина имеет ребро

System.out.println("удаление false");

model.endUpdate();

this.revalidate();

return false;

}

model.endUpdate();

this.revalidate();

System.out.println("удаление\_2 true");

return true;

}

public void functionVisual() { //функция визуализации графа

removeAll();

mxGraph graph = new mxGraph();

model = graph.getModel();

Object parent = graph.getDefaultParent();

model.beginUpdate();

double phi0 = 0;

double phi = 2 \* Math.PI / n;

int r = 230;

for (int i = 0; i < n; i++) { // создание вершин на поле

points.add(i, graph.insertVertex(parent, null, i + 1, 250 + r \* Math.cos(phi0), 250 + r \* Math.sin(phi0), 40, 40, "shape=ellipse"));

phi0 += phi;

}

for (int i = 0; i < adj.size(); i++) {

for (int j = 0; j < adj.get(i).size(); j++) { // создание ребер

graph.insertEdge(parent, null, null, points.get(i), points.get(adj.get(i).get(j)));

}

}

model.endUpdate();

graph.setAllowDanglingEdges(false);

graph.setCellsResizable(false);

mxGraphComponent graphComponent = new mxGraphComponent(graph);

graphComponent.setConnectable(false);

this.add(graphComponent);

this.revalidate();

}

public int dfsVisual\_2(int step, int kol) { // функция визуализации обратного пошагового обхода графа

mxGraph graph = new mxGraph();

graph.setModel(model);

model.beginUpdate();

provv = false;

if ((listDepht.size() == 1) && (step == -1)) { // если остался один элемен, то необходимо исползовать его( удалить)

step=0;

}

int st=-1;

for (int i = 0; i < n; i++) { // поиск ребра для вершины с индексом step

if (listEdges\_1.contains(new Pair<>( listDepht.get(step) - 1,i))) {

// покраска ребра, как не пройденного(неиспользованного)

graph.getModel().setStyle(graph.getEdgesBetween( points.get(listDepht.get(step) - 1),points.get(i))[0], "strokeColor=#0");

listEdges\_1.remove(new Pair<>( listDepht.get(step) - 1,i));

provv = true;

st=i;

break;

}

}

if (provv == false) { // если ребра для указанной вершины нет, то перекрашиваем только вершину

System.out.println("перекраска false");

model.endUpdate();

this.revalidate();

return st;

}

// если вершина имеет ребро

System.out.println("перекраска true");

model.endUpdate();

this.revalidate();

return st;

}

}