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

ОТЧЕТ

по учебной практике

Тема: Алгоритм Форда-Фалкерсона

Студентка гр. 7303 _____ Аплачкина Е.А.

Студентка гр. 7303 _____ Дегтярева А.А.

Студент гр. 7383 _____ Ласковенко Е.А.

Руководитель _____ Ефремов М.А.

Санкт-Петербург
2019

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Ласковенко Е.А. группы 7383

Студентка Дегтярева А.А. группы 7303

Студентка Аплачкина Е.А. 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: 11.07.2019

Дата защиты отчета: 11.07.2019

Студентка _____ Аплачкина Е.А.

Студентка _____ Дегтярева А.А.

Студент _____ Ласковенко Е.А.

Руководитель _____ Ефремов М.А.

АННОТАЦИЯ

В ходе данной практической работы будет представлена реализация алгоритма Форда-Фалкерсона на языке программирования Java, а также функция пошагового выполнения алгоритма, визуализация графа и его состояний после выполнения алгоритма. В отчете по практической работе будет подробно описана реализация алгоритма, план разработки, описание тестов, исходных литературных источников и представлены результаты работы программы.

SUMMARY

In the course of this practical work will be presented the realization of the Ford-Fulkerson algorithm in the Java programming language, step-by-step function, the visualization of the graph and its states after algorithm execution. The practical work report will describe in detail the implementation of the algorithm, the development plan, the description of the tests, source literature and the results of the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной работы - сделать визуализацию алгоритма Форда-Фалкерсона на языке программирования Java. Для достижения цели требуется определить функционал программы, разделить программу на три части(модель, контроллер и пользовательский интерфейс), реализовать визуализацию алгоритма, написать тесты для проверки правильности выполнения программы.

Алгоритм Форда—Фалкерсона решает задачу нахождения максимального потока в сети. Идея алгоритма заключается в следующем: изначально величине потока присваивается значение 0 для всех ребер, затем величина потока итеративно увеличивается посредством поиска увеличивающего пути (путь от источника к стоку, вдоль которого можно послать больший поток). Процесс повторяется, пока можно найти увеличивающий путь.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Все возможности пользователя, такие как создание графа различными способами, передвижение и изменение структуры графа, нахождение максимального потока, сброс состояния и просмотр информации о программе, показаны на рисунке 1.

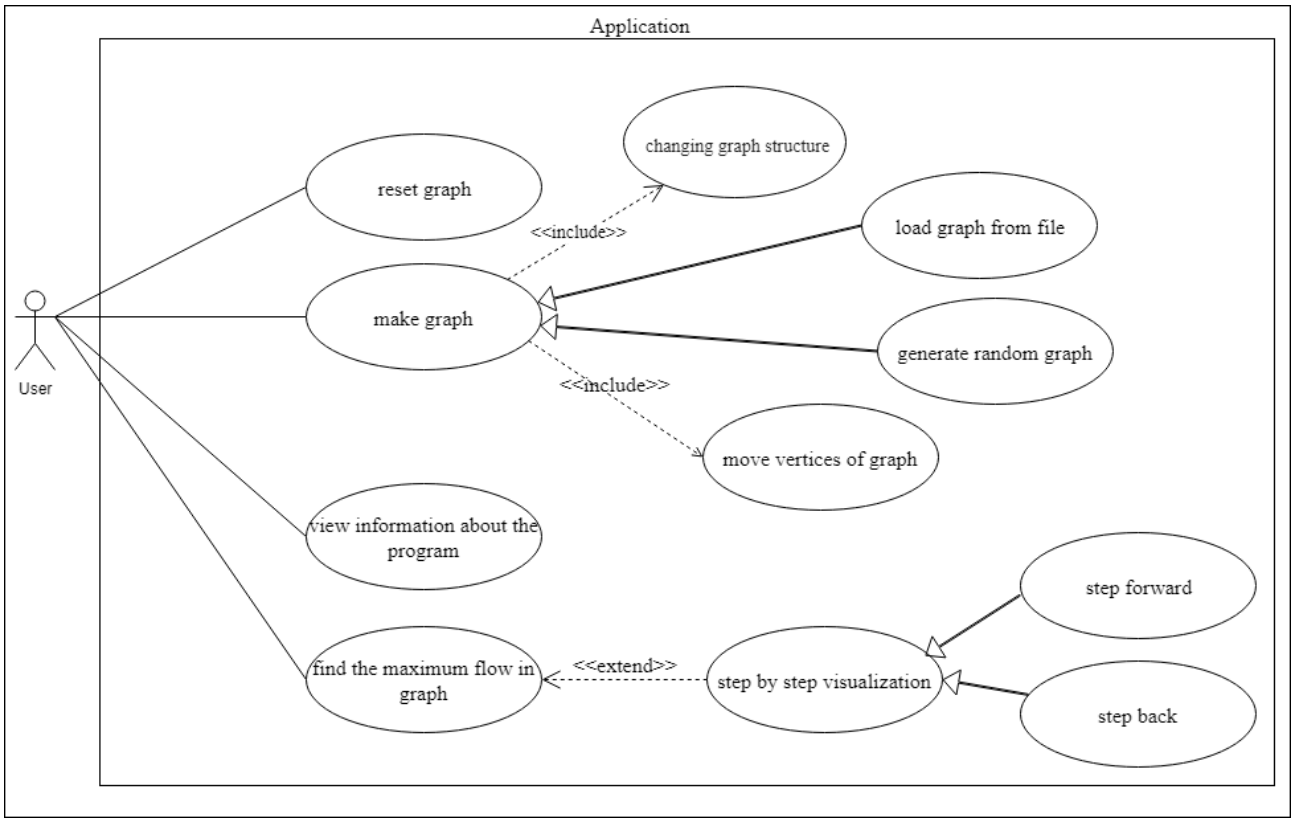

Рис. 1 — use-case диаграмма

1.1.1. Требования к вводу исходных данных:

Программа получает данные из файла. Сперва вводится вершина-исток и вершина-сток, затем вершины, которые соединяет ребро и вес данного ребра. В случае, если программа получила некорректные параметры, должно выводиться сообщение об ошибке.

1.1.2. Требования к визуализации:

По входным данным должен строиться граф, должны отображаться имена вершин, а так же вес ребра и проходящий по ним поток.

1.1.3. Требования к алгоритму:

Алгоритм должен выполняться по шагам или же сразу выводить значение максимального потока.

1.2. Уточнение требований после сдачи прототипа

Необходимо реализовать добавление и удаление вершины и ребер графа. Также нужно добавить кнопки для перевода графа в начальное и конечное состояние.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. Построение диаграммы сценариев использования (Use Case) используя нотацию UML. Создание прототипа пользовательского интерфейса (UI) на Swing-е.

2. Построение модели решения задачи, описав ее диаграммой классов (Class diagram) в нотации UML, а так же реализация алгоритма на собственноручно написанных структурах данных. Проверка корректности алгоритмов при помощи модульного тестирования.

3. Связывание UI и модели решения задачи посредством промежуточных управляющих классов, передающих данные между UI и моделью. На данном этапе программа уже должна уметь считывать/генерировать данные, а так же корректно показывать результат. Программа должна компилироваться, тестироваться, а так же собираться в jar-архив при помощи системы Maven.

4. Добавление оставленного на конец работы функционала, исправление ошибок программы. Написание отчета согласно шаблону.

2.2. Распределение ролей в бригаде

Евгений Ласковенко: составление Use case диаграммы и UI.

Алиса Дегтярева: создание диаграммы классов и модели задачи.

Екатерина Аплачкина: тестирование модели и соединение UI с моделью посредством реализации промежуточных управляющих классов, а так же оформление отчетов.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Использованные структуры данных

На рисунке 2 показана иерархия классов, требующихся для выполнения алгоритма. Класс Graph содержит список вершин и методы, для работы с ним, а классы Vertex и Edge реализуют свойства вершин и ребер. Класс BFS выполняет поиск в ширину, который используется классом Ford-Fulkerson.

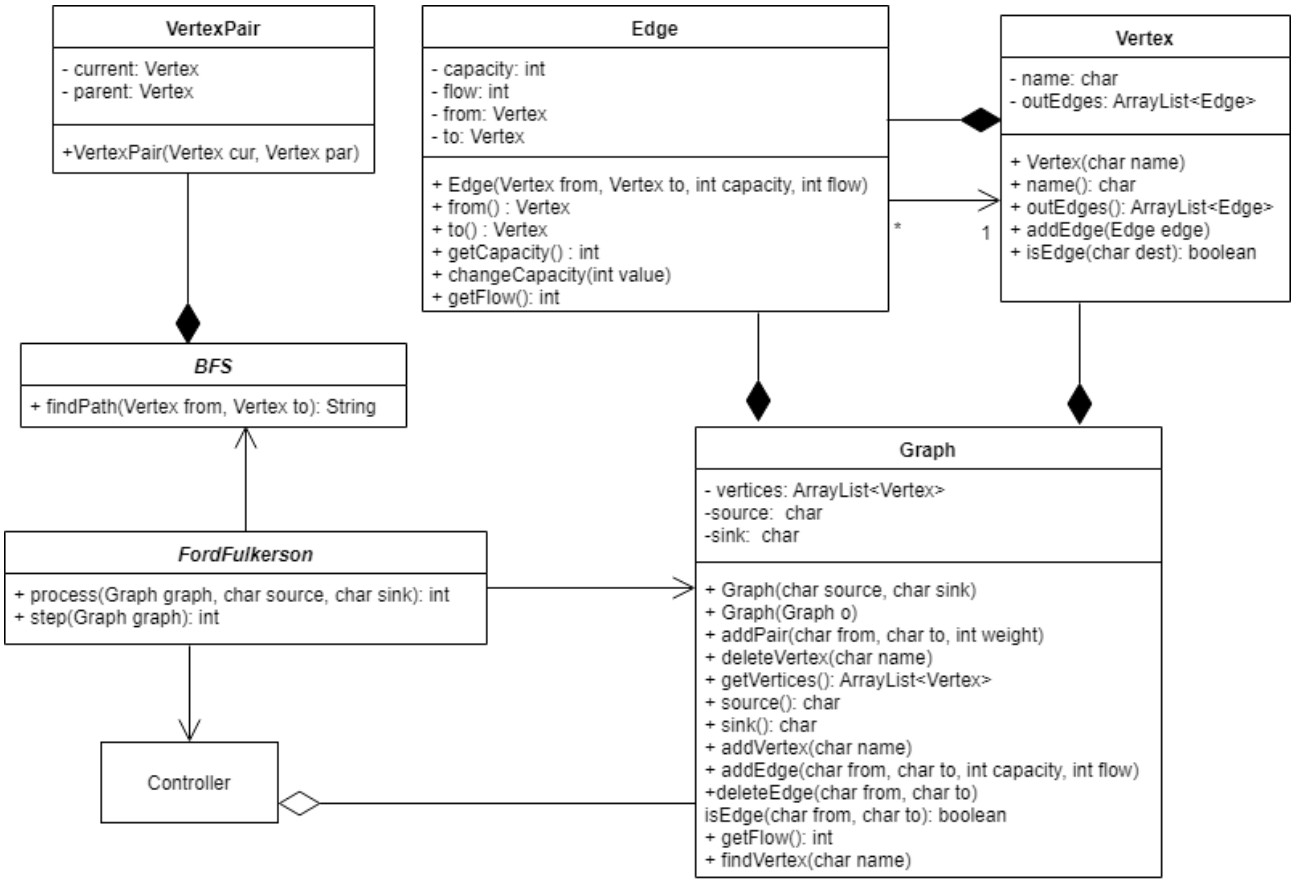

Рис. 2 — диаграмма классов

3.2. Основные методы

Edge(Vertex from, Vertex to, int capacity, int flow) — создание ребра.

Vertex(char name) — создание вершины.

public Graph(char source, char sink) — создание графа.

static int process(Graph graph) — метод, выполняющий алгоритм Форда-Фалкерсона.

static String findPath(Vertex from, Vertex to) — метод, выполняющий поиск пути.

3.3. Пример работы программы

Интерфейс состоит из меню в верхней части, поля для отрисовки графа, кнопок перехода в различные состояния графа и консоли, выводящей информацию и состоянии программы.

На рисунке 3 изображено состояние программы после случайной генерации графа.

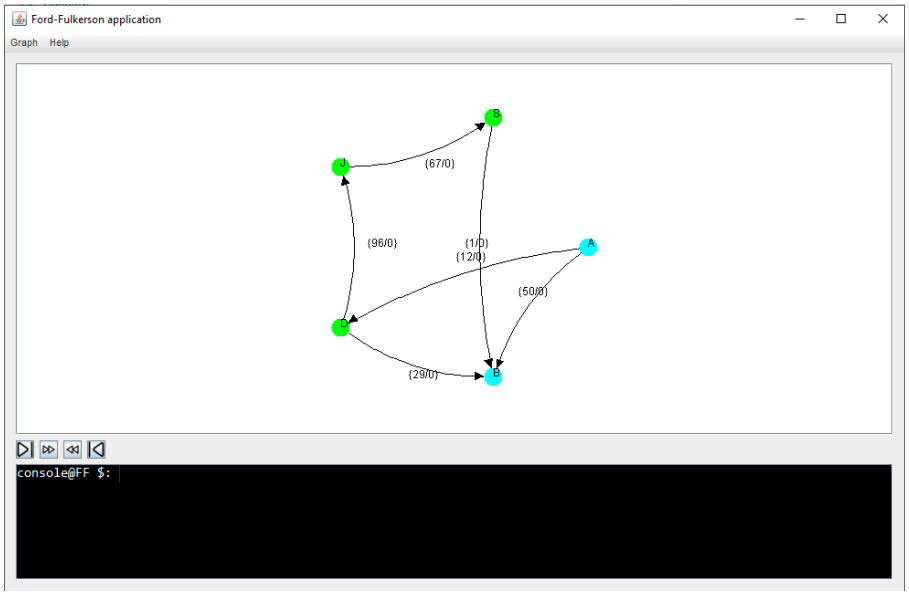

Рис. 3 — состояние приложения после генерации графа

На рисунке 4 показана функция добавления ребра в граф.

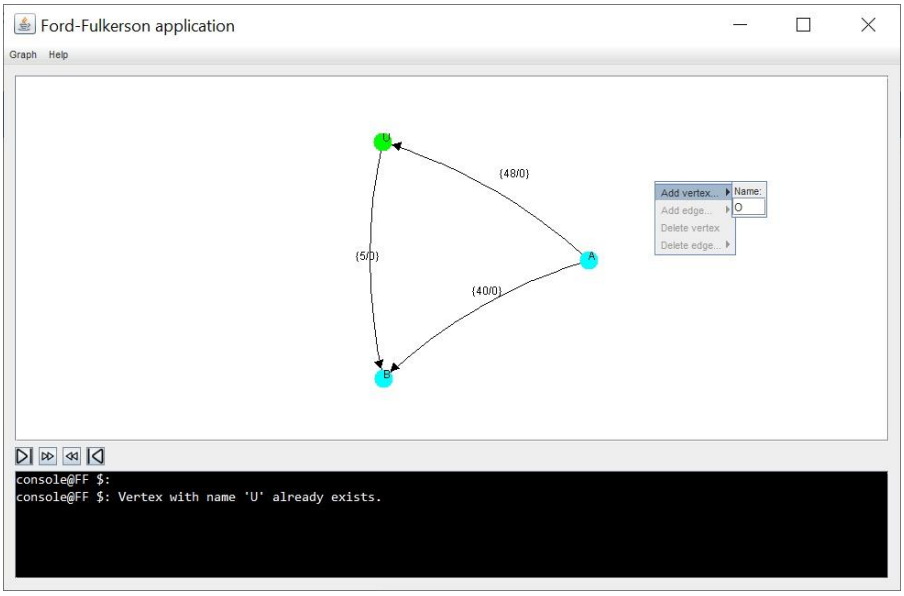

Рис. 4 — функция добавления ребра в граф

На рисунке 5 показано состояние программы после запуска алгоритма.

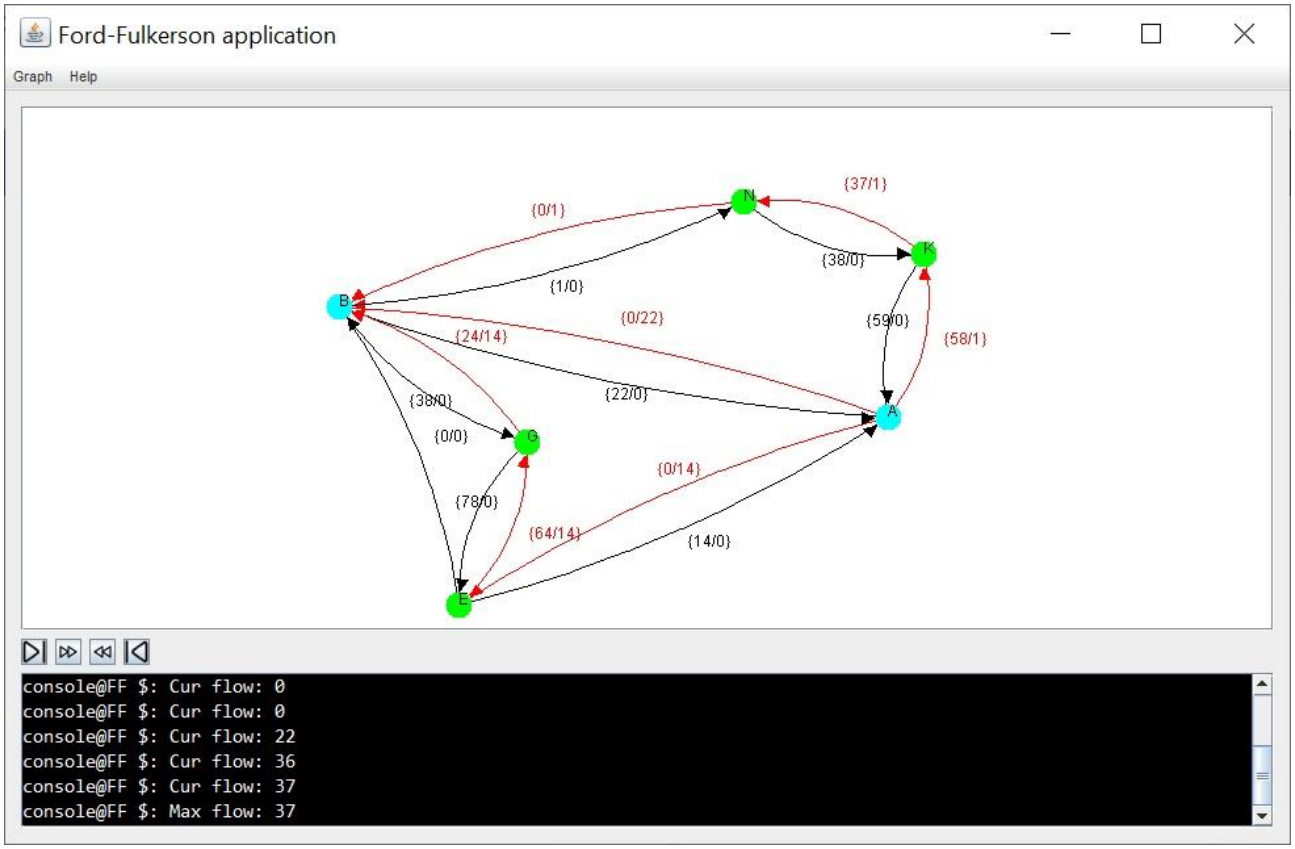

Рис.5 — состояние приложения после запуска алгоритма

4. ТЕСТИРОВАНИЕ

4.1.Тестирование кода алгоритма

Файл AlgorithmTest.java используется для тестирования алгоритма.

Методы test1() , test2(), test3(), test5() используются для тестирования алгоритма на различных наборах данных с вершинами, заданными буквами.

Метод test4() используется для тестирования работы алгоритма с вершинами, заданными цифрами.

Метод testOneVertex() используется для тестирования алгоритма на одной вершине.

4.2.Тестирование графа

Метод setNumericGraph() используется для тестирования графа с вершинами, обозначенными цифрами.

Метод testPathes() используется для тестирования путей в графе.

Метод testVertices() используется для тестирования вершин графа.

Метод testLoop() используется для тестирования случая, когда в граф добавляется петля.

Методы testEdgesNumber(), testEdgesCapacity(), testEdges(), testEdgesNegative(), testEdgeAddExist() используются для тестирования ребер графа.

ЗАКЛЮЧЕНИЕ

В ходе данной практической работы были решены поставленные задачи, созданы тесты для проверки корректной работы программы и представлены результаты их выполнения. Также была освоена работа с языком программирования Java и изучены его возможности. В итоге была написана программа, которая строит граф и визуализирует на нем работу алгоритма Форда-Фалкерсона.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кей Хорстманн, Гари Корнелл "Java. Библиотека профессионала", том 1, том 2: Пер. с англ. - М.: ООО „И. Д. Вильямс“, 2014.

2. Документация Java, Java Platform, Standard Edition API Specification https://docs.oracle.com/javase/7/docs/api/ 03.07.2019

3. Руководство по maven – что такое maven [Электронный ресурс], Apache Maven Project. URL: www.apache-maven.ru 04.07.2019

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

BFS.java:

```
package model;


import java.util.ArrayList;
import java.util.ArrayDeque;

abstract class BFS
{

    static String findPath(Vertex from, Vertex to) {
        class VertexPair {
            private VertexPair(Vertex cur, Vertex par) {
                current = cur;
                parent = par;
            }
            private Vertex current;
            private Vertex parent;
        }


        StringBuilder out = new StringBuilder();
        ArrayList <Vertex> checked = new ArrayList<>();
        ArrayList <VertexPair> pairs = new ArrayList<>();
        ArrayDeque <Vertex> vertexQueue = new ArrayDeque<>();

        pairs.add(new VertexPair(from, null));
        checked.add(from);
        vertexQueue.add(from);

        while (!(vertexQueue.isEmpty())) {
            Vertex curVertex = vertexQueue.pollFirst();

            if(curVertex == to) {

                while (curVertex != null){
                    out.append(curVertex.name());
                    for(VertexPair curPair: pairs) {
                        if (curPair.current == curVertex) {
                            curVertex = curPair.parent;
                        }
                    }
                }

                return out.reverse().toString();
            }

            for(Edge curEdge: curVertex.outEdges()) {
                if(curEdge.getCapacity() != 0 &&
checked.indexOf(curEdge.to()) == -1) {
                    pairs.add(new VertexPair(curEdge.to(), curVertex));
                    checked.add(curEdge.to());
                    vertexQueue.add(curEdge.to());
                }
```

```
            }
        }
        return null;
    }
}
```

Edge.java:

```
package model;

class Edge
{
    Edge(Vertex from, Vertex to, int capacity, int flow) {
        this.from = from;
        this.to = to;
        this.capacity = capacity;
        this.flow = flow;
    }

    //Getters:
    public Vertex from() {
        return from;
    }
    public Vertex to() {
        return to;
    }
    public int getCapacity() {
        return capacity;
    }
    public int getFlow() {
        return flow;
    }

    //Setter:
    void changeCapacity(int value) {
        if (capacity < value) {
            throw new IllegalArgumentException();
        }
        capacity -= value;
        flow += value;
    }


    private Vertex from;
    private Vertex to;
    private int capacity;
    private int flow;
}
```

FordFulkerson.java:

```
package model;

import java.util.ArrayList;

public abstract class FordFulkerson
{
```

```
    static int process(Graph graph) {

        if(graph.getVertices().get(0).name() ==
graph.getVertices().get(1).name()) {
            throw new IllegalArgumentException("Wrong source or sink.");
        }

        int flow = 0;
        int stepFlow = step(graph);
        while (stepFlow != 0) {
            flow += stepFlow;
            stepFlow = step(graph);
        }
        return flow;
    }

    public static int step(Graph graph) {

        Vertex _source = graph.getVertices().get(0);
        Vertex _sink = graph.getVertices().get(1);

        String path = BFS.findPath(_source, _sink);
        if (path == null) {

            return 0;
        }
        ArrayList<Edge> edgeList = new ArrayList<>();
        Vertex curVertex = graph.findVertex(path.charAt(0));
        int minCapacity = 0;
        for (int i = 1; i < path.length(); i++) {
            for (Edge curEdge: curVertex.outEdges()) {
                if (curEdge.to().name() == path.charAt(i)) {
                    edgeList.add(curEdge);
                    if(curEdge.getCapacity() < minCapacity || minCapacity
== 0){
                        minCapacity = curEdge.getCapacity();
                    }
                    graph.addEdge(path.charAt(i), path.charAt(i-1),
curEdge.getCapacity(), curEdge.getFlow());
                    curVertex = curEdge.to();
                    break;
                }
            }
        }
        for(Edge curEdge:edgeList) {
            curEdge.changeCapacity(minCapacity);
        }
        return minCapacity;
    }
}
```

Graph.java

```
package model;

import java.util.ArrayList;
import java.util.Iterator;
```

```
public class Graph
{
    public Graph(char source, char sink) {
        this.source = source;
        this.sink = sink;
        vertices = new ArrayList<>();
        vertices.add(new Vertex(source));
        vertices.add(new Vertex(sink));
    }

    public Graph(Graph o) {
        vertices = new ArrayList<>();
        source = o.source();
        sink = o.sink();
        addVertex(source);
        addVertex(sink);

        for(Vertex v : o.getVertices()) {
            addVertex(v.name());
            for(Edge e : v.outEdges()) {
                addEdge(e.from().name(), e.to().name(), e.getCapacity(),
e.getFlow());
            }
        }
    }

    //Getters:
    public ArrayList<Vertex> getVertices() {
        return vertices;
    }
    public char source() {
        return source;
    }
    public char sink() {
        return sink;
    }

    public void addVertex(char name) {
        if(findVertex(name) == null) {
            vertices.add(new Vertex(name));
        }
    }
    public void addEdge(char from, char to, int capacity, int flow) {
        if(capacity < 0) {
            throw new IllegalArgumentException("Negative weight of
edge.");
        }
        if(from == to) {
            throw new IllegalArgumentException("Loop is not allowed.");
        }

        addVertex(from);
        addVertex(to);
        Vertex _from = findVertex(from);
        Vertex _to = findVertex(to);
        if(!isEdge(from, to)) {
```

```
            _from.addEdge(new Edge(_from, _to, capacity, flow));
        }
    }

    public void deleteVertex(char name) {
        Vertex v = findVertex(name);
        if(v != null) {
            for(Vertex vt : vertices) {
                Iterator<Edge> it = vt.outEdges().iterator();
                while(it.hasNext()) {
                    Edge e = it.next();
                    if(e.to().name() == v.name()) {
                        it.remove();
                    }
                }
            }
            Iterator<Edge> it = v.outEdges().iterator();
            while(it.hasNext()) {
                Edge e = it.next();
                if(e.to().name() == v.name()) {
                    it.remove();
                }
            }
            vertices.remove(v);
        }
    }

    public void deleteEdge(char from, char to) {
        Vertex fr = findVertex(from);
        Vertex t = findVertex(to);
        if(fr != null && t != null) {
            Iterator<Edge> it = fr.outEdges().iterator();
            while(it.hasNext()) {
                Edge e = it.next();
                if(e.to().name() == t.name()) {
                    it.remove();
                    break;
                }
            }
        }
    }

    boolean isEdge(char from, char to) {
        Vertex _from = findVertex(from);
        Vertex _to = findVertex(to);
        if(_from == null || _to == null) {
            return false;
        }
        for(Edge e : _from.outEdges()) {
            if(e.to().name() == to) {
                return true;
            }
        }

        return false;
    }
```

```
    public int getFlow() {
        Vertex _source = getVertices().get(0);
        int curFlow = 0;
        for(Edge e : _source.outEdges()) {
            curFlow += e.getFlow();
        }
        return curFlow;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();
        for(Vertex v : vertices) {

str.append(v.name()).append(System.getProperty("line.separator"));
        }
        for(Vertex v : vertices) {
            for(Edge e : v.outEdges()) {
                str.append(e.from().name())
                        .append(' ')
                        .append(e.to().name())
                        .append(' ')
                        .append(e.getCapacity())
                        .append(' ')
                        .append(e.getFlow())
                        .append(System.getProperty("line.separator"));
            }
        }
        return str.toString();
    }

    Vertex findVertex(char name) {
        for (Vertex it : vertices) {
            if (it.name() == name) {
                return it;
            }
        }
        return null;
    }

    private ArrayList<Vertex> vertices;
    private char source;
    private char sink;
}
```

Vertex.java

```
package model;

import java.util.ArrayList;

public class Vertex
{
    Vertex(char name) {
        this.name = name;
        outEdges = new ArrayList<>();
    }
```

```
    //Getters:
    public char name() {
        return name;
    }
    public ArrayList<Edge> outEdges() {
        return outEdges;
    }

    //Setters:
    public void addEdge(Edge edge) {
        outEdges.add(edge);
    }

    private char name;
    private ArrayList<Edge> outEdges;
}
```

Controller.java

```
package controller;


import java.awt.*;
import java.util.ArrayList;

import ui.Frame;
import model.Graph;
import model.FordFulkerson;
import ui.GraphicVertex;

public class Controller
{
    private Frame ui;
    private Graph graph;
    private ArrayList<Graph> conditions;
    private int currentGraph;

    public Controller() {
        ui = new Frame(this);
        ui.setVisible(true);
        conditions = new ArrayList<>();
        currentGraph = 0;
    }

    public void restoreGraph() {
        if(conditions.size() > 0) {
            graph = new Graph(conditions.get(0));
            conditions.clear();
            currentGraph = 0;
            ui.clearPanel();
            drawGraph(graph);
        }
    }

    public void addVertex(char name, Point p) {
        graph.addVertex(name);
        ui.addVertex(new GraphicVertex(name, p));
```

```
    }

    public void addEdge(char from, char to, int capacity, int flow) {
        graph.addEdge(from, to, capacity, flow);
        ui.addEdge(from, to, capacity, flow);
    }

    public void deleteVertex(char name) {
        graph.deleteVertex(name);
        ui.deleteVertex(name);
    }

    public void deleteEdge(char from, char to) {
        graph.deleteEdge(from, to);
        ui.deleteEdge(from, to);
    }

    public void makeGraph(String str) {
        String[] arr = str.split(System.getProperty("line.separator"));
        char source = arr[0].charAt(0);
        char sink = arr[1].charAt(0);
        graph = new Graph(source, sink);

        for(int i=2; i<arr.length; ++i) {
            String[] cur = arr[i].split(" ");
            graph.addEdge(cur[0].charAt(0), cur[1].charAt(0),
Integer.parseInt(cur[2]), 0);
        }

        drawGraph(graph);
    }

    public void randomGraph() {
        ArrayList<Character> vertices = new ArrayList<>();
        char source = 'A';
        char sink = 'B';
        vertices.add(source);
        vertices.add(sink);
        graph = new Graph(source, sink);

        for(int i=67; i!=123; ++i) {
            if(i>=91 && i<=96) continue;
            if(Math.random()<0.2) {
                vertices.add((char)i);
            }
        }

        int nWays = (int)(Math.random()*vertices.size()+1);
        ArrayList<String> ways = new ArrayList<>(nWays);
        for(int i=0; i<nWays; ++i) {
            StringBuilder str = new StringBuilder();
            str.append(vertices.get(0));
            for(int l=2; l<(int)(Math.random()*(vertices.size()-1)); ++l)
{
                if(Math.random()<0.5) str.append(vertices.get(l));
            }
            str.append(vertices.get(1));
```

```
            ways.add(str.toString());
        }

        for(String way : ways) {
            for(int i=0; i<way.length()-1; ++i) {
                graph.addEdge(way.charAt(i), way.charAt(i+1),
(int)(Math.random()*100), 0);
            }
        }

        drawGraph(graph);
    }

    public void reset() {
        ui.clearPanel();
        conditions.clear();
        currentGraph = 0;
        graph = null;
        ui.repaint();
    }

    private void drawGraph(Graph graph) {
        int j = 0;
        String[] s =
graph.toString().split(System.getProperty("line.separator"));
        ArrayList<GraphicVertex> vrt = new ArrayList<>();
        for(GraphicVertex v : ui.getVertices()) {
            vrt.add(new GraphicVertex(v.getName(), v.getP()));
        }

        ui.clearPanel();

        if(vrt.isEmpty()) {
            ArrayList<Point> coord = new ArrayList<>();
            int r = 150, n = graph.getVertices().size();
            for (int i = 0; i < n; i++) {
                int x = 475 + (int)(r * Math.cos(2*Math.PI/n*i));
                int y = 200 + (int)(r* Math.sin(2*Math.PI/n*i));
                coord.add(new Point(x,y));
            }
            for(; j < s.length && s[j].length() == 1; j++) {
                ui.addVertex(new GraphicVertex(s[j].charAt(0),
coord.get(j)));
            }
        } else {
            j = vrt.size();
            for(GraphicVertex gv: vrt) {
                ui.addVertex(gv);
            }
        }

        for(;j < s.length && s[j].length() > 1; j++) {
            String[] temp = s[j].split(" ");
            ui.addEdge(temp[0].charAt(0), temp[1].charAt(0),
Integer.parseInt(temp[2]), Integer.parseInt(temp[3]));
        }
```

```
    ui.repaint();
}

private void copyGraph() {
    Graph newGraph = new Graph(graph);
    conditions.add(newGraph);
}

public int toStart() {
    if (graph == null) {
        return -1;
    }
    if (conditions.isEmpty()) {
        initFordFulkerson();
    }
    currentGraph = 0;
    drawGraph(conditions.get(currentGraph));
    return 0;
}

public int toEnd() {
    if (graph == null) {
        return -1;
    }
    if (conditions.isEmpty()) {
        initFordFulkerson();
    }
    currentGraph = conditions.size() - 1;
    drawGraph(conditions.get(currentGraph));
    return conditions.get(currentGraph).getFlow();
}

public int stepForward(){
    if (graph == null) {
        return -1;
    }
    if (conditions.isEmpty()) {
        initFordFulkerson();
    }
    if (currentGraph == conditions.size() - 1) {
        return conditions.get(conditions.size() - 1).getFlow();
    }
    currentGraph ++;
    drawGraph(conditions.get(currentGraph));
    return conditions.get(currentGraph).getFlow();
}

public int stepBack(){
    if (graph == null) {
        return -1;
    }
    if (conditions.isEmpty()) {
        initFordFulkerson();
    }
    if (currentGraph == 0) {
        return 0;
    }
```

```
        --currentGraph;
        drawGraph(conditions.get(currentGraph));
        return conditions.get(currentGraph).getFlow();
    }

    private void initFordFulkerson() {
        int flow;
        do {
            copyGraph();
            flow = FordFulkerson.step(graph);
        } while (flow != 0);
    }
}
```

Frame.java

```
package ui;


import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.*;
import java.net.URI;
import java.net.URL;
import java.rmi.server.ExportException;
import java.util.Scanner;
import java.lang.StringBuilder;
import java.util.ArrayList;

import controller.Controller;

public class Frame extends JFrame {
    public Frame(final Controller controller) {
        //Controller:
        this.controller = controller;
        //Menu bar:
        JMenuBar menuBar = new JMenuBar();
        JMenu graphMenu = new JMenu("Graph");
        JMenu helpMenu = new JMenu("Help");
        JMenuItem randomItem = new JMenuItem("Generate graph");
        JMenuItem fileItem = new JMenuItem("Load from file...");
        JMenuItem resetItem = new JMenuItem("Reset");
        JMenuItem helpItem = new JMenuItem("Help");
        //Fonts:
        Font font = new Font("TimesNewRoman", Font.PLAIN, 11);
        graphMenu.setFont(font);
        helpMenu.setFont(font);
        randomItem.setFont(font);
        fileItem.setFont(font);
        resetItem.setFont(font);
        helpItem.setFont(font);
        //Add
        graphMenu.add(randomItem);
        graphMenu.add(fileItem);
        graphMenu.addSeparator();
        graphMenu.add(resetItem);
```

```
        helpMenu.add(helpItem);
        menuBar.add(graphMenu);
        menuBar.add(helpMenu);
        //

        //Graph panel:
        graphPanel = new GraphPanel(this);
        JScrollPane graphPanelScroll = new JScrollPane(graphPanel);
        //

        //Buttons:
        JButton toEnd = new JButton(new
ImageIcon(Frame.class.getResource("images/end.png")));
        toEnd.setPreferredSize(new Dimension(20,20));
        toEnd.setToolTipText("To end");
        JButton stepForward = new JButton(new
ImageIcon(Frame.class.getResource("images/forward.png")));
        stepForward.setPreferredSize(new Dimension(20,20));
        stepForward.setToolTipText("Step forward");
        JButton stepBack = new JButton(new
ImageIcon(Frame.class.getResource("images/back.png")));
        stepBack.setPreferredSize(new Dimension(20,20));
        stepBack.setToolTipText("Step back");
        JButton toStart = new JButton(new
ImageIcon(Frame.class.getResource("images/start.png")));
        toStart.setPreferredSize(new Dimension(20,20));
        toStart.setToolTipText("To start");

        //Console:
        console = new JTextArea("console@FF $: ");
        console.setFont(new Font("Consolas", Font.PLAIN, 15));
        console.setBackground(Color.BLACK);
        console.setForeground(Color.WHITE);
        console.setCaretPosition(consoleStr.length());
        console.addKeyListener(new KeyListener() {
            @Override
            public void keyTyped(KeyEvent e) {
                // TODO Auto-generated method stub
            }
            @Override
            public void keyReleased(KeyEvent e) {
                if (e.getKeyCode() == 0x0A) {
                    console.setText(console.getText() + consoleStr);
                    console.moveCaretPosition(console.getText().length());
                }
            }
            @Override
            public void keyPressed(KeyEvent e) {
                // TODO Auto-generated method stub
            }
        });
        JScrollPane consolePane = new JScrollPane(console);
        //

        //Context menu:

        //
```

```
        //Frame:
        setBounds(150, 10, 1000, 650);
        setTitle("Ford-Fulkerson application");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setJMenuBar(menuBar);

        GroupLayout layout = new GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setAutoCreateGaps(true);
        layout.setAutoCreateContainerGaps(true);


layout.setHorizontalGroup(layout.createParallelGroup(GroupLayout.Alignmen
t.LEADING)
                .addComponent(graphPanelScroll)
                .addGroup(layout.createSequentialGroup()
                        .addComponent(toEnd)
                        .addComponent(stepForward)
                        .addComponent(stepBack)
                        .addComponent(toStart))
                .addComponent(consolePane)
        );

        layout.linkSize(SwingConstants.HORIZONTAL, toEnd, stepForward,
stepBack, toStart);

        layout.setVerticalGroup(layout.createSequentialGroup()
                .addComponent(graphPanelScroll)
                .addGroup(layout.createParallelGroup(GroupLayout.Alignmen
t.LEADING)
                        .addComponent(toEnd)
                        .addComponent(stepForward)
                        .addComponent(stepBack)
                        .addComponent(toStart))
                .addComponent(consolePane)
        );

        layout.linkSize(SwingConstants.VERTICAL, toEnd, stepForward,
stepBack, toStart);

        //Actions
        fileItem.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                controller.reset();
                console.setText(consoleStr);

                JFileChooser fileopen = new JFileChooser();
                int ret = fileopen.showDialog(null, "РћС‚РєСЂС‹С‚СЊ
С„Р°Р№Р»");
                if (ret == JFileChooser.APPROVE_OPTION) {
                    File file = fileopen.getSelectedFile();
                    Scanner sc;
                    StringBuilder input = new StringBuilder();
                    try {
                        sc = new Scanner(file);
```

```
                        for (int i = 0; i < 2; i++) {
                            String line = sc.nextLine();

input.append(line).append(System.getProperty("line.separator"));
                        }
                        while (sc.hasNextLine()) {
                            String line = sc.nextLine();
                            input.append(line).append(" 0
").append(System.getProperty("line.separator"));
                        }
                        sc.close();
                    } catch (FileNotFoundException ex) {
                        return;
                    }
                    controller.makeGraph(input.toString());
                }
            }
        });

        resetItem.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                controller.reset();
                console.setText(consoleStr);
            }
        });

        randomItem.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                controller.reset();
                console.setText(consoleStr);
                controller.randomGraph();
            }
        });

        toEnd.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                int flow = controller.toEnd();
                if (flow == -1) {
                    return;
                }
                print("Max flow: " + flow);
            }
        });

        stepForward.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                int flow = controller.stepForward();
                if (flow == -1) {
                    return;
                }
                print("Cur flow: " + flow);

            }
```

```
        });

        stepBack.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                int flow = controller.stepBack();
                if (flow == -1) {
                    return;
                }
                print("Cur flow: " + flow);
            }
        });

        toStart.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                int flow = controller.toStart();
                if (flow == -1) {
                    return;
                }
                print("Cur flow: 0");
            }
        });

        helpItem.addActionListener(new ActionListener()  {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    URL url = Frame.class.getResource("docs/help.html");
                    Desktop.getDesktop().browse(url.toURI());
                }
                catch(Exception ex) {}
            }
        });
    }

    public void addVertex(GraphicVertex e) {
        if(graphPanel.findVertex(e.getName()) == null) {
            graphPanel.addVertex(e);
        }
    }

    public void addEdge(char from, char to, int capacity, int flow) {
        graphPanel.addEdge(from, to, capacity, flow);
    }

    public void deleteVertex(char name) {
        graphPanel.deleteVertex(name);
    }

    public void deleteEdge(char from, char to) {
        graphPanel.deleteEdge(from, to);
    }

    public void clearPanel() {
        graphPanel.clear();
    }
```

```
    public final Controller getController() {
        return controller;
    }

    public void print(String str) {
        console.setText(console.getText() +
System.getProperty("line.separator")+ consoleStr + str);
    }

    public ArrayList <GraphicVertex> getVertices(){
        return graphPanel.getVertices();
    }

    private final GraphPanel graphPanel;
    private final JTextArea console;
    final String consoleStr = "console@FF $: ";
    private final Controller controller;
}


package ui;


import java.awt.*;
import java.awt.geom.AffineTransform;
import java.awt.geom.Point2D;
import java.awt.geom.QuadCurve2D;

public class GraphicEdge
{
     private final GraphicVertex from;
     private final GraphicVertex to;
     private int capacity;
     private int flow;


     public GraphicEdge(GraphicVertex from, GraphicVertex to, int
capacity, int flow) {
          this.from = from;
          this.to = to;
          this.capacity = capacity;
          this.flow = flow;
     }

     public void draw(Graphics g) {
          if(getFlow() != 0)  g.setColor(Color.RED);
          else g.setColor(Color.BLACK);

          double angle = Math.atan2(to.getP().y - from.getP().y,
to.getP().x - from.getP().x);
          double offsetX = 10*Math.cos(angle);
          double offsetY = 10*Math.sin(angle);
          int distance =
(int)Math.round(from.getP().distance(to.getP()));

          AffineTransform t = new AffineTransform();
          t.setToIdentity();
```

```
            t.translate(from.getP().x, from.getP().y);
            t.rotate(angle);
            Point2D p = new Point(distance/2, 30);
            t.transform(p, p);
            Graphics2D g2d = (Graphics2D)g;
            QuadCurve2D qc2d = new QuadCurve2D.Double(
                        from.getP().x,
                        from.getP().y,
                        p.getX(),
                        p.getY(),
                        to.getP().x,
                        to.getP().y
            );
            g2d.draw(qc2d);

            t.setToIdentity();
           double arrowAngle = Math.atan2(p.getY() - to.getP().y, p.getX()
- to.getP().x);
            t.translate(to.getP().x, to.getP().y);
            t.rotate(angle - Math.PI/2 - Math.PI/12);
            Point2D p1 = new Point(-5,-20);
            Point2D p2 = new Point(0,-10);
            Point2D p3 = new Point(5,-20);
            t.transform(p1, p1);
            t.transform(p2, p2);
            t.transform(p3, p3);
            g.fillPolygon(
                        new int[]{
                                    (int)p1.getX(),
                                    (int)p2.getX(),
                                    (int)p3.getX()
                        },
                        new int[]{
                                    (int) p1.getY(),
                                    (int) p2.getY(),
                                    (int) p3.getY()
                        },
                        3
            );

            g.drawString(String.format("{%s/%s}", this.capacity,
this.flow), (int)p.getX(), (int)p.getY());
      }

      public void updatePosition(Point d) {
            from.getP().x += d.x;
            from.getP().y += d.y;
            to.getP().x += d.x;
            to.getP().y += d.y;
      }

      public GraphicVertex getFrom() {
            return from;
      }

      public GraphicVertex getTo() {
            return to;
```

```
	}

	public int getCapacity() {
		return capacity;
	}

	public int getFlow() {
		return flow;
	}

}


package ui;


import java.awt.*;
import java.util.ArrayList;

public class GraphicVertex
{
	private static final int RADIUS = 10;
	private Color COLOR;
	private final char name;
	private Point p;
	private Rectangle b;
	private ArrayList<GraphicEdge> edges;
	private boolean selected;


	public GraphicVertex(char name, Point p) {
		this.name = name;
		this.p = p;
		b = new Rectangle();
		setBoundary();
		COLOR = Color.GREEN;
		edges = new ArrayList<>();
		selected = false;
	}

	//Getters:
	public final char getName() {
		return name;
	}
	public Point getP() {
		return p;
	}
	public Rectangle getB() {
		return b;
	}
	public ArrayList<GraphicEdge> getEdges() {
		return edges;
	}
	//Setters:
	public void setSelected(boolean f) {
		selected = f;
	}
```

```
	public void draw(Graphics g) {

		if(selected) {
			g.setColor(Color.PINK);

		} else {
			g.setColor(COLOR);
			g.fillOval(b.x, b.y, b.width, b.height);
		}
		g.fillOval(b.x, b.y, b.width, b.height);
		g.setColor(Color.BLACK);
		g.drawString(String.format("%s", name), p.x, p.y);
	}

	public void updatePosition(Point d) {
		p.x += d.x;
		p.y += d.y;
		setBoundary();
	}

	private final void setBoundary() {
		b.setBounds(p.x - RADIUS, p.y - RADIUS, 2 * RADIUS, 2 *
RADIUS);
	}

	public boolean contains(Point p) {
		return b.contains(p);
	}

	public void setColor(Color color) {
		COLOR = color;
	}
}


package ui;


import java.awt.*;
import java.awt.event.*;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.*;
import javax.swing.event.*;

public class GraphPanel extends JPanel
{
	private static final int WIDTH = 950;
	private static final int HEIGHT = 300;

	private final Frame frame;

	private ArrayList<GraphicVertex> vertices;

	private Point mouseP;
	private GraphicVertex selected;
```

```
	private char prevSelectedName;

	public GraphPanel(final Frame parent) {
		frame = parent;

		setPreferredSize(new Dimension(WIDTH, HEIGHT));
		setBackground(Color.WHITE);
		addMouseListener(new MouseHandler());
		addMouseMotionListener(new MouseMotionHandler());

		vertices = new ArrayList<>();
		mouseP = new Point(WIDTH/2, HEIGHT/2);
		selected = null;
		prevSelectedName = '\0';
	}

	public Frame getFrame() {
		return frame;
	}

	public Point getMouseP() {
		return mouseP;
	}

	public void addVertex(GraphicVertex el) {
		vertices.add(el);
		if(el.getName() == vertices.get(0).getName() || el.getName()
== vertices.get(1).getName()){
			el.setColor(Color.CYAN);
		}
	}

	public void addEdge(char from, char to, int capacity, int flow) {
		GraphicVertex _from = findVertex(from);
		GraphicVertex _to = findVertex(to);
		if(_from == null || _to == null) {
			return;
		}

		_from.getEdges().add(new GraphicEdge(_from, _to, capacity,
flow));
	}

	public void deleteVertex(char name) {
		GraphicVertex v = findVertex(name);
		if(v != null) {
			for(GraphicVertex vt : vertices) {
				Iterator<GraphicEdge> it = vt.getEdges().iterator();
				 while(it.hasNext()) {
					GraphicEdge e = it.next();
					if(e.getTo().getName() == v.getName()) {
						it.remove();
					}
				 }
			}
			Iterator<GraphicEdge> it = v.getEdges().iterator();
			while(it.hasNext()) {
```

```
                GraphicEdge e = it.next();
                if(e.getTo().getName() == v.getName()) {
                    it.remove();
                }
            }
            vertices.remove(v);
        }
    }

    public void deleteEdge(char from, char to) {
        GraphicVertex fr = findVertex(from);
        GraphicVertex t = findVertex(to);
        if(fr != null && t != null) {
            Iterator<GraphicEdge> it = fr.getEdges().iterator();
            while(it.hasNext()) {
                GraphicEdge e = it.next();
                if(e.getTo().getName() == t.getName()) {
                    it.remove();
                    break;
                }
            }
        }
    }

    public void clear() {
        vertices.clear();
    }

    public GraphicVertex findVertex(char name) {
        for(GraphicVertex v : vertices) {
            if(v.getName() == name) {
                return v;
            }
        }
        return null;
    }

    public ArrayList<GraphicVertex> getVertices() {
        return vertices;
    }

    private void doPop(MouseEvent e) {
        PopupMenu menu = new PopupMenu(this);
        if(selected == null) {
            menu.addVertexMenu.setEnabled(true);
            menu.addEdgeMenu.setEnabled(false);
            menu.deleteVertexItem.setEnabled(false);
            menu.deleteEdgeMenu.setEnabled(false);
        } else {
            menu.addVertexMenu.setEnabled(false);
            menu.addEdgeMenu.setEnabled(true);
            menu.deleteVertexItem.setEnabled(true);
            if(selected.getEdges().size() == 0) {
                menu.deleteEdgeMenu.setEnabled(false);
            } else {
                menu.deleteEdgeMenu.setEnabled(true);
            }
```

```
            }
            menu.show(e.getComponent(), e.getX(), e.getY());
      }

      @Override
      public void paintComponent(Graphics g) {
            g.setColor(getBackground());
            g.fillRect(0,0, getWidth(), getHeight());
            for(GraphicVertex v : vertices) {
                  for(GraphicEdge e : v.getEdges()) {
                        e.draw(g);
                  }
            }
            for(GraphicVertex v : vertices) {
                  v.draw(g);
            }
      }

      private class MouseHandler extends MouseAdapter {
            @Override
            public void mouseReleased(MouseEvent e) {
                  if(selected != null) {
                        selected.setSelected(false);
                        prevSelectedName = selected.getName();
                        selected = null;
                  }
                  e.getComponent().repaint();
            }
            @Override
            public void mousePressed(MouseEvent e) {
                  mouseP = e.getPoint();
                  for(GraphicVertex v : vertices) {
                        if(v.contains(mouseP)) {
                              selected = v;
                              selected.setSelected(true);
                              break;
                        }
                  }
                  if(e.getButton() == MouseEvent.BUTTON3 &&
vertices.size() != 0) {
                        doPop(e);
                  }

                  e.getComponent().repaint();
            }
      }

      private class MouseMotionHandler extends MouseMotionAdapter {
            Point delta = new Point();

            @Override
            public void mouseDragged(MouseEvent e) {
                  if(selected != null) {
                        delta.setLocation(e.getX() - mouseP.x, e.getY() -
mouseP.y);
                        e.getComponent().repaint();
                        selected.updatePosition(delta);
```

```
                    mouseP = e.getPoint();
                }
                e.getComponent().repaint();
            }
        }

        class PopupMenu extends JPopupMenu {
            GraphPanel graphPanel;
            JMenu addVertexMenu = new JMenu("Add vertex...");
            JMenu addEdgeMenu = new JMenu("Add edge...");
            JMenuItem deleteVertexItem = new JMenuItem("Delete vertex");
            JMenu deleteEdgeMenu = new JMenu("Delete edge...");

            JTextField addVertexName = new JTextField(1);
            JList<String> addEdgeList = new JList<>();
            JTextField addEdgeCap = new JTextField(1);
            JList<String> deleteEdgeList = new JList<>();

            PopupMenu(GraphPanel parent) {
                graphPanel = parent;

                Font font = new Font("TimesNewRoman", Font.PLAIN, 11);
                addVertexMenu.setFont(font);
                addEdgeMenu.setFont(font);
                deleteVertexItem.setFont(font);
                deleteEdgeMenu.setFont(font);

                JLabel addVertexNameLabel = new JLabel("Name:");
                addVertexNameLabel.setFont(font);
                addVertexMenu.add(addVertexNameLabel);
                addVertexMenu.add(addVertexName);
                addEdgeMenu.add(addEdgeList);
                JLabel addEdgeNameLabel = new JLabel("Capacity:");
                addEdgeNameLabel.setFont(font);
                addEdgeMenu.add(addEdgeNameLabel);
                addEdgeMenu.add(addEdgeCap);
                deleteEdgeMenu.add(deleteEdgeList);

                add(addVertexMenu);
                add(addEdgeMenu);
                add(deleteVertexItem);
                add(deleteEdgeMenu);

                GroupLayout layout = new GroupLayout(this);
                setLayout(layout);



        layout.setHorizontalGroup(layout.createParallelGroup(GroupLayout.Al
ignment.LEADING)
                                .addComponent(addVertexMenu)
                                .addComponent(addEdgeMenu)
                                .addComponent(deleteVertexItem)
                                .addComponent(deleteEdgeMenu)
                );
```

```
                layout.linkSize(SwingConstants.HORIZONTAL, addVertexMenu,
addEdgeMenu, deleteVertexItem, deleteEdgeMenu);

                layout.setVerticalGroup(layout.createSequentialGroup()
                           .addComponent(addVertexMenu)
                           .addComponent(addEdgeMenu)
                           .addComponent(deleteVertexItem)
                           .addComponent(deleteEdgeMenu)
                );

                addVertexName.addActionListener(new ActionListener() {
                      @Override
                      public void actionPerformed(ActionEvent e) {
                           if (addVertexName.getText().length() == 1 ) {

      graphPanel.getFrame().getController().restoreGraph();
                                graphPanel.getFrame()
                                           .getController()

      .addVertex(addVertexName.getText().charAt(0), getMouseP());
                           }
                      }
                });
                addVertexName.addKeyListener(new KeyAdapter() {
                      @Override
                      public void keyTyped(KeyEvent e) {
                           if (addVertexName.getText().length() >= 1 ) {
                                e.consume();
                           }
                      }
                });

                addEdgeMenu.addMenuListener(new MenuListener() {
                      @Override
                      public void menuSelected(MenuEvent e) {
                           String[] arr = new
String[graphPanel.getVertices().size()];
                           for(int i=0;
i<graphPanel.getVertices().size(); ++i) {

      if(graphPanel.getVertices().get(i).getName() == prevSelectedName)
                                      continue;
                                arr[i] =
String.valueOf(graphPanel.getVertices().get(i).getName());
                           }
                           addEdgeList.setListData(arr);
                      }

                      @Override
                      public void menuDeselected(MenuEvent e) {

                      }

                      @Override
                      public void menuCanceled(MenuEvent e) {

                      }
```

```
                });
                addEdgeCap.addActionListener(new ActionListener() {
                      @Override
                      public void actionPerformed(ActionEvent e) {

      graphPanel.getFrame().getController().restoreGraph();

      graphPanel.getFrame().getController().addEdge(
                                        prevSelectedName,

      addEdgeList.getSelectedValue().charAt(0),

      Integer.parseInt(addEdgeCap.getText()),
                                        0
                                  );
                      }
                });
                addEdgeCap.addKeyListener(new KeyAdapter() {
                      @Override
                      public void keyTyped(KeyEvent e) {
                           if(e.getKeyChar() < 48 || e.getKeyChar() > 57)
                                 e.consume();
                            else if (addEdgeCap.getText().length() >= 2)
                                 e.consume();
                            else if(addEdgeCap.getText().length() == 1)
                            {
                                 if(addEdgeCap.getText().charAt(0) == 48)
{
                                       addEdgeCap.setText("");
                                 }
                            }
                      }
                });

                deleteVertexItem.addActionListener(new ActionListener()
{
                      @Override
                      public void actionPerformed(ActionEvent e) {
                            if(prevSelectedName ==
graphPanel.getVertices().get(0).getName()) {
                                 graphPanel.getFrame().print("Unable to
delete source.");
                                 return;
                            }
                            if(prevSelectedName ==
graphPanel.getVertices().get(1).getName()) {
                                 graphPanel.getFrame().print("Unable to
delete sink.");
                                 return;
                            }

      graphPanel.getFrame().getController().restoreGraph();

      graphPanel.getFrame().getController().deleteVertex(prevSelectedName
);
                            prevSelectedName = '\0';
                      }
```

```
                });

                deleteEdgeMenu.addMenuListener(new MenuListener() {
                      @Override
                      public void menuSelected(MenuEvent e) {
                            GraphicVertex v =
graphPanel.findVertex(prevSelectedName);
                            String[] arr = new
String[v.getEdges().size()];
                            for(int i=0; i<v.getEdges().size(); ++i) {
                                  arr[i] =
String.valueOf(v.getEdges().get(i).getTo().getName());
                            }
                            deleteEdgeList.setListData(arr);
                      }

                      @Override
                      public void menuDeselected(MenuEvent e) {

                      }

                      @Override
                      public void menuCanceled(MenuEvent e) {

                      }
                });
                deleteEdgeList.addMouseListener(new MouseListener() {
                      @Override
                      public void mouseClicked(MouseEvent e) {

      graphPanel.getFrame().getController().restoreGraph();
                            graphPanel.getFrame()
                                        .getController()
                                        .deleteEdge(
                                                prevSelectedName,

      deleteEdgeList.getSelectedValue().charAt(0)
                                        );
                      }

                      @Override
                      public void mousePressed(MouseEvent e) {

                      }

                      @Override
                      public void mouseReleased(MouseEvent e) {

                      }

                      @Override
                      public void mouseEntered(MouseEvent e) {

                      }

                      @Override
                      public void mouseExited(MouseEvent e) {
```

```
                                }
                            });
                        }
                    }
                }
```