**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

По лабораторному практикуму

Тема: Поиск минимального остовного дерева, с помощью алгоритма Борувки.

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6383 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Никитин К.В. |
| гр. 6383 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Спас А.А. |
| гр.6383  Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Карамышев А.О.  Ефремов М.А. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Никитин К.В. группы 6383 | | |
| Студент Спас А.А. группы 6383 | | |
| Студент Карамышев А.О. группы 6383  Тема практики: Нахождение минимального остовного дерева с помощью алгоритма Борувки | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма Борувки на Java с графическим интерфейсом. | | |
| Сроки прохождения практики: 27.06.2018 – 10.07.2018 | | |
| Дата сдачи отчета: | | |
| Дата защиты отчета: | | |
|  | | |
| Студент гр.6383 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Никитин К.В. |
| Студент гр.6383 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Спас А.А. |
| Студент гр.6383 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Карамышев А.О. |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Ефремов М.А. |

**Аннотация**

В данной работе рассмотрена программа, которая ищет минимальное остовное дерево с помощью алгоритма Борувки. Программа разработана в среде IntelliJ IDEA. Язык разработанной программы Java. Программа подробно показывает процесс нахождения минимального остовного дерева.

**Summary**

In this paper, we consider a program that searches for a minimal spanning tree using the Borovka algorithm. The program is developed in the IntelliJ IDEA environment. The language of the developed Java program. The program details the process of finding the minimal spanning tree.

**Введение.**

**Формулировка задания.** Требуется разработать программу, поиск минимального остовного дерева с помощью алгоритма Борувки. При этом должен присутствовать графический интерфейс.

**Алгоритм Борувки** (англ. *Borůvka's algorithm*) — алгоритм поиска [минимального остовного дерева](https://neerc.ifmo.ru/wiki/index.php?title=Остовные_деревья:_определения,_лемма_о_безопасном_ребре) во взвешенном неориентированном связном графе. Впервые был опубликован в 1926 году Отакаром Борувкой.

Алгоритм состоит из нескольких шагов:

1. Изначально каждая вершина графа G— тривиальное дерево, а ребра не принадлежат никакому дереву.
2. Для каждого дерева T найдем минимальное инцидентное ему ребро. Добавим все такие ребра.
3. Повторяем шаг 2 пока в графе не останется только одно дерево T.

Данный алгоритм может работать неправильно, если в графе есть ребра равные по весу. Например, полный граф из трех вершин, вес каждого ребра равен один. В T могут быть добавлены все три ребра. Избежать эту проблему можно, например, выбирая в первом пункте среди ребер, равных по весу, ребро с наименьшим номером.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 4 |
| 1. | Требования к программе | 6 |
| 2. | План разработки и распределение ролей в бригаде | 6 |
| 2.1. | План разработки | 6 |
| 2.2. | Распределение ролей в бригаде | 6 |
| 3. | Особенности реализации | 6 |
| 3.1. | Описание классов | 6 |
| 3.2. | Алгоритм работы | 9 |
| 3.3  3.4 | Входные/выходные данные  Описание интерфейса | 9  9 |
| 4. | Тестирование | 10 |
| 4.1 | Тестирование графического интерфейса | 10 |
| 4.2 | Тестирование кода алгоритма | 10 |
|  | Заключение | 10 |
|  | Список использованных источников | 10 |
|  | Приложение А. Код программы. | 11 |
|  | Приложение Б. UML - диаграмма | 29 |

1. **Требование к программе**

Данная программа должна корректно работать на всевозможных входных данных и выводить минимальное остовное дерево.

Программа должна подробно демонстрировать результат работы алгоритма и пошагово показывать его выполнение.

1. **План разработки и распределение ролей в бригаде**
   1. **План разработки:**
   2. **Распределение ролей в бригаде:**

Спас Артём: Визуализация алгоритма.

Карамышев Альберт: Архитектура программы.

Никитин Кирилл: Тестирование, написание отчёта.

1. **Особенности реализации**

**3.1. Описание классов:**

**Canvas.java**

**(DESCRIPTOIN)**

|  |  |
| --- | --- |
| public void setContent(Graph data) |  |
| public void setMST(Graph data) |  |
| public void checkBox() |  |

**Coord.java**

**(DESCRIPTION)**

**Edge.java**

**(DESCRIPTION)**

|  |  |
| --- | --- |
| public double weight() |  |
| public int either() |  |
| public int other() |  |
| public int compareTo(Edge that) |  |
| public boolean equals(Edge that) |  |
| public String toString() |  |

**Graph.java**

**(DESCRIPTION)**

|  |  |
| --- | --- |
| public int V() |  |
| public int E() |  |
| public void addEdge(Edge e) |  |
| public Collection<Edge> adj(int v) |  |
| public Collection<Edge> edges() |  |
| public String toString() |  |

**InputOutput.java**

**(DESCRIPTION)**

|  |  |
| --- | --- |
| public Graph getData(Graph g, BufferedReader cin, BufferedReader cinV) throws IOException |  |

**Launcher.java**

**(DESCRIPTION)**

**????**

**MST.java**

**(DESCRIPTION)**

|  |  |
| --- | --- |
| public void build() |  |
| public void buildStep() |  |
| public Graph MSTtoGraph() |  |
| public boolean equal(MST x) |  |
| private boolean less(Edge e, Edge f) |  |
| public Collection<Edge> edges() |  |
| public double weight() |  |
| public String toString() |  |

**UnionField.java**

**(DESCRIPTION)**

|  |  |
| --- | --- |
| public int count() |  |
| public boolean connected(int p, int q) |  |
| public int find(int p) |  |
| public void unoin(int p, int q) |  |

**VGraph.java**

**(DESCRIPTION)**

|  |  |
| --- | --- |
| public void paint(Graphics g) |  |
| boolean attend(HashMap<String, Object> e) |  |
| private void addVertex(String name) |  |
| private void addEdge(String fromName, String toName, double weight) |  |
| private void addMstEdge(String fromName, String toName, double weight) |  |
| private HashMap<String,Object> vertexLookup(String name) |  |
| private void reposition() |  |
| public void recolor(int id, Color c) |  |
| public void SetMST(MST original) |  |
| public VGraph(Graph original) |  |
| public HashMap<String, Object> find(Point2D p) |  |
| private class MyMouse extends MouseAdapter |  |
| private class MyMove implements MouseMotionListener |  |

**3.2.Алгоритм работы:**

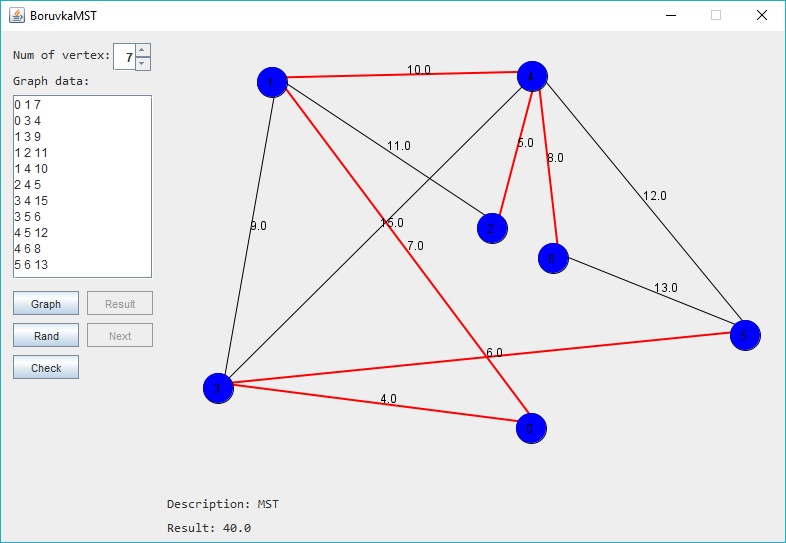
**NAPISHITE PLEASE**

**3.3. Входные/выходные данные:**

На *вход* пользователь вводит количество вершин и все пути в графе.

На *выходе* пользователь получит минимальное остовное дерево.

**3.4. Описание интерфейса:**



Пользовательский интерфейс состоит из :

* Поле ввода количества вершин
* Поле ввода путей в графе
* Кнопка Graph, которая строит граф
* Кнопка Result, которая выводит на экран минимальное остовное дерево
* Кнопка Rand, которая генерирует случайно дерево
* Кнопка Next, которая пошагово ищет минимальное остовное дерево
* Кнопка Check, которая включает/отключает отображение весов

**4. Тестирование**

**4.1. Тестирование графического интерфейса**

Все кнопки нашего графического интерфейса работают корректно. В ходе тестирования никаких ошибок выявлено не было.

**4.2. Тестирование кода алгоритма**

**IN PROGRESS**

**Заключение:**

В ходе данной учебной практики были изучены основы программирования на языке Java, пройден интерактивный курс «Java.Базовый курс». После чего была разработана программа, которая с помощью алгоритма Борувки находит минимальное остовное дерево, наглядно демонстрируя его пользователю.

Программа соответствует требованиям спецификации и прошла проверку на всевозможных тестах, никаких ошибок выявлено не было.

**Список использованных источников:**

1. Java Базовый курс.URL: stepik.org/course/Java-Базовый-курс-187/syllabus

**Приложение А.**

**Код программы.**

**Canvas.java**

import java.awt.\*;

import javax.swing.\*;

public class Canvas extends JPanel {

private VGraph content;

private boolean wght = false;

private static final Color[] compColor = {Color.white, Color.blue, Color.green, Color.yellow, Color.magenta,

Color.cyan, Color.gray, Color.pink, Color.darkGray, Color.red};

//конструктор

public Canvas() {

content = null;

this.setLayout(null);

}

//обновление содержимого

public void setContent(Graph data) {

if (content != null) {

content.setVisible(false);

}

content = new VGraph(data);

this.add(content);

this.revalidate();

this.repaint();

}

public void setMST(Graph data){

content.SetMST(new MST(data));

this.add(content);

this.revalidate();

this.repaint();

}

public void checkBox(){

wght = wght ? false : true;

content.wght = wght;

this.add(content);

this.revalidate();

this.repaint();

}

}

**Coord.java**

public class Coord {

public int x;

public int y;

public Coord(int x, int y) {

this.x = x;

this.y = y;

}

public Coord(Coord old) {

this.x = old.x;

this.y = old.y;

}

}

**Edge.java**

//package com.etu.practice.graph;

public class Edge implements Comparable<Edge>{

private final int v;

private final int w;

private final double weight;

public Edge(int v, int w, double weight){

this.v=v;

this.w=w;

this.weight=weight;

}

public double weight(){ return weight;}

public int either(){ return v;}

public int other(int vertex){

if(vertex == v) return w;

else if(vertex == w) return v;

else throw new RuntimeException("Недопустимое ребро");

}

@Override

public int compareTo(Edge that) {

if(this.weight()< that.weight()) return -1;

else if(this.weight()>that.weight()) return 1;

else return 0;

}

public boolean equals(Edge that){

if(this.compareTo(that)==0 && this.either()==that.either() && this.other(this.either())==that.other(that.either())){

return true;

}

return false;

}

@Override

public String toString() {

return String.format("%d-%d %.2f", v, w, weight);

}

}

**Graph.java**

**//package com.etu.practice.graph;**

import java.util.Collection;

import java.util.LinkedList;

public class Graph {

private final int V;

private int E;

private LinkedList<Edge> [] adj;

public Graph(int V){

this.V=V;

this.E=0;

adj = (LinkedList<Edge>[]) new LinkedList[V];

for(int i=0; i<V; i++)

adj[i]=new LinkedList<>();

}

public int V(){return V;}

public int E(){return E;}

public void addEdge(Edge e){

adj[e.either()].add(e);

adj[e.other(e.either())].add(e);

}

public Collection<Edge> adj(int v){

return adj[v];

}

public Collection<Edge> edges(){

LinkedList<Edge> a= new LinkedList<Edge>();

for(int v=0; v<V; v++){

for(Edge e : adj[v]) {

if (e.other(v) > v) {

a.add(e);

}

}

}

return a;

}

public String toString(){

String str = "";

for(Edge e: edges()){

str += e.toString()+'\n';

}

return str;

}

}

**InputOutput.java**

**import java.io.BufferedReader;**

import java.io.IOException;

import java.io.PrintWriter;

import java.util.StringTokenizer;

public class InputOutput {

private PrintWriter cout=new PrintWriter(System.out);

//ввод данных с читателя cin

public Graph getData(Graph g, BufferedReader cin, BufferedReader cinV) throws IOException {//передаем ссылку на граф, в который надо считать данные,

StringTokenizer tokenizer = new StringTokenizer(cinV.readLine());

g = new Graph(Integer.parseInt(tokenizer.nextToken())); //считываем количество вершин графа

//g.numE = Integer.parseInt(tokenizer.nextToken()); //считываем количество ребер графа

String s;

while((s=cin.readLine())!=null){

tokenizer = new StringTokenizer(s);

int v = Integer.parseInt(tokenizer.nextToken());

int w = Integer.parseInt(tokenizer.nextToken());

double weight = Double.parseDouble(tokenizer.nextToken());

g.addEdge(new Edge(v, w, weight));

}

return g;

}

}

**Launcher.java**

import javax.swing.\*;

import java.awt.\*;

import java.io.BufferedReader;

import java.io.File;

import java.io.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.StringReader;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class Launcher extends JFrame{

private JLabel labelWeight;

private JSpinner spinner;

private JLabel descLabel;

private JLabel resLabel;

private JLabel label;

private JTextArea dataGraph;

private JButton buttonGraph;

private JButton buttonResult;

private JButton buttonNext;

private JButton buttonRand;

private JButton buttonCheck;

private Canvas canvas;

//Graphics g;

private InputOutput ios;

private Graph gr;

private MST mst;

private JPanel rootPanel;

// ключевое слово super, которое обозначает суперкласс, т.е. класс, производным от которого является текущий класс

public Launcher() {

this.setSize(800,550);

this.setResizable(false);

this.setMinimumSize(new Dimension(800,550));

this.setTitle("BoruvkaMST");

this.rootPanel = new JPanel();

rootPanel.setLayout(null); //абсолютное позиционирование

rootPanel.setBounds(0,0,800,550);

//this.setBounds(100,100,450,400);

setContentPane(rootPanel);

Font font = new Font("Consolas", 0, 13);

this.spinner = new JSpinner();

this.spinner.setValue(7);

this.spinner.setBounds(112,12,38,28);

this.spinner.setVisible(true);

this.labelWeight = new JLabel();

this.labelWeight.setText("Num of vertex:");

this.labelWeight.setFont(font);

this.labelWeight.setBounds(12,18,108,14 );

this.labelWeight.setVisible(true);

this.label = new JLabel();

this.label.setText("Graph data:");

this.label.setFont(font);

this.label.setBounds(12,44,85,14);

this.label.setVisible(true);

//кнопки управления

this.buttonGraph = new JButton("Graph");

this.buttonResult = new JButton("Result");

this.buttonNext = new JButton("Next");

this.buttonRand = new JButton("Rand");

this.buttonCheck = new JButton("Check");

Font fontButton = new Font(null, 0, 11);

this.buttonGraph.setFont(fontButton);

this.buttonResult.setFont(fontButton);

this.buttonNext.setFont(fontButton);

this.buttonRand.setFont(fontButton);

this.buttonCheck.setFont(fontButton);

//устанавливаем размеры кнопок

this.buttonGraph.setBounds(12,260,66,24);

this.buttonResult.setBounds(86,260,66,24);

this.buttonRand.setBounds(12,292,66,24);

this.buttonNext.setBounds(86,292,66,24);

this.buttonCheck.setBounds(12,324,66,24);

//убираем выделение текста кнопки

this.buttonGraph.setFocusPainted(false);

this.buttonResult.setFocusPainted(false);

this.buttonNext.setFocusPainted(false);

this.buttonRand.setFocusPainted(false);

this.buttonCheck.setFocusPainted(false);

//поясняющие надписи

this.descLabel = new JLabel();

this.descLabel.setBounds(166,450,418,48);

this.descLabel.setText("Description: ");

this.descLabel.setFont(font);

this.resLabel = new JLabel();

this.resLabel.setBounds(166,490,418,16);

this.resLabel.setText("Result: -");

this.resLabel.setFont(font);

//строка для указания кол-ва вершин

this.dataGraph = new JTextArea("");

this.dataGraph.setBounds(12,64,140,1840);

this.dataGraph.setText("0 1 7\n" +

"0 3 4\n" +

"1 3 9\n" +

"1 2 11\n" +

"1 4 10\n" +

"2 4 5\n" +

"3 4 15\n" +

"3 5 6\n" +

"4 5 12\n" +

"4 6 8\n" +

"5 6 13");

JScrollPane scroll = new JScrollPane(dataGraph);

scroll.setBounds(12,64,140,184);

scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL\_SCROLLBAR\_AS\_NEEDED);

scroll.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

this.canvas = new Canvas();

this.canvas.setBounds(166,0,634,460);

this.canvas.setVisible(true);

this.rootPanel.add(this.canvas);

//устанавливаем виимость всех объектов

this.buttonGraph.setVisible(true);

this.buttonResult.setVisible(true);

this.buttonNext.setVisible(true);

this.buttonRand.setVisible(true);

this.buttonCheck.setVisible(true);

this.descLabel.setVisible(true);

this.resLabel.setVisible(true);

scroll.setVisible(true);

//добавляем объекты на панель

this.rootPanel.add(this.buttonGraph);

this.rootPanel.add(this.buttonResult);

this.rootPanel.add(this.buttonNext);

this.rootPanel.add(this.buttonRand);

this.rootPanel.add(this.buttonCheck);

this.rootPanel.add(label);

this.rootPanel.add(scroll);

this.rootPanel.add(this.descLabel);

this.rootPanel.add(this.resLabel);

this.rootPanel.add(this.spinner);

this.rootPanel.add(this.labelWeight);

buttonResult.setEnabled(false);

buttonNext.setEnabled(false);

buttonCheck.setEnabled(false);

rootPanel.setVisible(true);

ios = new InputOutput();

//инициализация графа

buttonGraph.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent ev) {

try {

gr = ios.getData(gr, new BufferedReader(new StringReader(dataGraph.getText())),

new BufferedReader(new StringReader(spinner.getValue().toString())));

mst = new MST(gr);

canvas.setContent(gr);

descLabel.setText("Description: Graph is constructed");

resLabel.setText("Result: -");

buttonResult.setEnabled(true);

buttonNext.setEnabled(true);

buttonGraph.setEnabled(false);

buttonCheck.setEnabled(true);

} catch (Exception e) {

descLabel.setText("Description: exception! "+e.getClass().getName()+": "+e.getMessage());

}

}

});

buttonCheck.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ev) {

canvas.checkBox();

}

});

buttonNext.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ev) {

try {

MST mstEnd = new MST(gr);

mstEnd.build();

if(mstEnd.equal(mst)){

buttonResult.setEnabled(false);

buttonNext.setEnabled(false);

buttonGraph.setEnabled(true);

resLabel.setText("Result: "+ mst.weight());

}

else{

resLabel.setText("Result: step by step...");

}

canvas.setMST(mst.MSTtoGraph());

mst.buildStep();

descLabel.setText("Description: MST");

} catch (Exception e) {

descLabel.setText("Description: exception! "+e.getClass().getName()+": "+e.getMessage());

}

}

});

buttonResult.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent ev) {

canvas.setMST(gr);

descLabel.setText("Description: MST");

resLabel.setText("Result: "+ mst.weight());

buttonResult.setEnabled(false);

buttonNext.setEnabled(false);

buttonGraph.setEnabled(true);

}

});

setVisible(true);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

new Launcher();

}

}

/\*

7

0 1 7

0 3 4

1 3 9

1 2 11

1 4 10

2 4 5

3 4 15

3 5 6

4 5 12

4 6 8

5 6 13

2

0 1 100

\*/

**MST.java**

import java.util.Collection;

import java.util.LinkedList;

public class MST {

private final int V;

private LinkedList<Edge> mst;

private UnionField uf;

private Graph g;

private double weidht=0;

public int V(){

return V;

}

public MST(Graph G){

g = G;

V = G.V();

uf = new UnionField(G.V());

mst = new LinkedList<Edge>();

}

public void build(){

for(int i=1; i<g.V() && mst.size()<g.V()-1; i=i+i){

this.buildStep();

}

}

public void buildStep(){

Edge[] close=new Edge[g.V()];

for(Edge e : g.edges()){

int v= e.either(), w=e.other(v);

int vuf=uf.find(v), wuf=uf.find(w);

if(vuf==wuf) continue;

if(close[vuf]==null || less(e, close[vuf])) close[vuf] = e;

if(close[wuf]==null || less(e, close[wuf])) close[wuf] = e;

}

for(int j=0; j<g.V(); j++){

Edge e = close[j];

if(e!=null){

int v =e.either(), w=e.other(v);

if(!uf.connected(v, w)){

mst.add(e);

weidht += e.weight();

uf.unoin(v, w);

}

}

}

};

public Graph MSTtoGraph(){

Graph gr = new Graph(V());

if(mst != null){

for(Edge e : mst){

gr.addEdge(e);

}

}

return gr;

}

public boolean equal(MST x){

if(mst.size() != x.edges().size()) return false;

for(int i=0; i<mst.size() ; i++) {

Edge e = mst.get(i);

LinkedList<Edge> f =(LinkedList<Edge>) x.edges();

Edge f1 = f.get(i);

if (!e.equals(f1)) return false;

}

return true;

}

private boolean less(Edge e, Edge f) {

return e.weight() < f.weight();

}

public Collection<Edge> edges(){

return mst;

}

public double weight(){

return weidht;

}

public String toString(){

String str = "";

for(Edge e: mst){

str += e.toString()+'\n';

}

str += weidht;

return str;

}

}

**UnionField.java**

//package com.etu.practice.graph;

public class UnionField {

private int[] id;

private int count;

public UnionField(int N){

count = N;

id = new int[N];

for(int i=0; i<N; i++)id[i]=i;

}

public int count(){ return count;}

public boolean connected(int p, int q){

return find(p)==find(q);

}

public int find(int p){

return id[p];

}

public void unoin(int p, int q){

int ptr=find(p), qtr=find(q);

if(find(p)== find(q)) return;

for(int i=0; i<id.length; i++){

if(id[i] == ptr) id[i] = qtr;

}

}

}

**Vgraph.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.MouseMotionListener;

import java.awt.geom.Ellipse2D;

import java.awt.geom.Line2D;

import java.awt.geom.Point2D;

import java.util.\*;

import java.lang.Math;

//объект, превращающий абстрактный граф в его плоскостное представление

public class VGraph extends JPanel {

public ArrayList<HashMap<String,Object>> vertices;

public ArrayList<HashMap<String,Object>> edges;

public ArrayList<HashMap<String,Object>> mstEdges;

public boolean wght = false;

public HashMap<String, Object> current;

//public ArrayList<HashMap<String, Object>> weightes;

public void paint(Graphics g) {

super.paint(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint ( RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON );

for (HashMap<String,Object> e: edges) {

if(attend(e)){

g2d.setStroke(new BasicStroke(2));

g2d.setColor(Color.RED);

}

g2d.draw((Line2D) e.get("component"));

g2d.setStroke(new BasicStroke());

g2d.setColor(Color.BLACK);

if(wght) {

Coord From = (Coord) e.get("posFrom");

Coord To = (Coord) e.get("posTo");

g2d.drawString(e.get("weight").toString(), (To.x - From.x) / 2 + From.x, (To.y - From.y) / 2 + From.y);

}

}

//Работа с цветом линии/фигуры

// Запоминаем исходный цвет;

Color oldColor = g.getColor();

//рисование кружочков для вершин графа

for (int i = 0; i<vertices.size(); ++i) { //проходим по всем вершинам

Coord vertC = (Coord) vertices.get(i).get("pos"); //координата вершины

g2d.draw((Ellipse2D) vertices.get(i).get("component"));

//если надо закрасить кружочки:

Color newColor = (Color) vertices.get(i).get("color");

// Устанавливаем новый цвет;

g2d.setColor(newColor);

g2d.fill((Ellipse2D) vertices.get(i).get("component"));

// Восстанавливаем исходный цвет;

g2d.setColor(oldColor);

g2d.drawString((String) vertices.get(i).get("name"), vertC.x-10, vertC.y+10);

}

}

boolean attend(HashMap<String, Object> e){

if(mstEdges==null) return false;

for(HashMap<String, Object> er : mstEdges){

if(e.get("from").toString().equals(er.get("from").toString()) &&

e.get("to").toString().equals(er.get("to").toString())) return true;

}

return false;

}

private void addVertex(String name) {

HashMap<String, Object> vertex = new HashMap<>(5);

vertex.put("name", name);

vertex.put("color",Color.blue);

vertex.put("pos", new Coord(0,0));

vertex.put("component", new Ellipse2D.Double(0, 0, 30, 30));

vertices.add(vertex);

}

private void addEdge(String fromName, String toName, double weight) {

HashMap<String, Object> edge = new HashMap<>(6);

edge.put("from", fromName);

edge.put("to", toName);

edge.put("posFrom", new Coord(0,0));

edge.put("posTo", new Coord(0,0));

edge.put("component", new Line2D.Double(0,0,0,0));

edge.put("weight", weight);

edges.add(edge);

}

private void addMstEdge(String fromName, String toName, double weight){

HashMap<String, Object> edge = new HashMap<>(6);

edge.put("from", fromName);

edge.put("to", toName);

edge.put("weight", weight);

mstEdges.add(edge);

}

private HashMap<String,Object> vertexLookup(String name) {

for (int i = 0; i<vertices.size(); ++i) {

HashMap<String,Object> cur = vertices.get(i);

if (cur.get("name").equals(name)) return cur;

}

return null;

}

private void reposition() {

for (int i = 0; i<vertices.size(); ++i) {

HashMap<String,Object> cur = vertices.get(i);

Coord place = new Coord(314+(int)(150\*Math.cos(6.28/vertices.size()\*i)),220+(int)(150\*Math.sin(6.28/vertices.size()\*i)));

cur.replace("pos", place);

Ellipse2D.Double v = (Ellipse2D.Double) cur.get("component");

v.setFrame(place.x-20, place.y-10, 30, 30);

}

for (int i = 0; i<edges.size(); ++i) {

HashMap<String,Object> cur = edges.get(i);

HashMap<String,Object> v = vertexLookup((String) cur.get("from"));

Coord fromC = new Coord((Coord) v.get("pos"));

v = vertexLookup((String) cur.get("to"));

Coord toC = new Coord((Coord) v.get("pos"));

Line2D line = (Line2D) cur.get("component");

line.setLine(fromC.x, fromC.y, toC.x, toC.y);

cur.replace("posFrom",fromC);

cur.replace("posTo",toC);

}

}

public void recolor(int id, Color c) {

vertices.get(id).replace("color",c);

}

public void SetMST(MST original){

original.build();

LinkedList<Edge> tmp = new LinkedList<>(original.edges()) ;

mstEdges = new ArrayList<>(original.edges().size());

for(Edge e : tmp){

this.addMstEdge(String.valueOf(e.either()), String.valueOf(e.other(e.either())), e.weight());

}

}

public VGraph(Graph original) {

this.setBounds(0,0,634, 460);

this.setLayout(null);

addMouseListener(new MyMouse());

addMouseMotionListener(new MyMove());

vertices = new ArrayList<>(original.V());

edges = new ArrayList<>(original.edges().size());

LinkedList<Edge> tmp = new LinkedList<>(original.edges()) ;

for (int i = 0; i<original.V(); ++i) {

this.addVertex( String.valueOf(i));

}

for(Edge e : tmp){

this.addEdge(String.valueOf(e.either()), String.valueOf(e.other(e.either())), e.weight());

}

reposition();

this.revalidate();

this.repaint();

}

/\*public VGraph(MST original) {

this.setBounds(0,0,634, 460);

this.setLayout(null);

addMouseListener(new MyMouse());

addMouseMotionListener(new MyMove());

vertices = new ArrayList<>(original.V());

edges = new ArrayList<>(original.edges().size());

LinkedList<Edge> tmp = new LinkedList<>(original.edges()) ;

for (int i = 0; i<original.V(); ++i) {

this.addVertex( String.valueOf(i));

}

for(Edge e : tmp){

this.addEdge(String.valueOf(e.either()), String.valueOf(e.other(e.either())), e.weight());

}

reposition();

this.revalidate();

this.repaint();

}\*/

public HashMap<String, Object> find(Point2D p){

for(HashMap<String,Object> v: vertices){

Ellipse2D tmp = (Ellipse2D) v.get("component");

if(tmp.contains(p)) return v;

}

return null;

}

private class MyMouse extends MouseAdapter{

@Override

public void mousePressed(MouseEvent e) {

current = find(e.getPoint());

}

}

private class MyMove implements MouseMotionListener {

@Override

public void mouseMoved(MouseEvent e) {

if(find(e.getPoint()) == null)

setCursor(Cursor.getDefaultCursor());

else

setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT\_CURSOR));

}

@Override

public void mouseDragged(MouseEvent e) {

if(current != null){

Ellipse2D tmp = (Ellipse2D) current.get("component");

Coord crd = new Coord(e.getX()+10,e.getY());

if(crd.x<20) crd.x = 20;

if(crd.y<20) crd.y = 20;

if(crd.x>610) crd.x = 610;

if(crd.y>440) crd.y = 440;

tmp.setFrame(crd.x-20, crd.y-10,30,30);

current.replace("pos", crd);

for(HashMap<String, Object> ed : edges){

if(ed.get("from").toString().equals(current.get("name").toString())){

Coord f = (Coord) current.get("pos");

Coord t = (Coord) ed.get("posTo");

ed.replace("posFrom", f);

Line2D line = (Line2D) ed.get("component");

line.setLine(f.x, f.y, t.x, t.y);

}

if(ed.get("to").toString().equals(current.get("name").toString())){

Coord f = (Coord) ed.get("posFrom");

Coord t = (Coord) current.get("pos");

ed.replace("posTo", t);

Line2D line = (Line2D) ed.get("component");

line.setLine(f.x, f.y, t.x, t.y);

}

}

repaint();

}

}

}

}

**Приложение Б.**

**UML – диаграмма**

