**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: Визуализация алгоритм построения МОД Ярника-Прима-Дейкстры на языке Java

|  |  |  |
| --- | --- | --- |
| Студент   гр. 6383 |  | Азаревич А.Д. |
| Студентка гр. 6383 |  | Гомонова А. А. |
| Студент   гр. 6383 |  | Ульянов П. А. |
| Руководитель |  | Хатбуллина Л.Р. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Азаревич А.Д. группы 6383 | | |
| Студентка Гомонова А.А. группы 6383 | | |
| Студент Ульянов П.А. группы 6383  Тема практики: визуализация алгоритмов на языке Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: алгоритм Ярника-Прима-Дейкстры. | | |
| Сроки прохождения практики: 27.06.2018 – 10.07.2018 | | |
| Дата сдачи отчета: 10.07.2018 | | |
| Дата защиты отчета: 10.07.2018 | | |
|  | | |
| Студент   гр. 6383 |  | Азаревич А.Д. |
| Студентка гр. 6383 |  | Гомонова А. А. |
| Студент   гр. 6383 |  | Ульянов П. А. |
| Руководитель |  | Хатбуллина Л.Р. |

**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 построения минимального остовного дерева Ярника-Прима-Дейкстры.

**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 for constructing the minimal spanning tree Jarnika-Prima-Dijkstra.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc518828732)

[1. ТРЕБОВАНИЯ К ПРОГРАММЕ 6](#_Toc518828733)

[1.1 Исходные требования к программе 6](#_Toc518828734)

[1.2 Изменения к спецификации 7](#_Toc518828735)

[1.2.1 Изменения к спецификации после сдачи первой версии 7](#_Toc518828736)

[1.3 Описание входных и выходных данных 7](#_Toc518828737)

[2. ПЛАН РАЗАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ 8](#_Toc518828738)

[2.1. План разработки 8](#_Toc518828739)

[2.2 Распределение ролей в бригаде 9](#_Toc518828740)

[3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ 10](#_Toc518828741)

[3.1 Использованные структуры данных 10](#_Toc518828742)

[3.2 Использованные возможности библиотеки 15](#_Toc518828743)

[4. ТЕСТИРОВАНИЕ 17](#_Toc518828744)

[4.1 Тестирование графического интерфейса 17](#_Toc518828745)

[4.2 Тестирование кода алгоритма 23](#_Toc518828746)

[ЗАКЛЮЧЕНИЕ 24](#_Toc518828747)

[ПРИЛОЖЕНИЕ А Алгоритм построения МОД Ярника-Прима-Дейкстры 26](#_Toc518828748)

# **ВВЕДЕНИЕ**

Задача построения минимального остовного дерева встречается в различных областях. Интересным ее применением является проблема построения смешанного остовного дерева (*Dana Richards and Jeffrey S. Salowe. Mixed spanning trees in theory and practice. International Journal Of Computational Geometry & Applications. Vol. 9, No. 3 (1999), 277-292*): построить для графа дерево со свойствами минимального остовного дерева и дерева кратчайших путей. Другой важной задачей является быстрое обновление минимального остовного дерева при изменении графа. В статье *Sajal K. Das, Paolo Ferragina "An Erew Pram algorithm for updating minimum spanning trees"* показано, как для графа с *n* вершинами и *m* ребрами выполнить обновление одного ребра за учетное время O(log*n*).

Проблема построения минимального остовного дерева достаточно разносторонняя, и продолжает исследоваться и сегодня. В нашей работе мы рассмотрим один из базовых алгоритмов- алгоритм Ярника-Прима-Дейкстры.

 Алгоритм Ярника-Прима-Дейкстры – алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э. Дейкстрой в 1959 году.

В нашей работе будет реализован визуализатор данного алгоритма, с использованием графического интерфейса.

# **ТРЕБОВАНИЯ К ПРОГРАММЕ**

# Исходные требования к программе

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

1. Поле интерфейса работы с программой:

* Кнопка «добавить вершину» (добавляется в центр спец. поля и может передвигаться мышью).
* Кнопка «Добавить ребро» с полями «из», «в».
* Кнопка «Старт» (запускает работу алгоритма).
* Кнопка «Следующий шаг».
* Кнопка «Очистка» (завершает работу алгоритма, дает возможность дополнить граф).

1. Поле «Сведения о графе»
2. Поле «Логи»
3. Поле «Место, в котором рисуется граф»

При нажатии кнопки «Старт» начинает работу алгоритм ЯПД. Последующее нажатие кнопки «Следующий шаг» приводит к:

* Помечается рассматриваемое (согласно алгоритму) ребро в цвет2.
* Если оно подходит – помечается цветом3, иначе – цветом4.

Примерный вид интерфейса представлен на рис. 1

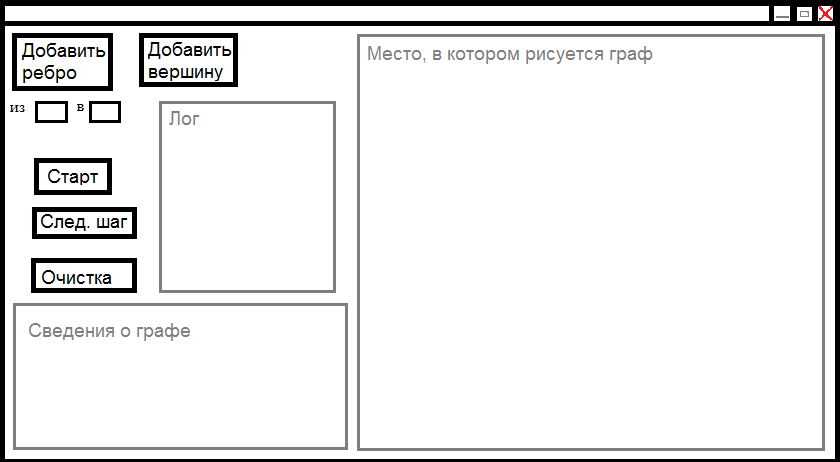


Рисунок 1- Интерфейс программы

# 1.2 Изменения к спецификации

# 1.2.1 Изменения к спецификации после сдачи первой версии

Для удобства пользователя было решено ввести дополнительные функции и кнопки:

«Удалить ребро» и «Удалить вершину» и кнопка «Результат».

Убрали окна "Лог" и "Сведения о графе"; при нажатии кнопки "Следующий шаг" к "ответу" прибавляется нужное ребро и вершина (без промежуточных результатов, как изначально).

Также вершины графа можно передвигать вручную, что обеспечивает значительное удобство для восприятия. Также была добавлена подсветка меток веса соответствующих пройденных ребер.

# 1.3 Описание входных и выходных данных

Входные данные:

Связный, неориентированный граф, можно задавать вручную (с помощью кнопок) или считыванием из файла, где:

Перечисление вершин-

n- число вершин

- Вершины 1 и 2

w- вес ребра между соответствующими вершинами

Выходные данные:

Графическое представление графа с подсветкой всех посещенных вершин и меток веса соответствующих пройденных ребер. Вывод результата (суммарных вес рёбер минимального остовного дерева) при нажатии кнопки «Результат».

# **ПЛАН РАЗАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**

## 2.1 План разработки

30.06.2018(суббота) – разработка спецификации.

01.07.2018 – согласование спецификации с руководителем.

(02.07-03.07).2018 – разработка интерфейса и реализация визуализации графа.

(04.07-05.07).2018 – реализация визуализации графа на интерфейсе, реализация графического отображения алгоритма.

06.07.2018(пятница)– сдача 1-й версии с добавлением частичной функциональности к пользовательскому интерфейсу.

(07.07-09.07).2018 – внесение правок, тестирование, оформление отчёта.

10.07.2018 – сдача финальной версии. Предоставление проекта с полной функциональностью.

## 2.2 Распределение ролей в бригаде

В таблице 1 представлен состав бригады и распределение обязанностей.

Таблица 1

|  |  |
| --- | --- |
| Имя студента | Обязанности |
| Азаревич Артём | Реализация алгоритма Ярника-Прима-Дейкстры. Осуществление пошаговой работы алгоритма и выведение результата. |
| Гомонова Анастасия | Осуществление пошаговой реализации алгоритма на графической части интерфейса. Тестирование программы. |
| Ульянов Пётр | Визуализация графа, реализация методов, отвечающих за работу с графом (удаление заданных ребер и вершин) |

# **ОСОБЕННОСТИ РЕАЛИЗАЦИИ**

## 3.1 Использованные структуры данных

Граф для реализации алгоритма представляется в виде списка смежности.

Программа состоит из восьми классов: Algorithm, APD, AdjacenyList, Graph, ActiveVertex, GraphField, Main, MainWindow и одного дополнительного PAR\_S с глобальными константами.

***Класс Main***

Класс Main отвечает за изображение графического интерфейса.

В основе работы класса лежат методы класса JFrame, запускаемые в отдельном потоке. Данные методы реализуют графическое поле программы и кнопки.

Таблица 2

|  |  |
| --- | --- |
| Метод | Предназначение |
| setDefaultCloseOperation(operation) | Указывает операцию, которая будет произведена при закрытии окна |
| setTitle(str) | Устанавливает название окна |
| setSize(int width, int height) | Устанавливает размеры окна |
| setVisible(bool) | Отображает окно |
| add( new MainWindow(), BorderLayout.WEST ); | Вызывается для размещения панельки с кнопками и изображения графа слева |

***Класс MainWindow***

Таблица 3

|  |  |
| --- | --- |
| Название панели | Функциональность |
| createButtons | Создает панель с кнопками |
| createAlgorithmButtons | Панель с кнопками, управляющими алгоритмом |
| createGraphButtons | Панель с кнопками, управляющими созданием графа |
| createAddEdge | Панель, отвечающая за добавление ребра |
| glueParametrs | Склейка окошка для ввода и текста |
| createAddVertex | Кнопка, добавляющая в граф вершины |

Таблица 4

|  |  |  |
| --- | --- | --- |
| Название метода | Title | Возможные сообщение |
| startAlgorithm | "Ошибка старта алгоритма" | 1. "Ошибка: алгоритм уже запущен" 2. "Ошибка: для запуска алгоритма нужен граф, состоящий " +     "минимум из 2-х вершин", 3. "Ошибка: у каждой вершины должно быть минимум по 1-му ребру" 4. "Ошибка: алгоритм уже закончил работу" |
| nextStep | "Ошибка алгоритма" | 1. "Ошибка: Чтобы сделать шаг, запустите алгоритм" 2. "Ошибка: алгоритм уже закончил работу" |
| result | "Ошибка алгоритма" | "Ошибка: Чтобы получить результат работы алгоритма, сперва запустите алгоритм" |
| clearAll | Функция очистки панели с построенным графом | - |
| createEdge | "Ошибка добавления ребра" | 1. "Ошибка: запрещено добавлять ребра пока алгоритм работает" 2. "Ошибка: концы и вес ребра должны задаваться натуральными числами" 3. "Ошибка: нельзя создавать петли" 4. "Ошибка: Вес ребра должен задаваться натуральным числом" |
| addVertex |  | Функция, которая запускает процесс добавления вершины |

***Класс Graph***

Таблица 5

|  |  |
| --- | --- |
| Название метода | Предназначение |
| addV | Функция добавления вершины в граф |
| addE | Функция добавления ребра в граф |
| checkV | Функция проверки вершины |
| checkE | Функция проверки ребра |
| сlear | Функция удаления графа/Очистка панели |
| getKolE | Функция вывода информации о количестве ребер |
| getKolV | Функция вывода информации о количестве вершин |

***Class ActiveVertex***

Таблица 6

|  |  |
| --- | --- |
| Название метода | Предназначение |
| Point | Изображение вершин |
| ActiveVertex | Функция передвижения вершин |

***Class AdjacenyList***

Таблица 7

|  |  |
| --- | --- |
| Название метода | Предназначение |
| adjacenyList HashMap:  <Integer, Vertex> | Для представления графа  (списки смежности) |

***Class Algorithm***

Таблица 8

|  |  |
| --- | --- |
| Название метода | Предназначение |
| start() | Функция запуска алгоритма |
| step() | Функция пошаговой реализации алгоритма (Поиск минимального ребра от i-ой вершины «во-вне» |
| result() | Функция подсчета суммарного веса ребер МОДа |
| clear() | Функция завершения запуска алгоритма и очистки поля с графом |

## 3.2 Использованные возможности библиотеки

*java.util.ArrayList*- вид списков

*java.util.HashMap* - HashMap - основан на хэш-таблицах, реализует интерфейс Map (что подразумевает хранение данных в виде пар ключ/значение). Ключи и значения могут быть любых типов, в том числе и null. Данная реализация не дает гарантий относительно порядка элементов с течением времени.   
 *java.util.Map* - Интерфейс Map располагается на вершине иерархии в [Java Collection](http://java-online.ru/java-collection.xhtml). Он входит в состав JDK начиная с версии 1.2, предоставляя разработчику базовые методы для работы с данными вида «ключ — значение».  
 *java.awt.event.MouseEven* - Интерфейс MouseEvent представляет собой событие, которое происходит в результате взаимодествия пользователя с манипулятором ( например, мышью).   
 *java.awt.event.MouseListener* - реализует обработку событий от мыши.  
 *java.awt.event.MouseMotionListener* - реализует обработку событий от мыши.  
 *java.util.Random* - генератор случайных чисел.

*javax.swing* - Библиотека Swing предназначениа для создания графического интерфейса desktop'ых приложений, разрабатываемых на языке Java.   
 *java.awt* - Abstract Window Toolkit (AWT) — это исходная платформо-независимая оконная библиотека графического интерфейса (Widget toolkit) языка [Java](https://ru.wikipedia.org/wiki/Java).  
 *java.io.FileInputStream* – для считывания данных с файла.  
 *java.io.IOException* - используется для исключений, возникающих при доступе к данным с помощью потоков, файлов и каталогов.

*java.util.Scanner* - используется для ввода данных.

*java.awt* - это исходная платформо-независимая оконная библиотека графического интерфейса (Widget toolkit) языка [Java](https://ru.wikipedia.org/wiki/Java).

3.3 UML-диаграмма классов

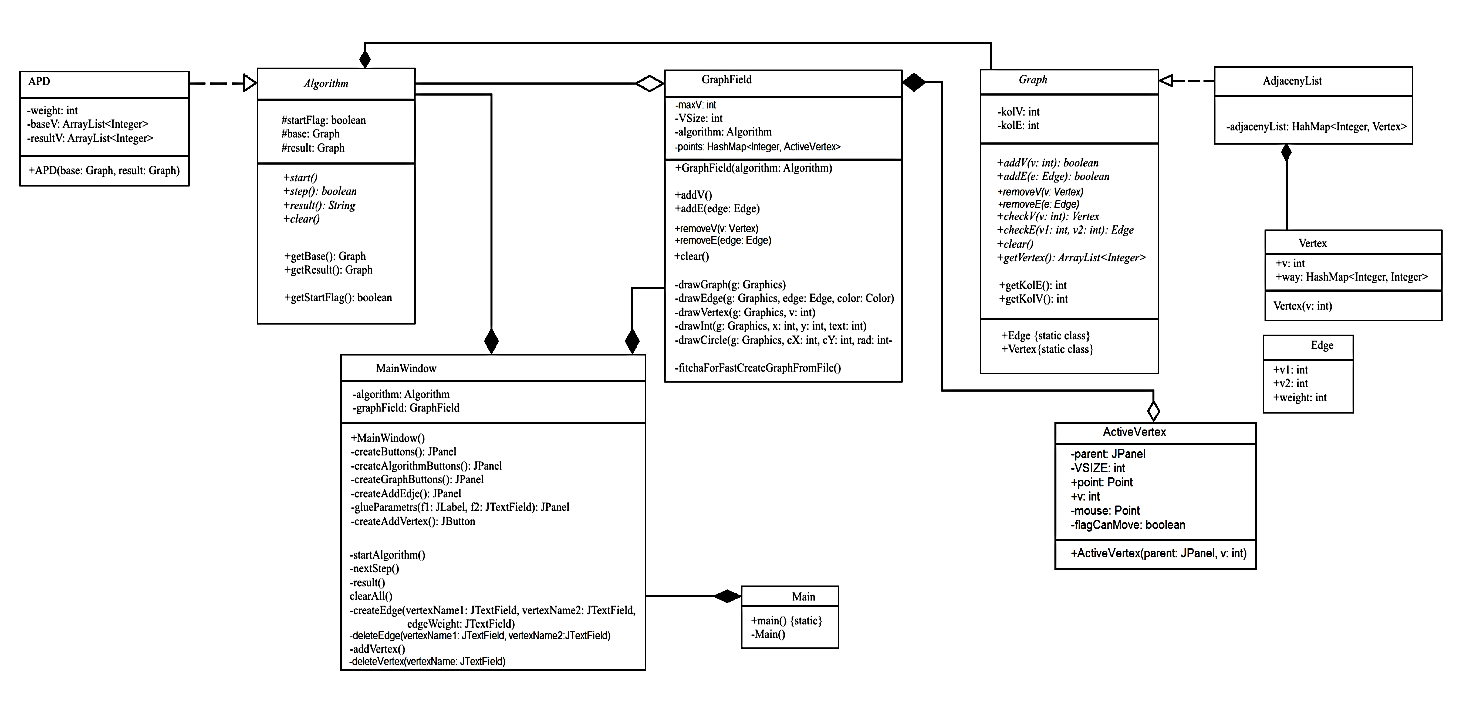


Рисунок 2- UML- диаграмма (построен в программе Paint)

# **4. ТЕСТИРОВАНИЕ**

# Тестирование графического интерфейса

1. Запускаем jar файл
2. Добавляем 5 вершин, нажатием на кнопку «Добавить вершину». На экране по порядку нажатия появляются 5 вершин (0-4). Вершины можно передвигать для удобства восприятия. (Рис.8)

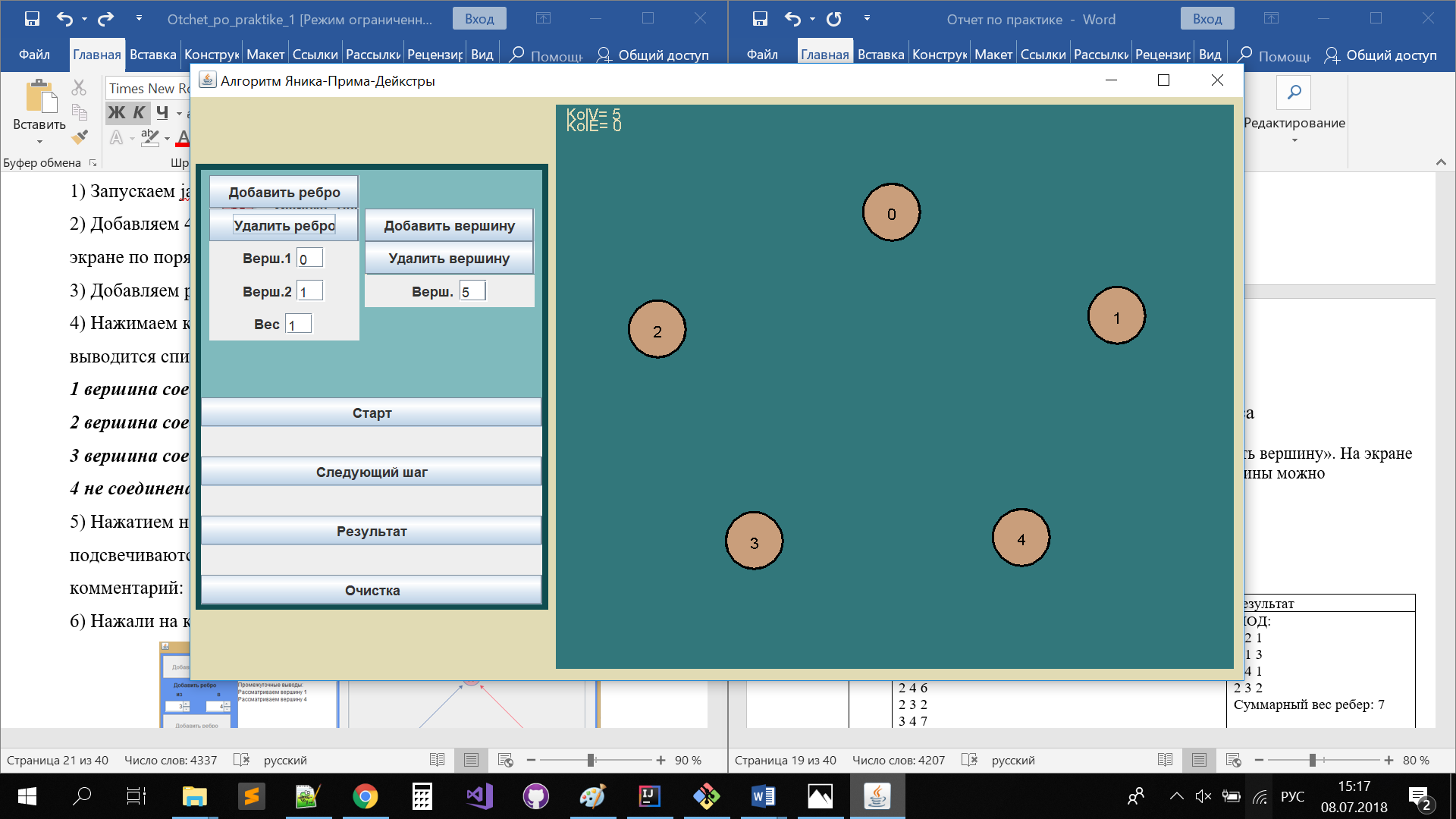


Рисунок 3- Добавление вершины

1. Добавляем ребра

В соответствующем окне задаем номера вершины1(Верш.1) и вершины2 (Верш.2) и Вес. Нажимаем кнопку «Добавить ребро» (Рис. 9, Рис.10)

0 1 1

0 2 2

0 3 3

2 3 1

3 4 3

4 0 2

4 1 1

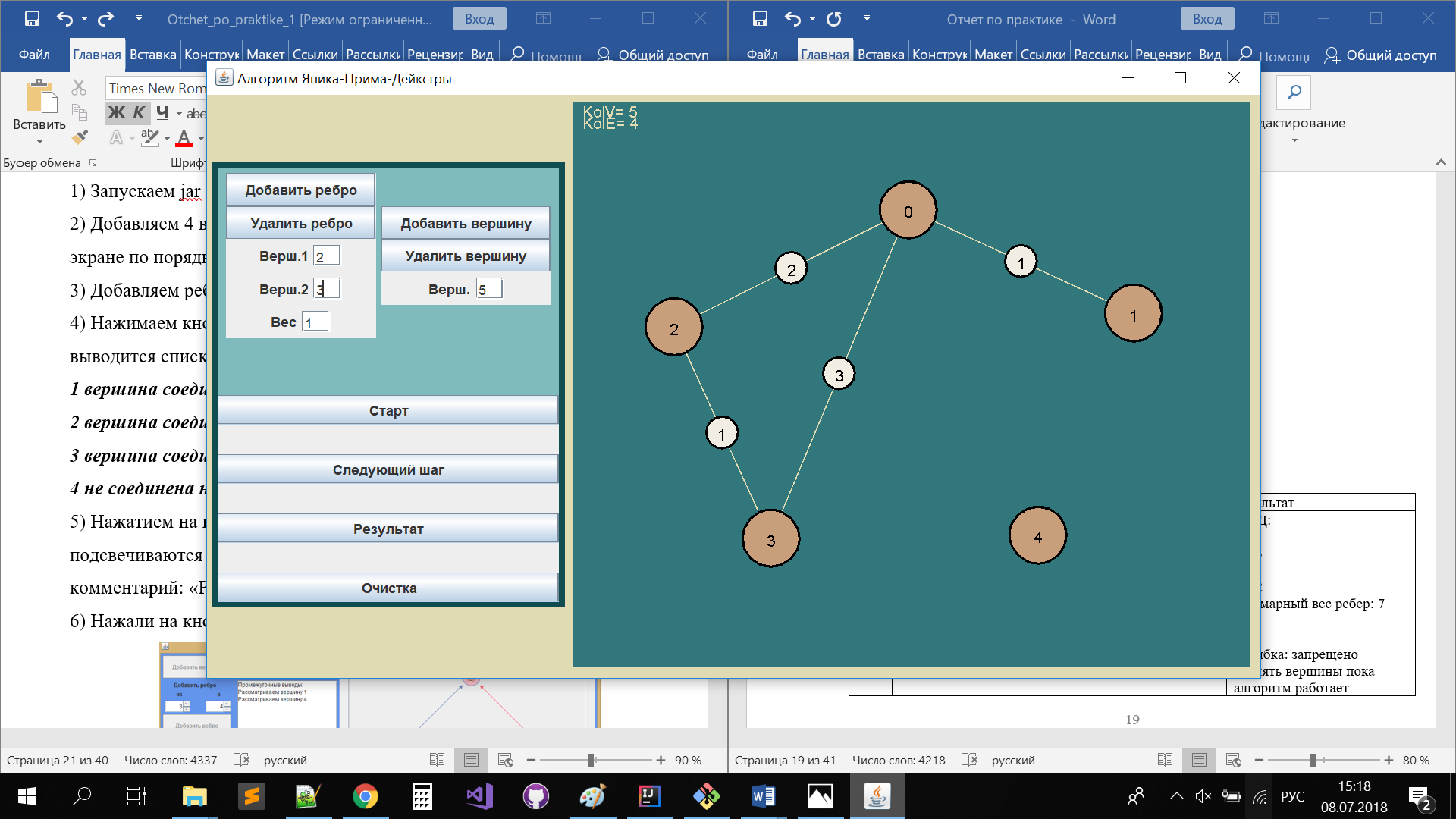


Рисунок 4- Промежуточный результат добавления ребер

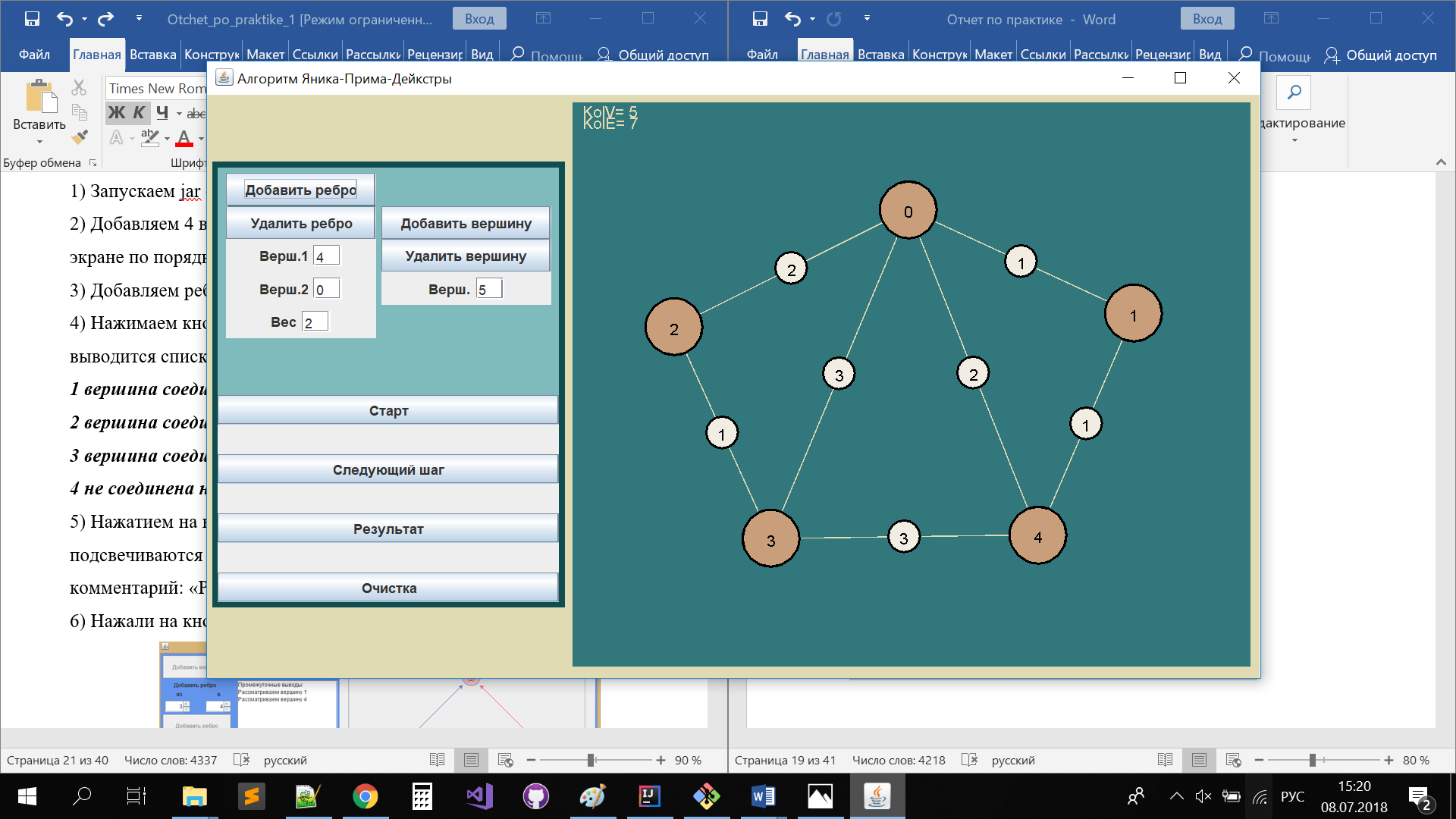


Рисунок 5- Результат добавления ребер

1. Нажимаем на кнопку «Старт», которая запускает алгоритм поиска МОД. (вершина 0 меняет цвет)

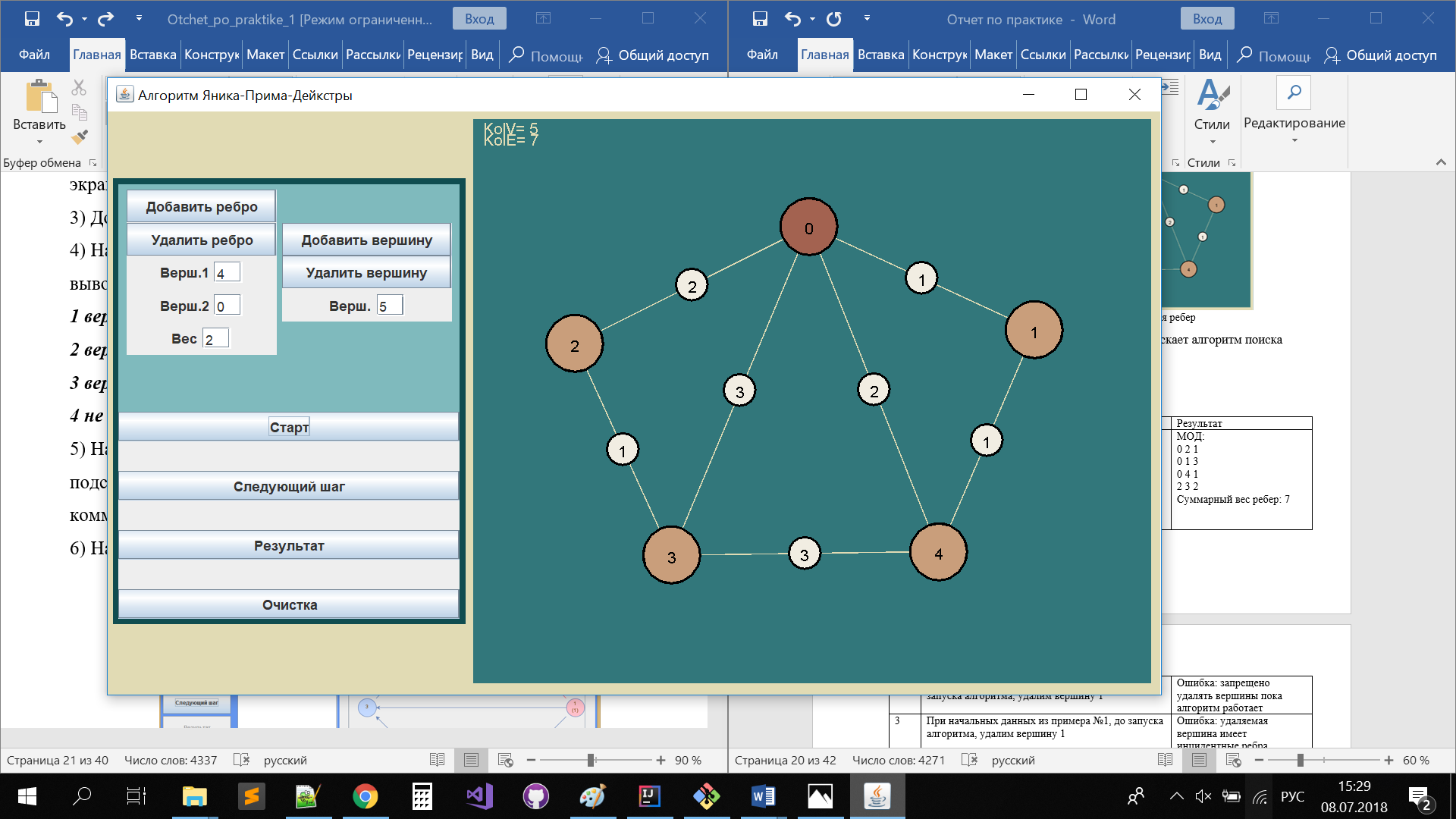


Рисунок 6- Запуск алгоритма поиска МОД

1. Нажатием на кнопку «Следующий шаг» на графическом поле подсвечивается вершина 1.

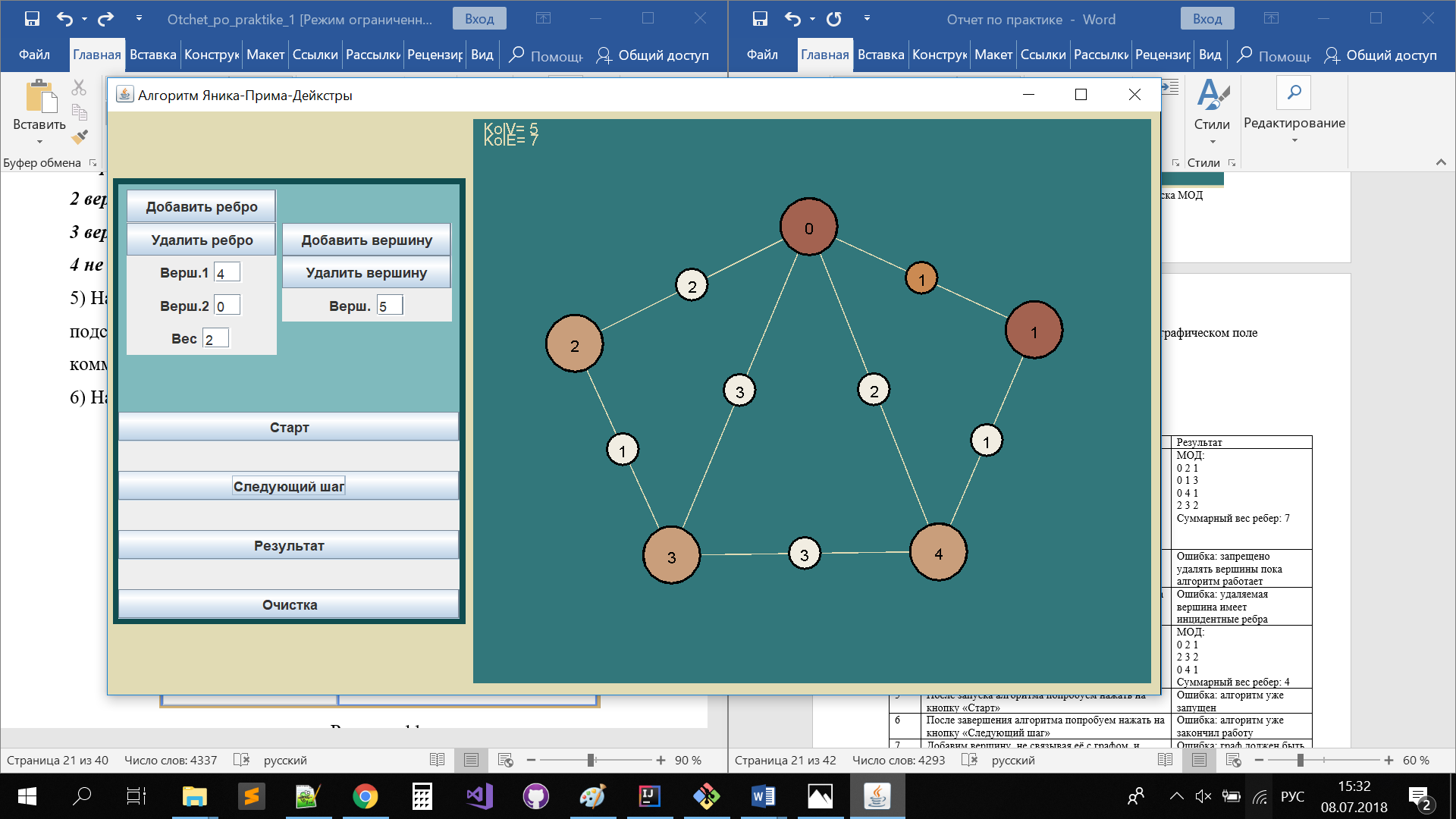


Рисунок 7- "Следующий шаг"

1. Нажали на кнопку «Следующий шаг»

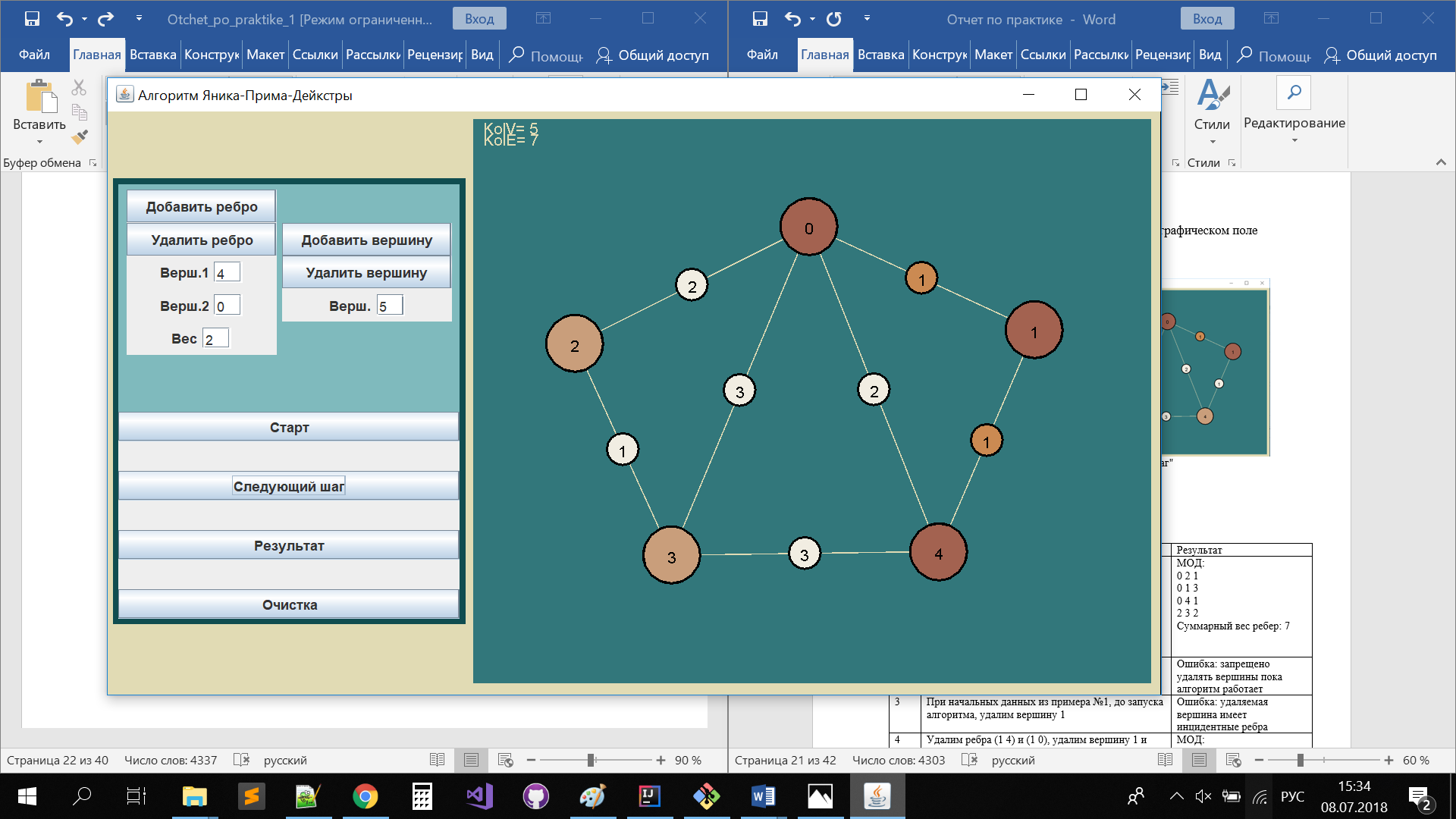


Рисунок 8-"Следующий шаг»

1. Нажали на кнопку «Следующий шаг»

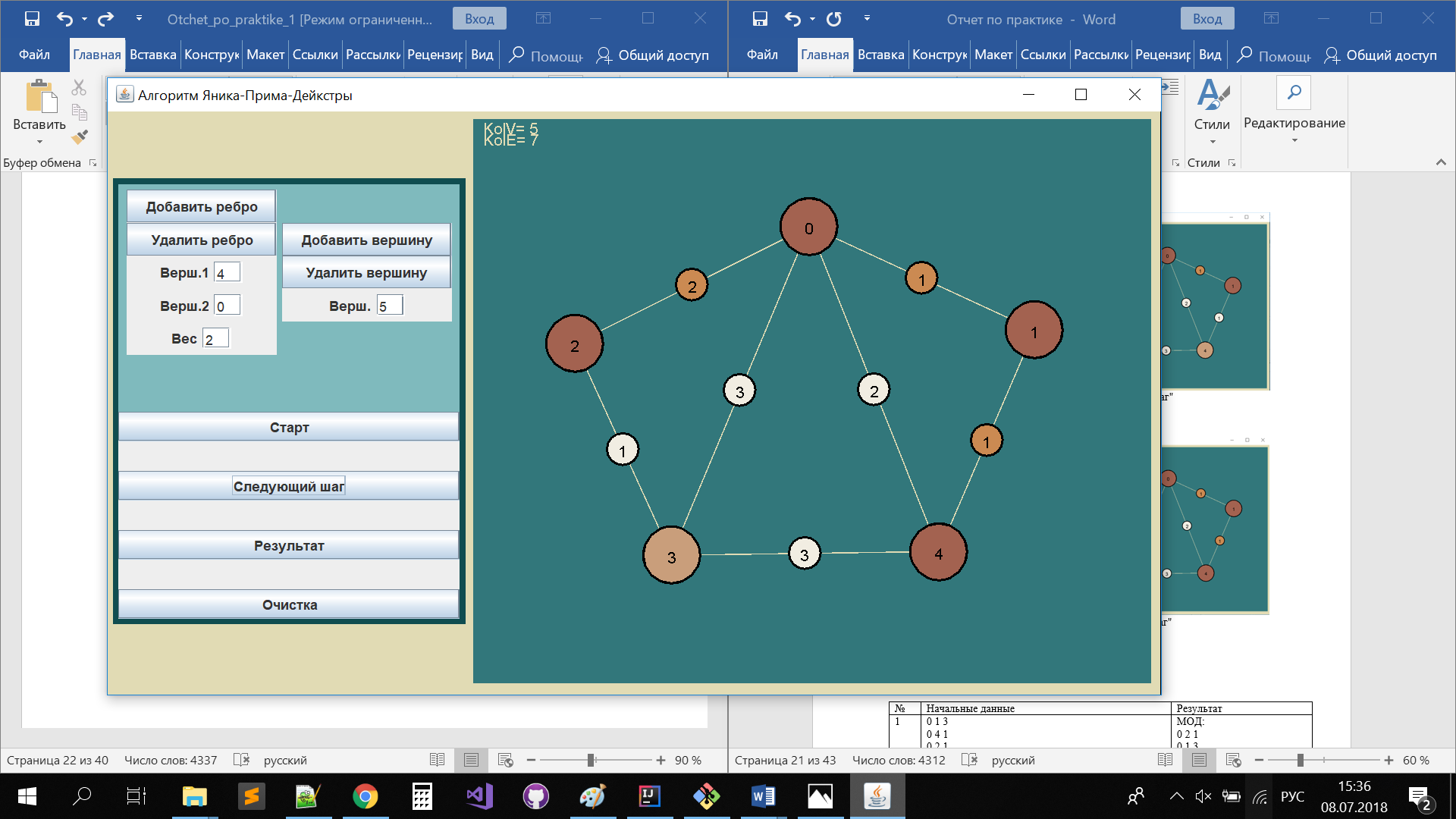


Рисунок 9- "Следующий шаг"

1. Нажали на кнопку «Следующий шаг»

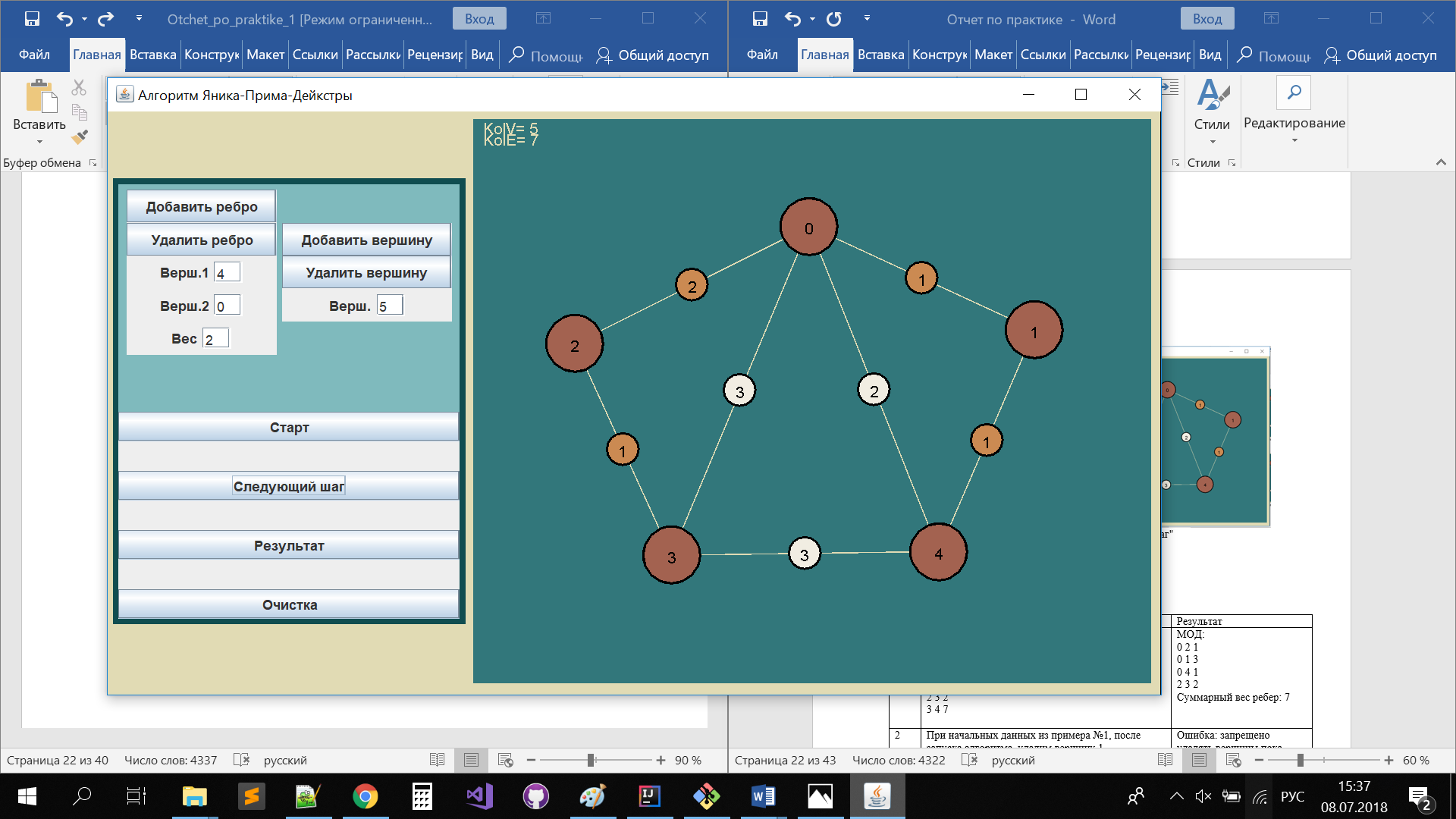


Рисунок 10- "Следующий шаг»

1. Нажали на кнопку «Следующий шаг» - Алгоритм закончил работу, вывод соответствующего сообщения.

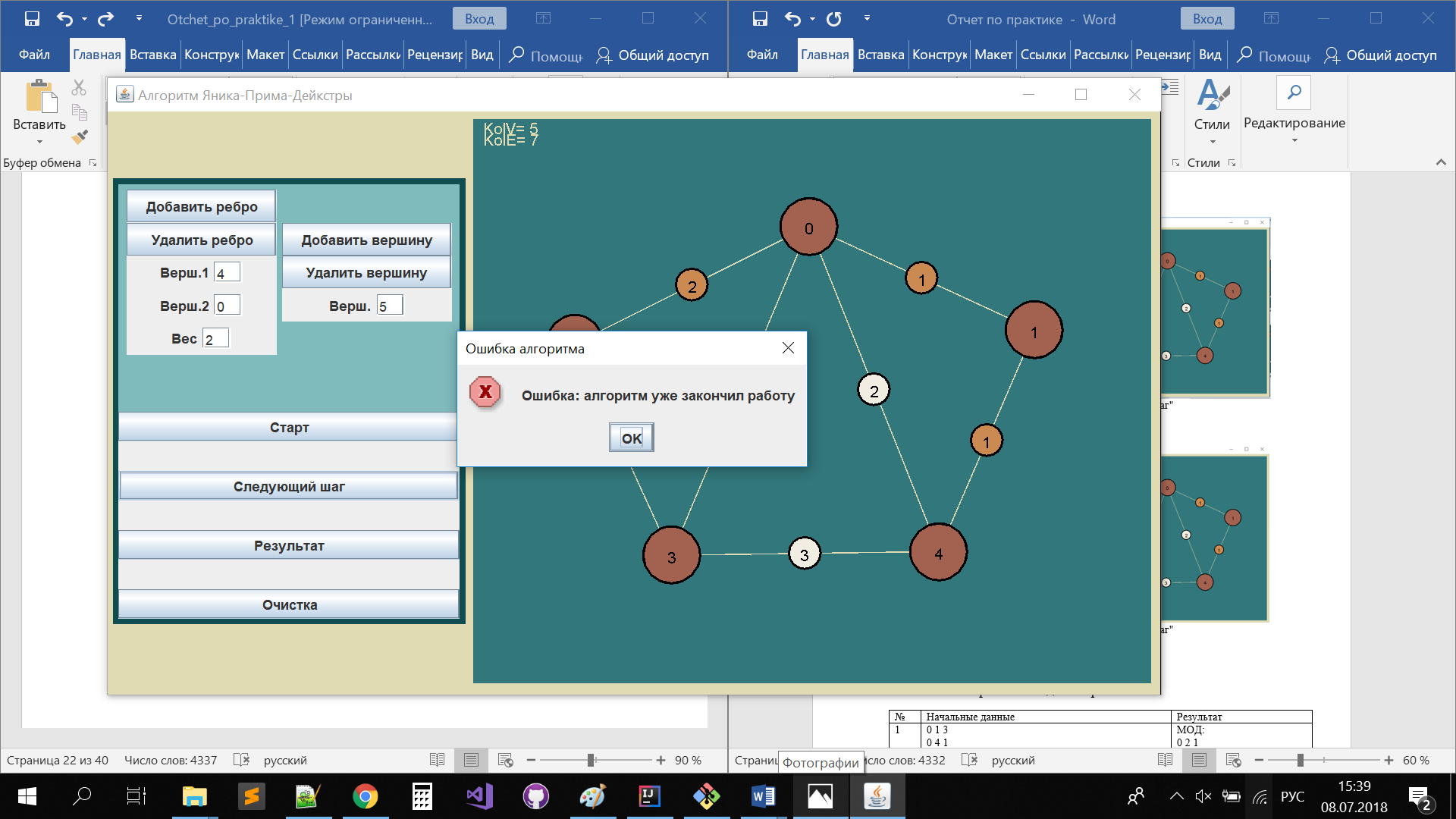


Рисунок 11- "Следующий шаг"

1. Нажали на кнопку «Результат»

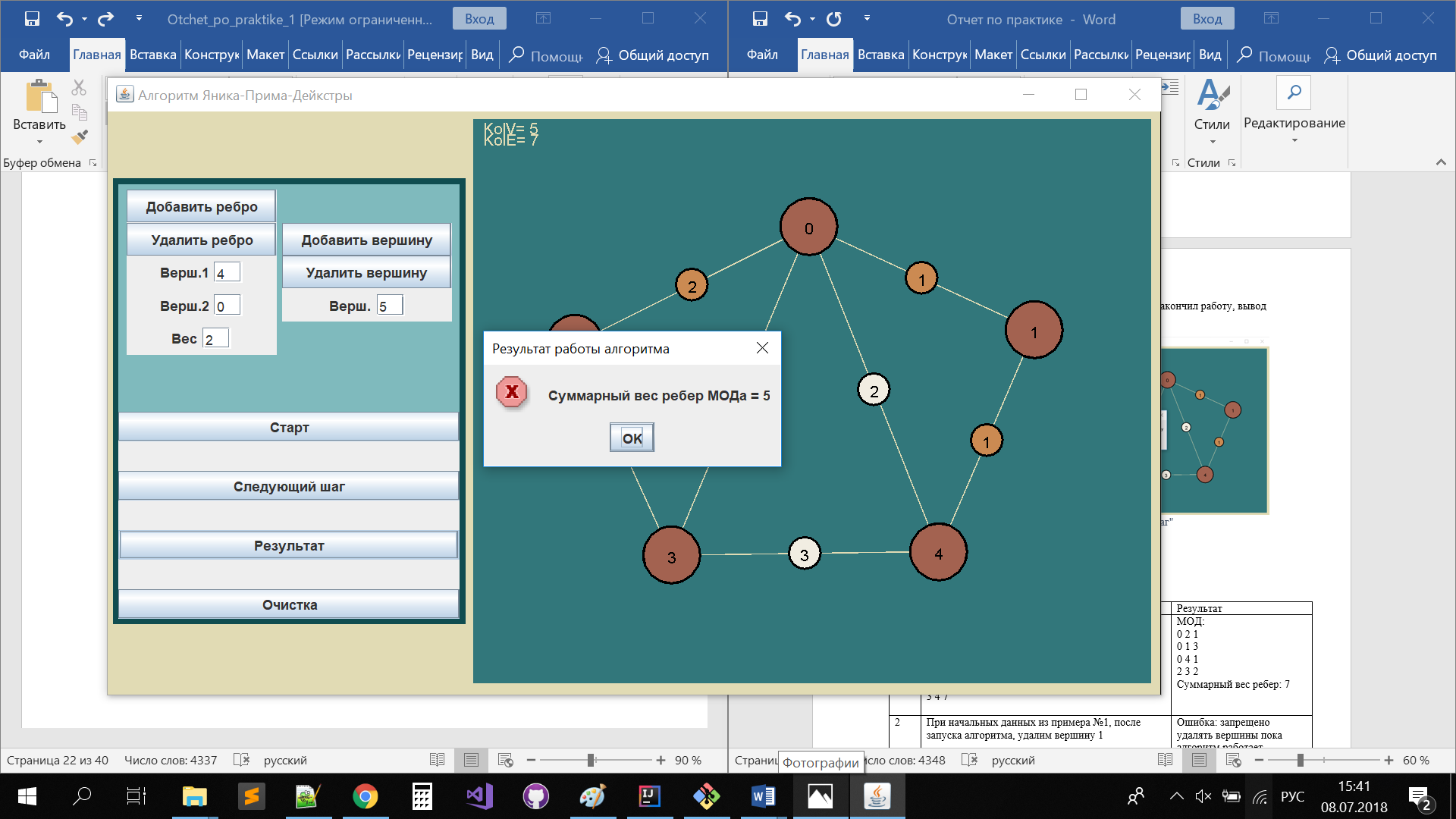


Рисунок 12- "Результат"

1. Нажали на кнопку «Очистка»

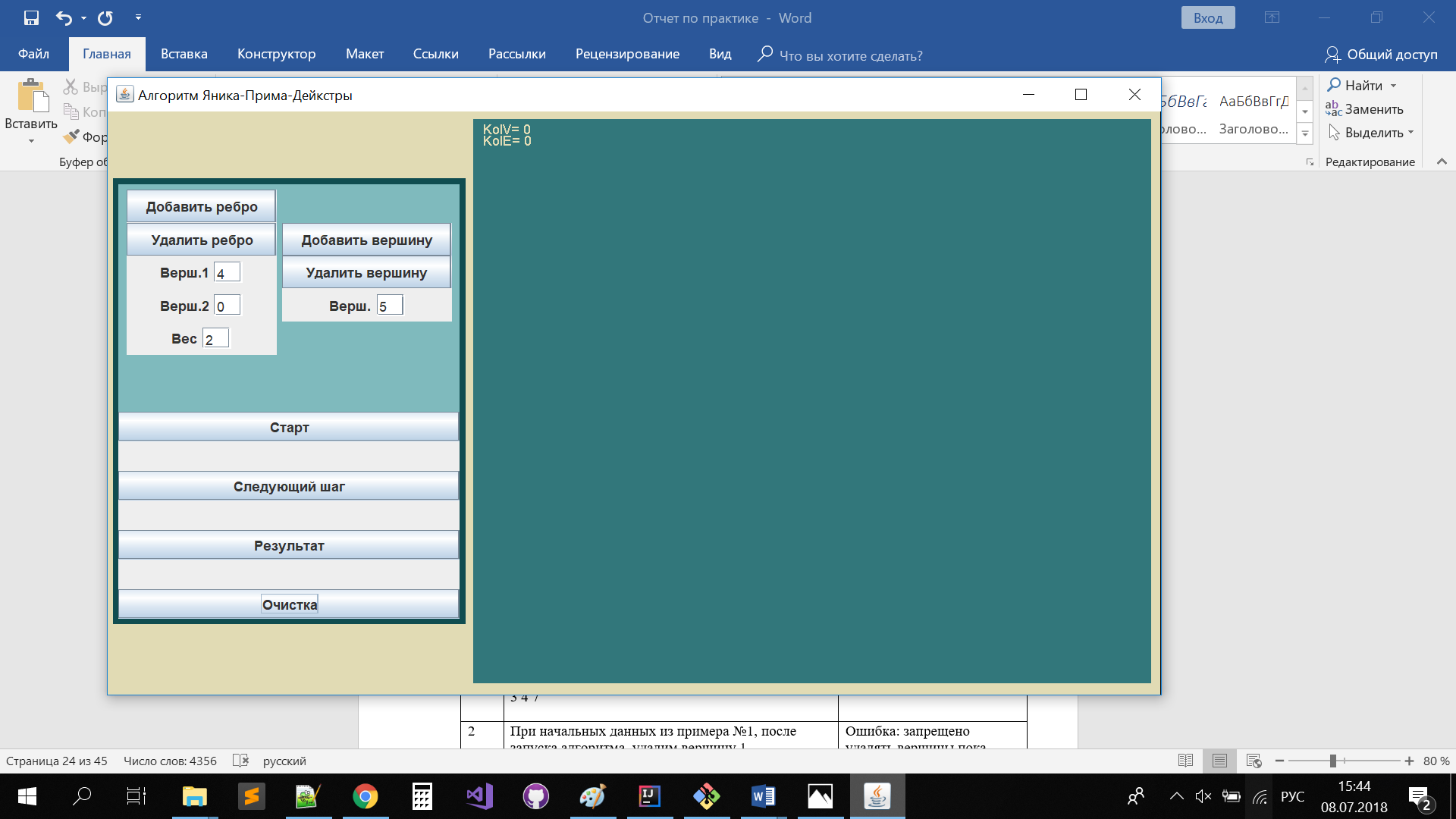


Рисунок 13- "Очистка"

# 4.2 Тестирование кода алгоритма

Таблица 9

|  |  |  |
| --- | --- | --- |
| № | Начальные данные | Результат |
| 1 | 0 1 3  0 4 1  0 2 1  1 4 4  2 4 6  2 3 2  3 4 7 | МОД:  0 2 1  0 1 3  0 4 1  2 3 2  Суммарный вес ребер: 7 |
| 2 | При начальных данных из примера №1, после запуска алгоритма, удалим вершину 1 | Ошибка: запрещено удалять вершины пока алгоритм работает |
| 3 | При начальных данных из примера №1, до запуска алгоритма, удалим вершину 1 | Ошибка: удаляемая вершина имеет инцидентные ребра |
| 4 | Удалим ребра (1 4) и (1 0), удалим вершину 1 и запустим алгоритм | МОД:  0 2 1  2 3 2  0 4 1  Суммарный вес ребер: 4 |
| 5 | После запуска алгоритма попробуем нажать на кнопку «Старт» | Ошибка: алгоритм уже запущен |
| 6 | После завершения алгоритма попробуем нажать на кнопку «Следующий шаг» | Ошибка: алгоритм уже закончил работу |
| 7 | Добавим вершину, не связывая её с графом, и запустим алгоритм | Ошибка: граф должен быть связным |
| 8 | Попробуем добавить ребро к несуществующим вершинам | Ошибка: Вершина 6 не существует |
| 9 | При удалении несуществующей вершины | Ошибка: Такой вершины нет |
| 10 | Запустим алгоритм, не создавая графа | Ошибка: для запуска алгоритм нужен граф, состоящий минимум из двух вершин |
| 11 | Не запуская алгоритма, нажмем на кнопку «Следующий шаг» | Ошибка: чтобы сделать шаг, запустите алгоритм |
| 12 | Нажмем на кнопку «Результат» без запуска алгоритма | Ошибка: чтобы получить результат работы алгоритма, сперва запустите алгоритм |
| 13 | Начальные данные:  1 2 -1 | Ошибка: Вес ребра должен задаваться натуральным числом |
| 14 | Начальные данные:  0 1 2  1 2 1  1 3 2  1 4 2  2 4 5  2 3 1  1 1 1 | Ошибка: нельзя создавать петли |
| 15 | Начальные данные:  1 2 1  1 3 4  0 1 2  3 1 1  1 3.2 5 | Ошибка: концы и вес ребра должны задавать натуральными числами |

# 

# 

# ЗАКЛЮЧЕНИЕ

Таким образом, был реализован алгоритм построения МОД Ярника-Прима-Дейкстры. 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 Алгоритм построения МОД Ярника-Прима-Дейкстры

**Main.java**

|  |
| --- |
| package Window; |
|  |  |
|  | import javax.swing.\*; |
|  | import java.awt.\*; |
|  |  |
|  | import static Window.PAR\_S.\*; |
|  |  |
|  | public class Main extends JFrame { |
|  |  |
|  |  |
|  | public static void main(String[] args) { |
|  | new Main(); |
|  | } |
|  |  |
|  | Main() { |
|  | setDefaultCloseOperation(EXIT\_ON\_CLOSE); |
|  |  |
|  | setTitle("Алгоритм Яника-Прима-Дейкстры"); |
|  | setSize( WINDOW\_SIZE ); |
|  |  |
|  | setResizable(false); |
|  |  |
|  | add( new MainWindow(), BorderLayout.WEST ); |
|  |  |
|  | setVisible(true); |
|  | } |
|  | } |

**MainWindow.java**

|  |
| --- |
| package Window; |
|  | import Algorithm.\*; |
|  | import Graph.\*; |
|  | import javax.swing.\*; |
|  | import java.awt.\*; |
|  | import java.awt.event.ActionEvent; |
|  | import static Window.PAR\_S.\*; |
|  | public class MainWindow extends JPanel { |
|  |  |
|  | private Algorithm algorithm; |
|  | private GraphField graphField; |
|  | public MainWindow() { |
|  | algorithm = new APD(new AdjacenyList(), new AdjacenyList()); |
|  | setBackground( BACKGROUND ); |
|  |  |
|  | add( createButtons() ); |
|  | graphField = new GraphField( (algorithm)); |
|  | add(graphField); |
|  | } |
|  | private JPanel createButtons() { |
|  | JPanel panel = new JPanel( ); |
|  | Box box = Box.createVerticalBox(); |
|  | panel.setBackground( BUTTENS\_BORDER ); |
|  |  |
|  | box.add(createGraphButtons()); |
|  | box.add(createAlgorithmButtons()); |
|  |  |
|  | panel.add(box, BorderLayout.SOUTH); |
|  | return panel; |
|  | } |
|  | private JPanel createAlgorithmButtons() { |
|  | JPanel algorithmButtons = new JPanel( new GridLayout(7, 1) ); |
|  |  |
|  | algorithmButtons.add(new JButton(new AbstractAction("Старт") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | startAlgorithm(); |
|  | } |
|  | })); |
|  | algorithmButtons.add(Box.createVerticalStrut(5)); |
|  | algorithmButtons.add(new JButton(new AbstractAction("Следующий шаг") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | nextStep(); |
|  | } |
|  | })); |
|  | algorithmButtons.add(Box.createVerticalStrut(5)); |
|  | algorithmButtons.add(new JButton(new AbstractAction("Результат") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | result(); |
|  | } |
|  | })); |
|  | algorithmButtons.add(Box.createVerticalStrut(5)); |
|  | algorithmButtons.add(new JButton(new AbstractAction("Очистка") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | clearAll(); |
|  | } |
|  | })); |
|  |  |
|  | return algorithmButtons; |
|  | } |
|  | private JPanel createGraphButtons() { |
|  | JPanel graphButtons = new JPanel(); |
|  | graphButtons.setBackground( CREATE\_BUTTONS\_BG ); |
|  |  |
|  | graphButtons.add(createAddEdge()); |
|  | graphButtons.add(createAddVertex()); |
|  |  |
|  | graphButtons.setPreferredSize( CREATE\_GRAPH\_PANEL\_SIZE ); |
|  |  |
|  | return graphButtons; |
|  | } |
|  | private JPanel createAddEdge(){ |
|  |  |
|  | JTextField vertexName1 = new JTextField(); vertexName1.setPreferredSize( SIZE\_OF\_INPUT\_FIELD); |
|  | JTextField vertexName2 = new JTextField(); vertexName2.setPreferredSize( SIZE\_OF\_INPUT\_FIELD); |
|  | JTextField edgeWeight = new JTextField(); edgeWeight.setPreferredSize (SIZE\_OF\_INPUT\_FIELD); |
|  |  |
|  | JButton addEdjeButton = new JButton(new AbstractAction("Добавить ребро") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | createEdge(vertexName1, vertexName2, edgeWeight); |
|  | } |
|  | }); |
|  |  |
|  | JButton deleteEdjeButton = new JButton(new AbstractAction("Удалить ребро") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | deleteEdge(vertexName1, vertexName2); |
|  | } |
|  | }); |
|  |  |
|  | JPanel addEdge = new JPanel( new GridLayout(5,1) ); |
|  | addEdge.add(addEdjeButton); |
|  | addEdge.add(deleteEdjeButton); |
|  | addEdge.add( glueParametrs(new JLabel("Верш.1"),vertexName1 ) ); |
|  | addEdge.add( glueParametrs(new JLabel("Верш.2"),vertexName2 ) ); |
|  | addEdge.add( glueParametrs(new JLabel("Вес" ), edgeWeight ) ); |
|  |  |
|  | return addEdge; |
|  | } |
|  | private JPanel glueParametrs(JLabel f1, JTextField f2) { |
|  | JPanel parametr = new JPanel(); |
|  | parametr.add(f1); |
|  | parametr.add(f2); |
|  | return parametr; |
|  | } |
|  | private JPanel createAddVertex(){ |
|  |  |
|  | JTextField vertexName = new JTextField(); vertexName.setPreferredSize( SIZE\_OF\_INPUT\_FIELD); |
|  |  |
|  | JButton addVertexButton = new JButton(new AbstractAction("Добавить вершину") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { addVertex(); } |
|  | }); |
|  |  |
|  | JButton deleteVertexButton = new JButton(new AbstractAction("Удалить вершину") { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | deleteVertex(vertexName); |
|  | } |
|  | }); |
|  |  |
|  | JPanel addVertex = new JPanel( new GridLayout(3,1) ); |
|  |  |
|  | addVertex.add(addVertexButton); |
|  | addVertex.add(deleteVertexButton); |
|  | addVertex.add( glueParametrs(new JLabel("Верш."),vertexName ) ); |
|  |  |
|  | return addVertex; |
|  | } |
|  | private void startAlgorithm() { |
|  | if (algorithm.getStartFlag()) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: алгоритм уже запущен", |
|  | "Ошибка старта алгоритма", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  |  |
|  | int check = algorithm.getBase().getKolV(); |
|  |  |
|  | if (check < 2 ){ |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: для запуска алгоритма нужен граф, состоящий " + |
|  | "минимум из 2-х вершин", |
|  | "Ошибка старта алгоритма", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  |  |
|  | for (int i=0; i < check; i++) { |
|  | if ( algorithm.getBase().kolEinV(i) == 0) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: у каждой вершины должно быть минимум по 1-му ребру", |
|  | "Ошибка старта алгоритма", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | } |
|  | algorithm.start(); |
|  | repaint(); |
|  | } |
|  |  |
|  | private void nextStep() { |
|  | if (!algorithm.getStartFlag()) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: Чтобы сделать шаг, запустите алгоритм", |
|  | "Ошибка алгоритма", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | if (!algorithm.step() ) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: алгоритм уже закончил работу", |
|  | "Ошибка алгоритма", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | repaint(); |
|  | } |
|  | private void result() { |
|  | if (!algorithm.getStartFlag()) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: Чтобы получить результат работы алгоритма, сперва запустите алгоритм", |
|  | "Ошибка алгоритма", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  |  |
|  | JOptionPane.showMessageDialog(this, |
|  | algorithm.result(), |
|  | "Результат работы алгоритма", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  |  |
|  | repaint(); |
|  | } |
|  | private void clearAll() { |
|  | graphField.clear(); |
|  | repaint(); |
|  | } |
|  |  |
|  | private void createEdge( JTextField vertexName1, JTextField vertexName2, JTextField edgeWeight) { |
|  | if (algorithm.getStartFlag()) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: запрещено добавлять ребра пока алгоритм работает", |
|  | "Ошибка добавления ребра", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | Graph.Edge edge = new Graph.Edge(-1,-1,-1); |
|  | try { |
|  | edge.v1 = Integer.parseInt(vertexName1.getText()); |
|  | edge.v2 = Integer.parseInt(vertexName2.getText()); |
|  | edge.weight = Integer.parseInt(edgeWeight.getText()); |
|  | } |
|  | catch (NumberFormatException e) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: концы и вес ребра должны задаваться натуральными числами", |
|  | "Ошибка добавления ребра", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  |  |
|  | if (edge.v1==edge.v2) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: нельзя создавать петли", |
|  | "Ошибка добавления ребра", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | if ( edge.weight <= 0) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: Вес ребра должен задаваться натуральным числом", |
|  | "Ошибка добавления ребра", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  |  |
|  | graphField.addE(edge); |
|  |  |
|  | } |
|  | private void deleteEdge( JTextField vertexName1, JTextField vertexName2) { |
|  | if (algorithm.getStartFlag()) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: запрещено удалять ребра пока алгоритм работает", |
|  | "Ошибка удаления ребра", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | Graph.Edge edge = new Graph.Edge(-1,-1,-1); |
|  |  |
|  | try { |
|  | edge.v1 = Integer.parseInt(vertexName1.getText()); |
|  | edge.v2 = Integer.parseInt(vertexName2.getText()); |
|  | } |
|  | catch (NumberFormatException e) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: концы ребра должны задаваться натуральными числами", |
|  | "Ошибка удаления ребра", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  |  |
|  | if (edge.v1==edge.v2) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: нельзя использовать петли", |
|  | "Ошибка удаления ребра", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  |  |
|  | graphField.removeE(edge); |
|  |  |
|  | } |
|  |  |
|  | private void addVertex(){ |
|  | graphField.addV(); |
|  | repaint(); |
|  | } |
|  |  |
|  | private void deleteVertex(JTextField vertexName){ |
|  | if (algorithm.getStartFlag()) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: запрещено удалять вершины пока алгоритм работает", |
|  | "Ошибка удаления вершины", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | Graph.Vertex v = new Graph.Vertex(-1); |
|  |  |
|  | try { |
|  | v.v = Integer.parseInt(vertexName.getText()); |
|  | } |
|  | catch (NumberFormatException e) { |
|  | JOptionPane.showMessageDialog(this, |
|  | "Ошибка: номер вершины должен задаваться натуральным числом", |
|  | "Ошибка удаления вершины", |
|  | JOptionPane.ERROR\_MESSAGE); |
|  | return; |
|  | } |
|  | graphField.removeV(v); |
|  | repaint(); |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**ActiveVertex.java**

package Window;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;  
import java.awt.event.MouseMotionListener;  
import java.util.Random;  
  
public class ActiveVertex extends JPanel implements MouseListener, MouseMotionListener {  
  
 JPanel parent;  
 final int VSIZE = 50;  
 Point point;  
 final int v;  
  
 private Point mouse = new Point();  
  
 private boolean flagCanMove = false;  
  
  
 ActiveVertex( JPanel parent, int v ) {  
 this.parent = parent;  
 this.v = v;  
  
 Random random = new Random();  
 point = new Point(random.nextInt(600-VSIZE), random.nextInt(500-VSIZE));  
  
  
 setSize(new Dimension(VSIZE,VSIZE));  
 setLocation(point.x-VSIZE/2, point.y-VSIZE/2);  
 setBackground(new Color(0xFFFFFF));  
  
 addMouseMotionListener(this);  
 addMouseListener(this);  
 }  
  
   
 @Override  
 public void mousePressed(MouseEvent e) {  
 flagCanMove = true;  
 mouse.x = e.getX();  
 mouse.y = e.getY();  
 }  
 @Override  
 public void mouseReleased(MouseEvent e) {  
 flagCanMove = false;  
 }  
 @Override  
 public void mouseDragged(MouseEvent e) {  
 if (flagCanMove) {  
 int dx = e.getX() - mouse.x;  
 int dy = e.getY() - mouse.y;  
  
 point.x += dx;  
 point.y += dy;  
  
 setLocation(point.x-VSIZE/2, point.y-VSIZE/2);  
 parent.repaint();  
 }  
 }  
  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 }  
 @Override  
 public void mouseEntered(MouseEvent e) {  
 }  
 @Override  
 public void mouseExited(MouseEvent e) {  
 }  
 @Override  
 public void mouseMoved(MouseEvent e) {  
  
 }  
}

**GraphField.java**

package Window;  
  
import Algorithm.Algorithm;  
import Graph.\*;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Scanner;  
  
  
public class GraphField extends JPanel {  
 int VSize = 50;  
 int maxV = 0;  
 Algorithm algorithm;  
 private HashMap<Integer, ActiveVertex> points = new HashMap<Integer, ActiveVertex>();  
  
 private int emptyPoint(){  
 int i;  
 for(i=0; points.containsKey(i); i++){};  
 return i;  
 }  
  
 GraphField(Algorithm algorithm) {  
  
 setLayout(null);  
 setPreferredSize( new Dimension(600,500));   
 this.algorithm = algorithm;  
  
 fitchaForFastCreateGraphFromFile();  
 }

public void addV() {  
  
 if (algorithm.getStartFlag()) {  
 JOptionPane.showMessageDialog(this,  
 "Ошибка: запрещено добавлять вершины пока алгоритм работает",  
 "Ошибка добавления вершины",  
 JOptionPane.ERROR\_MESSAGE);  
 return;  
 }  
  
 int n = emptyPoint();  
 if (n > maxV){maxV = n;}  
 algorithm.getBase().addV(n);  
  
 points.put(n, new ActiveVertex(this, n));  
 add(points.get(n));  
  
 }  
 public void addE(Graph.Edge edge){  
 try {  
 algorithm.getBase().addE(edge);  
 }  
 catch (RuntimeException exception) {  
 JOptionPane.showMessageDialog(this,  
 "Ошибка: " + exception.getLocalizedMessage(),  
 "Ошибка добавления ребра",  
 JOptionPane.ERROR\_MESSAGE);  
 return;  
 }  
  
 repaint();  
 }  
  
 public void removeV(Graph.Vertex v) {  
  
 if (algorithm.getBase().kolEinV(v.v) > 0) {  
 JOptionPane.showMessageDialog(this,  
 "Ошибка: удаляемая вершина имеет инцидентные рёбра",  
 "Ошибка удаления вершины",  
 JOptionPane.ERROR\_MESSAGE);  
 return;  
 }  
  
 try {  
 algorithm.getBase().removeV(v);  
 remove(points.get(v.v));  
 points.remove(v.v);  
 }  
 catch (RuntimeException exception) {  
 JOptionPane.showMessageDialog(this,  
 "Ошибка: " + exception.getLocalizedMessage(),  
 "Ошибка удаления вершины",  
 JOptionPane.ERROR\_MESSAGE);  
 return;  
 }  
  
 repaint();  
 }  
 public void removeE(Graph.Edge edge){  
 try {  
 algorithm.getBase().removeE(edge);  
 }  
 catch (RuntimeException exception) {  
 JOptionPane.showMessageDialog(this,  
 "Ошибка: " + exception.getLocalizedMessage(),  
 "Ошибка удаления ребра",  
 JOptionPane.ERROR\_MESSAGE);  
 return;  
 }  
  
 repaint();  
 }  
  
 @Override  
 public void paint(Graphics g) {  
  
 g.setColor( new Color(50, 119, 123));  
 g.fillRect(0,0,1000,1000);  
  
 drawGraph(g);  
  
 g.setColor( new Color(225, 219, 180));   
 ((Graphics2D)g).setStroke(new BasicStroke(4));   
 g.drawRect( 0, 0, 600, 500);   
  
 g.drawString("KolV= " + algorithm.getBase().getKolV() , 10,15);  
 g.drawString("KolE= " + algorithm.getBase().getKolE() , 10,25);  
  
 }  
 private void drawGraph(Graphics g) {  
 Graph.Edge edge;  
  
 Color baseV = new Color(201, 158, 123);  
 Color resultV = new Color(163, 98, 80);  
  
 Color baseE = new Color(241, 237, 226);  
 Color resultE = new Color(203, 138, 82);  
  
 ActiveVertex i;  
 int av=0;  
 for (int w=0;w<points.size();w++) {  
  
 for(;!points.containsKey(av);av++){};  
 i = points.get(av);  
 av++;  
  
 boolean inRes = algorithm.getResult().checkV(i.v)!=null ? true : false;  
  
 for(int j=i.v; j < maxV+1; j++){  
 if ( ( edge = algorithm.getBase().checkE(i.v,j)) != null ) {  
 Color color;  
 if (inRes && (algorithm.getResult().checkE(i.v,j)!= null) ) color = resultE;  
 else color = baseE;  
 drawEdge(g, edge, color);  
 }  
 g.setColor(inRes ? resultV : baseV);  
 drawVertex(g, i.v);  
 }  
 }  
 }  
 private void drawEdge(Graphics g, Graph.Edge edge, Color color){  
  
 Point v1 = new Point(points.get(edge.v1).point.x, points.get(edge.v1).point.y);  
 Point v2 = new Point(points.get(edge.v2).point.x, points.get(edge.v2).point.y);  
  
 ((Graphics2D)g).setStroke(new BasicStroke(1));   
  
 g.setColor( new Color(225, 219, 180) );  
 g.drawLine(v1.x, v1.y, v2.x, v2.y);  
  
 int x = (v1.x+v2.x)/2;  
 int y = (v1.y+v2.y)/2;  
  
 g.setColor(color);  
 g.fillOval(x-14, y-14, 28,28);  
  
 ((Graphics2D)g).setStroke(new BasicStroke(2));  
 g.setColor(new Color(0, 0, 0));  
 g.drawOval(x-14, y-14, 28,28);  
  
 drawInt(g, x, y, edge.weight);  
 }

private void drawVertex(Graphics g, int v) {  
 drawCircle(g, points.get(v).point.x, points.get(v).point.y, VSize/2);  
 drawInt(g, points.get(v).point.x, points.get(v).point.y, v);  
 }  
 private void drawInt(Graphics g, int x, int y, int text) {  
 g.setColor(Color.BLACK);  
 Font font = new Font("Default", Font.PLAIN, 14);  
  
 g.setFont(font);  
  
 FontMetrics fm = g.getFontMetrics(font);  
  
 g.drawString(Integer.toString(text),  
 x-fm.stringWidth(Integer.toString(text))/2,  
 y+fm.getAscent()/2);  
 }  
 private void drawCircle(Graphics g, int cX, int cY, int rad) {  
 g.fillOval(cX-rad, cY-rad, rad\*2, rad\*2);  
  
 ((Graphics2D)g).setStroke(new BasicStroke(2));  
 g.setColor(new Color(0, 0, 0));  
 g.drawOval(cX-rad, cY-rad, rad\*2, rad\*2);  
 }  
  
 public void clear() {  
 if (!algorithm.getStartFlag()) points = new HashMap<Integer, ActiveVertex>();  
 algorithm.clear();  
 }  
 private void fitchaForFastCreateGraphFromFile() {  
 try (FileInputStream file = new FileInputStream("FastGraph.txt");) {  
 Scanner scanner = new Scanner(file);  
  
 int kolV = scanner.nextInt();  
  
 HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();  
  
 for (int i=0; i < kolV; i++) {  
 map.put( scanner.nextInt(), i);  
 addV();  
 }  
  
 while (scanner.hasNext()) {  
 int v1 = scanner.nextInt();  
 int v2 = scanner.nextInt();  
 int weight = scanner.nextInt();  
  
 addE(new Graph.Edge(map.get(v1).intValue(), map.get(v2).intValue(), weight));  
 }  
 } catch (IOException e) { }  
 }  
}

**Algorithm.java**

package Algorithm;  
  
import Graph.Graph;  
  
public abstract class Algorithm {  
 protected boolean startFlag = false;  
 protected Graph base;   
 protected Graph result;   
  
 abstract public void start();  
 abstract public boolean step();  
 abstract public String result();  
  
 abstract public void clear();  
  
 public Graph getBase() {  
 return base;  
 }  
 public Graph getResult() {  
 return result;  
 }  
  
 public boolean getStartFlag() {return startFlag;}  
  
}

**APD.java**

package Algorithm;  
  
import Graph.Graph;  
  
import java.util.ArrayList;  
import java.util.Map;  
  
public class APD extends Algorithm {  
  
 private int weight;  
 private ArrayList<Integer> baseV;  
 private ArrayList<Integer> resultV;  
  
 public APD(Graph base, Graph result){  
 this.base = base;  
 this.result = result;  
 }  
  
  
 @Override  
 public void start() {  
 baseV = base.getVertexes();  
 resultV = new ArrayList<Integer>();  
 resultV.add(baseV.get(0));  
 result.addV(baseV.get(0));  
  
 startFlag = true;  
 weight = 0;  
 }  
  
 @Override  
 public boolean step() {  
  
 if (base.getKolV() != result.getKolV()) {  
 Graph.Edge minE = new Graph.Edge(0, 0, Integer.MAX\_VALUE);  
  
 for (int i : resultV) {  
 Graph.Edge minE\_i;  
 Graph.Vertex v\_i = base.checkV(i);  
 for(Map.Entry<Integer,Integer> j: v\_i.way.entrySet()) {  
 if ( (result.checkV(j.getKey().intValue())==null) &&   
 ( j.getValue().intValue() < minE.weight )) {   
 minE.v1 = i;  
 minE.v2 = j.getKey().intValue();  
 minE.weight = j.getValue().intValue();   
 }  
 }  
 }  
 result.addV(minE.v2);  
 resultV.add(minE.v2);  
 result.addE(minE);  
 weight += minE.weight;  
  
 return true;  
 }  
 return false;  
 }  
  
 @Override  
 public String result() {  
 while (step());  
 return "Суммарный вес ребер МОДа = "+weight;  
 }  
  
 @Override  
 public void clear() {  
 if (startFlag) {  
 startFlag = false;  
 result.clear();  
 baseV = null;  
 } else {  
 base.clear();  
 }  
 }  
  
}

**AdjacenyList.java**

|  |
| --- |
| package Graph; |
|  | import java.util.ArrayList; |
|  | import java.util.HashMap; |
|  | import java.util.Map; |
|  |  |
|  | public class AdjacenyList extends Graph { |
|  |  |
|  | private HashMap<Integer, Vertex> adjacenyList = new HashMap<Integer, Vertex>(); |
|  | @Override |
|  | public boolean addV(int v) { |
|  | if (adjacenyList.containsKey(v)) throw new RuntimeException("Такая вершина уже есть"); |
|  |  |
|  | adjacenyList.put(v, new Vertex(v)); |
|  | kolV++; |
|  | return true; |
|  | } |
|  | @Override |
|  | public boolean addE(Edge e) { |
|  | if ( !adjacenyList.containsKey(e.v1) ) throw new RuntimeException("Вершина "+e.v1+" не существует"); |
|  | if ( !adjacenyList.containsKey(e.v2) ) throw new RuntimeException("Вершина "+e.v2+" не существует"); |
|  | if ( adjacenyList.get(e.v1).way.containsKey(e.v2)) throw new RuntimeException("Данное ребро уже существует"); |
|  |  |
|  | adjacenyList.get(e.v1).way.put(e.v2, e.weight); |
|  | adjacenyList.get(e.v2).way.put(e.v1, e.weight); |
|  |  |
|  | kolE++; |
|  | return true; |
|  | } |
|  |  |
|  | @Override |
|  | public boolean removeV(Vertex v) { |
|  | if (!adjacenyList.containsKey(v.v)) throw new RuntimeException("Такой вершины нет"); |
|  |  |
|  | adjacenyList.remove(v.v); |
|  | kolV--; |
|  | return true; |
|  | } |
|  | @Override |
|  | public boolean removeE(Edge e) { |
|  |  |
|  | if ( !adjacenyList.containsKey(e.v1) ) throw new RuntimeException("Вершина "+e.v1+" не существует"); |
|  | if ( !adjacenyList.containsKey(e.v2) ) throw new RuntimeException("Вершина "+e.v2+" не существует"); |
|  | if ( !adjacenyList.get(e.v1).way.containsKey(e.v2)) throw new RuntimeException("Данное ребро не существует"); |
|  |  |
|  | adjacenyList.get(e.v1).way.remove(e.v2); |
|  | adjacenyList.get(e.v2).way.remove(e.v1); |
|  |  |
|  | kolE--; |
|  | return true; |
|  | } |
|  |  |
|  | @Override |
|  | public Vertex checkV(int v) { |
|  | return adjacenyList.get(v); |
|  | } |
|  |  |
|  | @Override |
|  | public Edge checkE(int v1, int v2) { |
|  | if (checkV(v1)!=null && checkV(v2)!=null) { |
|  | Integer i = adjacenyList.get(v1).way.get(v2); |
|  |  |
|  | return i==null ? null : new Edge(v1,v2,i.intValue()); |
|  | } |
|  | return null; |
|  | } |
|  |  |
|  | @Override |
|  | public int kolEinV(int v) { |
|  | if (!adjacenyList.containsKey(v)) return -1; |
|  | return adjacenyList.get(v).way.size(); |
|  | } |
|  |  |
|  | @Override |
|  | public void clear() { |
|  | adjacenyList = new HashMap<Integer, Vertex>(); |
|  | kolE = 0; |
|  | kolV = 0; |
|  | } |
|  | } |
|  | @Override |
|  | public ArrayList<Integer> getVertexes() { |
|  | ArrayList<Integer> ret = new ArrayList<Integer>(); |
|  |  |
|  | for(Map.Entry<Integer, Vertex> v: adjacenyList.entrySet()) { |
|  | ret.add(v.getKey()); |
|  | } |
|  | return ret; |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Graph.java**

|  |
| --- |
| package Graph; |
|  |  |
|  | import java.util.ArrayList; |
|  | import java.util.HashMap; |
|  |  |
|  | public abstract class Graph { |
|  |  |
|  | int kolV = 0; |
|  | int kolE = 0; |
|  |  |
|  | abstract public boolean addV(int v); |
|  |  |
|  | abstract public boolean addE(Edge e); |
|  |  |
|  | abstract public boolean removeV(Vertex v); |
|  |  |
|  | abstract public boolean removeE(Edge e); |
|  |  |
|  | abstract public Vertex checkV(int v); |
|  |  |
|  | abstract public Edge checkE(int v1, int v2); |
|  |  |
|  | abstract public int kolEinV(int v); |
|  |  |
|  | public int getKolE() { |
|  | return kolE; |
|  | } |
|  | public int getKolV() { |
|  | return kolV; |
|  | } |
|  | abstract public void clear(); |
|  |  |
|  | abstract public ArrayList<Integer> getVertexes(); |
|  |  |
|  |  |
|  | public static class Edge { |
|  | public Edge(int v1, int v2, int weight) { |
|  | this.v1 = v1; |
|  | this.v2 = v2; |
|  | this.weight = weight; |
|  | } |
|  |  |
|  | public int v1; |
|  | public int v2; |
|  | public int weight; |
|  | } |
|  | public static class Vertex { |
|  | public int v; |
|  | public HashMap<Integer,Integer> way = new HashMap<Integer,Integer>(); |
|  |  |
|  | public Vertex(int v) { |
|  | this.v = v; |
|  | } |
|  | } |
|  |  |
|  | } |
|  |  |
|  |  |
|  |  |

**Par\_S.java**

|  |
| --- |
| package Window; |
|  |  |
|  | import java.awt.\*; |
|  |  |
|  | public class PAR\_S { |
|  |  |
|  | final static Dimension WINDOW\_SIZE = new Dimension(941, 550); |
|  | final static Dimension CREATE\_GRAPH\_PANEL\_SIZE = new Dimension(300,200); |
|  | final static Dimension SIZE\_OF\_INPUT\_FIELD = new Dimension(25,19); |
|  | final static Dimension SIZE\_OF\_GRAPH\_FIELD = new Dimension(600,500); |
|  |  |
|  | final static Color TEXT\_COLOR = Color.BLACK; |
|  | final static Color BUTTENS\_BORDER = new Color(16, 77, 80); |
|  | final static Color BACKGROUND = new Color(225, 219, 180); |
|  | final static Color BASE\_EDGE\_COLOR = new Color(241, 237, 226); |
|  | final static Color EDGE\_LINE\_COLOR = new Color(225, 219, 180); |
|  | final static Color EDGE\_CIRKLE\_LINE\_COLOR = new Color(0, 0, 0); |
|  | final static Color CREATE\_BUTTONS\_BG = new Color(127, 186, 189); |
|  | final static Color RESULT\_EDGE\_COLOR = new Color(203, 138, 82); |
|  | final static Color BASE\_VERTEX\_COLOR = new Color(201, 158, 123); |
|  | final static Color CIRCLE\_BORDERLINE\_COLOR = new Color(0, 0, 0); |
|  | final static Color GRAPH\_FIELD\_BORDER = new Color(225, 219, 180); |
|  | final static Color RESULT\_VERTEX\_COLOR = new Color(163, 98, 80); |
|  | final static Color GRAPH\_FIELD\_BACKGROUND = new Color(50, 119, 123); |
|  |  |
|  | final static int VERTEX\_R = 25; |
|  | final static int TEXT\_SIZE = 14; |
|  | final static int EDJE\_CIRKLE\_R = 28; |
|  | final static int VERTEX\_D = VERTEX\_R\*2; |
|  |  |
|  | final static BasicStroke EDGE\_LINE\_THIKNESS = new BasicStroke(1); |
|  | final static BasicStroke EDGE\_CIRKLE\_LINE\_THKNESS = new BasicStroke(2); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |