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

ОТЧЕТ

по учебной практике

Тема: Алгоритм Форда-Фалкерсона

Студент гр. 7303 _____ Швец А.А.

Студент гр. 7303 _____ Мищенко М.А.

Студент гр. 7303 _____ Батурин И.

Руководитель _____ Ефремов М.А.

Санкт-Петербург
2019

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Швец А.А. группы 7303

Студент Мищенко М.А. группы 7303

Студент Батурин И. группы 7303

Тема практики: выполнение мини-проекта на языке java.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Форда-Фалкерсона.

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчета: 09.07.2019

Дата защиты отчета: 10.07.2019

Студент	_____	Швец А.А.
Студент	_____	Мищенко М.А.
Студент	_____	Батурин И.
Руководитель	_____	Ефремов М.А.

АННОТАЦИЯ

В ходе выполнения данной работы был реализован алгоритм Форда Фалкерсона. Практическое задание состоит из 4 частей. В первой части расписаны требования к программе в течении выполнения практического задания. Во второй описано распределение работы на всех студентов и план выполнения работы. В третьей части рассматриваются методы решения задачи и используемые структуры данных. В последнюю часть включено тестирование программы, а также сделано заключение, описаны исходные литературные источники, и прикреплен код проекта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Кратко описать цель и задачи практики, а также реализуемый алгоритм и его применение. Цель задачи – создание мини-проекта, который реализует алгоритм Форда-Фалкерсона. В задаче нужно использовать диаграмму всех использующихся классов, реализовать алгоритм и интерфейс к нему. Также нужно связать исходный код с интерфейсом, обработать исключительные ситуации и написать тесты к логике программы. К тому же, необходимо нарисовать диаграмму возможностей пользователя по использованию данной программы. В конце требуется написать отчет по выполненной работе.

SUMMARY

In the course of this work, the Ford Fulkerson 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 student for work and the work plan. The third part deals with the methods of implementation of the task and the data structures used. The last part includes testing of the program, as well as a conclusion, describes the source literature, and attached the project code.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. На рисунке 1 изображена use case диаграмма. На старте пользователь может выбрать 5 опций: создать граф самому, загрузить его из файла, создать произвольный граф и показать инструкцию применения и about. Для создания графа вручную пользователь может добавлять или удалять вершины и ребра. После создания графа, следует запустить алгоритм соответствующей кнопкой start и посмотреть результат на визуализации.

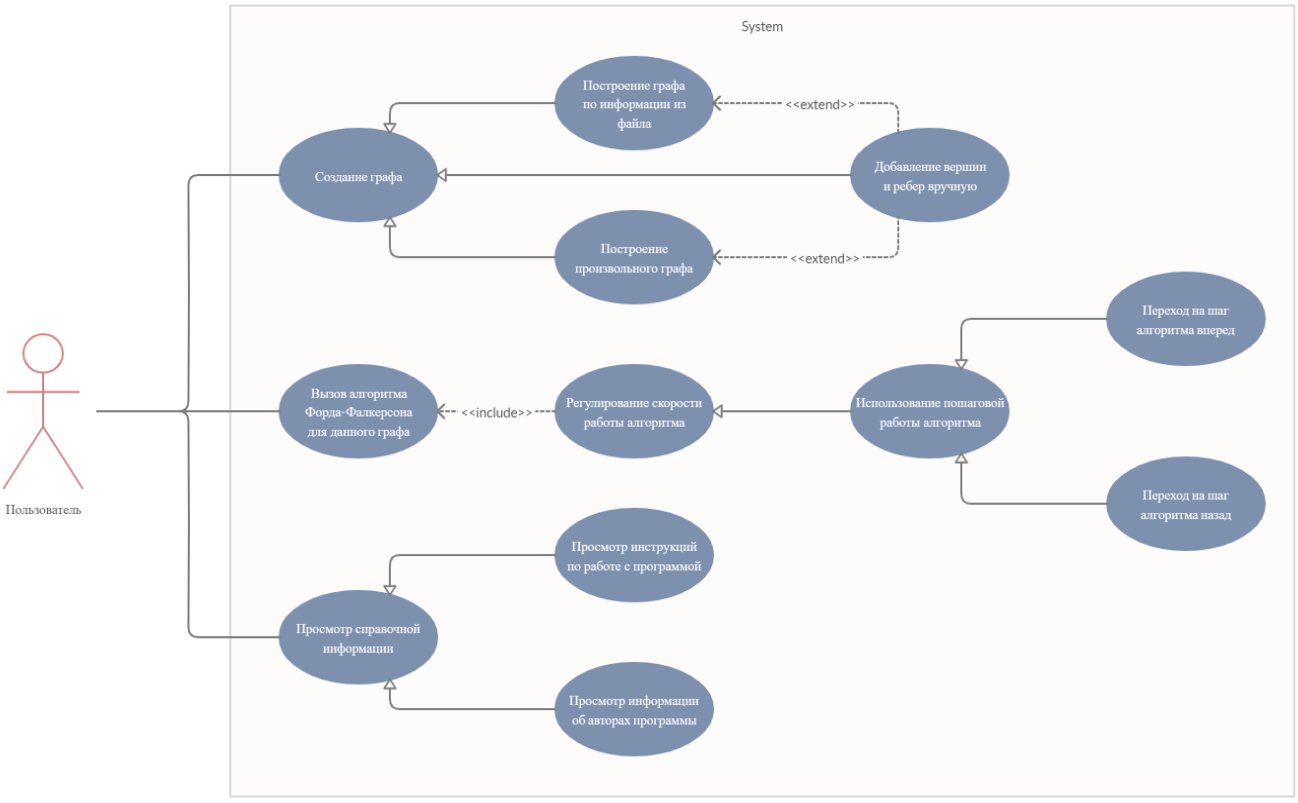

Рисунок 1 – use-case диаграмма

1.1.2. Отображение графа.

Ориентированность ребер показывается с помощью стрелки. На каждой итерации алгоритма Форда-Фалкерсона каждый

найденный новый путь в остаточной сети графа окрашивается в желтый цвет. Количество вершин, которые можно вместить граф - от 3 до 15, а количество ребер от 1 до 182. Именами для вершин могут быть любые последовательности 1 или более символов, включая !@#$%.

1.2. Уточнение требований после сдачи прототипа.

1.2.1. Заполнить файл с темой задания;

1.2.2. Сделать диаграмму классов для интерфейса и добавить логи работы программы (вывод в текстовое поле);

1.2.3. Исправить недочеты, связанные с перетаскиванием вершин.

1.3. Уточнение требований после сдачи 1-ой версии.

1.3.1. Перенести кнопку start вниз;

1.3.2. Защитить поле, отображающее индекс текущего шага, от записи;

1.3.3. Защитить поле, в которое выводятся логи, от записи;

1.3.4. Сделать недоступными для нажатия кнопки и очистить поле для вывода индекса шага алгоритма, если графа нет (очищен);

1.3.5. Добавить кнопки, отвечающие за переход на последнюю и первую итерацию работы алгоритма;

1.3.6. Добавить текстуры (иконки) для кнопок;

1.3.7. Установить maven в проект;

1.3.8. Сделать unit-тест для модели с проверкой существования экземпляров классов.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

до 04.07: use case диаграмма, интерфейс без реализации логики, прототип с демонстрацией функциональности программы.

до 06.07: прототип с демонстрацией функциональности программы.

до 08.07: 1 версия с обработкой ошибок и всплывающим контекстным меню, диаграмма классов.

до 10.07: 2 версия, исправление недочетов 1 версии.

2.2. Распределение ролей в бригаде

Мищенко М.А.: GUI, базовые классы, соединение интерфейса и логики программы.

Швец А.А.: диаграмма классов, use-case диаграмма, контроллер, отчет.

Батурин И.: юнит-тестирование, Maven и реализация алгоритма Форда-Фалкерсона.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Использованные структуры данных

На рисунке 2 показана иерархия классов, касающихся работы со структурой данных граф. Класс Graph – основной класс, который содержит списки вершин и ребер и методы работы с ними. Пакет классов Controller служит мостом между интерфейсом и логикой программы. Это запуск основной функции, переопределенные методы отрисовки графа и логика загрузки графа из файла, вызов алгоритма Форда-Фалкерсона, определенного в отдельном классе. Классы Node и Edge реализуют свойства вершин и ребер соответственно (имена, поток, цвет).

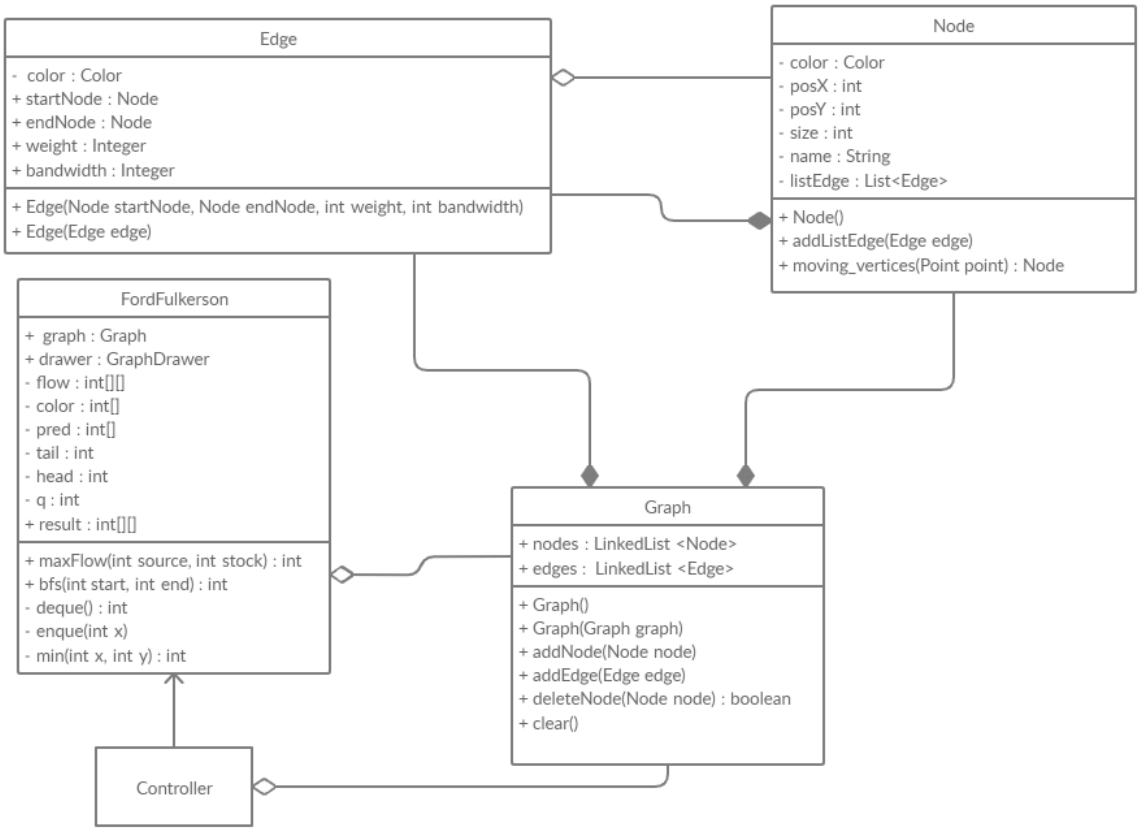

Рисунок 2 – классы логики программы.

3.2. Основные методы

1) public Graph() – инициализирует массивы для хранения вершин и ребер
2) public Edge(Node startNode, Node endNode, int weight, int bandwidth) – конструктор для ребра
3) public int getWeight() – получение текущего потока данного ребра
4) public Node(String name, int posX, int posY) – конструктор для вершины
6) int bfs(int start, int end) – поиск в глубину для путей в графе
7) public int maxFlow (int source, int srock) – алгоритм поиск потока

3.3. Использование интерфейса

На рисунке 3 изображен интерфейс программы. Он состоит из 3 главных частей. В левой части окна располагается текстовое поле, в которое выводятся логи по ходу программы. В правой части окна выводится сам граф. В нижней же части окна расположены кнопки для старта алгоритма, перехода по шагам алгоритма и текстовое поле, содержащее номер текущего шага.

В контекстном меню панели пользователь может:

1) Создавать произвольный граф
2) Загрузить граф из файла
3) Очистить (удалить) текущий граф
4) Просмотреть информацию о задании
5) Просмотреть информацию об авторах программы

Двойное нажатие по свободному месту на поле приведет к появлению диалогового окна создания вершины, в котором нужно ввести ее название.

Два последовательных двойных нажатия по двум различным вершинам приведут к появлению диалогового окна создания ребра, в котором нужно ввести пропускную способность и текущий поток (последний по умолчанию 0).

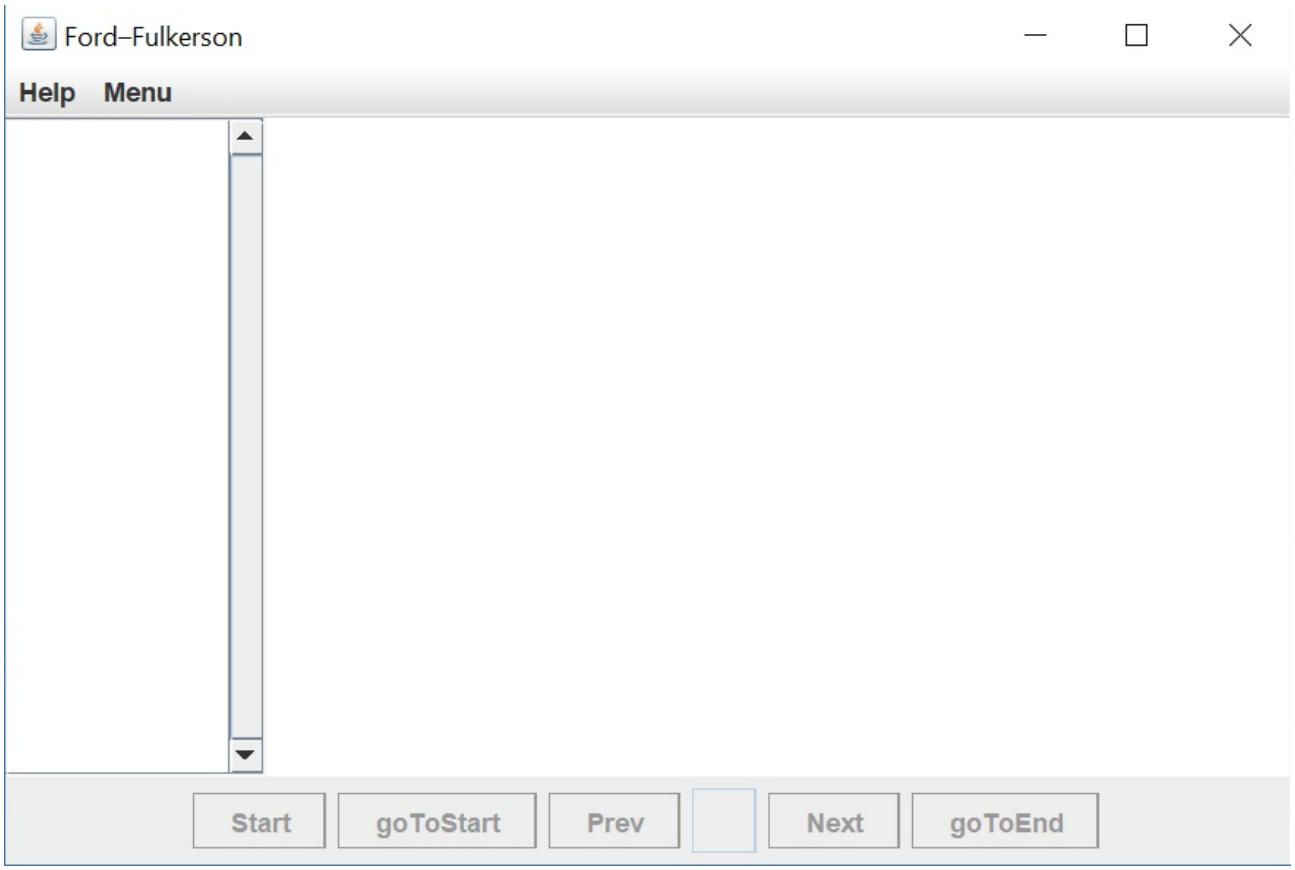

Рисунок 3 – интерфейс программы

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Был создан класс GraphTest, в котором хранятся тесты для проверки на правильность добавления/удаления рёбер/вершин. Были созданы 4 переменные с модификатором доступа private для облегченного тестирования и для того, чтобы избежать дублирование кода.

- В методе addNode проверяется добавлена ли в граф вершина. Это делается посредством добавления переменной типа Node в ArrayList.
- В методе addEdge добавляются две вершины на случай, если список вершин будет пусть. Затем проверяется было ли добавлено ребро в граф функцией Assert.Equals().
- В методе deleteNode ситуация аналогична добавлению вершины, только в этом методе проверяется можем ли мы удалить элемент из списка вершин.
- В методе deleteEdge добавляется вершина в список уже существующих вершин, записывается в переменную число – количество рёбер после добавления текущего ребра. Затем текущее ребро удаляется и вызывается функция Assert.Equals() для проверки, удалилось ли ребро и списка ребёр.

4.2. Тестирование кода алгоритма

Был создан класс FordFulkersonTest для проверки на корректность работы алгоритма Форда-Фалкерсона. Было написано два теста, которые проверяли правильность максимального потока в сети.

- В методах algorithmFirstTest и algorithmSecondTest в строке был записан путь до определенного файла .txt. Строка типа String передавалась на вход функции readFromFile, которая считывала данные с файла (матрица смежности) и записывала в двумерный массив. Затем функцией

Assert.Equals() проверялось правильный ли максимальный поток находит алгоритм.

ЗАКЛЮЧЕНИЕ

Таким образом был реализован алгоритм Форда-Фалкерсона. При этом были сделаны диаграммы классов и use-case, графический интерефейс, разделение проекта на файлы, обработка ошибок и исключений, класс тестирования и сборка при помощи maven. Реализованный проект полностью соответствует требованиям, поставленным в задаче. План работы описан в разделе 2.1, и подробную реализацию можно увидеть в разделах 3 и 4.

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

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

public class DialogView {

}

class InitNode extends JDialog {
    public InitNode(GraphDrawer graphDrawer, MouseEvent e) {
        this.setVisible(true);
        add(new JLabel("<html><h1><i>Task</i></h1><hr>" + "InitNode"),
                BorderLayout.CENTER);
        this.setLocation(e.getX(), e.getY());
        JTextField textField = new JTextField();
        add(textField);
        // При активизации кнопки OK диалогове окно закрывается.
        JButton ok = new JButton("ok");
        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                setVisible(false);
                if (textField.getText().length() > 0) {
                    graphDrawer.graph.addNode(new Node(textField.getText(), e.getX(),
e.getY()));
                    graphDrawer.repaint();
                }
            }
        });

        // Кнопка OK помещается в нижнюю часть окна.

        JPanel panel = new JPanel();
        panel.add(ok);
        add(panel, BorderLayout.SOUTH);
        setSize(100, 100);
    }
}
```

```
class InitEdge extends JDialog {
    private Node start = null;
    private Node end = null;
    private GraphDrawer drawer;
    private Color colorStart;
    private Color colorEnd;
    private Color color = Color.green;

    public InitEdge(GraphDrawer drawer) {
        this.drawer = drawer;
    }

    private boolean checkEdge(Node node1, Node node2) {
        for (Edge edge1 : node1.getListEdge()) {
            for (Edge edge2 : node2.getListEdge()) {
                if (edge1 == edge2)
                    return false;
            }
        }
        return true;
    }

    public void addNodetoEdge(Node node) {
        if (start == null) {
            start = node;
            colorStart = start.getColor();
            node.changeColor(color);
            drawer.repaint();
        } else if (start != node) {
            if (!checkEdge(start, node)) {
                JOptionPane.showMessageDialog(drawer,"There is already an edge between
the vertices!!!!!");
                return;
            }
            end = node;
            colorEnd = end.getColor();
            end.changeColor(color);
            drawer.repaint();
            paintEdge();
        }
```

```
    }

    public void paintEdge() {
        this.setVisible(true);
        add(new JLabel("<html><h1><i>Task</i></h1><hr>" + "InitEdge"),
                BorderLayout.CENTER);
        this.setLocation((start.getPosX() + end.getPosX()) / 2,
                (start.getPosY() + end.getPosY()) / 2);
        JPanel panelText = new JPanel();
        JTextField textField = new JTextField(2);
        JTextField textField1 = new JTextField(2);

        panelText.add(textField);
        panelText.add(textField1);
        add(panelText, BorderLayout.CENTER);
        // При активизации кнопки OK диалогове окно закрывается.
        JButton ok = new JButton("ok");
        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                setVisible(false);
                if (textField.getText().length() > 0 && textField1.getText().length() >
0) {
                    drawer.graph.addEdge(new Edge(start, end,
Integer.parseInt(textField.getText()), Integer.parseInt(textField1.getText())));

                }
                start.changeColor(colorStart);
                end.changeColor(colorEnd);
                drawer.repaint();
                textField.setText("");
                textField1.setText("");
                start = null;
                end = null;
            }
        });

        // Кнопка OK помещается в нижнюю часть окна.

        JPanel panel = new JPanel();
        panel.add(ok);
```

```
        add(panel, BorderLayout.SOUTH);
        setSize(100, 100);
    }
}

public class EdgeDrawer extends JPanel {

    public Graph graph;

    /**
     * saves reference on graph
     */
    public EdgeDrawer(Graph _graph) {
        graph = _graph;
    }

    public void setGraph(Graph _graph){
        graph = _graph;
    }

    public int[] Turn (int corn,int x,int y,int cenX,int cenY){
        int  []cord =new int[2];
        cord[0]=(int)((x-cenX)*Math.cos((corn*Math.PI)/180)-(y-
cenY)*Math.sin((corn*Math.PI)/180))+cenX;
        cord[1]=(int)((x-cenX)*Math.sin((corn*Math.PI)/180)+(y-
cenY)*Math.cos((corn*Math.PI)/180))+cenY;
        return cord;
    }
    public Object[] getCord(int x, int y, Node start){
        LinkedList<Integer> list=new LinkedList<Integer>();
        int vecX = x-(start.getPosX()+(start.getDiameter() / 2));
        int vecY =y-(start.getPosY()+(start.getDiameter() /2)) ;
        double vecSize = Math.sqrt((double) (Math.pow(vecX, 2) + Math.pow(vecY, 2)));
        int posX=(int)(vecX - vecX / vecSize*start.getDiameter() / 2 +start.getPosX() +
start.getDiameter() / 2);
        int posY=(int)(vecY - vecY / vecSize*start.getDiameter() / 2 +start.getPosY()+
start.getDiameter() / 2);
        list.add(Turn(30,posX,posY,x,y)[0]);
        list.add(Turn(30,posX,posY,x,y)[1]);
        list.add(Turn(-30,posX,posY,x,y)[0]);
        list.add(Turn(-30,posX,posY,x,y)[1]);
```

```
        return list.toArray();
    }
    /**
     * draws all graph edges
     */
    @Override
    public void paint(Graphics g) {
        for (Edge edge : graph.edges) {
            g.setColor(edge.getColor());
            g.setFont(new Font("Arial", Font.BOLD, edge.getStartNode().getDiameter() /
3));
            g.drawLine(edge.getStartNode().getPosX() +
edge.getStartNode().getDiameter() / 2, edge.getStartNode().getPosY() +
edge.getStartNode().getDiameter() / 2,
                    edge.getEndNode().getPosX() + edge.getEndNode().getDiameter() / 2,
edge.getEndNode().getPosY() + edge.getEndNode().getDiameter() / 2);
            int vecX = (edge.getEndNode().getPosX() + (edge.getEndNode().getDiameter()
/ 2))-(edge.getStartNode().getPosX() + (edge.getStartNode().getDiameter() / 2));
            int vecY =(edge.getEndNode().getPosY() + (edge.getEndNode().getDiameter() /
2))-(edge.getStartNode().getPosY() + (edge.getStartNode().getDiameter() / 2));
            double vecSize = Math.sqrt((double) (Math.pow(vecX, 2) + Math.pow(vecY,
2)));
            g.setColor(Color.BLACK);
            int posX=(int)(vecX - vecX / vecSize*edge.getStartNode().getDiameter() / 2
+edge.getStartNode().getPosX() + edge.getStartNode().getDiameter() / 2);
            int posY=(int) (vecY - vecY / vecSize *edge.getStartNode().getDiameter() /
2+ edge.getStartNode().getPosY() + edge.getStartNode().getDiameter() / 2);

            g.fillPolygon(new int[]{posX,
(int)(getCord(posX,posY,edge.getStartNode())[0]),(int)(getCord(posX,posY,edge.getStartN
ode())[2])},new int[]{posY,
(int)(getCord(posX,posY,edge.getStartNode())[1]),(int)(getCord(posX,posY,edge.getStartN
ode())[3])},3);
            g.setFont(new Font("Arial", Font.BOLD, edge.getStartNode().getDiameter() /
3));
            g.setColor(Color.RED);
            g.drawString(edge.getWeight().toString() + "|" +
edge.getBandwidth().toString(),
```

```
                    (edge.getEndNode().getPosX() + edge.getStartNode().getPosX()) / 2 -
5 - edge.getWeight().toString().length() + edge.getBandwidth().toString().length(),
                    (edge.getEndNode().getPosY() + edge.getStartNode().getPosY()) / 2);

        }
    }
}
public class GraphDrawer extends JPanel {
    public Graph graph;
    public LinkedList <Graph> graphList; // список состояний графа на каждой итерации
алгоритма

    public int iteration;

    public NodeDrawer nodeDrawer;
    public EdgeDrawer edgeDrawer;
    Node node = null;
    Graphicsview frame;
    private final InitEdge initEdge = new InitEdge(this);

    public GraphDrawer(Graph _graph, Graphicsview frame) {
        graph = _graph;
        graphList = new LinkedList<>();
        iteration = 0;
        this.frame=frame;
        nodeDrawer = new NodeDrawer(graph);
        edgeDrawer = new EdgeDrawer(graph);
        addMouseMotionListener(new MyMove());
        addMouseListener(new MyMouse());
        addMouseWheelListener(new MyMouse());

    }

    public void setGraph(Graph _graph){
        graph = _graph;
        nodeDrawer.setGraph(_graph);
        edgeDrawer.setGraph(_graph);
    }

    @Override
    public void paint(Graphics g) {
```

```
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, this.getWidth(), this.getHeight());
    edgeDrawer.paint(g);
    nodeDrawer.paint(g);
}

public Node moving_vertices(Point point) {
    for (Node elm : graph.getNodes()) {
        if (elm.moving_vertices(point) != null)
            return elm;
    }
    return null;
}

public void CallWindowNode(MouseEvent e) {
    JDialog dialog = new InitNode(this, e);
}

private class MyMouse extends MouseAdapter {
    @Override
    public void mousePressed(MouseEvent e) {
        if (e.getButton() == MouseEvent.BUTTON1)
            node = moving_vertices(e.getPoint());
    }

    @Override
    public void mouseClicked(MouseEvent e) {

        if (e.getButton() == MouseEvent.BUTTON3) {
            node = moving_vertices(e.getPoint());
            if (node == null) {
                CallWindowNode(e);
            }
            if (e.getClickCount() > 1 && node != null) {
                initEdge.addNodetoEdge(node);
                if (graph.edges.size()!=0)
                    frame.startButton.setEnabled(true);
            }
        }
```

```
        }


        @Override
        public void mouseWheelMoved(MouseWheelEvent e) {
            //Zoom in
            if (e.getWheelRotation() > 0) {
                for (Node node : graph.getNodes()) {
                    int vecX = node.getPosX() - e.getX();
                    int vecY = node.getPosY() - e.getY();
                    double vecSize = Math.sqrt((double) (Math.pow(vecX, 2) +
Math.pow(vecY, 2)));
                    node.setPosX((int) (vecX + vecX / vecSize * 20 + e.getX()));
                    node.setPosY((int) (vecY + vecY / vecSize * 20 + e.getY()));
                    node.setDiameter((int) (node.getDiameter() + 1));

                }
                repaint();

            }
            //Zoom out
            if (e.getWheelRotation() < 0) {
                for (Node node : graph.getNodes()) {
                    if (node.getDiameter() > 5) {
                        int vecX = node.getPosX() - e.getX();
                        int vecY = node.getPosY() - e.getY();
                        double vecSize = Math.sqrt((double) (Math.pow(vecX, 2) +
Math.pow(vecY, 2)));
                        node.setPosX((int) (vecX - vecX / vecSize * 20 + e.getX()));
                        node.setPosY((int) (vecY - vecY / vecSize * 20 + e.getY()));
                        double vecSize1 = Math.sqrt((double) (Math.pow(vecX, 2) +
Math.pow(vecY, 2)));
                        node.setDiameter((int) (node.getDiameter() - 1));
                    }
                }
                repaint();
            }
        }
    }

    private class MyMove implements MouseMotionListener {
```

```
        public void mouseDragged(MouseEvent e) {
            if (node != null) {
                node.setPosY(e.getY());
                node.setPosX(e.getX());
                repaint();
            }
        }

        public void mouseMoved(MouseEvent e) {

        }

    }
}public class LogString {
    JTextArea textField;
    public ArrayList<String> list = new ArrayList<String>();

    public LogString(JTextArea textField) {
        this.textField = textField;
    }

    public void addString(String str) {
        list.add(str);
    }

    public void chengeTextLog(int count) {
        String log = "";
        for (int i = 0; i <=count; i++) {
         log+=list.get(i)+"\n";
        }
        textField.setText("");
        textField.setText(log);
    }


}

public class LogString {
    JTextArea textField;
    public ArrayList<String> list = new ArrayList<String>();
```

```
    public LogString(JTextArea textField) {
        this.textField = textField;
    }

    public void addString(String str) {
        list.add(str);
    }

    public void chengeTextLog(int count) {
        String log = "";
        for (int i = 0; i <=count; i++) {
         log+=list.get(i)+"\n";
        }
        textField.setText("");
        textField.setText(log);
    }
}

public class NodeDrawer extends JPanel {

    public Graph graph;
    /**
     * saves reference on graph
     */
    public NodeDrawer(Graph _graph){
        graph = _graph;
    }

    public void setGraph(Graph _graph){
        graph = _graph;
    }

    /**
     * draws all graph nodes
     */
    @Override
    public void paint(Graphics g) {
        for (Node node : graph.nodes) {

            g.setColor(node.getColor());
```

```
            g.fillOval(node.getPosX(), node.getPosY(), node.getDiameter(),
node.getDiameter());
            g.setFont(new Font("Arial", Font.BOLD, node.getDiameter()/3));
            g.setColor(Color.red);
            g.drawString(node.getName(), node.getPosX()+node.getDiameter()/4,
node.getPosY() +node.getDiameter()/2+3);
        }
    }
}

public class Edge {
    private Color color = Color.BLACK;
    public Node startNode;
    public Node endNode;
    public Integer weight = 0;
    public Integer bandwidth = 0;

    public Edge(Node startNode, Node endNode, int weight, int bandwidth) {
        this.startNode = startNode;
        this.endNode = endNode;
        this.weight = weight;
        this.bandwidth = bandwidth;
        startNode.addListEdge(this);
        endNode.addListEdge(this);
    }

    public Edge(Edge edge) {
        this.startNode = edge.startNode;
        this.endNode = edge.endNode;
        this.weight = edge.weight;
        this.bandwidth = edge.bandwidth;
        startNode.addListEdge(this);
        endNode.addListEdge(this);
    }


    public Color getColor() {
        return  this.color;
    }

    public void changeColor(Color color) {
```

```
        this.color = color;
    }

    public Node getStartNode() {
        return startNode;
    }

    public Node getEndNode() {
        return endNode;
    }

    public Integer getWeight() {
        return weight;
    }

    public Integer getBandwidth() {
        return bandwidth;
    }

}

public class FordFulkerson {
   public Graph graph;
   public GraphDrawer drawer;
   Graphicsview frame;
    final int WHITE = 0;
    final int GREY = 1;
    final int BLACK = 2;


    private int[][] flow; // Матрица потока
    private int[] color;      // Цвета для вершин
    private int[] pred;       // Массив пути
    private int head, tail;  // Начало, Конец
    private int[] q;
    public int [][] result;

    public FordFulkerson(Graph graph, GraphDrawer drawer, Graphicsview frame) {
        this.graph = graph;
        this.drawer = drawer;
        this.frame = frame;
```

```
    }

    public  FordFulkerson(int[][] array, Graph graph, GraphDrawer graphDrawer) {
        this.result = array;
        this.graph = graph;
        this.drawer = graphDrawer;
    }

    private int min(int x, int y)
    {
        if (x < y)
            return x;
        else
            return y;
    }

    private void enque(int x)
    {
        q[tail] = x;     // записать x в хвост
        tail++;          // хвостом становиться следующий элемент
        color[x] = GREY; // Цвет серый (из алгоритма поиска)
    }

    private int deque()
    {
        int x = q[head];  // Записать в x значение головы
        head++;           // Соответственно номер начала очереди увеличивается
        color[x] = BLACK; // Вершина x - отмечается как прочитанная
        return x;         // Возвращается номер x прочитанной вершины
    }

    private int bfs(int start, int end)
    {
        for(int u = 0; u < result.length; u++ ) // Сначала отмечаем все вершины не
пройденными
            color[u] = WHITE;

        head = 0;   // Начало очереди 0
        tail = 0;   // Хвост 0
        enque(start); // Вступили на первую вершину
        int u;
```

```
        pred[start] = -1;   // Специальная метка для начала пути
        while(head != tail)  // Пока хвост не совпадёт с головой
        {
            u = deque();
            for(int v = 0; v < result.length; v++ ) // Смотрим смежные вершины
            {
                // Если не пройдена и не заполнена
                if(color[v] == WHITE && (result[u][v] - flow[u][v]) > 0) {
                    enque(v);  // Вступаем на вершину v
                    pred[v] = u; // Путь обновляем
                }
            }
        }
        if(color[end] == BLACK) // Если конечная вершина, дошли - возвращаем 0
            return 0;
        else return 1;
    }

    /**
     * возвращает список измененных ребер
     */
    LinkedList<Edge> changedEdges(Graph prev){

        LinkedList<Edge> list = new LinkedList<>(); // список измененных на данной
итерации ребер
        for (int i = 0; i < prev.edges.size(); i++) {
            if (prev.edges.get(i).weight != this.graph.edges.get(i).weight){ // если
веса не совпали
                list.addLast(prev.edges.get(i));
            }
        }

        return list;
    }
   void  setLog(LinkedList<Edge> list){
        if (list.size() != 0) {
            String log = "";
            for (Edge edge : list) {
                log += edge.getStartNode().getName() + "-(" + edge.getWeight() + "|" +
edge.getBandwidth() + ")->";
            }
```

```
            log += list.getLast().getEndNode().getName();
            frame.logString.addString(log);
        }
    }
     void colorEdges(LinkedList<Edge> list, Color color){
        for (Edge edge : list) {
            edge.changeColor(color);
        }
     }

    public int maxFlow(int[][] result, int source, int stock)
     {
        flow = new int[result.length][result.length]; // Матрица потока
        color = new int[result.length];      // Цвета для вершин
        pred = new int[result.length];
        q = new int[result.length];
        int max_flow = 0;            // Изначально нулевой
        for(int i = 0; i < result.length; i++ )  // Зануляем матрицу потока
        {
            for(int j = 0; j < result.length; j++)
                flow[i][j] = 0;
        }
        while (bfs(source,stock) == 0)             // Пока существует путь
        {
            int delta = 10000;
            for(int u = result.length - 1; pred[u] >= 0; u = pred[u]) // Найти
минимальный поток в сети
            {
                delta = min(delta, ( result[pred[u]][u] - flow[pred[u]][u] ) );
            }
            for(int u = result.length - 1; pred[u] >= 0; u = pred[u]) // По алгоритму
Форда-Фалкерсона
            {
                flow[pred[u]][u] += delta;
                flow[u][pred[u]] -= delta;
            }
            max_flow += delta;                        // Повышаем максимальный поток

            Graph prev = new Graph(this.graph); // запишем в граф предыдущее состояние
            matrixToGraph(); // обновим состояние графа
```

```
            colorEdges(changedEdges(prev), Color.ORANGE); // красим изменившиеся ребра
в красный
            setLog(changedEdges(prev));
            drawer.graphList.add(prev); // сохраним в список состояний
            drawer.iteration++;
            drawer.setGraph(drawer.graphList.getLast());
            drawer.repaint();
        }

        drawer.iteration--; // так как это количество элементов списка, индекс
последнего на 1 меньше

        return max_flow;
    }

    public void graphToMatrix() {
        result = new int[graph.nodes.size()][graph.nodes.size()];
        int index_edges = 0, index_nodes = 0, index_i, index_j, vertex_counter = 0;
        while (index_edges != graph.edges.size()) {
            index_i = 0;
            index_j = 0;
            while (index_nodes != graph.nodes.size()) {
                if (graph.nodes.get(index_nodes) ==
graph.edges.get(index_edges).startNode && index_i == 0) {
                    index_i = index_nodes;
                    vertex_counter++;
                }
                if (graph.nodes.get(index_nodes) ==
graph.edges.get(index_edges).endNode && index_j == 0) {
                    index_j = index_nodes;
                    vertex_counter++;
                }
                if (vertex_counter == 2){
                    result[index_i][index_j] = graph.edges.get(index_edges).bandwidth;
                    break;
                }
                index_nodes++;
            }
            vertex_counter = 0;
            index_nodes = 0;
            index_edges++;
```

```
        }
    }

    public void matrixToGraph() {
        int index_edges = 0, index_nodes = 0, index_i, index_j, vertex_counter = 0;
        while (index_edges != graph.edges.size()) {
            index_i = 0;
            index_j = 0;
            while (index_nodes != graph.nodes.size()) {
                if (graph.nodes.get(index_nodes) ==
graph.edges.get(index_edges).startNode && index_i == 0) {
                    index_i = index_nodes;
                    vertex_counter++;
                }
                if (graph.nodes.get(index_nodes) ==
graph.edges.get(index_edges).endNode && index_j == 0) {
                    index_j = index_nodes;
                    vertex_counter++;
                }
                if (vertex_counter == 2){
                    graph.edges.get(index_edges).weight = flow[index_i][index_j];
                    break;
                }
                index_nodes++;
            }
            vertex_counter = 0;
            index_nodes = 0;
            index_edges++;
        }
    }
}

public class Graph {

    public LinkedList <Node> nodes;
    public LinkedList <Edge> edges;

    public Graph(){
        nodes = new LinkedList<Node>();
        edges = new LinkedList<Edge>();
    }
```

```
    public Graph(Graph graph){
        this();
        for (Node node : graph.nodes) {
            this.nodes.add(node);
        }
        for (Edge edge : graph.edges) {
            this.edges.add(new Edge(edge));
        }
    }

    public void addNode(Node _node){
        nodes.add(_node);
    }

    public void addEdge(Edge _edge){
        edges.add(_edge);
    }

    public boolean deleteNode(Node node){
        for (Edge edge:node.getListEdge()){
            deleteEdge(edge);
        }
        return nodes.remove(node);
    }

    public boolean deleteEdge(Edge edge) {
        return edges.remove(edge);
    }
    public  LinkedList<Node> getNodes(){return nodes;}

    public void Clear(){
        nodes.clear();
        edges.clear();
    }
}

public class Node {
    private Color color = Color.PINK;
    private int posX;
    private int posY;
```

```
private static int size = 30;
private String name;
private List<Edge> listEdge = new ArrayList<Edge>();

public Node(String name, int posX, int posY) {
    JPanel.getDefaultLocale();
    this.name = name;
    this.posX = posX;
    this.posY = posY;
}

public int getDiameter() {
    return size;
}
public void setDiameter(int size) {
    this.size=size;
}

public String getName() {
    return name;
}

public int getPosX() {
    return posX;
}

public int getPosY() {
    return posY;
}
public void setPosX(int x) {
    posX=x;
}

public void setPosY(int y) {
    posY=y;
}

public void addListEdge(Edge edge) {
    listEdge.add(edge);
}
```

```
    public List<Edge> getListEdge() {
        return listEdge;
    }

    public void changeColor(Color color) {
        this.color = color;
    }
    public Color getColor() {
        return this.color;
    }

    public Node moving_vertices(Point point) {
        int center_of_the_circleX = posX + size/2;
        int center_of_the_circleY = posY + size/2;
        if (Math.sqrt(Math.pow(center_of_the_circleX - point.x, 2) +
Math.pow(center_of_the_circleY - point.y, 2)) <= size/2) return this;
        else return null;
    }

}

public abstract class ButtonCommand implements ActionListener {
    public abstract void actionPerformed(ActionEvent e);
}

class StartCommand extends ButtonCommand {
    FordFulkerson algorithm;
    Graphicsview frame;

    public StartCommand(Graph graph, GraphDrawer drawer, Graphicsview frame) {
        algorithm = new FordFulkerson(graph, drawer, frame);
        this.frame = frame;

    }

    public void actionPerformed(ActionEvent e) {
        frame.goToStart.setEnabled(true);
        frame.goToEnd.setEnabled(true);
        frame.nextButton.setEnabled(true);
        frame.prevButton.setEnabled(true);
        frame.startButton.setEnabled(false);
```

```
        algorithm.graphToMatrix();
        int max_flow = algorithm.maxFlow(algorithm.result, 0, algorithm.result.length -
1);
        System.out.println(max_flow);
        algorithm.matrixToGraph();
        algorithm.drawer.repaint();
        algorithm.drawer.updateUI();
        frame.counter.setText("" + frame.drawer.graphList.size() + "/" +
frame.drawer.graphList.size());
        frame.logString.chengeTextLog(frame.drawer.graphList.size()-1);
    }
}


class PrevCommand extends ButtonCommand {
    GraphDrawer drawer;
    Graphicsview frame;

    public PrevCommand(GraphDrawer drawer, Graphicsview frame) {
        this.drawer = drawer;
        this.frame = frame;
    }


    public void actionPerformed(ActionEvent e) {
        if (!drawer.graphList.isEmpty()) {
            if (drawer.iteration - 1 >= 0) // меняем индекс состояния в списке
                --drawer.iteration;

            drawer.setGraph(drawer.graphList.get(drawer.iteration));
            drawer.repaint();

            frame.counter.setText("" + (drawer.iteration + 1) + "/" +
drawer.graphList.size());
            frame.logString.chengeTextLog(drawer.iteration);


            drawer.setGraph(drawer.graphList.get(drawer.iteration));
            drawer.repaint();
        }
    }
```

```
}


class NextCommand extends ButtonCommand {

    GraphDrawer drawer;
    Graphicsview frame;

    public NextCommand(GraphDrawer drawer, Graphicsview frame) {
        this.drawer = drawer;
        this.frame = frame;
    }



    public void actionPerformed(ActionEvent e) {
        if (!drawer.graphList.isEmpty()) {
            if (drawer.iteration + 1 < drawer.graphList.size()) // меняем индекс
состояния в списке
                ++drawer.iteration;

            drawer.setGraph(drawer.graphList.get(drawer.iteration));
            drawer.repaint();

            frame.counter.setText("" + (drawer.iteration + 1) + "/" +
drawer.graphList.size());
            frame.logString.chengeTextLog(drawer.iteration);

        }
    }
}

class goToStartCommand extends ButtonCommand {

    GraphDrawer drawer;
    Graphicsview frame;

    public goToStartCommand(GraphDrawer drawer, Graphicsview frame) {
        this.drawer = drawer;
        this.frame = frame;
    }
```

```
    public void actionPerformed(ActionEvent e) {
        if (!drawer.graphList.isEmpty()) {
            drawer.setGraph(drawer.graphList.getFirst());

            drawer.iteration = 0;
            drawer.repaint();


            frame.counter.setText("" + 1 + "/" + drawer.graphList.size());
            frame.logString.chengeTextLog(0);

            drawer.repaint();
        }
    }
}

class goToEndCommand extends ButtonCommand {

    GraphDrawer drawer;
    Graphicsview frame;

    public goToEndCommand(GraphDrawer drawer, Graphicsview frame) {
        this.drawer = drawer;
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (!drawer.graphList.isEmpty()) {

            drawer.setGraph(drawer.graphList.getLast());
            drawer.iteration = drawer.graphList.size() - 1;
            drawer.repaint();

            frame.counter.setText("" + drawer.graphList.size() + "/" +
drawer.graphList.size());
            frame.logString.chengeTextLog(drawer.graphList.size()-1);
        }
    }
}
```

```
public class Graphicsview extends JFrame {
    private JMenuBar menuBar = new JMenuBar();
    public JTextArea jTextArea;
    public JButton prevButton = new JButton("Prev");
    public JButton nextButton = new JButton("Next");
    public  JButton startButton = new JButton("Start");
    public JButton goToStart = new JButton("goToStart");
    public JButton goToEnd = new JButton("goToEnd");
    public JTextField counter = new JTextField();
    private JSlider jSlider;
    private List<Node> listNode = new ArrayList<Node>();
    private Container contaner;
    private int n;
    private final double pi = 3.14159265359;
    public int[][] result;
    public Graph graph = new Graph();
    public GraphDrawer drawer = new GraphDrawer(graph,this);
    public LogString logString;


    public Graphicsview() {
        super("Ford–Fulkerson");
        setSize(600, 400);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        contaner = this.getContentPane();
        contaner.setLayout(new BorderLayout());
        /**
         * menu
         */
        JMenuBar menuBar = new JMenuBar();
        menuBar.add(createHelpMenu(this));
        menuBar.add(createMenu(this));
        setJMenuBar(menuBar);
        /**
         * textArea
         */
        JPanel panelText = new JPanel(new BorderLayout());
        jTextArea = new JTextArea();
        jTextArea.setSize(200,100);
        jTextArea.setLineWrap(true);
        jTextArea.setEditable(false);
```

```
        logString=new LogString(jTextArea);
        JScrollPane scroll = new JScrollPane(jTextArea,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS, ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        panelText.add(scroll);
        contaner.add(panelText, BorderLayout.LINE_START);
        /**
         * Slider and button
         */

        JPanel panelSlider = new JPanel();
        /**
         * start
         */
        startButton.setEnabled(false);
        startButton.setIcon(new ImageIcon("src/main/resources/Image/icon.jpeg"));
        startButton.addActionListener(new StartCommand(graph, drawer,this));/*сюда
передаешь нужные тебе данные*/
        panelSlider.add(startButton);
        /**
         *   goToStart
         */
        goToStart.setEnabled(false);
        goToStart.setIcon(new ImageIcon("src/main/resources/Image/icon.jpeg"));
        goToStart.addActionListener(new goToStartCommand(drawer,this));
        panelSlider.add(goToStart);
        /**
         *  prevButton
         */
        prevButton.setEnabled(false);
        prevButton.setIcon(new ImageIcon("src/main/resources/Image/icon.jpeg"));
        prevButton.addActionListener(new PrevCommand(drawer,this));
        panelSlider.add(prevButton);
        /**
         *  counter
         */
        counter.setEditable(false);
        counter.setPreferredSize(new Dimension(45, 30));
        panelSlider.add(counter);
        /**
         *  nextButton
         */
```

```
    nextButton.setEnabled(false);
    nextButton.setIcon(new ImageIcon("src/main/resources/Image/icon.jpeg"));
    nextButton.addActionListener(new NextCommand(drawer,this));
    panelSlider.add(nextButton);
    /**
     * goToEnd
     */
    goToEnd.setEnabled(false);
    goToEnd.setIcon(new ImageIcon("src/main/resources/Image/icon.jpeg"));
    goToEnd.addActionListener(new goToEndCommand(drawer,this));
    panelSlider.add(goToEnd);

    contaner.add(panelSlider, BorderLayout.SOUTH);
    /**
     * graph
     */
    contaner.add(drawer, BorderLayout.CENTER);
}

private JMenu createHelpMenu(JFrame frame) {

    JMenu helpMenu = new JMenu("Help");
    JMenuItem Task = new JMenuItem("Task");
    JMenuItem About = new JMenuItem("About");
    helpMenu.add(Task);
    helpMenu.addSeparator();
    helpMenu.add(About);
    Task.addActionListener(new ActionListener() {
        JDialog dialog = new HelpMenuTask(frame);

        public void actionPerformed(ActionEvent event) {
            if (dialog == null) // в первый раз
                dialog = new HelpMenuTask(frame);
            dialog.setVisible(true); // отобразить диалог
        }
    });
    About.addActionListener(new ActionListener() {
        JDialog dialog = new HelpMenuAbout(frame);

        public void actionPerformed(ActionEvent event) {
            if (dialog == null) // в первый раз
```

```
                dialog = new HelpMenuAbout(frame);
            dialog.setVisible(true); // отобразить диалог
        }
    });

    return helpMenu;
}

private JMenu createMenu(JFrame frame) {
    JMenu menu = new JMenu("Menu");
    JMenuItem file = new JMenuItem("File");
    JMenuItem random = new JMenuItem("Random");
    JMenuItem clear = new JMenuItem("Clear graph");
    menu.add(file);
    menu.addSeparator();
    menu.add(random);
    menu.addSeparator();
    menu.add(clear);
    file.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (graph.getNodes().size() > 0) {
                JOptionPane.showMessageDialog(drawer, "Graph already exist!!!!!");
                return;
            }
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showDialog(null, "Открыть файл");
            if (ret == JFileChooser.APPROVE_OPTION) {
                File file = fileopen.getSelectedFile();
                try {
                    Scanner scanner = new Scanner(file);
                    n = scanner.nextInt();
                    System.out.println(n);
                    result = new int[n][n];
                    int i = 0, j = 0;
                    while (i != n) {
                        while (j != n) {
                            result[i][j] = scanner.nextInt();
                            j++;
                        }
                        j = 0;
```

```
                    i++;
                }

            } catch (FileNotFoundException ex) {
                ex.printStackTrace();
            }
            DrawGraph();
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    System.out.print(result[i][j] + " ");
                }
                System.out.println();
            }
        }
    }
});
random.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (graph.getNodes().size() > 0) {
            JOptionPane.showMessageDialog(drawer, "Graph already exist!!!!!");
            return;
        }
        result = CreationRandomAdjacencyMatrix();
        DrawGraph();
    }
});
clear.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        graph.Clear();
        drawer.graphList.clear();
        drawer.edgeDrawer.graph.Clear();
        drawer.nodeDrawer.graph.Clear();
        drawer.iteration = 0;
        drawer.setGraph(graph);
        startButton.setEnabled(true);
        goToEnd.setEnabled(false);
        goToStart.setEnabled(false);
        nextButton.setEnabled(false);
        prevButton.setEnabled(false);
```

```
                startButton.setEnabled(false);
                jTextArea.setText("");
                logString.list.clear();

                drawer.repaint();
            }
        });
        return menu;
    }

    private void DrawGraph() {
        this.startButton.setEnabled(true);
        Node[] node = new Node[result.length];
        float alpha = 360 / result.length;
        float radius = 100;
        float x, y;
        node[0] = new Node("1" + 0, drawer.getSize().width / 2 - 12,
drawer.getSize().height / 2 - 116);
        graph.addNode(node[0]);
        System.out.println(drawer.getSize().height / 2);
        System.out.println(drawer.getSize().width / 2);
        for (int i = 1; i < result.length; i++) {
            if (i % 2 == 0) {
                float radian = (float) (alpha * (i / 2) * pi / 180);
                float horda = 2 * radius * (float) Math.sin(radian / 2);
                y = horda * (float) Math.sin(pi * (180 - alpha * (i / 2)) / 360);
                x = (float) Math.sqrt((horda * horda) - (y * y));
                node[i - 1] = new Node("1" + (i - 1), drawer.getSize().width / 2 +
Math.round(-12 - y), drawer.getSize().height / 2 + Math.round(-116 + x));
                graph.addNode(node[i - 1]);
                node[i] = new Node("1" + i, drawer.getSize().width / 2 + Math.round(-12
+ y), drawer.getSize().height / 2 + Math.round(-116 + x));
                graph.addNode(node[i]);
            }
        }
        if (result.length % 2 == 0) {
            node[result.length - 1] = new Node("1" + (result.length - 1),
drawer.getSize().width / 2 - 12, drawer.getSize().height / 2 + 84);
            graph.addNode(node[result.length - 1]);
        }
        drawer.repaint();
```

```
        for (int i = 0; i < result.length; i++)
            for (int j = 0; j < result.length; j++)
                if (result[i][j] != 0)
                    graph.addEdge(new Edge(node[i], node[j], 0, result[i][j]));
        drawer.repaint();
    }


    public int[][] CreationRandomAdjacencyMatrix() {
        final Random random = new Random();
        int matrix_dimension = random.nextInt(4) + 4;
        n = matrix_dimension;
        int[][] adjacency_matrix = new int[matrix_dimension][matrix_dimension];
        int edge_weight_1;
        for (int i = 0; i < matrix_dimension; i++) {
            for (int j = i; j < matrix_dimension; j++) {
                edge_weight_1 = random.nextInt(10);
                if (i != j) {
                    if (j - i == 1) {
                        adjacency_matrix[i][j] = random.nextInt(9) + 1;
                    } else if (edge_weight_1 % 5 != 0)
                        adjacency_matrix[i][j] = edge_weight_1;
                } else
                    adjacency_matrix[i][j] = 0;
                if (adjacency_matrix[i][j] == 0 && i != j)
                    adjacency_matrix[j][i] = random.nextInt(10);
            }
        }
        adjacency_matrix[0][adjacency_matrix[0].length - 1] = 0;
        return adjacency_matrix;
    }

}

public class HelpMenu {
}

class HelpMenuAbout extends JDialog {
    public HelpMenuAbout(JFrame owner) {
        super(owner, "About", true);
```

```
        // Метка с HTML-форматированием выравнивается по центру.
        add(new JLabel("<html><h1><i>Task</i></h1><hr>"
                        + "Состав<br>" +
                        "Мищенко М.А. 7303<br>" +
                        "Батурин И. 7303 <br>" +
                        "Швец А.А. 7303"),
                BorderLayout.CENTER);

        // При активизации кнопки OK диалогове окно закрывается.
        JButton ok = new JButton("ok");
        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                setVisible(false);
            }
        });

        // Кнопка OK помещается в нижнюю часть окна.
        JPanel panel = new JPanel();
        panel.add(ok);
        add(panel, BorderLayout.SOUTH);
        setSize(500, 400);
    }
}

class HelpMenuTask extends JDialog {
    public HelpMenuTask(JFrame owner) {
        super(owner, "Task", true);
        // Метка с HTML-форматированием выравнивается по центру.
        add(new JLabel("<html><h1><i>Task</i></h1><hr>" + "Задача \"Алгоритм Форда – Фалкерсона\"<br>" +
                        "Задача: Визуализация алгоритма Форда-Фалкерсона, который используется для нахождения максимального потока в сети<br>" +
                        "Решение: Создание программы, с помощью которой можно задать граф и выполнить алгоритм Форда-Фалкерсона в нескольких<br>"+ "режимах(пошаговое выполнение, просмотр хода выполнения алгоритма, мгновенное получение ответа)<br>"
                ),
                BorderLayout.CENTER);
        // При активизации кнопки OK диалогове окно закрывается.
        JButton ok = new JButton("ok");
        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
```

```
                setVisible(false);
            }
        });
        // Кнопка OK помещается в нижнюю часть окна.
        JPanel panel = new JPanel();
        panel.add(ok);
        add(panel, BorderLayout.SOUTH);
        setSize(500, 400);
    }
}

public class Main {
    public static void main(String[] args) {
        Graphicsview app = new Graphicsview();
        app.setVisible(true);

    }
}

public class FordFulkersonTest {
    private int [][] array;
    private int size;
    private Graph graph = new Graph();
    Graphicsview frame;
    private GraphDrawer graphDrawer = new GraphDrawer(graph, frame);


    public void readFromFile(String path) {
        try
        {
            Scanner sc = new Scanner(new File(path));
            size = sc.nextInt();
            array = new int[size][size];
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    array[i][j] = sc.nextInt();
                }
            }
            sc.close();
```

```
        }
        catch (FileNotFoundException e)
        {
            System.out.println("Sorry, File not found!");
        }
        catch (InputMismatchException e)
        {
            System.out.println("Sorry, InputMismatchException");
        }
    }

    @Test
    public void algorithmFirstTest()  {
        String path =
"E:/java_projects/sources/JavaPractice/Practice4/src/Recources/Test/test1.txt";
        readFromFile(path);
        FordFulkerson fordFulkerson = new FordFulkerson(array, graph, graphDrawer);
        Assert.assertEquals(fordFulkerson.maxFlow(array, 0, array.length - 1), 9);
    }

    @Test
    public void algorithmSecondTest()  {
        String path =
"E:/java_projects/sources/JavaPractice/Practice4/src/Recources/Test/test2.txt";
        readFromFile(path);
        FordFulkerson fordFulkerson = new FordFulkerson(array, graph, graphDrawer);
        Assert.assertEquals(fordFulkerson.maxFlow(array, 0, array.length - 1), 11);
    }
}

public class GraphTest {
    private Graph graph = new Graph();
    private Node node1 = new Node("name1", 50, 70);
    private Node node2 = new Node("name2", 70, 50);
    private int edges_count = 0, nodes_count = 0;


    @Test
    public void addNode() {
        nodes_count = graph.nodes.size();
        graph.addNode(node1);
```

```
        Assert.assertEquals(graph.nodes.size(), nodes_count + 1);
    }

    @Test
    public void addEdge() {
        edges_count = graph.edges.size();
        Edge edge = new Edge(node1, node2, 0, 15);
        graph.addEdge(edge);
        Assert.assertEquals(graph.edges.size(), edges_count + 1);
    }

    @Test
    public void deleteNode() {
        graph.addNode(node1);
        nodes_count = graph.nodes.size();
        graph.deleteNode(node1);
        Assert.assertEquals(graph.nodes.size(), nodes_count - 1);
    }

    @Test public void deleteEdge() {
        edges_count = graph.edges.size();
        Edge edge = new Edge(node1, node2, 0, 15);
        graph.addEdge(edge);
        edges_count++;
        graph.deleteEdge(edge);
        Assert.assertEquals(graph.edges.size(), edges_count - 1);
    }
}
```