**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 №6**

По дисциплине «СиАОД»

На тему:

«Сетевые алгоритмы. Динамические алгоритмы поиска путей»

Выполнил: студент группы БВТ1801

Алмамма Башар

Руководитель:

Кутейников Иван Алексеевич

Москва 2020

**Цель работы:**

Используя технологию модульного программирования разработать программу нахождения наикратчайшего пути между вершинами графа с помощью определенного алгоритма, заданного заранее преподавателем, в соответствии с индивидуальным заданием. Разработать графический интерфейс пользователя.

**Задача (Вариант 1):**

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.

По результатам работы проанализировать временную сложность работы заданного алгоритма в зависимости от числа узлов и ребер графа. Заданный алгоритм - Алгоритм Флойда-Уоршелла.

**Программа:**

**1.** Код программы:

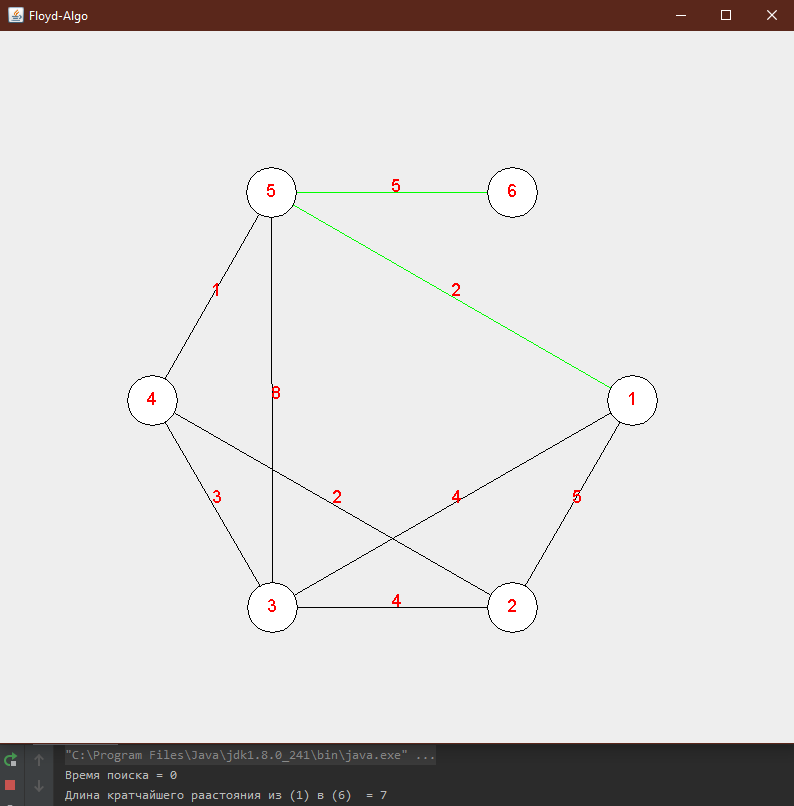
import java.io.IOException;  
  
public class Main {  
  
 public static void main(String[] args) throws IOException {  
 Floyd floyd = new Floyd(800,"D:\\Projects\\SiAOD\\Laba6-Java-NetworkAlgo\\test2.txt");  
 System.*out*.println("Время поиска = " + floyd.getElapsedTime());  
 System.*out*.println("Длина кратчайшего раастояния из (1) в (6) = " + floyd.getMinPath());  
 }  
  
}

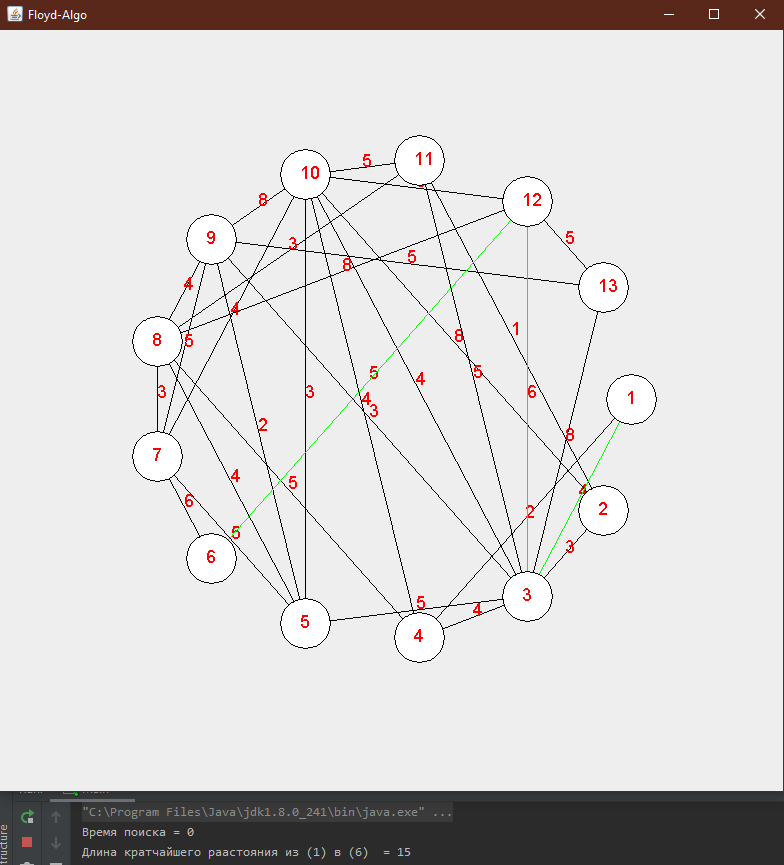
import javax.swing.\*;  
import java.awt.\*;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Floyd extends JFrame {  
 public static final int INF = 999999999;  
 ArrayList<Point> point;  
 ArrayList<Integer> grafPath= new ArrayList<Integer>();  
 long elapsedTime;  
 int minPath;  
 int size;  
 int n;  
 int[][] d;  
 int[][] oldD;  
 int[][] grafHistory;  
  
 Floyd(int size, String path) throws IOException {  
 intil(path);  
 long start = System.currentTimeMillis();  
 minPath = FloydMin(0,5);  
 long end = System.currentTimeMillis();  
 elapsedTime = end - start;  
 this.size = size;  
 setTitle("Floyd-Algo");  
 setSize(size,size);  
 setVisible(true);  
 setDefaultCloseOperation(EXIT\_ON\_CLOSE);  
 }

public void paint(Graphics g) {  
 int R = size / 2 - size / 5; //радиус  
 int X = size / 2, Y = size / 2;//координаты центра  
 point = new ArrayList<Point>();  
 g.setFont(new Font("TimesRoman", Font.PLAIN, 18));  
  
 double angle = 360.0 / n;  
 for (int i = 0; i < n; i++) {  
 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 (int i = 0; i < n; i++){  
 for(int j = i; j < n; j++){  
 if(i != j && oldD[i][j] != INF) {  
 int x1 = point.get(i).x;  
 int y1 = point.get(i).y;  
 int x2 = point.get(j).x;  
 int y2 = point.get(j).y;  
 g.setColor(Color.BLACK);  
 g.drawLine(x1, y1, x2, y2);  
 g.setColor(Color.RED);  
 g.drawString(String.valueOf(oldD[i][j]), (x1 + x2) / 2, (y1 + y2) / 2);  
 }  
 }  
 }  
 for (int i = 0; i < grafPath.size() - 1; i++) {  
 g.setColor(Color.GREEN);  
 int x1 = point.get(grafPath.get(i)).x;  
 int y1 = point.get(grafPath.get(i)).y;  
 int x2 = point.get(grafPath.get(i+1)).x;  
 int y2 = point.get(grafPath.get(i+1)).y;  
 g.drawLine(x1, y1, x2, y2);  
 }  
  
 int l = 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(l + 1), p.x - 5, p.y + 5);  
 l++;  
 }  
 }  
  
 private void intil(String string) throws IOException {  
 Path path = Paths.*get*(string);  
 List<String> contents = Files.readAllLines(path);  
 n = contents.size() - 1;  
 d = new int[n][n];  
 oldD = new int[n][n];  
 grafHistory = new int[n][n];  
 for(int i = 1; i < n + 1; i++){  
 String[] row = contents.get(i).split(" ");  
 for(int j = 1; j < n + 1; j++) {  
 d[i- 1][j- 1 ] = Floyd.*INF*;  
 grafHistory[i- 1][j- 1] = - 1;  
 int temp = Integer.*parseInt*(row[j]);  
 if(i == j )d[i- 1][j- 1] = 0;  
 else if(temp != 0) {  
 d[i- 1][j- 1] = temp;  
 grafHistory[i- 1][j- 1] = j - 1;  
 }  
 oldD[i- 1][j - 1] = d[i- 1][j - 1];  
 }  
 }  
 }

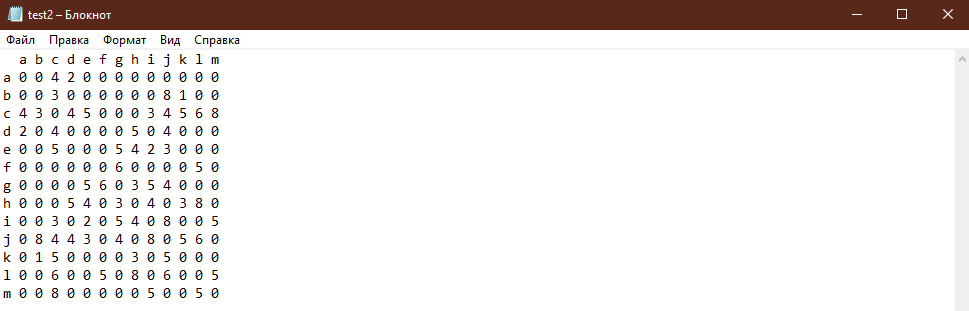
private int FloydMin(int i1, int j1) {  
 for (int k = 0; k < n; k++) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 if(d[i][k] + d[k][j] < d[i][j]) {  
 d[i][j] = Math.*min*(d[i][j], d[i][k] + d[k][j]);  
 grafHistory[i][j] = grafHistory[i][k];  
 }  
 }  
 }  
 }  
 getPath(i1,j1);  
 return d[i1][j1];  
 }  
 private void getPath(int i1, int j1) {  
 if(d[i1][j1] >= *INF*)return;  
 int c = i1;  
 while (c != j1){  
 grafPath.add(c);  
 c = grafHistory[c][j1];  
 }  
 grafPath.add(j1);  
 }  
  
 public long getElapsedTime() {  
 return elapsedTime;  
 }  
  
 public int getMinPath() {  
 return minPath;  
 }  
}

**2.** Тестирование программы





Пример входных данных:



**Вывод:** в данной лабораторной работе я научился реализовывать алгоритмы поиска кратчайшего пути, и разработать графический интерфейс пользователя, проанализировав результаты программы, можно делать вывод, что время поиска очень мал для такого объема вершин и ребер.