|  |  |
| --- | --- |
| haut | 河南工业大学信息科学与工程学院 |

**《算法设计与分析》实验报告**

|  |  |
| --- | --- |
| 课程名称： | 算法设计与分析 |
| 学生姓名： | 刘文博 |
| 学生学号： | 201716040224 |
| 学生专业： | 软件工程1702 |

**实验四：最小生成树**

苗建雨

2019 年 5 月 9 日

实验日期： 2019 年 5 月 23 日 班级： 软件工程1702

学号（后四位）：\_\_\_0224\_\_\_\_\_\_ 姓名：\_\_\_\_刘文博\_\_\_\_\_ 成绩：

**一、 实验目的：**

熟悉贪心算法的基本原理与使用范围；

熟悉和掌握贪心算法求最小生成树问题；

**二、实验要求：**

熟悉贪心算法的基本原理与使用范围； 熟悉和掌握贪心算法求最小生成树问题；并用代码实现。

**三、实验内容：**

**代码实现：**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

/\*\*

\* 最小生成树(普里姆算法(Prim算法))

\* @author liuwenbo

\* @date 2019/05/23

\*/

public class MinTreePrimer {

private static List<Vertex> visitedVertexs,leftedVertexs; //分别为添加到集合U中的节点集和剩余的集合V中的节点集

private static List<Edge> searchEdges;

//初始化图的信息

public static void initGraph(Graph g){

visitedVertexs = new ArrayList<Vertex>();

leftedVertexs = new ArrayList<Vertex>();

searchEdges = new ArrayList<Edge>();

Scanner sc = new Scanner(System.in);

System.out.print("输入顶点数: ");

int vertexNumber = sc.nextInt();

System.out.print("请输入边数: ");

int edgeNumber = sc.nextInt();

String[] allVertex = new String[vertexNumber];

String[] allEdge = new String[edgeNumber];

System.out.println("=================================");

System.out.println("请输入各个顶点:");

Scanner scanner = new Scanner(System.in);

for(int i=0;i<vertexNumber;i++){

System.out.print("顶点"+(i+1)+":");

allVertex[i] = scanner.nextLine();

}

System.out.println("=================================");

for(int i=0;i<edgeNumber;i++){

System.out.print("输入边(Vi,Vj)中的顶点名称和权值W(如:A B 7): ");

allEdge[i] = scanner.nextLine();

}

g.vertex = new Vertex[allVertex.length];

g.edge = new Edge[allEdge.length];

g.minWeight = 0;

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

g.vertex[i] = new Vertex();

g.vertex[i].vName = allVertex[i];

leftedVertexs.add(g.vertex[i]); //初始化剩余点集合

}

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

g.edge[i] = new Edge();

g.edge[i].startVertex = new Vertex();

g.edge[i].endVertex = new Vertex();

String edgeInfo[] = allEdge[i].split(" ");

g.edge[i].startVertex.vName = edgeInfo[0];

g.edge[i].endVertex.vName = edgeInfo[1];

g.edge[i].weight = Integer.parseInt(edgeInfo[2]);

}

}

public static void onChangeVertex(Vertex vertex){

visitedVertexs.add(vertex); //添加初始节点,作为默认的开始节点

leftedVertexs.remove(vertex);

}

public static Vertex findOneVertex(Graph g){

int minValue = Integer.MAX\_VALUE;

Vertex findVertex = new Vertex();

Edge findEdge = new Edge();

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

for(int j=0;j<leftedVertexs.size();j++){

Vertex v1 = visitedVertexs.get(i);

Vertex v2 = leftedVertexs.get(j); //获取两个顶点的名称

for(int k=0;k<g.edge.length;k++){

String startName = g.edge[k].startVertex.vName;

String endName = g.edge[k].endVertex.vName;

if((v1.vName.equals(startName) && v2.vName.equals(endName))

||(v1.vName.equals(endName) && v2.vName.equals(startName))){

if(g.edge[k].weight < minValue){

findEdge = g.edge[k];

minValue = g.edge[k].weight;

if(leftedVertexs.contains(v1)){ //会调用对象的equals方法比较对象,需重写equals方法

findVertex = v1;

}else if(leftedVertexs.contains(v2)){

findVertex = v2;

}

}

}

}

}

}

g.minWeight+= minValue;

searchEdges.add(findEdge);

return findVertex;

}

public static void prim(Graph g){

while(leftedVertexs.size()>0){ //直到剩余节点集为空时结束循环

Vertex findVertex = findOneVertex(g);

onChangeVertex(findVertex);

}

System.out.print("\n最短路径包含的边: ");

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

System.out.print("("+searchEdges.get(i).startVertex.vName+","+searchEdges.get(i).endVertex.vName+")"+" ");

}

System.out.println("\n最短路径长度: "+g.minWeight);

}

public static void main(String[] args) {

Graph g = new Graph();

initGraph(g);

onChangeVertex(g.vertex[0]);

prim(g);

}

}

/\*\*

\* 顶点类Vertex

\*/

class Vertex{

String vName; //顶点的名称

@Override

public boolean equals(Object obj) {

if(obj instanceof Vertex){

Vertex vertex = (Vertex)obj;

return this.vName.equals(vertex.vName);

}

return super.equals(obj);

}

}

/\*\*

\* 边类Edge

\*/

class Edge{

Vertex startVertex;

Vertex endVertex;

int weight;

}

/\*\*

\* 图的存储结构

\*/

class Graph{

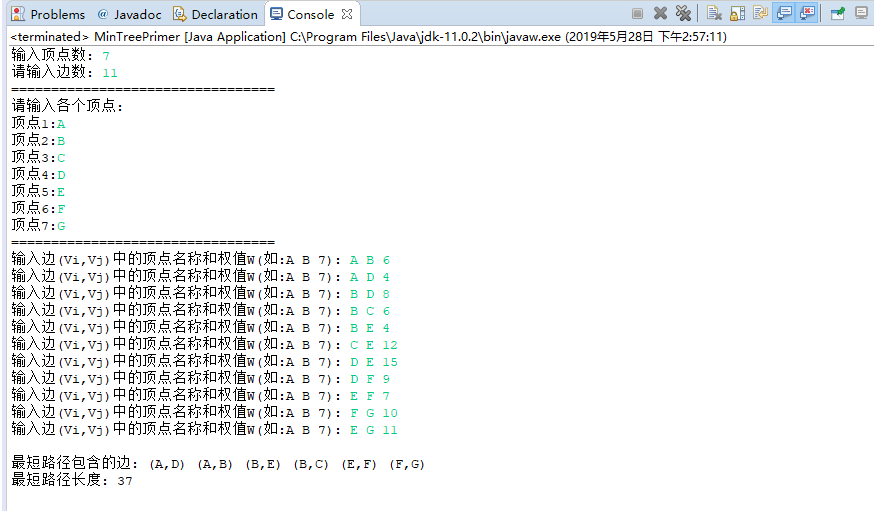
Vertex[] vertex; //顶点集

Edge[] edge; //边集

int minWeight; //最短路径

}

**实验截图：**



**四、实验结果&总结：**

假设G＝(V，E)是连通的，TE是G上最小生成树中边的集合。算法从U＝{u0}（u0∈V）、TE＝{}开始。重复执行下列操作：

 在所有u∈U，v∈V－U的边(u，v)∈E中找一条权值最小的边(u0,v0)并入集合TE中，同时v0并入U，直到V＝U为止。

   此时，TE中必有n-1条边，T=(V，TE)为G的最小生成树。

   Prim算法的核心:始终保持TE中的边集构成一棵生成树。

prim算法适合稠密图，其时间复杂度为O(n^2)，其时间复杂度与边得数目无关，而kruskal算法的时间复杂度为O(eloge)跟边的数目有关，适合稀疏图。