**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по учебной практике**

**“Визуализация алгоритма Флойда - Уоршелла в ориентированном графе на языке Java.”**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Тимофеев Д.А. |
| Студентка гр. 6383 |  | Михеева Е.Е. |
| Руководитель |  | Чайка К.В. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент группы 6383 Тимофеев Д.А. | | |
| Студентка группы 6383 Михеева Е.Е. | | |
| Тема практики: визуализация алгоритмов на языке Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: алгоритм Флойда – Уоршелла | | |
| Сроки прохождения практики: 27.06.2018 – 10.07.2018 | | |
| Дата сдачи отчета: \_\_.07.2018 | | |
| Дата защиты отчета: \_\_.07.2018 | | |
|  | | |
| Студент гр.6383 |  | Тимофеев Д.А. |
| Студентка гр.6383 |  | Михеева Е.Е. |
| Руководитель |  | Чайка К.В. |

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 Флойда – Уоршелла с пользовательским интерфейсом.

Summary

The subject of this training practice is the command iterative development of the algorithm visualizer in 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 Floyd –Warshall algorithm in an oriented graph with the user interface.

**Содержание**

[ВBЕДЕНИЕ 5](#_Toc518405341)

[1. ТРЕБОВАНИЯ К ПРОГРАММЕ 5](#_Toc518405342)

[1.1 Исходные Требования к программе 5](#_Toc518405343)

[1.2 Примерная работа программы 6](#_Toc518405344)

[1.2 Изменения к спецификации 9](#_Toc518405345)

[1.2.1 Изменения к спецификации после сдачи прототипа 9](#_Toc518405346)

[1.2.2 Изменения к спецификации после сдачи первой версии 10](#_Toc518405347)

[1.3 Описание входных и выходных данных 10](#_Toc518405348)

[1.4 Примерная работа программы 10](#_Toc518405349)

[2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ 10](#_Toc518405350)

[2.1. План разработки 10](#_Toc518405351)

[2.2. Распределение ролей в бригаде 11](#_Toc518405352)

[3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ 11](#_Toc518405353)

[3.1. Используемые структуры данных 11](#_Toc518405354)

[3.2 Описание классов 12](#_Toc518405355)

[3.3 UML – диаграмма классов 15](#_Toc518405356)

[4. ТЕСТИРОВАНИЕ ПРОГРАММЫ 15](#_Toc518405357)

[4.1 Тестирование интерфейса 15](#_Toc518405358)

[4.2 Тестирование алгоритма 19](#_Toc518405359)

[5. ЗАКЛЮЧЕНИЕ 20](#_Toc518405360)

[ПРИЛОЖЕНИЕ А 21](#_Toc518405361)

ВBЕДЕНИЕ

Алгоритм Флойда — Уоршелла — алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного [графа](http://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) без циклов с отрицательными весами с использованием метода динамического программирования. В нашей работе будет реализован визуализатор данного алгоритма, с использованием графического интерфейса.

Этот алгоритм был одновременно опубликован в статьях Роберта Флойда ([Robert Floyd](http://en.wikipedia.org/wiki/Robert_Floyd)) и Стивена Уоршелла ([Stephen Warshall](http://en.wikipedia.org/wiki/Stephen_Warshall)) в 1962 г., хотя в 1959 г. Бернард Рой ([Bernard Roy](http://en.wikipedia.org/wiki/Bernard_Roy)) опубликовал практически такой же алгоритм, но это осталось незамеченным.

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 3 поля:

1. Поле графического представления графа и визуализации алгоритма. Граф строится по данным, введенными пользователем с помощью графического интерфейса. Для удобства пользователя предусмотрено перемещение вершин графа с помощью мыши.
2. Поле с элементами управления:

1) Следующий шаг

2) Предыдущий шаг

1. Диалог создания графа:
2. Ввод количества вершин
3. добавить ребро
4. старт

1.2 Примерная работа программы

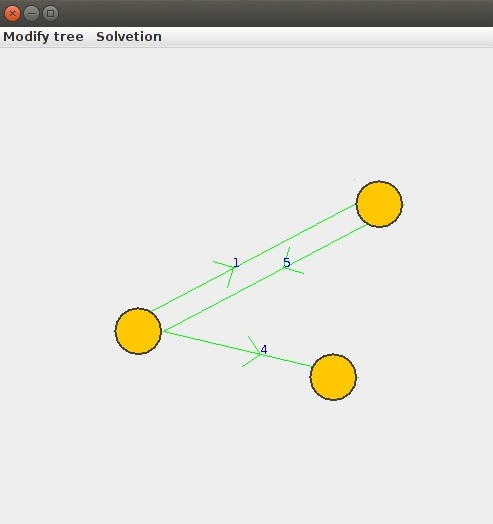


Рисунок 1. Интерфейс поля визуализации алгоритма.

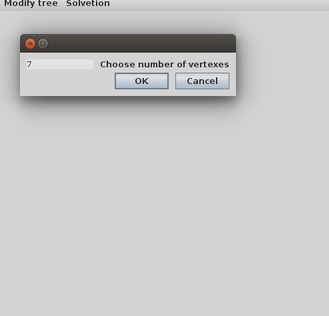


Рисунок 2. Диалог задания количества вершин в графе.



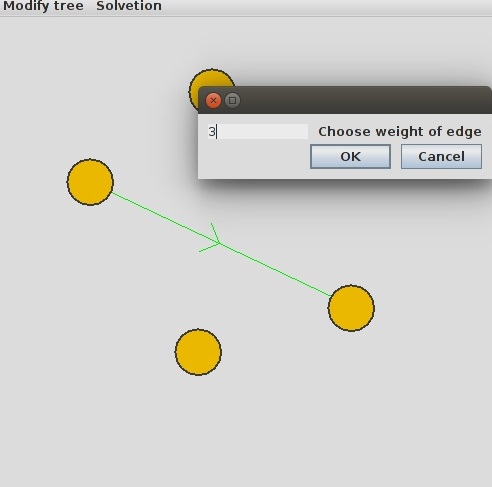
Рисунок 3. Задание ребер графа

Рисунок 4. Диалог задания веса ребра.

1.2 Изменения к спецификации

1.2.1 Изменения к спецификации после сдачи прототипа

Для удобства пользователя было решено добавить матрицу смежности текущего шага и результирующую матрицу смежности. Также в окне начальных данных было решено подсвечивать рассматриваемые ребра графа на текущем шаге.

1.2.2 Изменения к спецификации после сдачи первой версии

После сдачи первой версии было решено добавить возможность динамически добавлять вершины к уже построенному графу.

1.3 Описание входных и выходных данных

Входные данные:

Граф, введенный пользователем с помощью графического интерфейса.

Выходные данные:

Графическое представление графа по шагам. Вывод кратчайшего пути между заданной парой вершин.

1.4 Примерная работа программы

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

29.06.2018 (пятница) – разработка спецификации

30.06.2018 (суббота) – согласование спецификации с руководителем

(31.06 -03.07). 2018 – разработка интерфейса

04.07.2018 (среда) - предоставление планируемого интерфейса (прототип), с отсутствующей функциональностью.

05.07.2018 (четверг)– реализация графического представления графа, на основе входных данных и осуществление работы алгоритма с выводом результата в соответствующее окно (частичная функциональность)

06.07.2018 (среда) - сдача 1-ой версии с добавлением частичной функциональности к пользовательскому интерфейсу;

07.07.2018 – реализация пошаговой работы алгоритма, тестирование программы. Оформление отчета.

10 июля (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 студента | Обязанность |
| Тимофеев Дмитрий | Реализация отдельных графических компонентов (интерактивные вершины, ребра графа). Реализация алгоритма Флойда – Уоршелла с учетом специфики пошаговой визуализации. |
| Михеева Екатерина | Реализация диалога ввода графа пользователем. Пошаговая визуализация алгоритма. Тестирование. |

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Используемые структуры данных

Граф для реализации алгоритма представляется в виде матрицы смежности. Для визуализации алгоритма граф представляется с помощью класса Graph, содержащего матрицу смежности и атрибуты ребер и вершин (цвет и способ изображения). Для реализации графического интерфейса разрабатываемого приложения используется библиотека Swing. Граф визуализируется без использования прочих библиотек.

3.2 Описание классов

Программа состоит из классов: Main, Graph, DraggedPanel, MatrixPanel, FloydWarshallAlgorithm.

***Класс Main***

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

В основе работы класса лежат методы класса JFrame. Данные методы реализуют весь графический интерфейс.

Кнопки, используемые в программе представлены в таблице 2:

Таблица 2

|  |  |
| --- | --- |
| Название кнопки | Функциональность |
| Add graph | Создание графа |
| Add vertex | Добавление вершин |
| Add edges | Добавление ребер |
| Move vertexes | Позволяет двигать вершины |
| Solve | Результат |
| Next | Следующий шаг |
| Back | Предыдущий шаг |

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

***Класс Graph***

Класс Graph содержит в себе поле для хранения промежуточных данных при реализации алгоритма.

Методы класса:

Таблица 3.

|  |  |
| --- | --- |
| Метод | Предназначение |
| public Graph getLarge() | Возвращает граф с дополнительной вершиной |
| public void setMatr(int[][] matr) | Устанавливает матрицу смежности |
| public int getV() | Возвращает количество вершин |
| public void addChangeListeners(Consumer<Void>) | Добавляет слушателей изменения графа |
| public Cell getMatrixCell(int i, int j) | Возвращает пару вес/промежуточная вершина для ребра из i в j. |
| public EdgeState getEdgeState(int i, int j) | Возвращает атрибуты ребра |
| public VertexState getVertexState(int i) | Возвращает атрибуты вершины |
| public boolean setStepOfAlgorithm(int step) | Устанавливает атрибуты графа на соответствующий шаг алгоритма.  Возвращает false если достигнут последний шаг. |
| public setEdgeWeight(int I,int ) | Устанавливает вес ребра |

***Класс FloydWarshallAlgorithm***

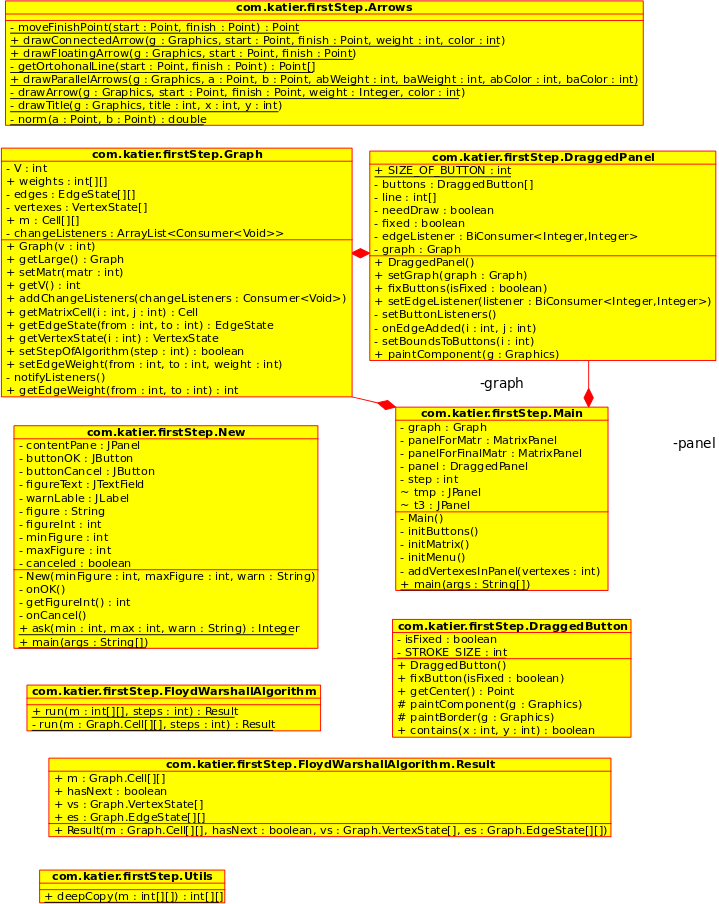
Состоит из одного метода Result run(int[][] weights, int step) возвращающего состояние графа на заданном шаге.

***Классы MatrixPanel и DraggedPanel***

Используются для динамического отображения матрицы смежности и графа соответственно. Оба класса содержат метод void setGraph(Graph graph) в котором они подписываются на события изменения графа.

Таким образом изменения графа автоматически влекут обновления на экране.

3.3 UML – диаграмма классов



4. ТЕСТИРОВАНИЕ ПРОГРАММЫ

4.1 Тестирование интерфейса

1) Запускаем jar файл

2) Добавляем 5 вершины, нажатием на кнопку “New Tree”. На экране по порядку нажатия появляются 3 вершин по окружности.

3) Добавляем ребра с помощью графического интерфейса;

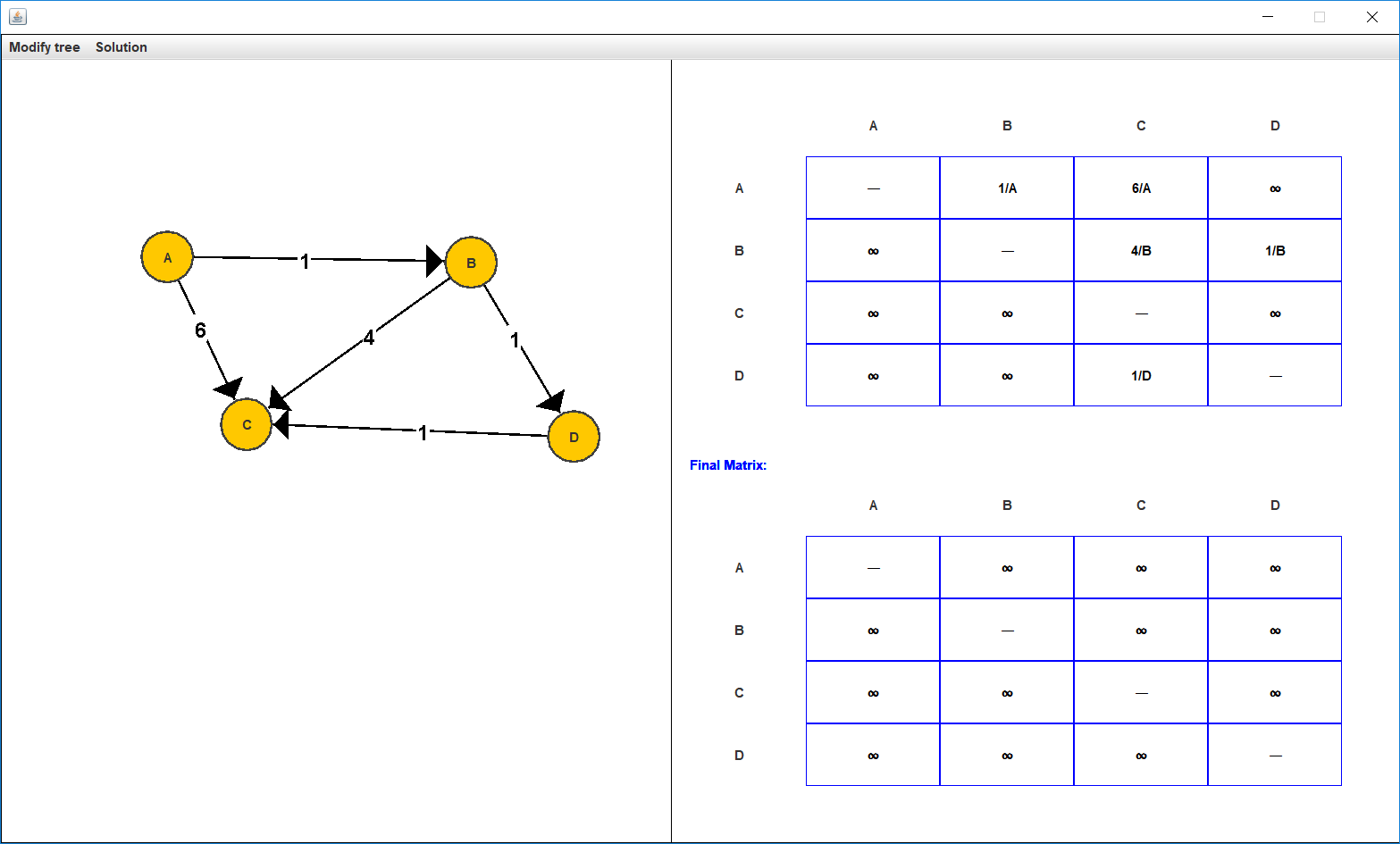


Рисунок 11.

4) Нажимаем кнопку “Solve” который запускает алгоритм и отображает изменения на результирующей матрице смежности.

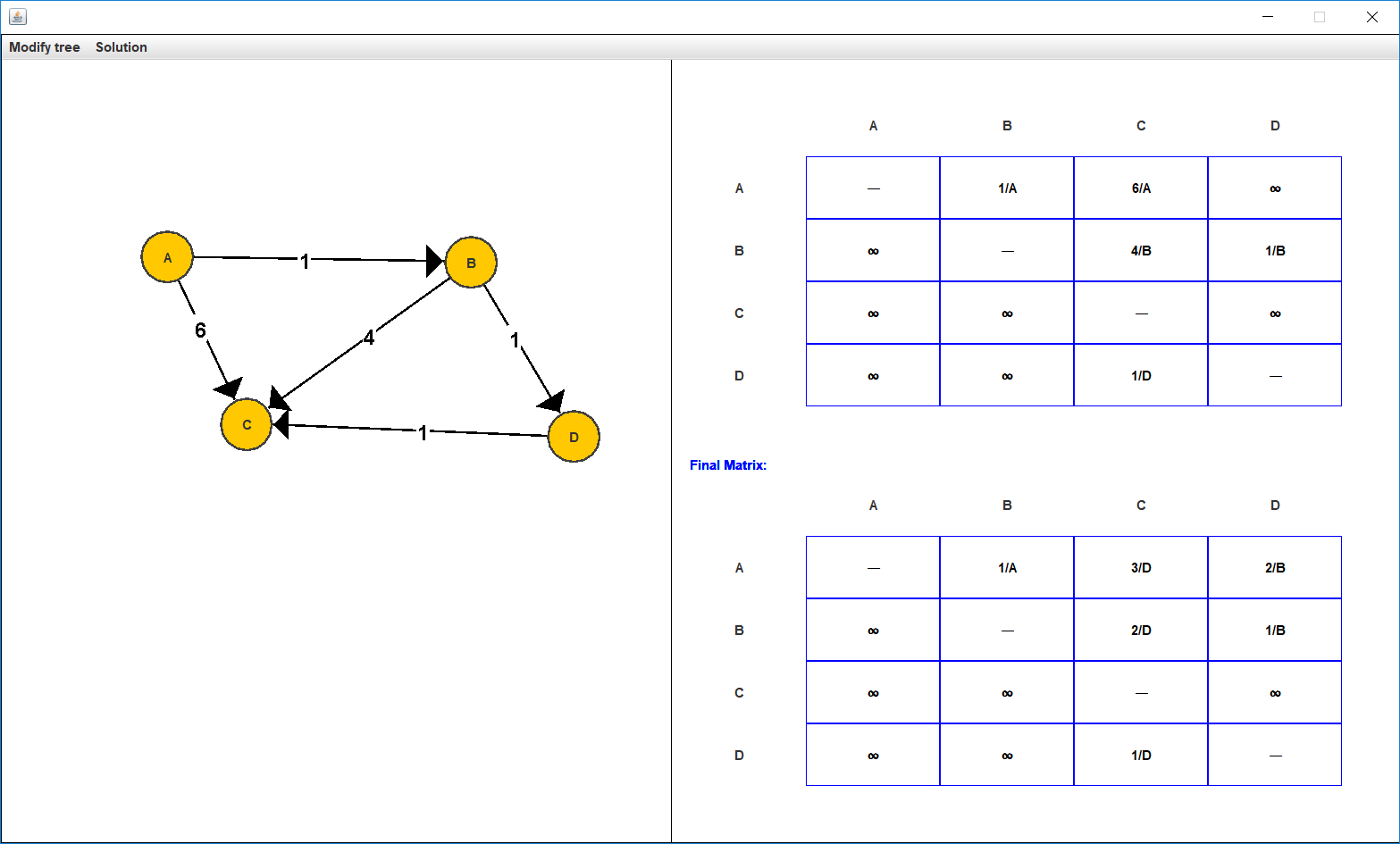


Рисунок 12.

5) Нажатием на кнопку “Next” и на матрице смежности текущего шага подсвечивается текущий шаг, а на графе подсвечиваются активные ребра текущего шага.

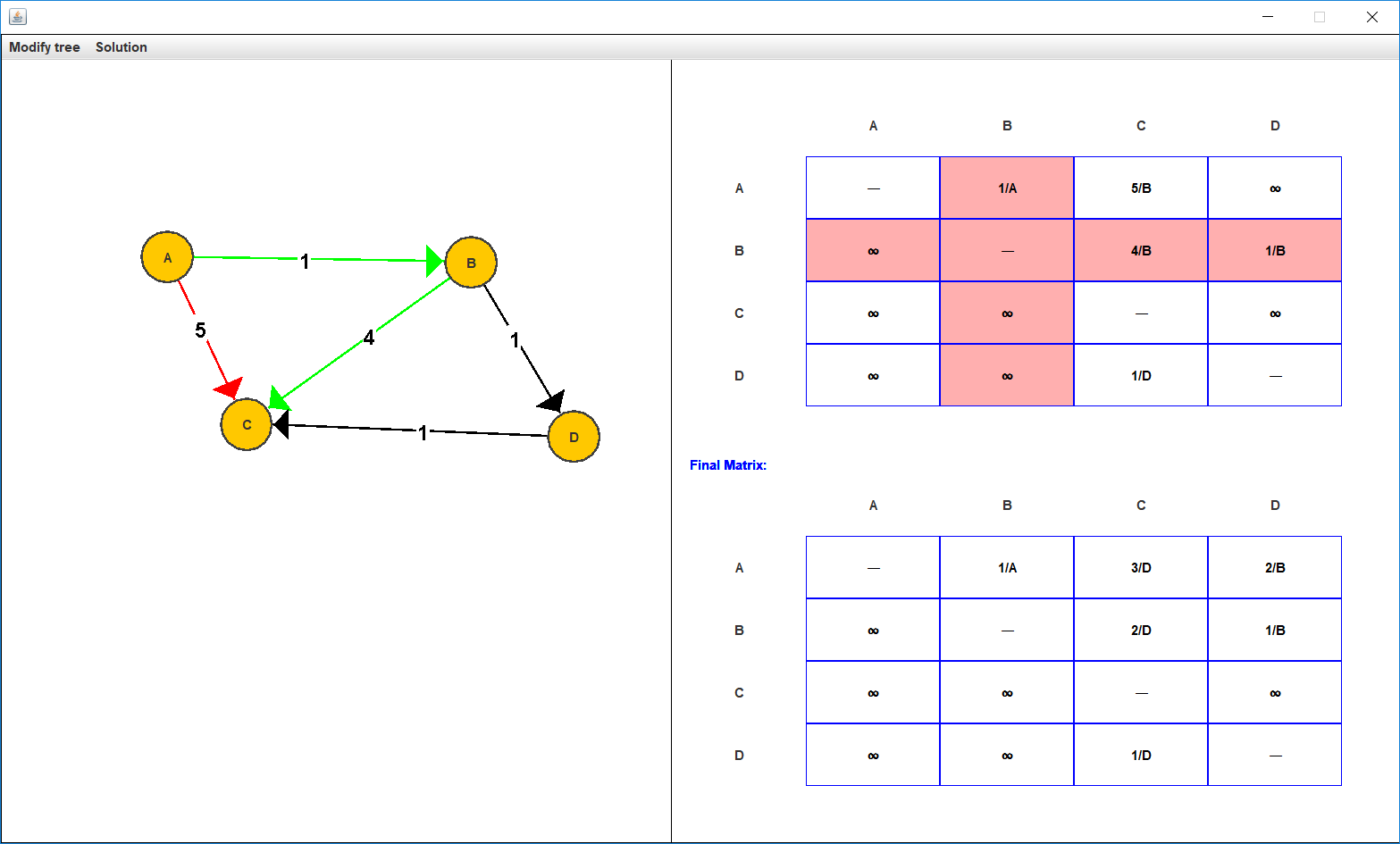


Рисунок 13.

7) Нажали на кнопку “Next”

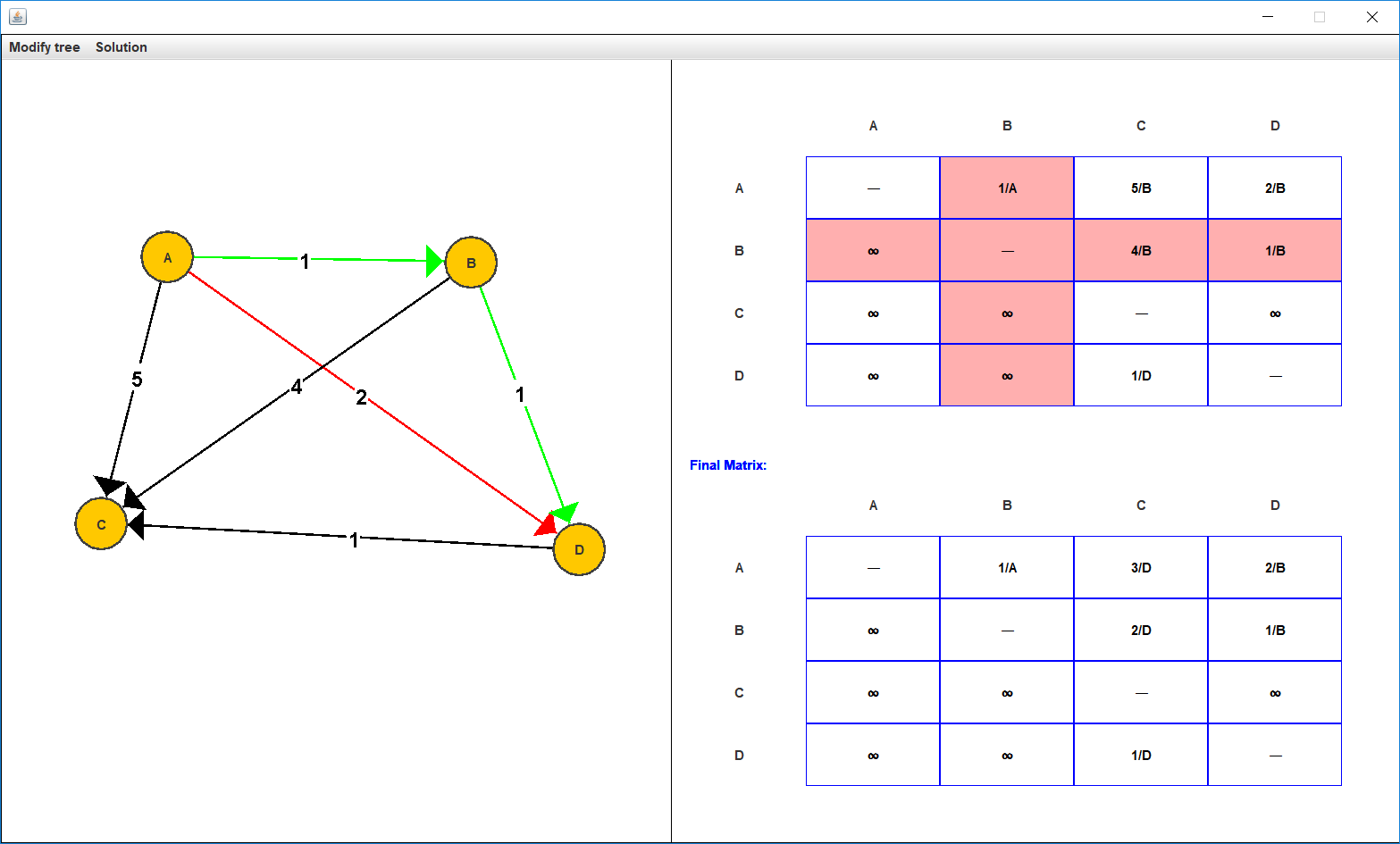


Рисунок 14.

8) Нажали на кнопку “Next”

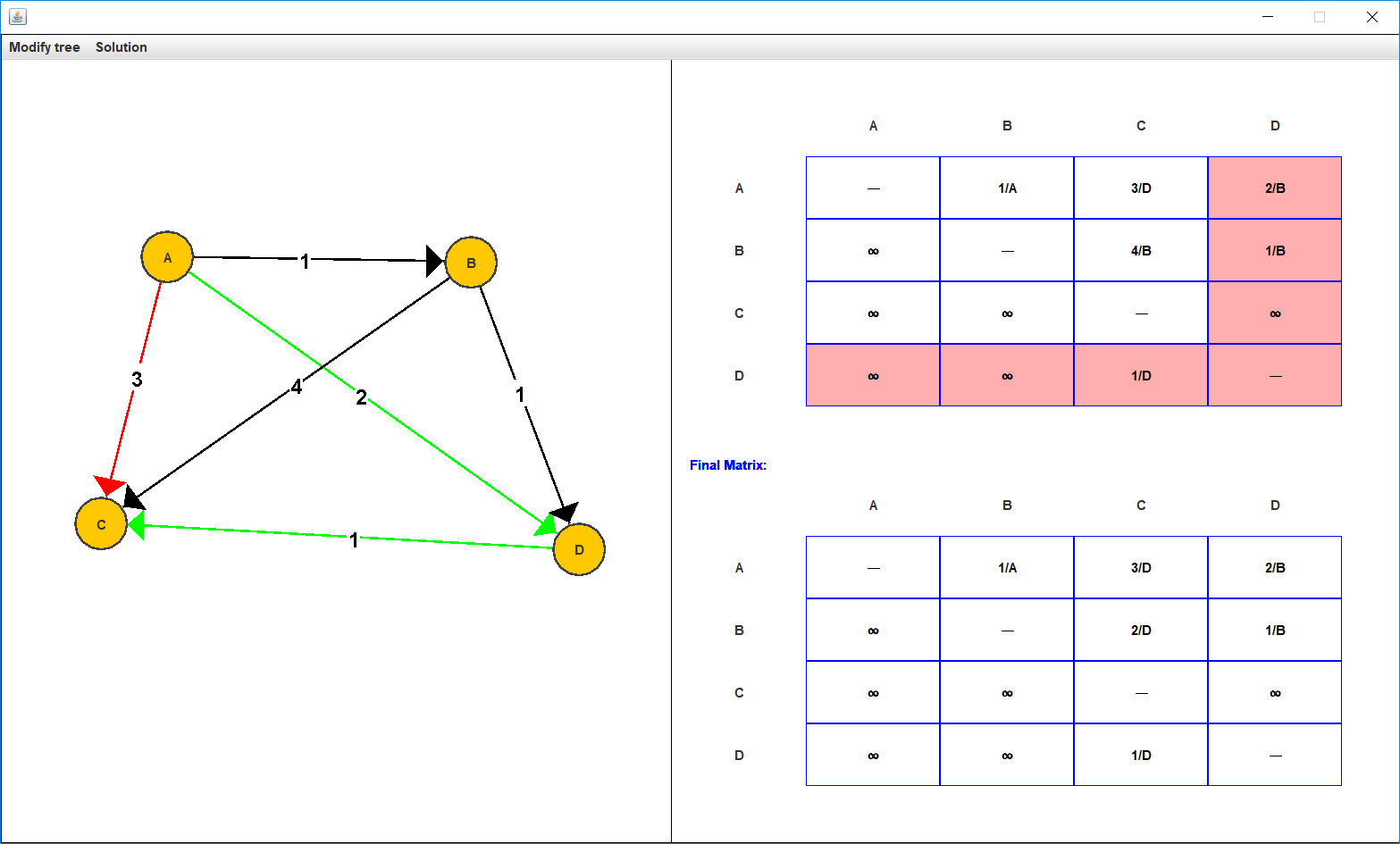


Рисунок 15.

9) Нажали на кнопку “Next”

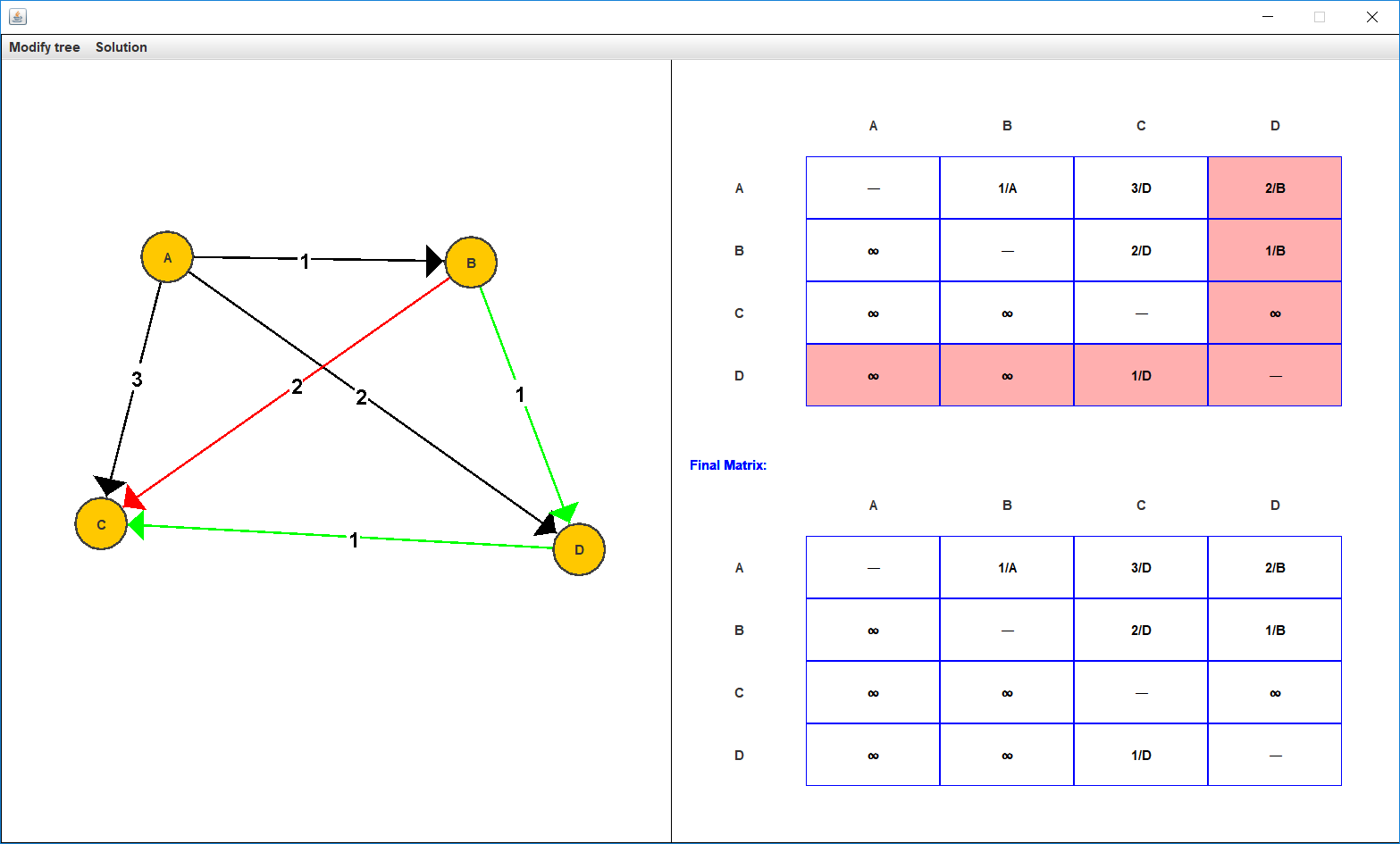


Рисунок 16.

4.2 Тестирование алгоритма

Таблица 5.

|  |  |  |
| --- | --- | --- |
| № | Начальные данные | Результат (порядок использования) |
| 1 | Начальные данные:  x 1 6 ∞  ∞ x 4 1 ∞ ∞ x ∞ ∞ ∞ 1 x | Результат:  x 1 3 2  ∞ x 2 1 ∞ ∞ x ∞ ∞ ∞ 1 x |
| 2 | Начальные данные:  x ∞ ∞ 3  1 x ∞ ∞ ∞ 5 x ∞ ∞ ∞ 4 x | Результат:  x 12 7 3  1 x 8 4 6 5 x 9 10 9 4 x |
| 3 | Начальные данные:  x ∞ ∞ 2  1 x ∞ ∞ ∞ ∞ x ∞ ∞ ∞ 4 x | Результат:  x ∞ 6 2  1 x 7 3 ∞ ∞ x ∞ ∞ ∞ 4 x |

5. ЗАКЛЮЧЕНИЕ

Таким образом, был реализован алгоритм Флойда-Уоршелла 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

Класс Arrows

**package** com.katier.firstStep;  
  
**import** java.awt.\*;  
**import** java.awt.geom.Rectangle2D;  
  
**import static** java.lang.Math.*pow*;  
**import static** java.lang.Math.*sqrt*;  
  
**public class** Arrows {  
  
 **private static** Point moveFinishPoint(Point start, Point finish){  
 **final double** norm=*norm*(start,finish);  
 **final double** p=1- DraggedPanel.***SIZE\_OF\_BUTTON***/2/norm;  
 **return new** Point((**int**)((finish.**x**-start.**x**)\*p+start.**x**),(**int**)((finish.**y**-start.**y**)\*p+start.**y**));  
 }  
  
 **public static void** drawConnectedArrow(Graphics g, Point start, Point finish, **int** weight, **int** color){  
 *drawArrow*(g,start,*moveFinishPoint*(start,finish),weight, color);  
 }  
  
 **public static void** drawFloatingArrow(Graphics g, Point start, Point finish){  
 *drawArrow*(g,start,finish,**null**,Color.***BLACK***.getRGB());  
 }  
  
 **private static** Point[] getOrtohonalLine(Point start, Point finish){  
 **final double** size=10;  
 **final double** norm=*norm*(start,finish);  
 Point st=**new** Point(start.**x**-finish.**x**,start.**y**-finish.**y**);  
 Point l=**new** Point((**int**)(st.**y**/norm\*size),(**int**)(-st.**x**/norm\*size));  
 Point r=**new** Point(-l.**x**,-l.**y**);  
 Point outL=**new** Point(l.**x**+finish.**x**,l.**y**+finish.**y**);  
 Point outR=**new** Point(r.**x**+finish.**x**,r.**y**+finish.**y**);  
 **return new** Point[]{outL,outR};  
 }  
  
 **public static void** drawParallelArrows(Graphics g, Point a, Point b, **int** abWeight, **int** baWeight, **int** abColor, **int** baColor){  
 Point[] ob=*getOrtohonalLine*(a,b);  
 Point[] oa=*getOrtohonalLine*(b,a);  
 *drawConnectedArrow*(g,oa[0],ob[1],abWeight,abColor);  
 *drawConnectedArrow*(g,ob[0],oa[1],baWeight,baColor);  
 }  
  
 **private static void** drawArrow(Graphics g, Point start, Point finish, Integer weight, **int** color){  
 **if**(start.equals(finish))**return**;  
 ((Graphics2D)g).setStroke(**new** BasicStroke(2));  
 g.setFont(**new** Font(Font.***SANS\_SERIF***,Font.***BOLD***,18));  
 g.setColor(**new** Color(color));  
 g.drawLine(start.**x**,start.**y**,finish.**x**,finish.**y**);  
 **final double** size= 30;  
 **final double** norm=*norm*(start,finish);  
 Point startFictive=**new** Point((**int**)((start.**x**-finish.**x**)/norm\*size)+finish.**x**,((**int**)((start.**y**-finish.**y**)/norm\*size))+finish.**y**);  
 Point center=**new** Point((startFictive.**x**+finish.**x**)/2,(startFictive.**y**+finish.**y**)/2);  
 Point finishTranslated=**new** Point(finish.**x**-center.**x**,finish.**y**-center.**y**);  
 Point rp=**new** Point(finishTranslated.**y**,-finishTranslated.**x**);  
 Point lp=**new** Point(-rp.**x**,-rp.**y**);  
 Point lpTranslated=**new** Point(lp.**x**+center.**x**,lp.**y**+center.**y**);  
 Point rpTranslated=**new** Point(rp.**x**+center.**x**,rp.**y**+center.**y**);  
 Polygon polygon=**new** Polygon(  
 **new int**[]{lpTranslated.**x**,rpTranslated.**x**,finish.**x**},  
 **new int**[]{lpTranslated.**y**,rpTranslated.**y**,finish.**y**},  
 3  
 );  
 g.fillPolygon(polygon);  
 **if**(weight==**null**)**return**;  
 *drawTitle*(g,weight,(start.**x**+finish.**x**)/2,(start.**y**+finish.**y**)/2);  
 }  
  
 **private static void** drawTitle(Graphics g,**int** title, **int** x,**int** y){  
 g.setColor(Color.***WHITE***);  
 FontMetrics fm=g.getFontMetrics();  
 Rectangle2D r=fm.getStringBounds(Integer.*toString*(title),g);  
 x-=(**int**)r.getWidth()/2;  
 y+=fm.getAscent()/2;  
 g.fillRect(x-1,y-fm.getAscent()-1,(**int**)r.getWidth()+2,(**int**)r.getHeight()+2);  
 g.setColor(Color.***BLACK***);  
 g.drawString(Integer.*toString*(title),x,y);  
 }  
  
 **private static double** norm(Point a, Point b){  
 **return** *sqrt*(*pow*(a.**x**-b.**x**,2)+*pow*(a.**y**-b.**y**,2));  
 }  
}

Класс DraggedButton

**package** com.katier.firstStep;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.MouseAdapter;  
**import** java.awt.event.MouseEvent;  
**import** java.awt.geom.Ellipse2D;  
  
**public class** DraggedButton **extends** JButton {  
 **private boolean isFixed**=**false**;  
 **public** DraggedButton(){  
 MouseListener mouseListener=**new** MouseListener();  
 addMouseMotionListener(mouseListener);  
 addMouseListener(mouseListener);  
 setFocusPainted(**false**);  
 setBackground(Color.***orange***);  
 setContentAreaFilled(**false**);  
 }  
 **public void** fixButton(**boolean** isFixed){  
 **this**.**isFixed**=isFixed;  
 }  
 **public** Point getCenter(){  
 **return new** Point(  
 getBounds().**x**+getBounds().**width**/2,  
 getBounds().**y**+getBounds().**height**/2  
 );  
 }  
  
 **private class** MouseListener **extends** MouseAdapter {  
 **private** Point **offset**=**new** Point();  
 @Override  
 **public void** mousePressed(MouseEvent e) {  
 **if**(**isFixed**)**return**;  
 **offset**.**x**=e.getX();  
 **offset**.**y**=e.getY();  
 }  
  
 @Override  
 **public void** mouseDragged(MouseEvent e) {  
 **if**(**isFixed**)**return**;  
 **int** x=getBounds().**x**+e.getX()-**offset**.**x**;  
 **int** y=getBounds().**y**+e.getY()-**offset**.**y**;  
 **if**(x<0)x=0;  
 **if**(y<0)y=0;  
 **if**(x>getParent().getWidth()-getWidth())x=getParent().getWidth()-getWidth();  
 **if**(y>getParent().getHeight()-getHeight())y=getParent().getHeight()-getHeight();  
 setBounds(x, y, getWidth(), getHeight());  
 }  
 }  
  
 **private static final int *STROKE\_SIZE***=2;  
 **protected void** paintComponent(Graphics g) {  
 **if** (getModel().isArmed()) g.setColor(Color.***gray***);  
 **else** g.setColor(getBackground());  
 ((Graphics2D)g).setStroke(**new** BasicStroke(***STROKE\_SIZE***));  
 g.fillOval(***STROKE\_SIZE***, ***STROKE\_SIZE***, getSize().**width** - 2\****STROKE\_SIZE***, getSize().**height** - 2\****STROKE\_SIZE***);  
 **super**.paintComponent(g);  
 }  
  
 **protected void** paintBorder(Graphics g) {  
 g.setColor(Color.***darkGray***);  
 g.drawOval(***STROKE\_SIZE***, ***STROKE\_SIZE***, getSize().**width** - 2\****STROKE\_SIZE***, getSize().**height** - 2\****STROKE\_SIZE***);  
 }  
  
 **public boolean** contains(**int** x, **int** y) {  
 **return new** Ellipse2D.Float(0, 0, getWidth(), getHeight()).contains(x, y);  
 }  
}

Класс DraggedPanel

**package** com.katier.firstStep;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.util.function.BiConsumer;  
**import static** java.lang.Math.***PI***;  
  
**public class** DraggedPanel **extends** JPanel {  
 **public static final int *SIZE\_OF\_BUTTON***=50;  
 **private** DraggedButton[] **buttons**;  
 **private final int**[] **line**={0,0,0,0};  
 **private boolean needDraw**=**false**;  
 **private boolean fixed**=**false**;  
 **private** BiConsumer<Integer,Integer> **edgeListener**=**null**;  
 **private** Graph **graph**=**new** Graph(0);  
  
 **public** DraggedPanel(){  
 setBackground(Color.***WHITE***);  
 setLayout(**null**);  
 }  
  
 **public void** setGraph(Graph graph){  
 removeAll();  
 **fixed**=**false**;  
 **needDraw**=**false**;  
 **this**.**graph**=graph;  
 graph.addChangeListeners((v)->updateUI());  
  
 ComponentListener buttonMoveListener=**new** ComponentAdapter(){  
 @Override  
 **public void** componentMoved(ComponentEvent e) {  
 updateUI();  
 }  
 };  
  
 **buttons**=**new** DraggedButton[graph.getV()];  
 **for**(**int** i=0;i<graph.getV();i++){  
 **buttons**[i]=**new** DraggedButton();  
 **buttons**[i].addComponentListener(buttonMoveListener);  
 setBoundsToButtons(i);  
 **char**[] c={(**char**) (i+**'A'**)};  
 **buttons**[i].setText(**new** String(c));  
 add(**buttons**[i]);  
 }  
 setButtonListeners();  
 }  
  
 **public void** fixButtons(**boolean** isFixed){  
 **this**.**fixed**=isFixed;  
 **if**(**buttons**==**null**)**return**;  
 **for**(DraggedButton db:**buttons**){  
 db.fixButton(isFixed);  
 }  
 }  
  
 **public void** setEdgeListener(BiConsumer<Integer,Integer> listener){  
 **edgeListener**=listener;  
 }  
  
 **private void** setButtonListeners(){  
 **final int**[] dowend = {-1};  
 **final int**[] entered = {-1};  
 **for**(**int** i=0;i<**graph**.getV();i++){  
 **final int** ii=i;  
 **buttons**[i].addMouseListener(**new** MouseAdapter() {  
 @Override  
 **public void** mousePressed(MouseEvent mouseEvent) {  
 **if**(!**fixed**)**return**;  
 dowend[0] =ii;  
 **needDraw**=**true**;  
 }  
  
 @Override  
 **public void** mouseReleased(MouseEvent mouseEvent) {  
 **if**(!**fixed**)**return**;  
 **if**(entered[0] !=-1&& dowend[0] != entered[0])onEdgeAdded(dowend[0],entered[0]);  
 dowend[0] =-1;  
 **needDraw**=**false**;  
 updateUI();  
 }  
  
 @Override  
 **public void** mouseEntered(MouseEvent mouseEvent) {  
 **if**(!**fixed**)**return**;  
 entered[0] =ii;  
 }  
  
 @Override  
 **public void** mouseExited(MouseEvent mouseEvent) {  
 **if**(!**fixed**)**return**;  
 entered[0] =-1;  
 }  
 });  
  
 **buttons**[i].addMouseMotionListener(**new** MouseMotionListener() {  
 @Override  
 **public void** mouseDragged(MouseEvent mouseEvent) {  
 **if**(!**fixed**)**return**;  
 **line**[0]=**buttons**[ii].getCenter().**x**;  
 **line**[1]=**buttons**[ii].getCenter().**y**;  
 **line**[2]=**buttons**[ii].getX()+mouseEvent.getX();  
 **line**[3]=**buttons**[ii].getY()+mouseEvent.getY();  
 updateUI();  
 }  
  
 @Override  
 **public void** mouseMoved(MouseEvent mouseEvent) { }  
 });  
 }  
 }  
  
 **private void** onEdgeAdded(**int** i, **int** j){  
 **if**(**edgeListener**!=**null**)**edgeListener**.accept(i,j);  
 }  
  
 **private void** setBoundsToButtons(**int** i){  
 DraggedButton b = **buttons**[i];  
 Dimension size=getSize();  
 Point center=**new** Point(size.**width**/2-***SIZE\_OF\_BUTTON***/2,size.**height**/2-***SIZE\_OF\_BUTTON***/2);  
 **final int** n=**buttons**.**length**;  
 **final int** r=8\*(center.**x**<center.**y**?center.**x**:center.**y**)/10;  
  
 b.setBounds(  
 -(**int**)(Math.*cos*(***PI***/2-2\****PI***\*i/n)\*r)+center.**x**,  
 -(**int**)(Math.*sin*(***PI***/2-2\****PI***\*i/n)\*r)+center.**y**,  
 ***SIZE\_OF\_BUTTON***, ***SIZE\_OF\_BUTTON*** );  
 }  
  
 @Override  
 **public void** paintComponent(Graphics g) {  
 **super**.paintComponent(g);  
  
 **for**(**int** i=0;i<**graph**.getV();i++){  
 **for**(**int** j=i;j<**graph**.getV();j++){  
 Point a=**buttons**[i].getCenter();  
 Point b=**buttons**[j].getCenter();  
 **int** wab=**graph**.getEdgeWeight(i,j);  
 **int** wba=**graph**.getEdgeWeight(j,i);  
 **if**(**graph**.getMatrixCell(j,i).**weight**!=0) wba = **graph**.getMatrixCell(j,i).**weight**;  
 **if**(**graph**.getMatrixCell(i,j).**weight**!=0) wab = **graph**.getMatrixCell(i,j).**weight**;  
 **int** abColor=**graph**.getEdgeState(i,j).getColor();  
 **int** baColor=**graph**.getEdgeState(j,i).getColor();  
 *//int abColor=Color.BLACK.getRGB();  
 //int baColor=Color.BLACK.getRGB();* **if**(wab != 0 && wba != 0)  
 Arrows.*drawParallelArrows*(g,a,b,wab,wba,abColor,baColor);  
  
 **else if**(wab != 0 ) Arrows.*drawConnectedArrow*(g,a,b,wab,abColor);  
 **else if**(wba != 0) Arrows.*drawConnectedArrow*(g,b,a,wba,baColor);  
 }  
 }  
  
 **if**(**needDraw**) Arrows.*drawFloatingArrow*(g,**new** Point(**line**[0],**line**[1]),**new** Point(**line**[2],**line**[3]));  
 }  
}

Класс FloydWarshallAlrotihtm

**package** com.katier.firstStep;  
  
**public class** FloydWarshallAlgorithm{  
 **public static class** Result{  
 **public final** Graph.Cell[][] **m**;  
 **public final boolean hasNext**;  
 **public final** Graph.VertexState[] **vs**;  
 **public final** Graph.EdgeState[][] **es**;  
  
 **public** Result(Graph.Cell[][] m, **boolean** hasNext, Graph.VertexState[] vs, Graph.EdgeState[][] es) {  
 **this**.**m** = m;  
 **this**.**hasNext** = hasNext;  
 **this**.**vs** = vs;  
 **this**.**es** = es;  
 }  
 }  
 **public static** Result run(**int**[][] m, **int** steps){  
 Graph.Cell[][] mat=**new** Graph.Cell[m.**length**][m.**length**];  
 **for** (**int** i = 0; i < m.**length**; i++) {  
 **for** (**int** j = 0; j < m.**length**; j++) {  
 mat[i][j]=**new** Graph.Cell(m[i][j],i);  
 }  
 }  
 **return** *run*(mat,steps);  
 }  
  
 **private static** Result run(Graph.Cell[][] m, **int** steps){  
 Graph.VertexState[] vs=**new** Graph.VertexState[m.**length**];  
 Graph.EdgeState[][] es=**new** Graph.EdgeState[m.**length**][m.**length**];  
 **for**(**int** i=0;i<vs.**length**;i++)vs[i]= Graph.VertexState.***NORMAL***;  
 **for** (**int** i = 0; i < m.**length**; i++)  
 **for** (**int** j = 0; j <m.**length**; j++)  
 es[i][j]= Graph.EdgeState.***NORMAL***;  
  
 main\_loop: **for**(**int** i=0;i<m.**length**;i++){  
 **for**(**int** j=0;j<m.**length**;j++){  
 **if**(i==j)**continue**;  
 **if**(m[j][i].**weight**==0)**continue**;  
 **for**(**int** k=0;k<m.**length**;k++){  
 **if**(k==i)**continue**;  
 **if**(k==j)**continue**;  
 **if**(m[i][k].**weight**==0)**continue**;  
 steps--;  
 **if**(steps<0)**break** main\_loop;  
  
 **if**(steps==0){  
 vs[i]= Graph.VertexState.***ACTIVE***;  
 es[j][k]= Graph.EdgeState.***SELECTED***; *//inner* es[j][i]= Graph.EdgeState.***SELECTED***;  
 es[i][k] = Graph.EdgeState.***SELECTED***;  
 }  
 **if**(m[j][i].**weight**+m[i][k].**weight**<(m[j][k].**weight**==0?Integer.***MAX\_VALUE***:m[j][k].**weight**)){  
 m[j][k]=**new** Graph.Cell(m[j][i].**weight**+m[i][k].**weight**,i);  
 **if**(steps==0) {  
 es[j][k] = Graph.EdgeState.***LOSE***; *//inner* es[j][i] = Graph.EdgeState.***WIN***;  
 es[i][k] = Graph.EdgeState.***WIN***;  
  
 }  
 }  
 **else** {  
 **if**(steps==0){es[j][k]= Graph.EdgeState.***WIN***; *//inner* es[j][i]= Graph.EdgeState.***LOSE***;  
 es[i][k] = Graph.EdgeState.***LOSE***;}  
 }  
  
 }  
 }  
 }  
 **return new** Result(m,steps<0, vs, es);  
 }  
}

Класс Graph

**package** com.katier.firstStep;  
  
**import** com.sun.istack.internal.NotNull;  
  
**import** java.awt.\*;  
**import** java.util.ArrayList;  
**import** java.util.function.Consumer;  
  
**public class** Graph{  
 **private final int V**;  
 **public int**[][] **weights**;  
  
 **private** EdgeState[][] **edges**;  
 **private** VertexState[] **vertexes**;  
 **public** Cell[][] **m**;  
 **private** @NotNull ArrayList< Consumer<Void>> **changeListeners** =**new** ArrayList<>();  
 **public** Graph(**int** v) {  
 **V**=v;  
 **weights**=**new int**[**V**][**V**];  
 FloydWarshallAlgorithm.Result result=FloydWarshallAlgorithm.*run*(**weights**,0);  
 **m**=result.**m**;  
 **edges**=result.**es**;  
 **vertexes**=result.**vs**;  
 }  
  
 **public** Graph getLarge(){  
 Graph g=**new** Graph(**V**+1);  
 **for** (**int** i = 0; i < **V**; i++) {  
 System.*arraycopy*(**weights**[i], 0, g.**weights**[i], 0, **V**);  
 }  
 FloydWarshallAlgorithm.Result res=FloydWarshallAlgorithm.*run*(g.**weights**,0);  
 g.**m**=res.**m**;  
 g.**vertexes**=res.**vs**;  
 g.**edges**=res.**es**;  
 **return** g;  
 }  
  
 **public void** setMatr(**int** matr [][]){  
 **weights** = Utils.*deepCopy*(matr);  
 }  
 **public int** getV(){  
 **return V**;  
 }  
  
 **public void** addChangeListeners(Consumer<Void> changeListeners){  
 **if** (changeListeners != **null**) **this**.**changeListeners**.add(changeListeners);  
 }  
  
 **public** Cell getMatrixCell(**int** i,**int** j){  
 **return m**[i][j];  
 }  
  
 **public** EdgeState getEdgeState(**int** from, **int** to){  
 **return edges**[from][to];  
 }  
  
 **public** VertexState getVertexState(**int** i){  
 **return vertexes**[i];  
 }  
  
 **public boolean** setStepOfAlgorithm(**int** step){  
 FloydWarshallAlgorithm.Result r=FloydWarshallAlgorithm.*run*(**weights**,step);  
 **m**=r.**m**;  
 **edges**=r.**es**;  
 **vertexes**=r.**vs**;  
 notifyListeners();  
 **return** r.**hasNext**;  
 }  
  
 **public void** setEdgeWeight(**int** from, **int** to, **int** weight){  
 **weights**[from][to]=weight;  
 **m**=FloydWarshallAlgorithm.*run*(**weights**,0).**m**;  
 notifyListeners();  
 }  
  
 **private void** notifyListeners(){  
 **for**(Consumer<Void> cl: **changeListeners**) cl.accept(**null**);  
 }  
  
 **public int** getEdgeWeight(**int** from, **int** to){  
 **return weights**[from][to];  
 }  
  
 **public enum** EdgeState{  
 ***NORMAL*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***BLACK***.getRGB();  
 }  
 },***SELECTED*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***PINK***.getRGB();  
 }  
 },***WIN*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***GREEN***.getRGB();  
 }  
 },***LOSE*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***RED***.getRGB();  
 }  
 }  
 ,***TMP*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***GRAY***.getRGB();  
 }  
 };  
 **public abstract int** getColor();  
 }  
  
 **public enum** VertexState{  
 ***NORMAL*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***BLACK***.getRGB();  
 }  
 },***ACTIVE*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***RED***.getRGB();  
 }  
 },***PROCESSED*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***YELLOW***.getRGB();  
 }  
 },***NEIGHBOR*** {  
 @Override  
 **public int** getColor() {  
 **return** Color.***PINK***.getRGB();  
 }  
 };  
 **public abstract int** getColor();  
 }  
 **public static class** Cell{  
 **public final int weight**;  
 **public final int prev**;  
  
 **public** Cell(**int** weight, **int** prev) {  
 **this**.**weight** = weight;  
 **this**.**prev** = prev;  
 }  
 }  
}

Класс Main

**package** com.katier.firstStep;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** Main **extends** JFrame {  
 **private** Graph **graph** = **new** Graph(0);  
 **private final** MatrixPanel **panelForMatr** = **new** MatrixPanel();  
 **private final** MatrixPanel **panelForFinalMatr** = **new** MatrixPanel();  
 **private final** DraggedPanel **panel** = **new** DraggedPanel();  
 **private int step**=0;  
 JPanel **tmp**;  
 JPanel **t3** = **new** JPanel(**new** BorderLayout());  
 **private** Main() {  
 setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
  
 initMenu();  
  
 setLayout(**new** BorderLayout());  
  
 **panel**.setEdgeListener((i, j) -> {  
 Integer weight = New.*ask*(1, 99, **"Choose weight of edge"**);  
 **if** (weight == **null**) **return**;  
 **graph**.setEdgeWeight(i, j, weight);  
  
 Graph g = **new** Graph(**graph**.getV());  
 **panelForFinalMatr**.setGraph(g);  
 g.setStepOfAlgorithm(-1);  
 });  
 **panel**.setPreferredSize(**new** Dimension(600, 700));  
 **panel**.setBorder(BorderFactory.*createMatteBorder*(0,0,0,1,Color.***BLACK***));  
 add(**panel**, BorderLayout.***WEST***);  
  
 initButtons();  
  
 initMatrix();  
  
 setResizable(**false**);  
 pack();  
 setVisible(**true**);  
 }  
  
 **private void** initButtons(){  
 }  
  
 **private void** initMatrix(){  
 JPanel t = **new** JPanel();  
 t.setPreferredSize(**new** Dimension(50,700));  
 t.setBackground(Color.***WHITE***);  
 **tmp** = **new** JPanel();  
 **tmp**.setLayout(**new** BoxLayout(**tmp**,BoxLayout.***Y\_AXIS***));  
 **tmp**.setPreferredSize(**new** Dimension(600, 700));  
 **tmp**.setBackground(Color.***WHITE***);  
 JPanel t1 = **new** JPanel();  
 t1.setPreferredSize(**new** Dimension(600,30));  
 t1.setBackground(Color.***WHITE***);  
 **tmp**.add(t1);  
 **tmp**.add(**panelForMatr**);  
 **t3**.setPreferredSize(**new** Dimension(600,20));  
 **t3**.setBackground(Color.***WHITE***);  
 **tmp**.add(**t3**);  
 **tmp**.add(**panelForFinalMatr**);  
 JPanel t2 = **new** JPanel();  
 t2.setPreferredSize(**new** Dimension(600,50));  
 t2.setBackground(Color.***WHITE***);  
 **tmp**.add(t2);  
 add(t,BorderLayout.***EAST***);  
 add(**tmp**);  
 }  
  
 **private void** initMenu(){  
 JMenuBar jToolBar = **new** JMenuBar();  
 JMenu modifyTree = **new** JMenu(**"Modify tree"**);  
 JMenuItem newTree = **new** JMenuItem(**"New Graph"**);  
 JMenuItem addEdges = **new** JMenuItem(**"Add Edges"**);  
 JMenuItem moveVert = **new** JMenuItem(**"Move Vertexes"**);  
 JMenuItem addVert = **new** JMenuItem(**"Add Vertex"**);  
 newTree.addActionListener(actionEvent -> {  
 Integer k = New.*ask*(2, 12, **"Choose number of vertexes"**);  
 **if** (k == **null**) **return**;  
 **int** vertexes = k;  
 remove(**tmp**);  
 Graph g = **new** Graph(k);  
 **panelForFinalMatr**.setGraph(g);  
 initMatrix();  
 addVertexesInPanel(vertexes);  
 });  
 moveVert.addActionListener(actionEvent -> **panel**.fixButtons(**false**));  
 addEdges.addActionListener(actionEvent -> **panel**.fixButtons(**true**));  
 addVert.addActionListener(actionEvent->{  
 **if**(**graph**.getV()==0)**return**;  
 **if**(**graph**.getV()==12){  
 JOptionPane.*showMessageDialog*(**null**,**"Too many vertexes"**);  
 **return**;  
 }  
 **graph**=**graph**.getLarge();  
 **step**=0;  
 **panelForMatr**.setGraph(**graph**);  
 **panel**.setGraph(**graph**);  
 Graph g = **new** Graph(**graph**.getV());  
 **panelForFinalMatr**.setGraph(g);  
 g.setStepOfAlgorithm(-1);  
 });  
 JMenu solution = **new** JMenu(**"Solution"**);  
 JMenuItem solve = **new** JMenuItem(**"Solve"**);  
 JMenuItem next = **new** JMenuItem(**"Next"**);  
 JMenuItem back = **new** JMenuItem(**"Back"**);  
 solve.addActionListener(actionEvent -> {  
 *//floyd-warshall* Graph g = **new** Graph(**graph**.getV());  
 **panelForFinalMatr**.setGraph(g);  
 **panelForFinalMatr**.setListeners();  
 g.setMatr(**graph**.**weights**);  
 g.setStepOfAlgorithm(g.getV()\*g.getV()\*g.getV()-1);  
  
 });  
 next.addActionListener(actionEvent -> {  
 **if**(!**graph**.setStepOfAlgorithm(**step**)){  
 JOptionPane.*showMessageDialog*(**null**,**"This is already last step"**);  
 **return**;  
 }  
 **step**++;  
 **graph**.setStepOfAlgorithm(**step**);  
 });  
 back.addActionListener(actionEvent -> {  
 **if**(**step**==0){  
 JOptionPane.*showMessageDialog*(**null**,**"This is already 0 step"**);  
 **return**;  
 }  
 **step**--;  
 **graph**.setStepOfAlgorithm(**step**);  
 });  
 solution.add(solve);  
 solution.add(next);  
 solution.add(back);  
 jToolBar.add(modifyTree);  
 jToolBar.add(solution);  
 modifyTree.add(newTree);  
 modifyTree.add(addEdges);  
 modifyTree.add(addVert);  
 modifyTree.add(moveVert);  
 setJMenuBar(jToolBar);  
 }  
  
 **private void** addVertexesInPanel(**int** vertexes) {  
 **graph** = **new** Graph(vertexes);  
 **step**=0;  
 **panel**.setGraph(**graph**);  
 **panelForMatr**.setGraph(**graph**);  
 JLabel label = **new** JLabel(**"Final Matrix:"**);  
 label.setPreferredSize(**new** Dimension(100, 20));  
 label.setForeground(Color.***BLUE***);  
 label.setVerticalAlignment(SwingConstants.***BOTTOM***);  
 label.setHorizontalAlignment(SwingConstants.***CENTER***);  
 **t3**.add(label, BorderLayout.***WEST***);  
  
 }  
 **public static void** main(String[] args) {  
 **new** Main();  
 }  
}

Класс Utils

**package** com.katier.firstStep;  
  
  
**public class** Utils{  
 **public static int**[][] deepCopy(**int**[][] m){  
 **int**[][] r=**new int**[m.**length**][];  
 **for**(**int** i=0;i<m.**length**;i++)r[i]=m[i].clone();  
 **return** r;  
 }  
}

Класс New

**package** com.katier.firstStep;  
  
**import** org.jetbrains.annotations.Nullable;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
  
**public class** New **extends** JDialog {  
 **private** JPanel **contentPane**;  
 **private** JButton **buttonOK**;  
 **private** JButton **buttonCancel**;  
 **private** JTextField **figureText**;  
 **private** JLabel **warnLable**;  
 **private** String **figure**=**""**;  
 **private int figureInt**=0;  
 **private int minFigure**,**maxFigure**;  
  
 **private boolean canceled** =**false**;  
  
 **private** New(**int** minFigure,**int** maxFigure,String warn) {  
 **this**.**minFigure** = minFigure;  
 **this**.**maxFigure** = maxFigure;  
 **warnLable**.setText(warn);  
 setContentPane(**contentPane**);  
 setModal(**true**);  
 getRootPane().setDefaultButton(**buttonOK**);  
  
 **buttonOK**.addActionListener(e -> onOK());  
  
 **buttonCancel**.addActionListener(e -> onCancel());  
  
 setDefaultCloseOperation(***DO\_NOTHING\_ON\_CLOSE***);  
 addWindowListener(**new** WindowAdapter() {  
 **public void** windowClosing(WindowEvent e) {  
 onCancel();  
 }  
 });  
 }  
  
 **private void** onOK() {  
 **figure** = **figureText**.getText();  
 **try** {  
 **figureInt** = getFigureInt();  
 **warnLable**.setText(**"Good!"**);  
 **warnLable**.setForeground(Color.***GREEN***);  
 Timer timer = **new** Timer(500, actionEvent -> dispose());  
 timer.start();  
 }  
 **catch** (NumberFormatException ex){  
 **warnLable**.setText(**"Illegal input!"**);  
 **warnLable**.setForeground(Color.***RED***);  
 }  
 **catch** (TooBigFigureExeption ex){  
 **warnLable**.setText(**"Too big figure!"**);  
 **warnLable**.setForeground(Color.***RED***);  
 }  
 **catch** (TooSmallFigureException ex){  
 **warnLable**.setText(**"Too small figure!"**);  
 **warnLable**.setForeground(Color.***RED***);  
 }  
 }  
  
 **private int** getFigureInt() **throws** TooBigFigureExeption, TooSmallFigureException {  
 **int** f = Integer.*parseInt*(**figure**);  
 **if**(f>**maxFigure**) **throw new** TooBigFigureExeption();  
 **if**(f<**minFigure**) **throw new** TooSmallFigureException();  
 **return** f;  
 }  
  
 **private void** onCancel() {  
 **canceled**=**true**;  
 dispose();  
 }  
  
 @Nullable  
 **public static** Integer ask(**int** min, **int** max, String warn){  
 New dialog=**new** New(min,max,warn);  
 dialog.pack();  
 dialog.setVisible(**true**);  
 **if**(dialog.**canceled**)**return null**;  
 **else return** dialog.**figureInt**;  
 }  
  
 **private static class** TooBigFigureExeption **extends** Exception{ }  
  
 **private static class** TooSmallFigureException **extends** Exception{}  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"user enter: "**+*ask*(0,100,**"hghv3"**));  
 System.*exit*(0);  
 }  
}

Файл New.form

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

<form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-class="com.katier.firstStep.New">

<grid id="cbd77" binding="contentPane" layout-manager="GridLayoutManager" row-count="2" column-count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">

<margin top="10" left="10" bottom="10" right="10"/>

<constraints>

<xy x="48" y="54" width="293" height="103"/>

</constraints>

<properties/>

<border type="none"/>

<children>

<grid id="94766" layout-manager="GridLayoutManager" row-count="1" column-count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1" vgap="-1">

<margin top="0" left="0" bottom="0" right="0"/>

<constraints>

<grid row="1" column="0" row-span="1" col-span="2" vsize-policy="1" hsize-policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false"/>

</constraints>

<properties/>

<border type="none"/>

<children>

<hspacer id="98af6">

<constraints>

<grid row="0" column="0" row-span="1" col-span="1" vsize-policy="1" hsize-policy="6" anchor="0" fill="1" indent="0" use-parent-layout="false"/>

</constraints>

</hspacer>

<grid id="9538f" layout-manager="GridLayoutManager" row-count="1" column-count="2" same-size-horizontally="true" same-size-vertically="false" hgap="-1" vgap="-1">

<margin top="0" left="0" bottom="0" right="0"/>

<constraints>

<grid row="0" column="1" row-span="1" col-span="1" vsize-policy="3" hsize-policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false"/>

</constraints>

<properties/>

<border type="none"/>

<children>

<component id="e7465" class="javax.swing.JButton" binding="buttonOK">

<constraints>

<grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-layout="false"/>

</constraints>

<properties>

<text value="OK"/>

</properties>

</component>

<component id="5723f" class="javax.swing.JButton" binding="buttonCancel">

<constraints>

<grid row="0" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-layout="false"/>

</constraints>

<properties>

<text value="Cancel"/>

</properties>

</component>

</children>

</grid>

</children>

</grid>

<component id="a3db2" class="javax.swing.JLabel" binding="warnLable">

<constraints>

<grid row="0" column="1" row-span="1" col-span="1" vsize-policy="0" hsize-policy="6" anchor="0" fill="0" indent="0" use-parent-layout="false"/>

</constraints>

<properties>

<text value="Enter count of vertexes"/>

</properties>

</component>

<component id="d6750" class="javax.swing.JTextField" binding="figureText">

<constraints>

<grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="0" anchor="0" fill="3" indent="0" use-parent-layout="false">

<preferred-size width="100" height="-1"/>

</grid>

</constraints>

<properties>

<text value=""/>

</properties>

</component>

</children>

</grid>

</form>