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

ОТЧЕТ

по учебной практике

Тема: Мосты

Студент гр. 7303 _____ Алексо А.А

Студент гр. 7303 _____ Бондарчук Н.Р.

Студент гр. 7304 _____ Овчинников Н.В.

Руководитель _____ Ефремов М.А.

Санкт-Петербург
2019

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Алексо А.А. группы 7303

Студент Бондарчук Н.Р. группы 7303

Студент Овчинников Н.В. группы 7304

Тема практики: выполнение мини-проекта на языке java.

Задание на практику:

Командная разработка приложения визуализации алгоритма «Поиска Мостов» на Java с графическим интерфейсом.

Алгоритм: Поиск Мостов.

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчета: 09.07.2019

Дата защиты отчета: 10.07.2019

Студент	_____	Алексо А.А.
Студент	_____	Бондарчук Н.Р.
Студент	_____	Овчинников Н.В.
Руководитель	_____	Ефремов М.А.

АННОТАЦИЯ

В ходе выполнения данной работы был реализован алгоритм поиска мостов для автоматического поиска мостов в графе. Практическое задание состоит из 4 частей. В первой части расписаны требования к программе в течении выполнения практического задания. Во второй описано распределение работы на группу студентов и план выполнения работы. В третьей части рассматриваются методы решения задачи и используемые структуры данных. В четвертую часть включено тестирование элементов программы. Также сделано заключение, описаны используемая литература.

SUMMARY

In the course of this work, the bridge finder algorithm was implemented. The practical task consists of 4 parts. The first part describes the requirements for the program during the practical task. The second describes the responsibilities of students for work and the work plan. The third part deals with the methods of implementation of the task and the data structures used. The forth part includes testing of the program, a conclusion, describes the source literature.

СОДЕРЖАНИЕ

	Введение	5
1.	Требования к программе	6
1.1.	Исходные требования к программе	6
1.2.	Уточнение требований после сдачи прототипа	6
1.3.	Уточнение требований после сдачи 1-ой версии	6
2.	План разработки и распределение ролей в бригаде	7
2.1.	План разработки	7
2.2.	Распределение ролей в бригаде	7
3.	Особенности реализации	8
3.1.	Использованные структуры данных	8
3.2.	Основные методы	9
3.3.	Использование интерфейса	10
4.	Тестирование	12
4.1	Тестирование кода алгоритма	12
	Заключение	13
	Список использованных источников	14
	Приложение А. Исходный код – только в электронном виде	17

ВВЕДЕНИЕ

Цель задачи – создание мини-проекта, который реализует алгоритм для автоматического поиска мостов в графе. В задаче требуется реализовать диаграмму всех использующихся классов, разработать алгоритм решения задачи и интерфейс к алгоритму. Также необходимо связать исходный код с интерфейсом и написать тесты к логике программы. Далее необходимо создать use-case диаграмму для данной программы в нотации UML.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

1.1.1. На рисунке 1 изображена use-case диаграмма. На старте пользователь может создать граф самостоятельно или считать его из файла. После выбора «Считать из файла» появляется окно с файловым менеджером. Выбрав файл, пользователь загрузит его содержимое в программу, где она автоматически отрисует граф на графической панели. На самой панели доступна кнопка «Поиск мостов».

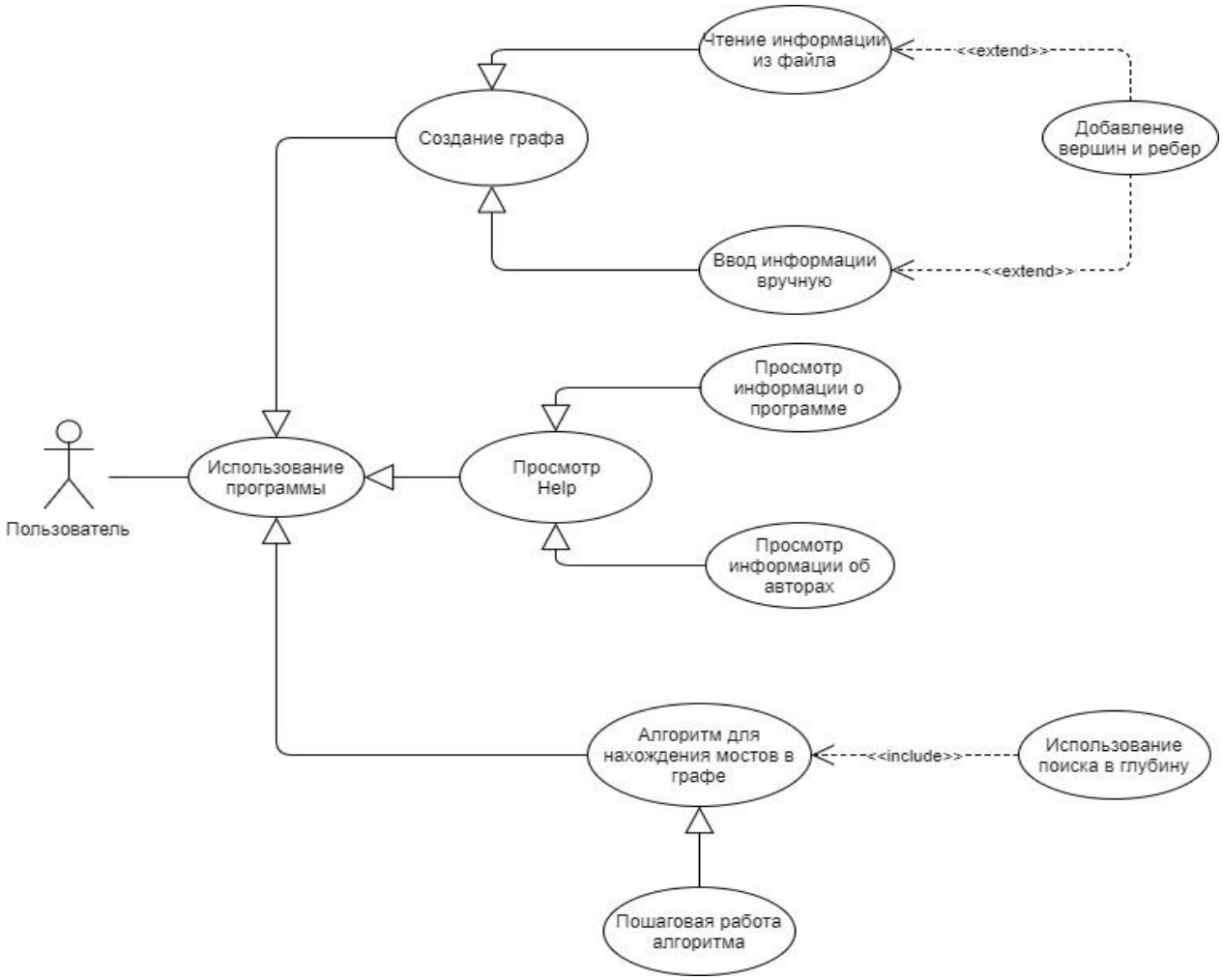

Рисунок 1 – Use-case диаграмма

1.2. Уточнение требований после сдачи прототипа.

1.2.1. Заполнить файл с темой задания

1.2.2. Сделать диаграмму классов.

1.2.3. Исправить недочеты в use-case диаграмме.

1.3. Уточнение требований после сдачи 1-ой версии.

1.3.1. Реализовать сборку проекта с помощью maven.

1.3.2. Сделать unit-тесты для модели.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

до 04.07: use case диаграмма, интерфейс без реализации логики.

до 06.07: прототип с демонстрацией функциональности программы.

до 08.07: рабочая версия со сборкой maven.

до 10.07: исправление недочетов первой версии.

2.2. Распределение ролей в бригаде

Алексо А.А.: базовые классы, логика алгоритма, отчет.

Бондарчук Н.Р.: юнит-тестирование, use-case диаграмма, сборка на maven.

Овчинников Н.В.: GUI, соединение интерфейса и логики программы, контроллер.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Использованные структуры данных

На рисунке 2 показана иерархия логики классов. Класс Drawing выполняет роль координатора между UI и состоянием приложения. В нем содержиться объект класса Graph, с которым идет взаимодействие программы.

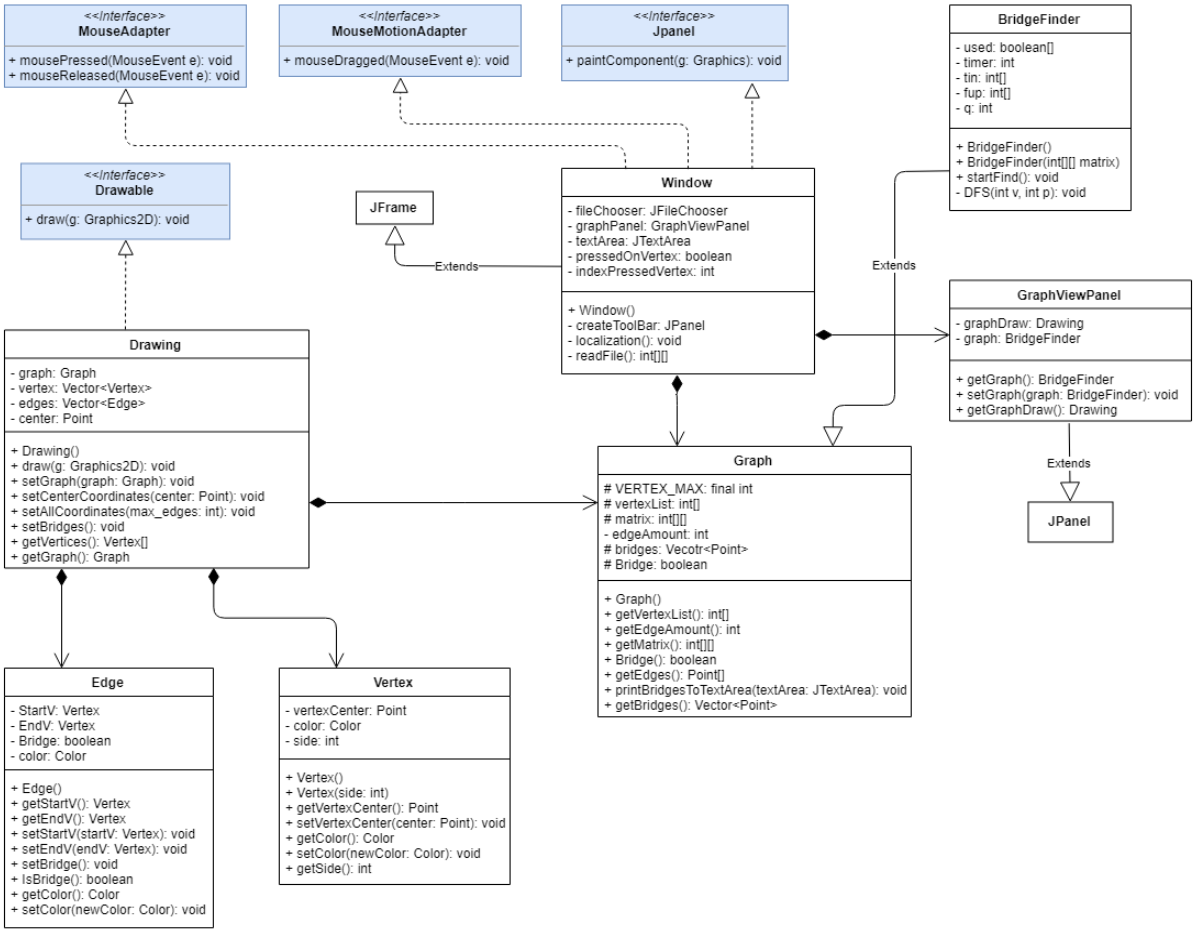

Рисунок 2 – классы логики программы.

3.3. Использование интерфейса

На следующих рисунках (рисунок 3, рисунок 4, рисунок5, рисунок6, рисунок7, рисунок8, рисунок9) представлен интерфейс программы. На рисунке 3 – главное мен. На рисунке 4 – меню выбора файла. На рисунке 5 – панель с графом. На рисунке 6, 7, 8, 9 – функционал программы и пример работы алгоритма поиска мостов.

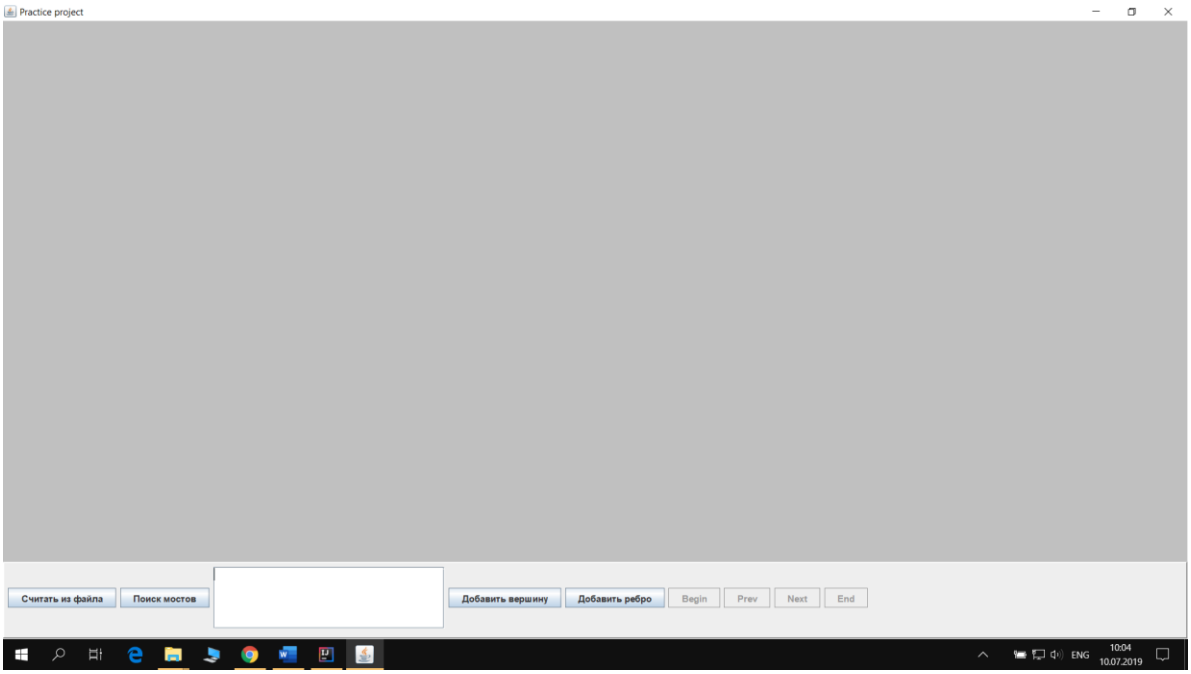

Рисунок 3 – интерфейс стартового меню программы

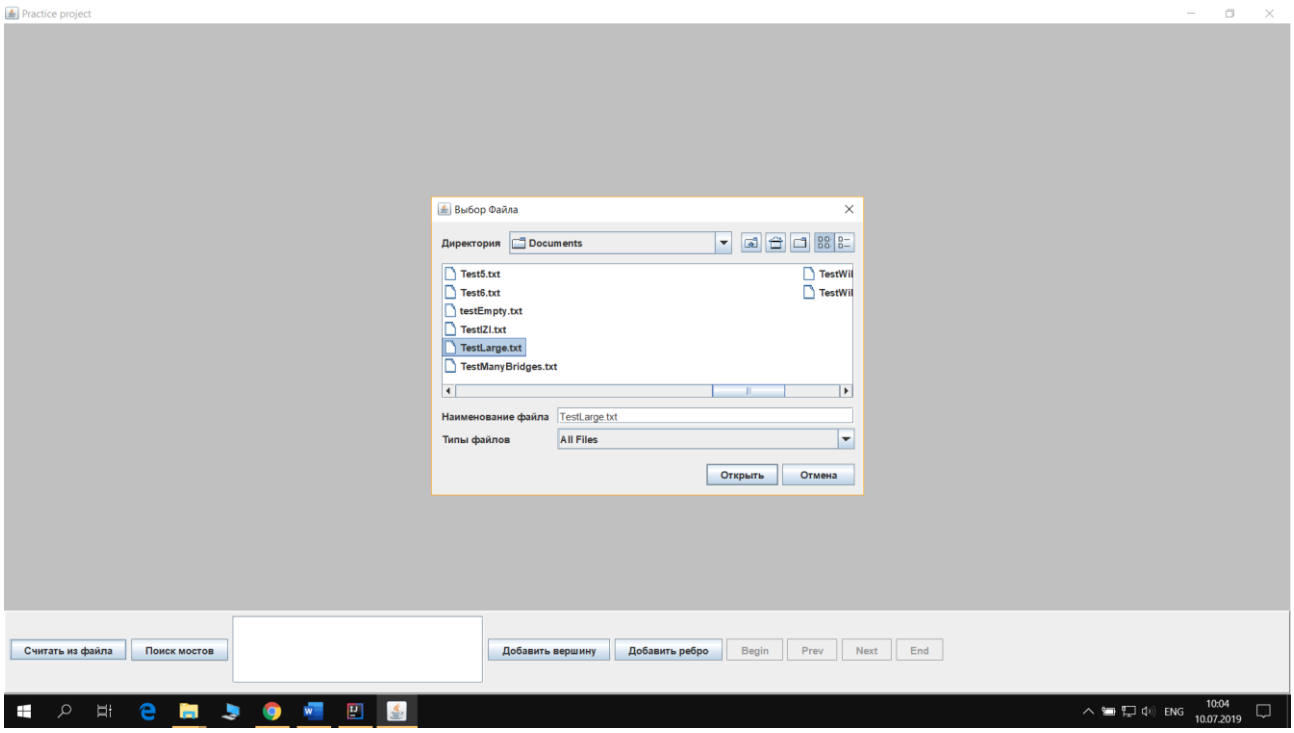

Рисунок 4 – меню выбора файла

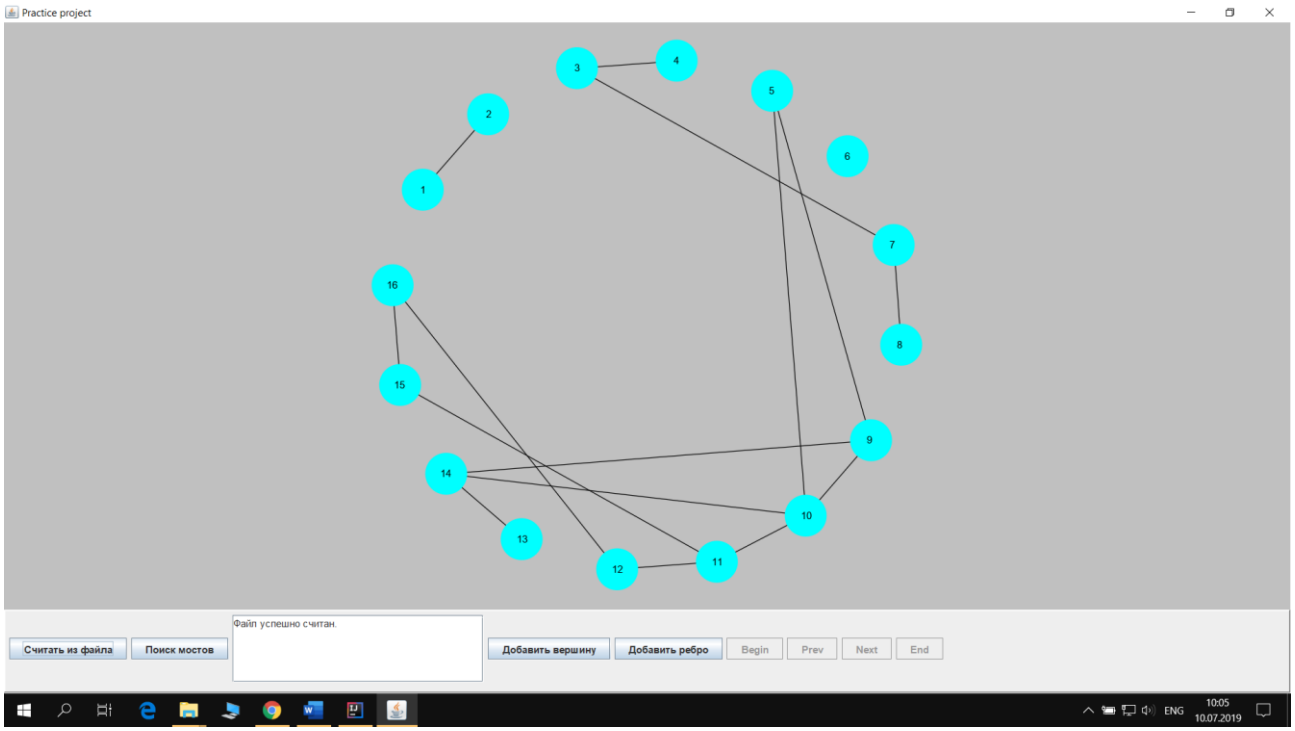

Рисунок 5 – графическая панель и граф

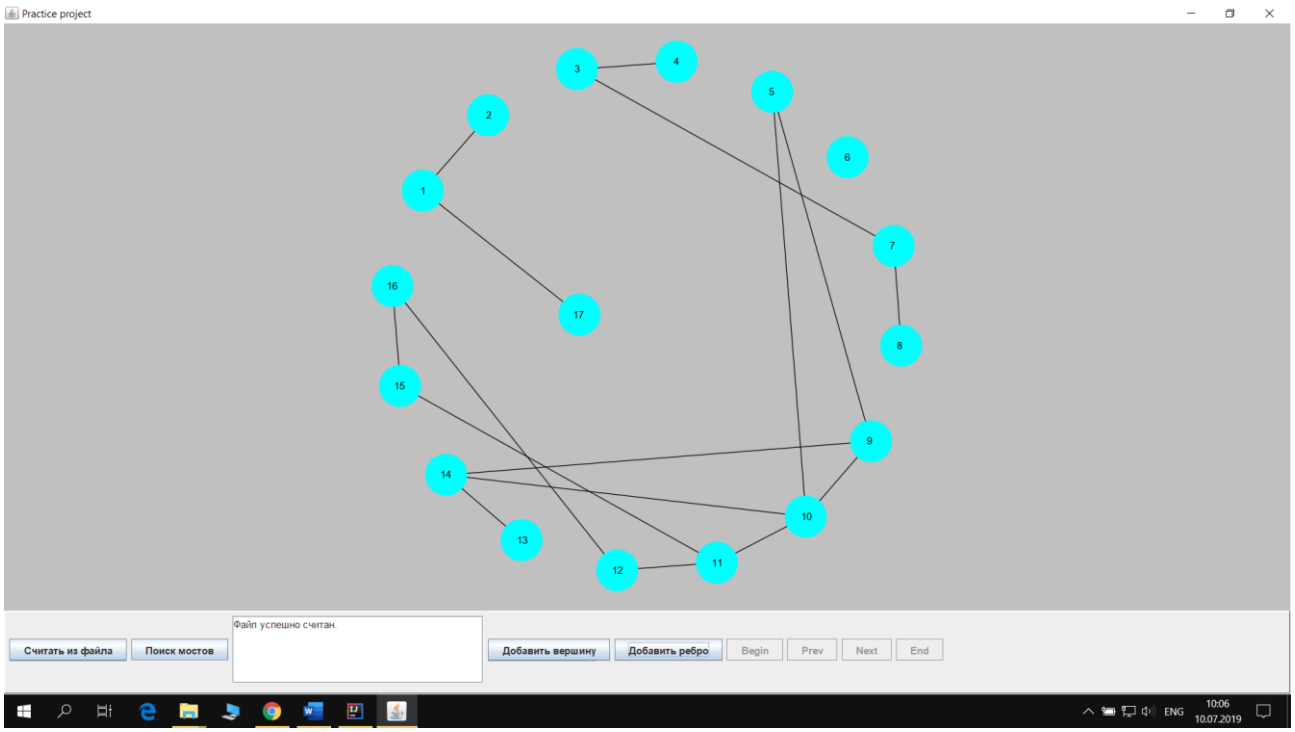

Рисунок 6 – добавление вершин и ребер

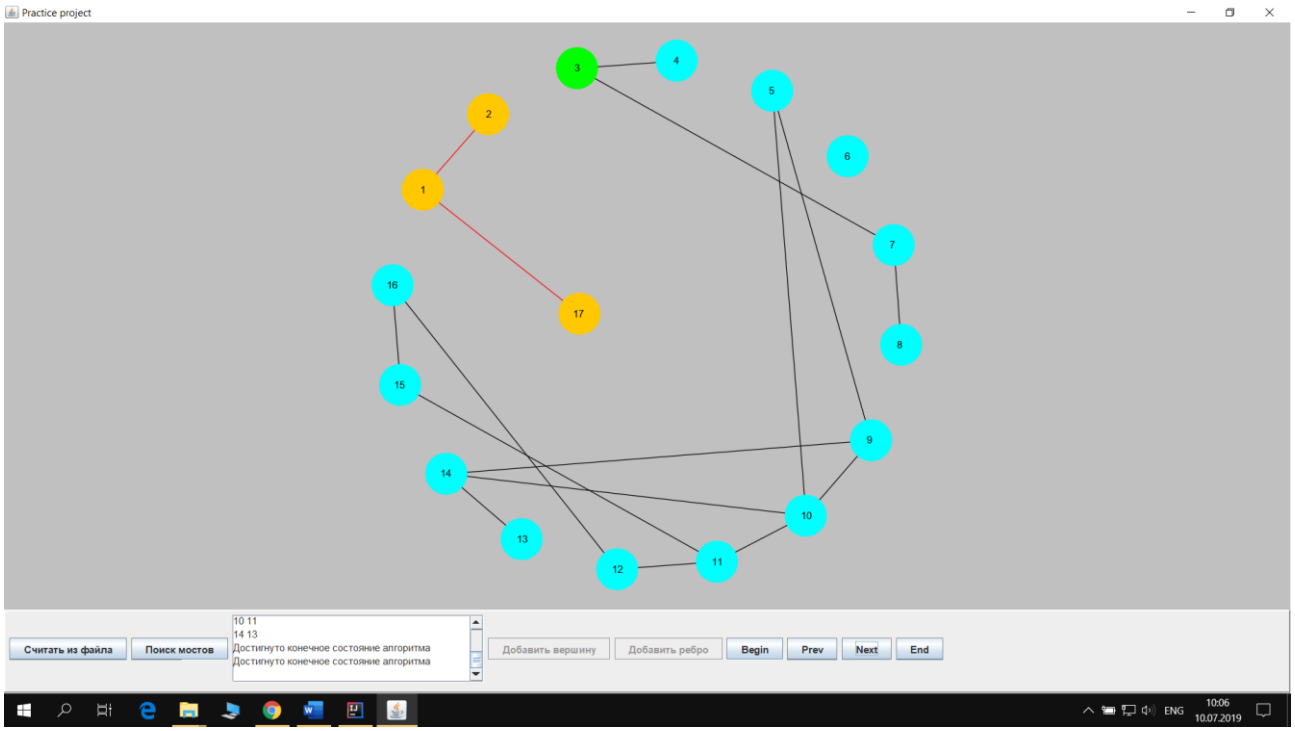

Рисунок 7 – пошаговая работа алгоритма

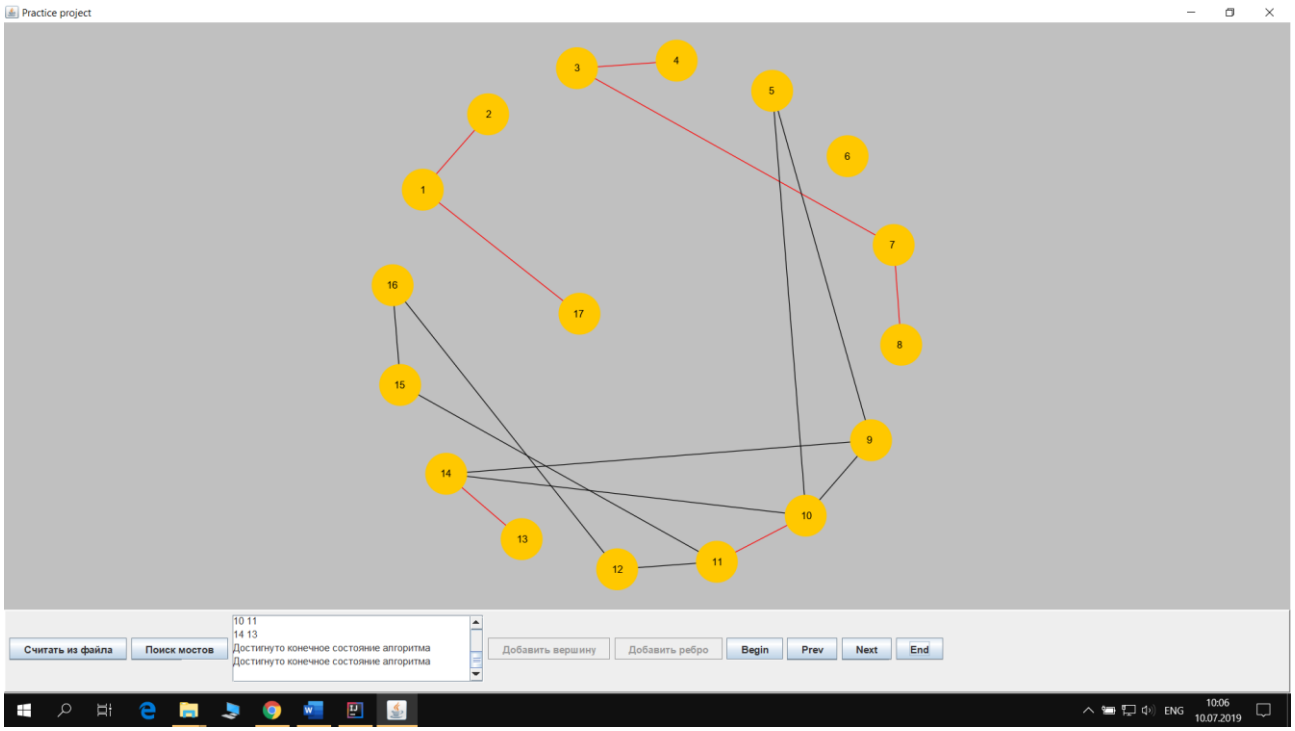

Рисунок 8 – конечное состояние графа

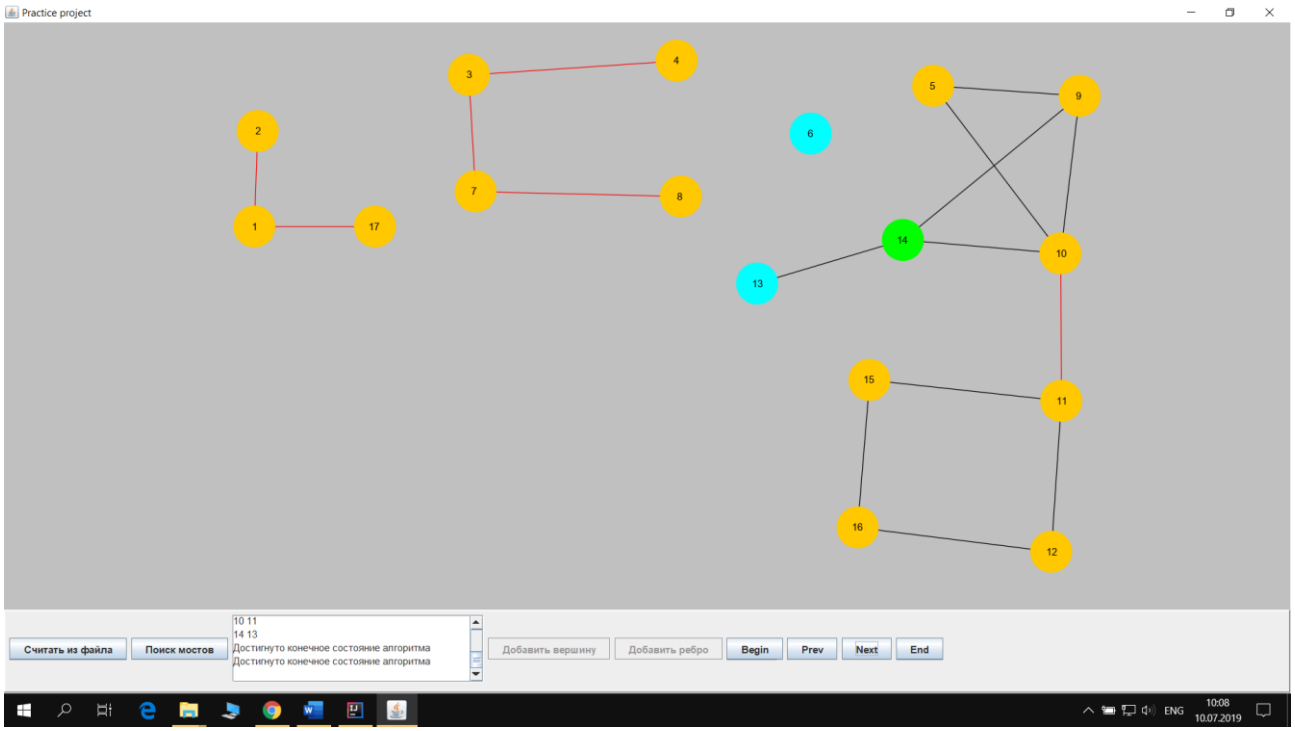

Рисунок 9 – возможность свободного перемещения вершин

4. ТЕСТИРОВАНИЕ

4.1. Тестирование кода алгоритма

Создается классы BridgeFinderTest и GraphTest для создания и хранения модульных тестов.

В методе *FindBridgeT()* создается объект класса BridgeFinder для использования во всех модульных тестах.

5 методов FindBridge() проверяют работу алгоритма.

2 метода getVertexList() проверяют правильность ввода матрицы.

2 метода getEdgeAmount() проверяет правильность ввода ребер в матрицу.

2 метода getMatrixTest() правильность передачи матрицы.

2 метода getEdgesTest() правильность метода getEdges() в графе.

2 метода getBridgesTest() правильность метода getBridges() в графе.

ЗАКЛЮЧЕНИЕ

В ходе разработки мини-проекта в нотации UML были созданы use-case диаграмма и диаграмма классов, по которым была построена модель решения задачи - класс, хранящий данные о состоянии головоломки и класс, вносящий изменения в эти данные по сигналам от UI. Пользовательский интерфейс "Мостов" был реализован при помощи библиотеки Swing. Сборка проекта осуществлялась фреймворком Apache Maven. В итоге после тестирования были получены полноценная реализация алгоритма "Поиска Мостов" с выбором файла, а также опыт разработки на языке Java.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кэти Сьерра и Берт Бейтс Изучаем Java: Москва, 2016, 383-428с.

2. Документация Java™ Platform, Standard Edition 7 API Specification [Электронный ресурс] // Copyright © 1993, 2018, Oracle and/or its affiliates. URL: https://docs.oracle.com/javase/7/docs/api/index.html 04.07.2019

3. Руководство по maven – что такое maven [Электронный ресурс], Apache Maven Project. URL: www.apache-maven.ru 03.07.2019

Приложение А

Исходный код программы

FindBridges.java

```
import gui.*;

import javax.swing.*;

public class FindBridges {
   public static void main(String[] args){
      JFrame window = new Window();
   }
}
```

Drawable.java

```
package draw;

import java.awt.*;


public interface Drawable {
   void draw(Graphics2D g);
}
```

Drawing.java

```
package draw;

import java.awt.*;
import java.util.Vector;
import source.*;

public class Drawing implements Drawable {
   private Graph graph;
   private Vector<Vertex> vertex;
   private Vector<Edge> edges;
   private Point center;
   private Condition condition;

   public Drawing() {
      graph = null;
      vertex = new Vector<Vertex>();
   }

   public void draw(Graphics2D g) {
      if(vertex == null)
```

```
            return;
        int ovalSide = vertex.elementAt(0).getSide();
        for(int i = 0; i < edges.size(); i++) {
            if(edges.elementAt(i).IsBridge()) {
                g.setColor(Color.RED);
            }
            else {
                g.setColor(edges.elementAt(i).getColor());
            }
            g.drawLine(edges.elementAt(i).getStartV().getVertexCenter().x,
edges.elementAt(i).getStartV().getVertexCenter().y,
                    edges.elementAt(i).getEndV().getVertexCenter().x,
edges.elementAt(i).getEndV().getVertexCenter().y);
        }
        for(int i = vertex.size()-1; i > -1; i--) {
            g.setColor(vertex.elementAt(i).getColor());
            g.fillOval(vertex.elementAt(i).getVertexCenter().x - ovalSide/2,
vertex.elementAt(i).getVertexCenter().y - ovalSide/2,
                    ovalSide, ovalSide);
            g.setColor(Color.BLACK);
            g.drawString(Integer.toString(i + 1), vertex.elementAt(i).getVertexCenter().x
- 3 - 3*(i+1)/10, vertex.elementAt(i).getVertexCenter().y + 4);
        }
    }

    public void drawWithCondition(Graphics2D g){
        graph.setBridges(condition.getBridges());
        setBridges();
        if(vertex == null)
            return;
        int ovalSide = vertex.elementAt(0).getSide();
        for(int i = 0; i < edges.size(); i++) {
            if(edges.elementAt(i).IsBridge()) {
                g.setColor(Color.RED);
            }
            else {
                g.setColor(edges.elementAt(i).getColor());
            }
            g.drawLine(edges.elementAt(i).getStartV().getVertexCenter().x,
edges.elementAt(i).getStartV().getVertexCenter().y,
                    edges.elementAt(i).getEndV().getVertexCenter().x,
edges.elementAt(i).getEndV().getVertexCenter().y);
        }
        for(int i = vertex.size()-1; i > -1; i--) {
```

```
            if(condition.getCurrentV() == i) {
               g.setColor(Color.GREEN);
            }
            else if(condition.getUsed()[i]) {
               g.setColor(Color.ORANGE);
            }
            else {
               g.setColor(vertex.elementAt(i).getColor());
            }
            g.fillOval(vertex.elementAt(i).getVertexCenter().x - ovalSide/2,
vertex.elementAt(i).getVertexCenter().y - ovalSide/2,
                  ovalSide, ovalSide);
            g.setColor(Color.BLACK);
            g.drawString(Integer.toString(i + 1), vertex.elementAt(i).getVertexCenter().x
- 3 - 3*(i+1)/10, vertex.elementAt(i).getVertexCenter().y + 4);
         }
      }

      public void setGraph(Graph graph){
         this.graph = graph;
         for(int i = 0; i < graph.getVertexList().length; i++) {
            vertex.add(new Vertex(50 + vertex.size()));
         }
         edges = new Vector<Edge>();
         setAllCoordinates();
         //setBridges();
      }

      public void setCenterCoordinates(Point center){ this.center = center; }

      private void setAllCoordinates(){
         int R = 50 + graph.getVertexList().length * graph.getVertexList().length;
         for(int i = 0; i < graph.getVertexList().length; i++){
            vertex.elementAt(i).setVertexCenter(new Point((int)(center.x + R *
Math.cos((double)360/graph.getVertexList().length + 2 * Math.PI * i /
graph.getVertexList().length)),
                  (int)(center.y + R * Math.sin((double)360/graph.getVertexList().length +
2 * Math.PI * i / graph.getVertexList().length))));
         }
         Point[] edges_arr = graph.getEdges();
         if(edges_arr == null)
            System.out.println("Ребер нет, но вы держитесь");
         else {
            for (int i = 0; i < graph.getEdgeAmount(); i++) {
```

```
            edges.add(new Edge(vertex.elementAt(edges_arr[i].x),
vertex.elementAt(edges_arr[i].y)));
         }
      }
   }

   public void setBridges() {
      if(!graph.getBridges().isEmpty()) {
         for(int i = 0; i < graph.getBridges().size(); i++){
            for(int j = 0; j < graph.getEdgeAmount(); j++) {
               if (vertex.elementAt(graph.getBridges().elementAt(i).x - 1) ==
edges.elementAt(j).getStartV() ||
                     vertex.elementAt(graph.getBridges().elementAt(i).x - 1) ==
edges.elementAt(j).getEndV()) {
                  if (vertex.elementAt(graph.getBridges().elementAt(i).y - 1) ==
edges.elementAt(j).getEndV() ||
                        vertex.elementAt(graph.getBridges().elementAt(i).y - 1) ==
edges.elementAt(j).getStartV())
                  {
                     edges.elementAt(j).setBridge(true);
                  }
               }
            }
         }
      }
   }

   public Vector<Vertex> getVertices(){ return vertex; }
   public Vector<Edge> getEdges(){ return edges; }

   public void removeBridges(){
      graph.clearBridges();
      for(int i=0; i < edges.size(); i++){
         edges.elementAt(i).setBridge(false);
      }
   }

   public void clearBridges(){
      for(int i=0; i < edges.size(); i++){
         edges.elementAt(i).setBridge(false);
      }
   }

   public void setCondition(Condition condition){ this.condition = condition; }
```

```
}
```

EdgeDrawer.java

```
package draw;

import java.awt.Graphics;
import java.awt.Point;
import javax.swing.JPanel;

public class EdgeDrawer extends JPanel {
   public EdgeDrawer(Point one, Point two) {
   }

   public void paint(Graphics g) {
   }
}
```

VertexDrawer.java

```
package draw;

import java.awt.Graphics;
import java.awt.Point;
import javax.swing.JPanel;

public class VertexDrawer extends JPanel {
   public VertexDrawer(Point one) {
   }

   public void paint(Graphics g) {
   }
}
```

GraphViewPanel.java

```
package gui;

import draw.*;
import source.*;
import javax.swing.*;
import java.awt.*;

public class GraphViewPanel extends JPanel {
   private Drawing graphDraw;
   private BridgeFinder graph;
```

```
    public GraphViewPanel(){
        graphDraw = new Drawing();
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        //обязательная проверка
        if(graph == null)
            return;

        if(graph.getConditionList() == null || graph.getConditionList().size() < 1) {
            graphDraw.draw(g2);
        }
        else {
            graphDraw.drawWithCondition(g2);
        }
    }

    public BridgeFinder getGraph(){
        return graph;
    }

    public void setGraph(BridgeFinder graph){
        this.graph = graph;
        graphDraw.setCenterCoordinates(new Point(this.getVisibleRect().width/2,
this.getVisibleRect().height/2));
    }

    public Drawing getGraphDraw(){ return graphDraw; }
}
```

Window.java

```
package gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import source.*;
```

```
import java.io.*;
import java.util.*;

public class Window extends JFrame {

    private JFileChooser fileChooser = null;
    private GraphViewPanel graphPanel;
    private JTextArea textArea;

    private boolean pressedVertexForDrag;
    private int indexVertexForDrag;

    private boolean pressedAddEdge;
    private boolean pressedVertexForEdge;
    private int indexVertexForEdge;

    private int conditionIterator = 0;

    public Window(){
        super("Practice project");

        // Frame
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(500, 500);
        setMinimumSize(Toolkit.getDefaultToolkit().getScreenSize());
        setExtendedState(MAXIMIZED_BOTH);

        /*
        // MenuBar
        //JMenuBar menuBar = new JMenuBar();
        //menuBar.add(new JMenu("Справка"));
        //setJMenuBar(menuBar);
         */

        // Panels
        graphPanel = new GraphViewPanel();
        graphPanel.setBackground(Color.LIGHT_GRAY);
        JPanel toolBar = createPanel();
        toolBar.setPreferredSize(new Dimension(1000, 100));

        add(graphPanel);
        add(toolBar, BorderLayout.SOUTH);

        setVisible(true);
```

```
        // Mouse Listeners
        graphPanel.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                super.mousePressed(e);
                indexVertexForDrag = -1;

                if(pressedAddEdge) {
                    if(pressedVertexForEdge == false) {
                        Vector<Vertex> vertices = graphPanel.getGraphDraw().getVertices();
                        for (int i = 0; i < vertices.size(); i++) {
                            if (Math.sqrt(Math.pow(e.getX() -
vertices.elementAt(i).getVertexCenter().x, 2) + Math.pow(e.getY() -
vertices.elementAt(i).getVertexCenter().y, 2))
                                    < (double) vertices.elementAt(i).getSide() / 2) {
                                indexVertexForEdge = i;
                                pressedVertexForEdge = true;
                                break;
                            }
                        }
                    }
                    else {
                        int current = -1;
                        Vector<Vertex> vertices = graphPanel.getGraphDraw().getVertices();
                        for (int i = 0; i < vertices.size(); i++) {
                            if (Math.sqrt(Math.pow(e.getX() -
vertices.elementAt(i).getVertexCenter().x, 2) + Math.pow(e.getY() -
vertices.elementAt(i).getVertexCenter().y, 2))
                                    < (double) vertices.elementAt(i).getSide() / 2) {
                                current = i;
                                break;
                            }
                        }

                        if(current > -1 && current != indexVertexForEdge) {

if(graphPanel.getGraph().getMatrix()[indexVertexForEdge][current] == 1){
                                indexVertexForEdge = -1;
                                pressedAddEdge = false;
                                pressedVertexForEdge = false;
                                return;
                            }
```

```
                    graphPanel.getGraphDraw().getEdges().add(new
Edge(graphPanel.getGraphDraw().getVertices().elementAt(indexVertexForEdge),

graphPanel.getGraphDraw().getVertices().elementAt(current)));
                    graphPanel.getGraph().addNewEdge(indexVertexForEdge,
current);
                    graphPanel.getGraphDraw().removeBridges();
                    graphPanel.repaint(graphPanel.getVisibleRect());
                }

                indexVertexForEdge = -1;
                pressedAddEdge = false;
                pressedVertexForEdge = false;
            }
        }
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        super.mouseReleased(e);


    }
});

graphPanel.addMouseMotionListener(new MouseMotionAdapter() {
    @Override
    public void mouseDragged(MouseEvent e) {
        super.mouseDragged(e);

        if(graphPanel.getGraphDraw().getVertices() == null)
            return;

        if(indexVertexForDrag == -1) {
            Vector<Vertex> vertices = graphPanel.getGraphDraw().getVertices();
            for (int i = 0; i < vertices.size(); i++) {
                if (Math.sqrt(Math.pow(e.getX() -
vertices.elementAt(i).getVertexCenter().x, 2) + Math.pow(e.getY() -
vertices.elementAt(i).getVertexCenter().y, 2))
                        < (double) vertices.elementAt(i).getSide() / 2) {
                    pressedVertexForDrag = true;
                    indexVertexForDrag = i;
                    break;
                }
```

```
                }
            }
            else {
                pressedVertexForDrag = true;
            }

            if(pressedVertexForDrag){

graphPanel.getGraphDraw().getVertices().elementAt(indexVertexForDrag).setVertex
Center(new Point(e.getX(), e.getY()));
            }

            pressedVertexForDrag = false;
            //indexVertexForDrag = -1;
            graphPanel.repaint(graphPanel.getVisibleRect());
        }
    });
}

private JPanel createPanel(){
    // Buttons
    final JButton buttonAddVertex = new JButton("Добавить вершину");
    buttonAddVertex.addActionListener(new ButtonAddVertex());

    final JButton buttonAddEdge = new JButton("Добавить ребро");
    buttonAddEdge.addActionListener(new ButtonAddEdge());

    final JButton buttonFirstCondition = new JButton("Begin");
    buttonFirstCondition.addActionListener(new ButtonFirstCondition());
    buttonFirstCondition.setEnabled(false);

    final JButton buttonPrevCondition = new JButton("Prev");
    buttonPrevCondition.addActionListener(new ButtonPrevCondition());
    buttonPrevCondition.setEnabled(false);

    final JButton buttonNextCondition = new JButton("Next");
    buttonNextCondition.addActionListener(new ButtonNextCondition());
    buttonNextCondition.setEnabled(false);

    final JButton buttonEndCondition = new JButton("End");
    buttonEndCondition.addActionListener(new ButtonEndCondition());
    buttonEndCondition.setEnabled(false);

    JButton buttonReadFile = new JButton("Считать из файла");
```

```
        //buttonReadFile.addActionListener(new ButtonFileRead());
        buttonReadFile.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    graphPanel.setGraph(new BridgeFinder(readFile()));
                }
                catch (NullPointerException ex){
                    textArea.append("Файл не считан.\n");
                    return;
                }

                buttonAddVertex.setEnabled(true);
                buttonAddEdge.setEnabled(true);
                buttonFirstCondition.setEnabled(false);
                buttonPrevCondition.setEnabled(false);
                buttonNextCondition.setEnabled(false);
                buttonEndCondition.setEnabled(false);

                //для того, чтобы при инициализированном графе, когда вызывается
кнопка "считать файл" и закрывается,
                //а затем вызыввался поиск мостов, не падала программа
                graphPanel.getGraphDraw().getVertices().clear();
                graphPanel.getGraph().getConditionList().clear();

                graphPanel.getGraphDraw().setGraph(graphPanel.getGraph());
                textArea.append("Файл успешно считан.\n");
                graphPanel.repaint(graphPanel.getVisibleRect());
            }
        });

        JButton buttonFindBridge = new JButton("Поиск мостов");
        buttonFindBridge.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if(graphPanel.getGraph() == null){
                    textArea.append("Граф не инициализирован.\n");
                    return;
                }

                buttonAddVertex.setEnabled(false);
                buttonAddEdge.setEnabled(false);
                buttonFirstCondition.setEnabled(true);
                buttonPrevCondition.setEnabled(true);
                buttonNextCondition.setEnabled(true);
                buttonEndCondition.setEnabled(true);
```

```
            graphPanel.getGraphDraw().removeBridges();
            graphPanel.getGraph().startFind();
            graphPanel.getGraph().printBridgesToTextAre(textArea);

            conditionIterator = 0;

graphPanel.getGraphDraw().setCondition(graphPanel.getGraph().getConditionList().
elementAt(conditionIterator));

            graphPanel.repaint(graphPanel.getVisibleRect());
        }
    });

    // Text Area
    textArea = new JTextArea();
    textArea.setLineWrap(true);
    JScrollPane log = new JScrollPane(textArea);
    log.setPreferredSize(new Dimension(300,80));

    // Tool Bar
    JPanel toolBar = new JPanel(new FlowLayout(FlowLayout.LEFT));
    toolBar.add(buttonReadFile);
    toolBar.add(buttonFindBridge);
    toolBar.add(log);
    toolBar.add(buttonAddVertex);
    toolBar.add(buttonAddEdge);
    toolBar.add(buttonFirstCondition);
    toolBar.add(buttonPrevCondition);
    toolBar.add(buttonNextCondition);
    toolBar.add(buttonEndCondition);
    toolBar.setBorder(BorderFactory.createRaisedBevelBorder());
    return toolBar;
}

/*
//кнопка "Считать с файла"
public class ButtonFileRead implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        try {
            graphPanel.setGraph(new BridgeFinder(readFile()));
        }
        catch (NullPointerException ex){
            textArea.append("Файл не считан.\n");
```

```
            return;
        }

        //для того, чтобы при инициализированном графе, когда вызывается
кнопка "считать файл" и закрывается,
        //а затем вызыввался поиск мостов, не падала программа
        graphPanel.getGraphDraw().getVertices().clear();
        graphPanel.getGraph().getConditionList().clear();

        graphPanel.getGraphDraw().setGraph(graphPanel.getGraph());
        textArea.append("Файл успешно считан.\n");
        graphPanel.repaint(graphPanel.getVisibleRect());
    }
}

 */

/*
//кнопка "Поиск мостов"
public class ButtonFindBridge implements ActionListener {
    public void actionPerformed(ActionEvent e){

        if(graphPanel.getGraph() == null){
            textArea.append("Граф не инициализирован.\n");
            return;
        }
        graphPanel.getGraphDraw().removeBridges();
        graphPanel.getGraph().startFind();
        graphPanel.getGraph().printBridgesToTextAre(textArea);

        conditionIterator = 0;

graphPanel.getGraphDraw().setCondition(graphPanel.getGraph().getConditionList().
elementAt(conditionIterator));

        graphPanel.repaint(graphPanel.getVisibleRect());
    }
}
 */

//кнопка "Добавить вершину"
public class ButtonAddVertex implements ActionListener {
    public void actionPerformed(ActionEvent e){
        if(graphPanel.getGraph() == null || graphPanel.getGraphDraw() == null) {
```

```
            graphPanel.setGraph(new BridgeFinder(new int[0][0]));
            graphPanel.getGraphDraw().setGraph(graphPanel.getGraph());
        }

        //добавляем в вектор вершин для рисования вершину с центром в
середине окна
        graphPanel.getGraphDraw().getVertices().add(new Vertex(new
Point(graphPanel.getVisibleRect().width/2, graphPanel.getVisibleRect().height/2),
                50));

        //добавляем вершину в vertexList и matrix текущего графа графа
        graphPanel.getGraph().addNewVertex();

        graphPanel.repaint(graphPanel.getVisibleRect());
      }
   }

   //кнопка "Добавить ребро"
   public class ButtonAddEdge implements ActionListener {
      public void actionPerformed(ActionEvent e){
         pressedAddEdge = true;
      }
   }

   //кнопка "Начальное состояние"
   public class ButtonFirstCondition implements ActionListener {
      public void actionPerformed(ActionEvent e){
         graphPanel.getGraphDraw().clearBridges();
         conditionIterator = 0;

graphPanel.getGraphDraw().setCondition(graphPanel.getGraph().getConditionList().
elementAt(conditionIterator));
         graphPanel.repaint(graphPanel.getVisibleRect());
      }
   }

   //кнопка "Следующее состояние"
   public class ButtonNextCondition implements ActionListener {
      public void actionPerformed(ActionEvent e){
         if(graphPanel.getGraph().getConditionList().size()-1 > conditionIterator) {

graphPanel.getGraphDraw().setCondition(graphPanel.getGraph().getConditionList().
elementAt(++conditionIterator));
         }
```

```
            else {
                textArea.append("Достигнуто конечное состояние алгоритма\n");
            }
            graphPanel.repaint(graphPanel.getVisibleRect());
        }
    }

    //кнопка "Предыдущее состояние"
    public class ButtonPrevCondition implements ActionListener {
        public void actionPerformed(ActionEvent e){
            if(conditionIterator > 0) {
                graphPanel.getGraphDraw().clearBridges();

graphPanel.getGraphDraw().setCondition(graphPanel.getGraph().getConditionList().
elementAt(--conditionIterator));
            }
            else {
                textArea.append("Достигнуто конечное состояние алгоритма\n");
            }
            graphPanel.repaint(graphPanel.getVisibleRect());
        }
    }

    //кнопка "Конечное состояние"
    public class ButtonEndCondition implements ActionListener {
        public void actionPerformed(ActionEvent e){
            conditionIterator = graphPanel.getGraph().getConditionList().size() - 1;

graphPanel.getGraphDraw().setCondition(graphPanel.getGraph().getConditionList().
elementAt(conditionIterator));
            graphPanel.repaint(graphPanel.getVisibleRect());
        }
    }

    private void localization(){
        UIManager.put(
                "FileChooser.openButtonText", "Открыть");
        UIManager.put(
                "FileChooser.cancelButtonText", "Отмена");
        UIManager.put(
                "FileChooser.fileNameLabelText", "Наименование файла");
        UIManager.put(
                "FileChooser.filesOfTypeLabelText", "Типы файлов");
        UIManager.put(
```

```
            "FileChooser.lookInLabelText", "Директория");
        UIManager.put(
            "FileChooser.saveInLabelText", "Сохранить в директории");
        UIManager.put(
            "FileChooser.folderNameLabelText", "Путь директории");
        fileChooser = new JFileChooser();
    }

    private int[][] readFile(){
        localization();
        int[][] matrix = new int[0][0];
        fileChooser.setDialogTitle("Выбор Файла");
        // Определение режима - только файл
        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        int result = fileChooser.showOpenDialog(Window.this);
        // Создаем наш файл
        File newfile;
        try {
            newfile = new File(fileChooser.getSelectedFile().toString());
        }
        catch (NullPointerException e){
            throw e;
        }
        if(newfile.exists())
        {
            try {
                int count = 0;
                int i = 0;
                int number = 0;
                int numbervertex = 0;
                int k = 0;

                FileInputStream fstream = new FileInputStream(newfile);
                BufferedReader br = new BufferedReader(new
InputStreamReader(fstream));
                String strLine;
                if ((strLine = br.readLine()) != null) {
                    i = Integer.parseInt(strLine);
                    matrix = new int[i][i];
                }
                while ((strLine = br.readLine()) != null) {
                    char[] ary = strLine.toCharArray();
                    for(number = 0; number < ary.length; number++) {
                        if(number + 1 < ary.length) {
```

```
                    if ((ary[number + 1] == ' ')) {
                        matrix[count][numbervertex] =
Character.getNumericValue(ary[number]);
                        numbervertex++;
                        number++;
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(Window.this, "Ошибка,
неверные данные");
                        System.exit(0);
                    }
                }
                if(number + 1 == ary.length){
                    matrix[count][numbervertex] =
Character.getNumericValue(ary[number]);
                    count ++;
                    numbervertex = 0;
                }
            }
        }
      }catch (IOException q){
        textArea.append("Файл не считан.\n");
      }
    }
    return matrix;
  }
}
```

BridgeFinder.java

```
package source;

import java.awt.*;
import java.util.Vector;

public class BridgeFinder extends Graph {
  private boolean[] used;
  private int timer;
  private int[] tin;
  private int[] fup;
  private int q = -1;
  private Vector<Condition> conditionList;
  private int lastVisitedVertex;
```

```
public BridgeFinder(){
   used = new boolean[vertexList.length];
   tin = new int[vertexList.length];
   fup = new int[vertexList.length];
   conditionList = new Vector<Condition>();
}

public BridgeFinder(int[][] matrix){
   if(matrix.length != 0) {
      bridges = new Vector<Point>();
      setMatrixValues(matrix);
      vertexList = new int[matrix[0].length];
      for (int i = 0; i < vertexList.length; i++) {
         vertexList[i] = i;
      }

      used = new boolean[vertexList.length];
      tin = new int[vertexList.length];
      fup = new int[vertexList.length];
   }
   conditionList = new Vector<Condition>();
}

//метод добавления вершины к vertexList
public void addNewVertex(){
   vertexList = new int[vertexList.length+1];
   for (int i = 0; i < vertexList.length; i++) {
      vertexList[i] = i;
   }
   used = new boolean[vertexList.length];
   tin = new int[vertexList.length];
   fup = new int[vertexList.length];
}

//инициализирует ребра в основной матрице 100x100 по переданной
private void setMatrixValues(int[][] newMatrix){
   for(int i=0; i < newMatrix.length; i++){
      for(int k=0; k < newMatrix[i].length; k++){
         this.matrix[i][k] = newMatrix[i][k];
      }
   }
}

public void addNewEdge(int startV, int endV){
```

```
        matrix[startV][endV] = 1;
        matrix[endV][startV] = 1;
        edgeAmount++;
    }

    public void startFind(){
        timer = 0;
        for(int i = 0; i < vertexList.length; i++){
            used[i] = false;
        }
        for(int i=0; i < vertexList.length; i++){
            if(!used[i]){
                DFS(i, q);
                q = -1;
            }
        }
        conditionList.add(new Condition(new Vector<Point>(bridges),
makeNewUsed(used), lastVisitedVertex));
        used[lastVisitedVertex] = true;
        conditionList.add(new Condition(new Vector<Point>(bridges),
makeNewUsed(used), -1));
    }

    private void DFS(int v, int p){
        conditionList.add(new Condition(new Vector<Point>(bridges),
makeNewUsed(used), v));
        lastVisitedVertex = v;
        used[v] = true;
        tin[v] = fup[v] = timer++;
        for(int i=0; i < matrix[v].length; ++i){
            if(matrix[v][i] == 0 || i == p){
                continue;
            }
            if(used[i]){
                fup[v] = Math.min(fup[v], tin[i]);
            }
            else {
                DFS(i, v);
                fup[v] = Math.min(fup[v], fup[i]);
                if(fup[i] > tin[v]){
                    bridges.add(new Point(v+1, i+1));
                }
            }
        }
```

```
    }

    public Object[] FindBridgeT(int[][] matrix, int vertex){
        if(vertex <= 0 || vertex > 100){
            BridgeFinder BridgeT = new BridgeFinder(matrix);
            BridgeT.bridges.add(new Point(vertex, vertex));
            return BridgeT.bridges.toArray();
        }
        BridgeFinder BridgeT = new BridgeFinder(matrix);
        BridgeT.startFind();
        return BridgeT.bridges.toArray();

    }

    public Vector<Condition> getConditionList(){ return conditionList; }

    private boolean[] makeNewUsed(boolean[] used){
        boolean[] newUsed = new boolean[used.length];
        for(int i=0; i< used.length; i++){
            newUsed[i] = used[i];
        }
        return newUsed;
    }
}
```

Condition.java

```
package source;

import java.awt.*;
import java.util.Vector;

public class Condition {
    private Vector<Point> bridges;
    private boolean[] used;
    private int currentV;

    public Condition(Vector<Point> bridges, boolean[] used, int currentV){
        this.bridges = bridges;
        this.used = used;
        this.currentV = currentV;
    }

    public int getCurrentV() {
        return currentV;
```

```
    }
    public Vector<Point> getBridges() {
        return bridges;
    }
    public boolean[] getUsed() {
        return used;
    }
}
```

Edge.java

```
package source;

import java.awt.*;

public class Edge {
    private Vertex StartV;
    private Vertex EndV;
    private boolean Bridge;
    private Color color;

    public Edge() {
        StartV = new Vertex();
        EndV = new Vertex();
        Bridge = false;
        color = Color.BLACK;
    }

    public Edge(Vertex startV, Vertex endV){
        this.StartV = startV;
        this.EndV = endV;
        Bridge = false;
        color = Color.BLACK;
    }

    public Vertex getStartV() {
        return StartV;
    }
    public Vertex getEndV() {
        return EndV;
    }
    public void setStartV(Vertex startV) { StartV = startV; }
    public void setEndV(Vertex endV) {
        EndV = endV;
    }
```

```
    public void setBridge(boolean bool) { Bridge = bool; }
    public boolean IsBridge() { return Bridge; }
    public Color getColor() {
        return color;
    }
    public void setColor(Color newColor) {
        color = newColor;
    }
}
```

Vertex.java

```
package source;

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

public class Graph {
    protected final int VERTEX_MAX = 100;
    protected int[] vertexList;
    protected int[][] matrix;
    protected int edgeAmount = 0;
    protected Vector<Point> bridges;

    public Graph(){
        bridges = new Vector<Point>();
        vertexList = new int[0];
        matrix = new int[VERTEX_MAX][VERTEX_MAX];
    }

    public int[] getVertexList(){
        return vertexList;
    }
    //public void setVertexList(int[] vertexList){ this.vertexList = vertexList; }

    public int[][] getMatrix(){
        return matrix;
    }
    //public void setMatrix(int[][] matrix){ this.matrix = matrix; }

    public int getEdgeAmount() {
        return edgeAmount;
    }
```

```
    public Point[] getEdges() {  //Возвращает массив пар индексов: "вершина старт" - "вершина конец"
        Point[] alledges = new Point[2 * getVertexList().length * (getVertexList().length - 1) / 2];
        for(int i = 0; i < (getVertexList().length * (getVertexList().length - 1) / 2); i++) {
            alledges[i] = new Point();
        }
        int iter = 0;
        for(int i = 0; i < vertexList.length; i++) {
            for(int j = i; j < vertexList.length; j++) {
                if(matrix[i][j] == 1){
                    alledges[iter].x = i;
                    alledges[iter].y = j;
                    iter++;
                }
            }
        }
        edgeAmount = iter;
        return alledges;
    }

    public void printBridgesToTextAre(JTextArea textArea){
        if(bridges.size() <= 0)
            textArea.append("Мостов не обнаружено" + "\n");
        for(int i = 0; i < bridges.size(); i++)
            textArea.append(bridges.elementAt(i).x + " " + bridges.elementAt(i).y + "\n");
    }

    public Vector<Point> getBridges(){
        return bridges;
    }

    public void setBridges(Vector<Point> bridges){
        this.bridges = bridges;
    }

    public void clearBridges(){
        bridges.clear();
    }
}
```

BridgeFinder.java

```
package source;
```

```
import java.awt.*;
import java.util.Vector;

import static org.junit.Assert.*;

public class BridgeFinderTest {

    @org.junit.Test
    public void FindBridgeNotNull() {
        BridgeFinder Bridge = new BridgeFinder();
        Vector<Point> bridges = new Vector();
        bridges.add(new Point(2, 3));
        bridges.add(new Point(1, 2));
        int[][] matrix = {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}};
        Object[] a = Bridge.FindBridgeT(matrix, 3);
        assertArrayEquals(bridges.toArray(), a);
    }

    @org.junit.Test
    public void FindBridgeNull() {
        BridgeFinder Bridge = new BridgeFinder();
        Vector<Point> bridges = new Vector();
        bridges.add(new Point(0, 0));
        int[][] matrix = new int[0][0];
        Object[] a = Bridge.FindBridgeT(matrix, 0);
        assertArrayEquals(bridges.toArray(), a);
    }

    @org.junit.Test
    public void FindBridgeNegative() {
        BridgeFinder Bridge = new BridgeFinder();
        Vector<Point> bridges = new Vector();
        int i = -5;
        bridges.add(new Point(i, i));
        int[][] matrix = new int[0][0];
        Object[] a = Bridge.FindBridgeT(matrix, i);
        assertArrayEquals(bridges.toArray(), a);
    }



    @org.junit.Test
    public void FindBridge100() {
        BridgeFinder Bridge = new BridgeFinder();
```

```
        Vector<Point> bridges = new Vector();
        bridges.add(new Point(1, 100));
        int[][] matrix = new int[100][100];
        for(int i = 0; i < 99; i++) {
            for (int j = 0; j < 99; j++) {
                matrix[i][j] = 1;
            }
        }
            matrix[0][99] = 1;
            for(int k = 1; k < 99; k++){
                matrix[k][99] = 0;
            }
            matrix[99][0] = 1;
        for(int k = 1; k < 100; k++){
            matrix[99][k] = 0;
        }


        Object[] a = Bridge.FindBridgeT(matrix, 100);
        assertArrayEquals(bridges.toArray(), a);
    }

    @org.junit.Test
    public void FindBridgeMoreThan100() {
        BridgeFinder Bridge = new BridgeFinder();
        Vector<Point> bridges = new Vector();
        int i = 110;
        bridges.add(new Point(i, i));
        int[][] matrix = new int[0][0];
        Object[] a = Bridge.FindBridgeT(matrix, i);
        assertArrayEquals(bridges.toArray(), a);
    }
}
```

GraphTest.java

```
package source;

import org.junit.Test;

import java.awt.*;
import java.util.Vector;

import static org.junit.Assert.*;
```

```
public class GraphTest {

    @Test
    public void getVertexListTest1() {
        int[][] matrix = {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}};
        int [] a = new int[] {0, 1, 2};
        BridgeFinder n = new BridgeFinder(matrix);
        assertArrayEquals(a, n.getVertexList());
    }

    @Test
    public void getVertexListTest2() {
        int[][] matrix = new int[0][0];
        int [] a = new int[0];
        BridgeFinder n = new BridgeFinder(matrix);
        assertArrayEquals(a, n.getVertexList());
    }

    @Test
    public void getEdgeAmountTest1() {
        int[][] matrix = new int[0][0];
        BridgeFinder n = new BridgeFinder(matrix);
        assertEquals(0, n.getEdgeAmount());
    }

    @Test
    public void getEdgeAmountTest2() {
        int[][] matrix = {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}};
        BridgeFinder n = new BridgeFinder(matrix);
        n.getEdges();
        assertEquals(2, n.getEdgeAmount());
    }

    @Test
    public void getMatrixTest1() {
        int[][] matrix = new int[100][100];
        matrix[0][0] = 0;
        matrix[0][1] = 1;
        matrix[0][2] = 0;
        matrix[1][0] = 1;
        matrix[1][1] = 0;
        matrix[1][2] = 1;
        matrix[2][0] = 0;
        matrix[2][1] = 1;
```

```
    matrix[2][2] = 0;
    BridgeFinder n = new BridgeFinder(matrix);
    assertArrayEquals(matrix, n.getMatrix());
}

@Test
public void getMatrixTest2() {
    int[][] matrix = new int[0][0];
    int[][] matrix1 = new int[100][100];
    BridgeFinder n = new BridgeFinder(matrix);
    assertArrayEquals(matrix1, n.getMatrix());
}

@Test
public void getEdgesTest1() {
    int[][] matrix = new int[0][0];
    int[][] matrix1 = new int[100][100];
    BridgeFinder n = new BridgeFinder(matrix);
    Point [] bridges = new Point[0];
    Object[] k = n.getEdges();
   // for(int i = 0; i < 4950; i++) {
   //     bridges[i] = new Point();
   // }
    assertArrayEquals(bridges,k);

}

@Test
public void getEdgesTest2() {
    int[][] matrix = {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}};
    BridgeFinder n = new BridgeFinder(matrix);
    Point [] bridges = new Point[6];
    for(int i = 0; i < 3; i++) {
        bridges[i] = new Point();
    }
    bridges[0].x = 0;
    bridges[0].y = 1;
    bridges[1].x = 1;
    bridges[1].y = 2;

    Object[] k = n.getEdges();
    assertArrayEquals(bridges,k);

}
```

```
    @Test
    public void getBridgesTest1() {
        int[][] matrix = {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}};
        BridgeFinder n = new BridgeFinder(matrix);
        Vector<Point> bridges = new Vector();
        bridges.add(new Point(2, 3));
        bridges.add(new Point(1, 2));
        assertArrayEquals(bridges.toArray(), n.FindBridgeT(matrix, 3));
    }

    @Test
    public void getBridgesTest2() {
        int[][] matrix = new int[0][0];
        BridgeFinder n = new BridgeFinder(matrix);
        Vector<Point> bridges = new Vector();
        bridges.add(new Point(0, 0));

        assertArrayEquals(bridges.toArray(), n.FindBridgeT(matrix, 0));
    }
}
```