**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математической кибернетики и информационных технологий

**Отчет по лабораторной работе**

по предмету «СИАОД»

на тему:

«Сетевые алгоритмы. Динамические алгоритмы поиска путей»

Выполнил: студент группы

Клюшкин Дмитрий Алексеевич

Руководитель:

Кутейников Иван Александрович

Москва 2020

*Цель работы:* Реализовать алгоритм поиска кратчайшего расстояния между двумя вершинами ориентированного взвешенного графа в соответствии с вариантом. Предусмотреть задание графа в виде матрицы смежности/инцидентности, читаемой из файла, либо графически с помощью пользовательского интерфейса. Разработать графический интерфейс пользователя с визуализацией графа и отображением кратчайшего расстояния между задаваемыми пользователем вершинами.

*Вариант:* 14. Алгоритм Дейкстры

*Ход работы:*

*Код программы:*

import javax.swing.\*;  
import java.awt.\*;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Stack;  
  
  
import static java.util.Arrays.*fill*;  
  
public class Dejkstry extends JFrame{  
  
 static int *INF* = Integer.*MAX\_VALUE* / 2; // "Бесконечность"  
 static int *vNum*; // количество вершин  
 static MultiList *graph*; // описание графа  
 static Graphics *graphics*;  
 static int *size*=800;  
 static int[] *sp*;  
  
 //рисование  
 static ArrayList<Vertex> *vertex* = new ArrayList<>();  
 static ArrayList<Point> *point* = new ArrayList<>();  
  
  
 public Dejkstry(int size){  
 setTitle("Dejkstry");  
 setSize(size,size);  
 setVisible(true);  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 }  
  
  
 public static void main(String[] args) throws IOException {  
 Dejkstry dejkstry = new Dejkstry(*size*);  
 *graphics* = dejkstry.getGraphics();  
  
 BufferedReader reader = new BufferedReader(new FileReader("Input.txt"));  
 String str=reader.readLine();  
  
 *vNum* = Integer.*parseInt*(str.split(",")[0]);  
 int eNum = Integer.*parseInt*(str.split(",")[1]);  
 int start = Integer.*parseInt*(str.split(",")[2]);  
 int end = Integer.*parseInt*(str.split(",")[3]);  
  
 *graph* = new MultiList(*vNum*,eNum);  
 for (int i=0;i<eNum; i++){  
 str=reader.readLine();  
 *graph*.add(Integer.*parseInt*(str.split(",")[0]),Integer.*parseInt*(str.split(",")[1]),Integer.*parseInt*(str.split(",")[2]));  
 *vertex*.add(new Vertex(Integer.*parseInt*(str.split(",")[0]),Integer.*parseInt*(str.split(",")[1]),Integer.*parseInt*(str.split(",")[2])));  
 }  
  
 *dijkstraRMQ*(start,end);  
 *paintGraph*(*graphics*);  
 System.*out*.println();  
 }  
  
 public void paint(Graphics g) {  
  
 }  
  
 static void paintGraph(Graphics g){  
 int R = *size*/2-*size*/5; //радиус  
 int X = *size*/2, Y = *size*/2;//координаты центра  
  
 g.setFont(new Font("TimesRoman", Font.*PLAIN*, 18));  
  
 double angle = 360.0 / *vNum* ;  
 //цикл создающий массив из точек  
 for (int i=0;i< *vNum*;i++){  
 //x0,y0 - центр, a - угол, r - радиус.  
 int x=(int) (X + R \* Math.*cos*(Math.*toRadians*(angle\*i)));  
 int y=(int) (Y + R \* Math.*sin*(Math.*toRadians*(angle\*i)));  
 *point*.add(new Point(x,y));  
 }  
  
 for (Vertex v: *vertex*) {  
 int x1=*point*.get(v.start).x;  
 int y1=*point*.get(v.start).y;  
 int x2=*point*.get(v.end).x;  
 int y2=*point*.get(v.end).y;  
  
 g.setColor(Color.*BLACK*);  
 g.drawLine(x1,y1,x2,y2);  
 g.setColor(Color.*RED*);  
 g.drawString(String.*valueOf*(v.weight),(x1+x2)/2,(y1+y2)/2);  
 }  
  
 for(int i=0;i<*sp*.length-1;i++){  
 g.setColor(Color.*GREEN*);  
 int x1=*point*.get(*sp*[i]).x;  
 int y1=*point*.get(*sp*[i]).y;  
 int x2=*point*.get(*sp*[i+1]).x;  
 int y2=*point*.get(*sp*[i+1]).y;  
 g.drawLine(x1,y1,x2,y2);  
 }  
  
 int i=0;  
 for (Point p: *point*) {  
 g.setColor(Color.*WHITE*);  
 g.fillOval(p.x-25,p.y-25,50,50);  
 g.setColor(Color.*BLACK*);  
 g.drawOval(p.x-25,p.y-25,50,50);  
 g.setColor(Color.*RED*);  
 g.drawString(String.*valueOf*(i),p.x-5,p.y+5);  
  
 i++;  
 }  
 }  
  
  
 /\* Алгоритм Дейкстры за O(E log V) \*/  
 static void dijkstraRMQ(int start, int end) {  
 boolean[] used = new boolean[*vNum*]; // массив пометок  
 int[] prev = new int[*vNum*]; // массив предков  
 int[] dist = new int[*vNum*]; // массив расстояний  
 RMQ rmq = new RMQ(*vNum*); // RMQ  
  
 /\* Инициализация \*/  
 *fill*(prev, -1);  
 *fill*(dist, *INF*);  
 rmq.set(start, dist[start] = 0);  
  
 for (; ; ) {  
 int v = rmq.minIndex(); // выбираем ближайшую вершину  
 if (v == -1 || v == end) break; // если она не найдена, или является конечной, то выходим  
  
 used[v] = true; // помечаем выбранную вершину  
 rmq.set(v, *INF*); // и сбрасываем ее значение в RMQ  
  
 for (int i = *graph*.head[v]; i != 0; i = *graph*.next[i]) { // проходим пр смежным вершинам  
 int nv = *graph*.vert[i];  
 int cost = *graph*.cost[i];  
 if (!used[nv] && dist[nv] > dist[v] + cost) { // если можем улучшить оценку расстояния  
 rmq.set(nv, dist[nv] = dist[v] + cost); // улучшаем ее  
 prev[nv] = v; // помечаем предка  
 }  
 }  
 }  
  
 /\* Восстановление пути \*/  
 Stack stack = new Stack();  
 for (int v = end; v != -1; v = prev[v]) {  
 stack.push(v);  
 }  
 *sp* = new int[stack.size()];  
 for (int i = 0; i < *sp*.length; i++)  
 *sp*[i] = (int)stack.pop(); */\* Вывод результата \*/* System.*out*.printf("Кратчайшее расстояние между %d и %d = %d%n", start, end, dist[end]);  
 System.*out*.println("Кратчайший путь: " + Arrays.*toString*(*sp*));  
 }  
  
 /\* Класс списка с несколькими головами \*/  
 static class MultiList {  
 int[] head;  
 int[] next;  
 int[] vert;  
 int[] cost;  
 int cnt = 1;  
  
 MultiList(int vNum, int eNum) {  
 head = new int[vNum];  
 next = new int[eNum + 1];  
 vert = new int[eNum + 1];  
 cost = new int[eNum + 1];  
 }  
  
 void add(int u, int v, int w) {  
 next[cnt] = head[u];  
 vert[cnt] = v;  
 cost[cnt] = w;  
 head[u] = cnt++;  
 }  
 }

/\* Класс RMQ \*/  
 static class RMQ {  
 int n;  
 int[] val;  
 int[] ind;  
  
 RMQ(int size) {  
 n = size;  
 val = new int[2 \* n];  
 ind = new int[2 \* n];  
 *fill*(val, *INF*);  
 for (int i = 0; i < n; i++)  
 ind[n + i] = i;  
 }  
  
 void set(int index, int value) {  
 val[n + index] = value;  
 for (int v = (n + index) / 2; v > 0; v /= 2) {  
 int l = 2 \* v;  
 int r = l + 1;  
 if (val[l] <= val[r]) {  
 val[v] = val[l];  
 ind[v] = ind[l];  
 } else {  
 val[v] = val[r];  
 ind[v] = ind[r];  
 }  
 }  
 }  
  
 int minIndex() {  
 return val[1] < *INF* ? ind[1] : -1;  
 }  
 }  
  
 static class Vertex{  
 int start;  
 int end;  
 int weight;  
  
 Vertex(int start, int end, int weigth){  
 this.start=start;  
 this.end=end;  
 this.weight=weigth;  
 }  
 }  
}

*Результат программы:*

|  |  |  |  |
| --- | --- | --- | --- |
| *Число узлов* | *Число ребер* | *Время* | *Рисунок* |
| *5* | *10* | *0.022*  *сек.* |  |
| *7* | *20* | *0.022*  *сек.* |  |

Вывод: сложность алгоритма находится по формуле   
(кол-во ребер \* log(кол-во вершин)). Получили знания по поиску кратчайшего расстояния между вершинами ориентированного взвешенного графа.