**上海电力大学**

**数据结构（JAVA）课程设计**



**题　　目:**  景区旅游信息管理

**姓 名：** 刘天启

**学 号：** 20181545

**院　　系：**　　 计算机科学与技术学院

**专业年级：**　 大二

**2020年　1 月　5 日**

目录

[一、设计题目 1](#_Toc13910)

[二、需求分析 2](#_Toc6886)

[三、概要设计 3](#_Toc16691)

[四、详细设计 3](#_Toc19012)

[五、调试分析 5](#_Toc4952)

[六、测试结果 6](#_Toc17077)

[七 、附录：程序设计源代码 8](#_Toc11903)

# 设计题目

1. 实验名称

景区旅游信息管理

## 2、实验目的

（1） 熟练掌握图的创建及遍历基本操作算法。

（2） 熟练掌握最短路径算法。

（3） 利用图的基本操作及求最短路径等方法，设计一个全国城市间的交通咨询程序，为旅客提供两种或三种最优先决策的交通咨询。

## 3、 实验内容

【问题描述】

在旅游景点经常会有游客询问从一个景点到另一个景点的最短路径和最短距离。景区的景点需要互相连通，且景区道路铺设代价要最小。

【基本要求】

1. 构建景区景点图。
2. 景点信息管理：包括查询、增加、删除景点信息。

(2)建立景区导游线路图。给出某一个景点，可以遍历所有景点，且不会有回路。

(3)从一个景点到另一个景点的最短路径和最短距离。

(4) 景区建设的建议，景区的景点需要互相连通，且景区道路铺设代价要最小。

【测试数据】

自行设计某旅游景点信息图。

# 二、需求分析

**1）运行环境（软、硬件环境）**

该程序要求在java环境下运行，生成图时要求在python环境下运行，运行时需要2G以上内存。

**2）输入的形式和输入值的范围**

选择功能时应输入0~9的整数

输入景区景点时可以输入任何类型的名称。

**3）输出的形式描述**

景区图，景区导游图道路规划图的输出均由图的邻接矩阵来表示。

**4）功能描述**

①创建景区分布图

ps：程序为了方便测试，避免每次测试重新输入数据，可以在选择功能界面按0使用默认的景区图，景区图如 5)测试数据。

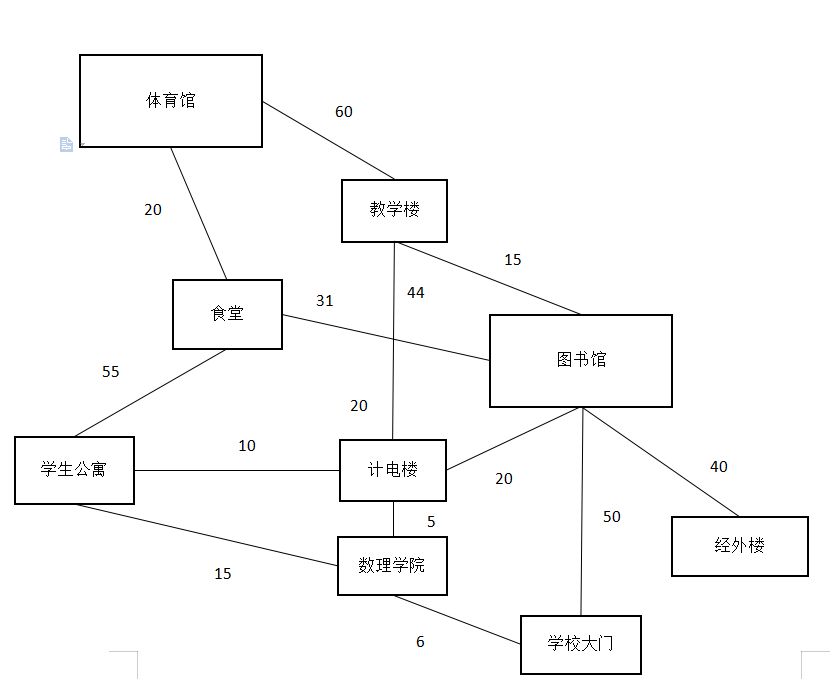
②查询、增加、删除景点

③创建景区导游图。给出某一个景点，可以遍历所有景点，且不会有回路。

④查找从一个景点到另一个景点的最短路径和最短距离。

⑤给出景区建设建议，让景区的景点互相连通且道路铺设代价最小。

1. 测试数据



# 三、概要设计

**1）抽象数据类型定义描述**

**private** **int** vexNum; //顶点数量

**private** **int** arcNum; //边数

**private** T[] vexs; //顶点数组

**private** **int**[][] arcs; //二维数组表示方阵

**private static final int** *MAX*=0x0000ffff; //最大权重

1. **功能模块设计（如主程序模块设计）**

1、创建景区景点分布图

通过一个**邻接矩阵**（实质是一个二维数组,arcs[i][j]表示从i到j的权值大小，为零表示没有直达的路径）记录景区景点的分布图。

2、输出景区景点分布图（**邻接矩阵**）

通过扫描邻接矩阵输出景区景点分布图。

3、输出导游线路图：深度优先策略

首先通过遍历景点，通过用户给出的一个入口景点c，建立一个导游线路图，导游线路图用有向图表示。遍历采用**深度优先策略**（递归），这个也是正常的游客的心理。

4、求两个景点间的最短路径和最短距离：**floyd算法**

在导游线路图中，还为一些不愿按线路走的游客提供信息服务，比如从一个景点到另一个景点的最短路径和最短距离。在本线路图中将输出任意景点间的最短路径和最短距离。

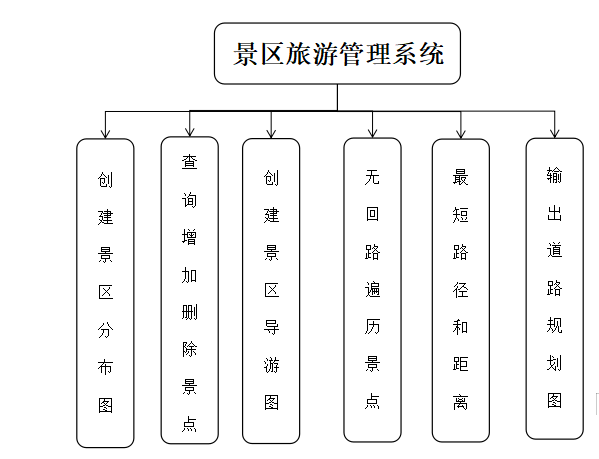
5、输出道路修建规划图：**prime算法**

在景区建设中，道路建设是其中一个重要的内容。道路建设首先要保证能连通所有景点，但又要花最小的代价，可以通过求最小生成树来解决这个问题，通过prime算法来求最小生成树。

1. 将当前图的邻接矩阵转化为图：

Java程序中图只能由邻接矩阵来表示景区图，并不能直观表现出地图路线和坐标，利用python的networkx图论扩展库、matplotlib数据可视化扩展库和NumPy矩阵扩展库自动生成一个新图，使得生成的图更形象。

**3）模块层次调用关系图**



# 详细设计

1. 单链表节点类

**public** Node(T data,Node<T> next)：初始化节点。

**public** String toString()：输出节点数据域。

2、单链表类

**public** SinglyList()：初始化单链表，初始化头结点。

**public** Node<T> insert(**int** i,T x)：在单链表i位置插入元素T。

**public** String toString()：输出单链表。

3、三元组类

**public** Triple(**int** row,**int** column,**int** value)：初始化三元组的行号、列号、元素值。

4、邻接矩阵类

**public** **void** createUW():创建景区图的邻接矩阵。

**private** **int** locateVex(T vex)：寻找元素在顶点数组的位置。几乎每个方法都要调用此方法。

**public** String toString():输出当前图，当输出值为MAX时将MAX的值替换为字符“∞”。

**public** **void** find():输入景点名称，查找与该景点连接的其他景点并输出距离。

**private void** insertVex(T x)：在顶点数组中插入一个顶点并调用setRowsColumns(int m,int n)方法扩充二维数组。

**private void** setRowsColumns(**int** m,**int** n)：扩充二维数组。

**public** **void** add()：新增景点。通过调用insertVex(T x)方法和setRowsColums(int m,int n)方法新增顶点和边。

**public** **void** removeVex()：删除景点。

**protected** **int** next(**int** v,**int** w)：返回vexs[v]在vexs[w]后的后继邻接顶点序号。

**private** **void** depthfs(**int** i,**boolean**[] visited)：深度优先遍历

**public** **void** DFS()：深度优先遍历输出景点导游图（无回路）。

**private** String toPath(Matrix path,**int** i,**int** j)：返回path路径矩阵中从顶点i到j的一条字符串路径。

**public** **void** shortestPath()：求两个景点间的最短路径和最短距离

**public** **void** minSpanTree()：prim算法，构造带权无向图的最小生成树，输出景区道路建设建议以及最小总距离。

**public** String toGraph()：生成标准邻接矩阵（该标准为：将“∞”用“0”代替）并输出，用于粘贴到python程序中生成景区图。

1. python程序

将java程序中生成的标准邻接矩阵复制在python程序中，根据该邻接矩阵向图论包中逐一添加顶点和边，最后生成标准邻接矩阵对应的图。

# 五、调试分析

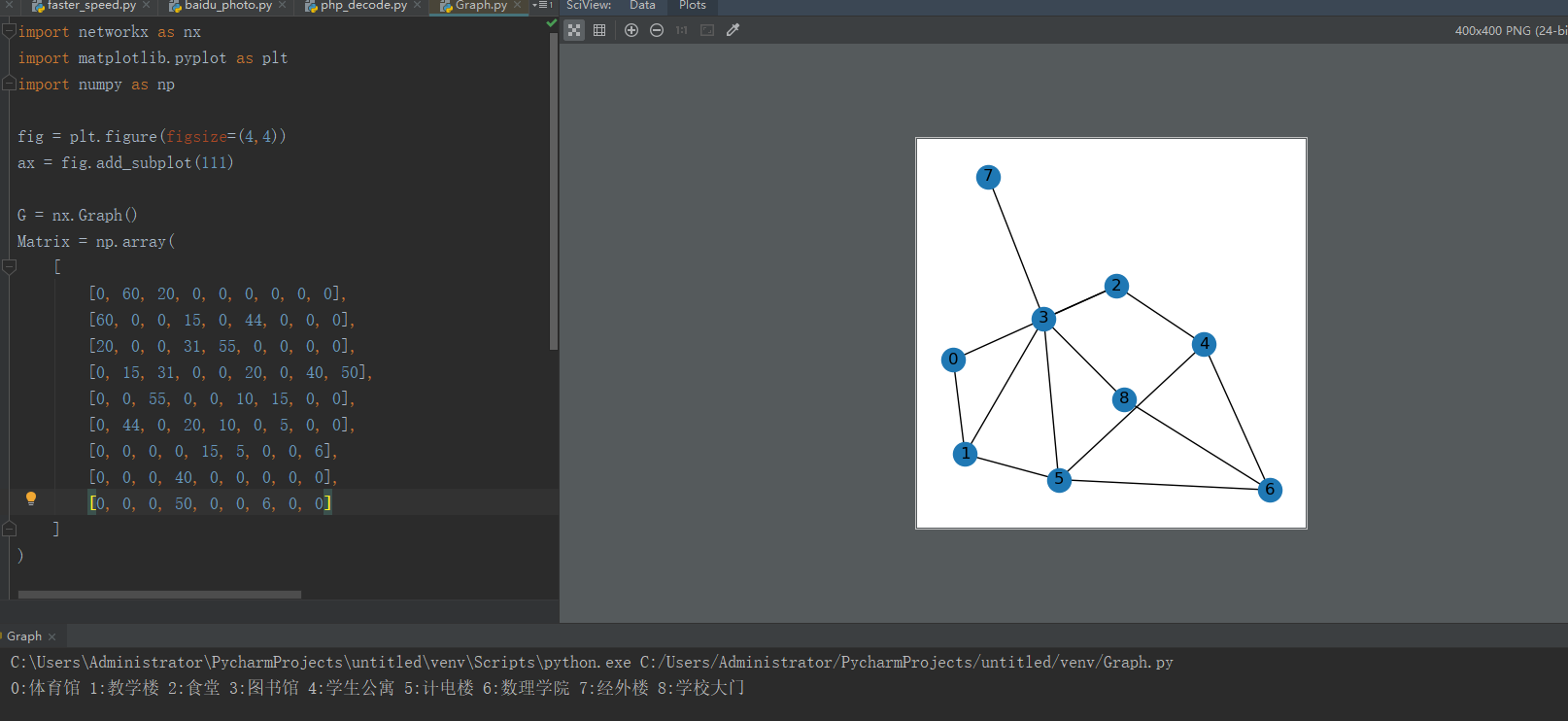
包括调试过程中遇到的问题及解决的方法、算法的时间空间复杂性分析、经验体会。

遇到的问题及解决的方法：

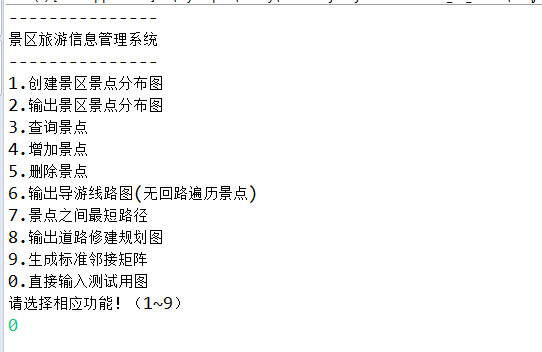
1. 在编写find()方法时，如果输入查找景点的名称错误，编译器会报错，用了**try/catch语句**让程序不会因为输入错误而中断。
2. 由于使用的是邻接矩阵表示景区图，因而在增加和删除景点时会比较麻烦，期间不仅要考虑数组的扩充和压缩，还要保证顶点数量vexNum和边数arcNum与数组的操作同时增加或减少。
3. 由于使用的是邻接矩阵表示景区图，因而在搜索景点时**时间复杂度**要优于使用邻接表表示，为O(n),但在增加和删除景点时**空间复杂度**要差于邻接矩阵表示，空间复杂度为O(n²)。
4. 在程序编写过程中总是遇到空指针报错，总结发现自己在编程时总是忘记**初始化**指针，例如初始化单链表头结点，初始化邻接矩阵二位数组等，这是以后需要注意的地方。
5. 在生成最短路径时，如下代码dist.getArcs()[i][k]+dist.getArcs()[k][j]

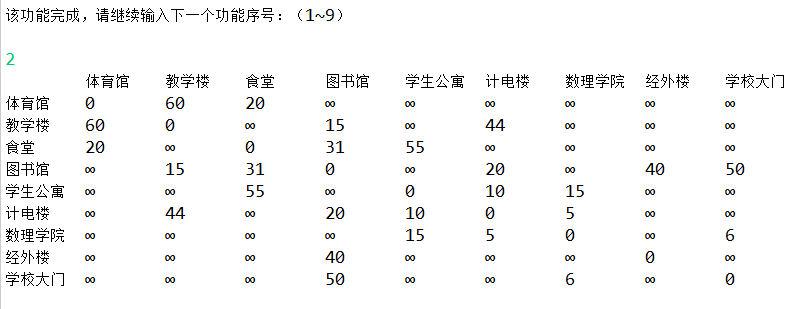
由于最开始最大权重设置的是*MAX*=Integer.*MAX\_VALUE，*在相加时若是两个无穷大相加，**int类型数据溢出**，导致相加结果为-2147483631，进而导致后续比较时结果出错，影响最短路径矩阵的值。为了解决此情况，将最大权重的设置值改为***MAX***=0x0000ffff，顺利解决了此问题。

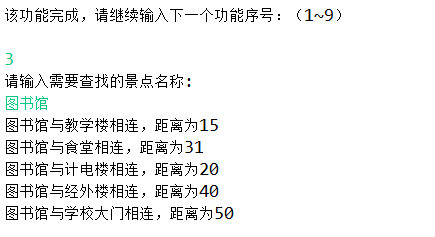
**程序创新点**：在运行程序时，可选择序号9，生成“标准邻接矩阵”，将生成的邻接矩阵复制到pycharm中，利用python的networkx图论扩展库、matplotlib数据可视化扩展库和NumPy扩展库自动生成一个新图，使得生成的图更形象。

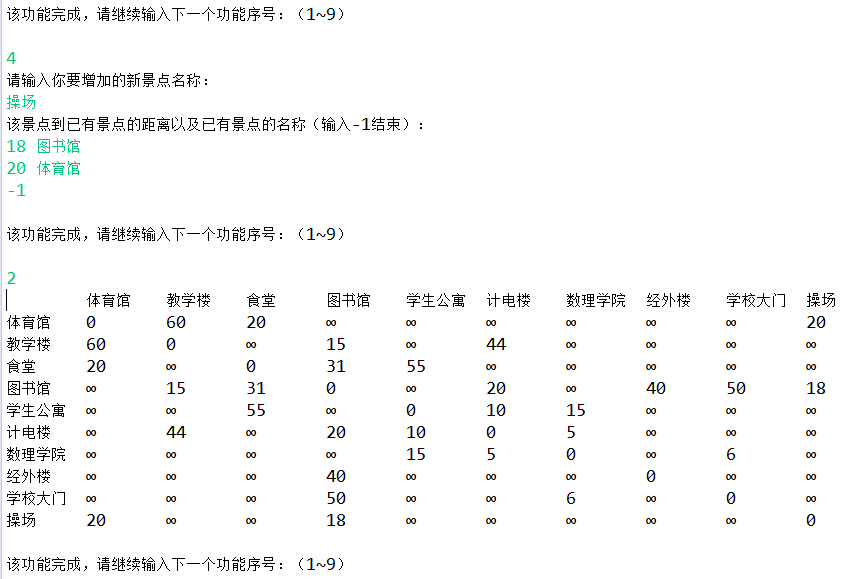


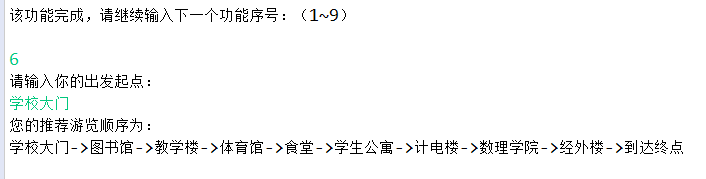
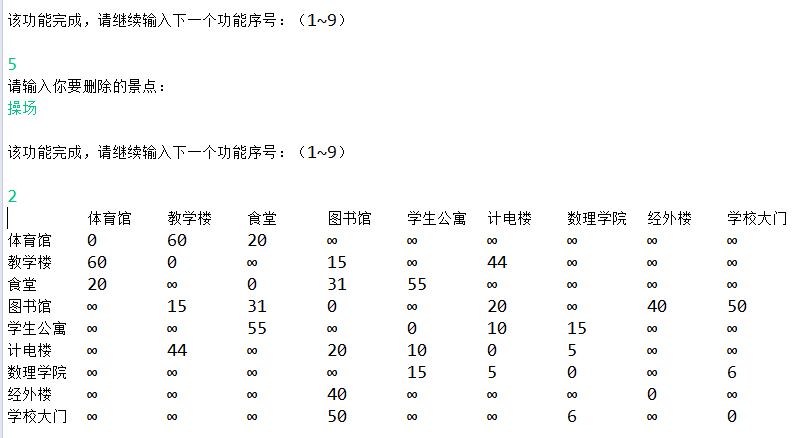
# 六、测试结果

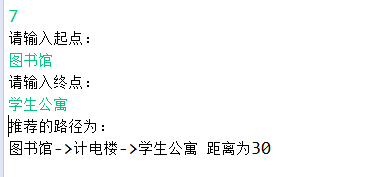


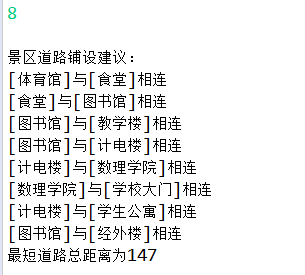


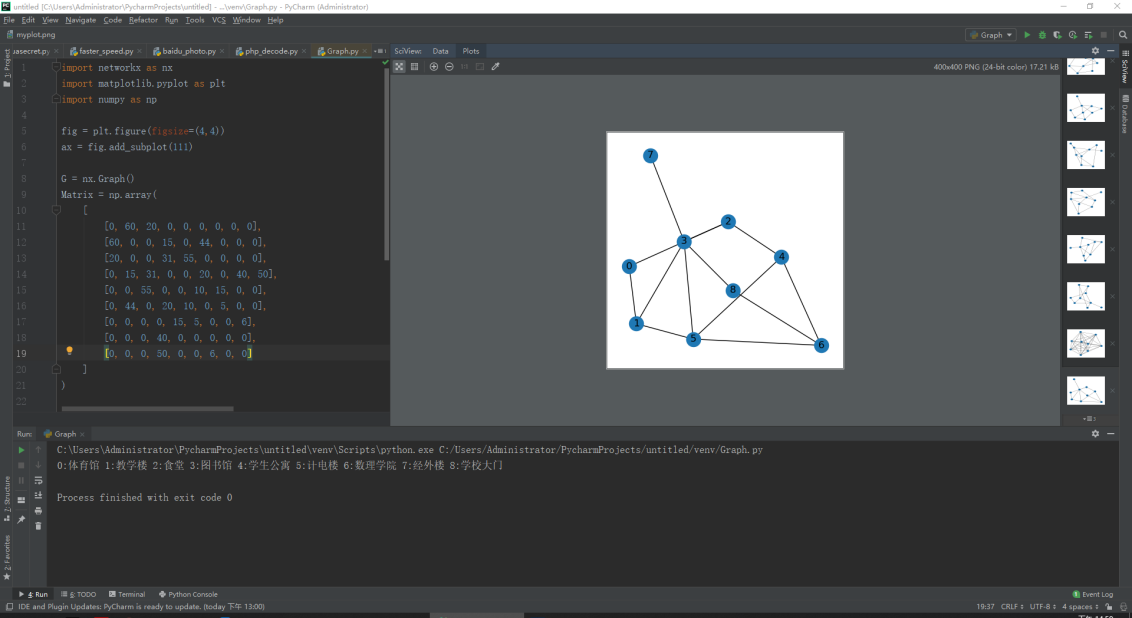
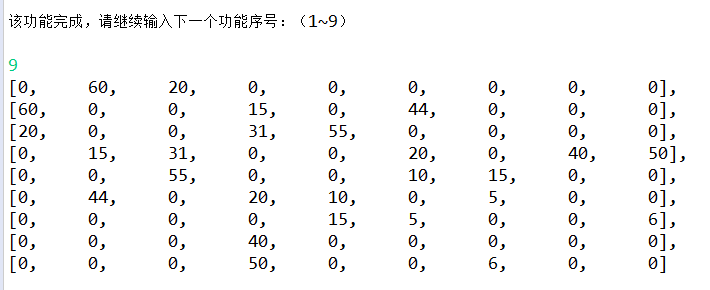












# 七、附录：程序设计源代码

**package** matrixGraph;

**public** **class** Node<T>{

**public** T data;

**public** Node<T> next;

**public** Node(T data,Node<T> next) {

**this**.data=data;

**this**.next=next;

}

**public** Node() {

**this**(**null**,**null**);

}

**public** String toString() {

**return** **this**.data.toString();

}

}

//////////////////////////////////////

**package** matrixGraph;

**public** **class** SinglyList<T> {

**public** Node<T> head;

**public** SinglyList() {

**this**.head = **new** Node<T>();

}

**public** Node<T> insert(**int** i,T x){ //在单链表i位置插入元素T

**if**(x==**null**)

**throw** **new** NullPointerException("x==null");

Node<T> front = **this**.head;

**for**(**int** j=0;front.next!=**null** && j<i;j++)

front = front.next;

front.next = **new** Node<T>(x,front.next);

**return** front.next;

}

**public** String toString() {

String str = "";

**for**(Node<T> p=head.next; p!=**null**; p=p.next) {

str+= p.toString();

**if**(p.next!=**null**)

str+="->";

}

**return** str;

}

}

////////////////////////////////

**package** matrixGraph;

**public** **class** Triple { //三元组类

**int** row,column,value; //行号、列号、元素值

**public** Triple(**int** row,**int** column,**int** value) {

**if**(row>=0 && column>=0) {

**this**.row = row;

**this**.column = column;

**this**.value = value;

}

**else** **throw** **new** IllegalArgumentException("行、列号不能为负数");

}

}

///////////////////////////////////////////////

**package** matrixGraph;

**import** java.util.Scanner;

**public** **class** Matrix<T> {

**private** **int** vexNum; //顶点数量

**private** **int** arcNum; //边数

**private** T[] vexs; //顶点数组

**private** **int**[][] arcs; //二维数组表示方阵

**private** **static** **final** **int** ***MAX***=0x0000ffff; //最大权重

**public** Matrix(**int** vexNum,**int** arcNum,T[] vexs,**int**[][] arcs) {

**this**.vexNum = vexNum;

**this**.arcNum = arcNum;

**this**.vexs = vexs;

**this**.arcs = arcs;

}

**public** Matrix(**int** vexNum) {

**this**.vexNum = vexNum;

**this**.arcNum = 0;

**this**.vexs = (T[]) **new** Object[vexNum];

**this**.arcs = **new** **int**[vexNum][vexNum];

}

**public** Matrix() {

**this**(0,0,**null**,**null**);

}

**public** **void** createM(**int** vexNum,**int** arcNum,T[] vexs,**int**[][] arcs) {

**this**.vexNum = vexNum; //顶点数n

**this**.arcNum = arcNum; //边数e

//输入图中各顶点

**this**.vexs = (T[]) **new** Object[vexNum];

**for** (**int** i = 0; i < vexNum; i++) //构造顶点向量

**this**.vexs[i] = (T)vexs[i];

//初始化邻接矩阵

**this**.arcs = **new** **int**[vexNum][vexNum];

**for** (**int** i = 0; i < vexNum; i++)

**for** (**int** j = 0; j < vexNum; j++) {

**if**(i==j) **this**.arcs[i][j]=0;

**else** **this**.arcs[i][j]=***MAX***;

}

**for**(**int** i=0;i<vexNum;i++) {

**for**(**int** j=0;j<vexNum;j++) {

**this**.arcs[i][j]=arcs[i][j];

}

}

}

// 无向网邻接矩阵的创建

**public** **void** createUW() {

Scanner in = **new** Scanner(System.***in***);

**try** {

System.***out***.println("请分别输入景区的的景点数、景区的道路数:");

vexNum = in.nextInt(); //顶点数n

arcNum = in.nextInt(); //边数e

}

**catch**(Exception e) {

System.***out***.println("您输入出现错误！输入样例[5 20]");

}

//输入图中各顶点

vexs = (T[]) **new** Object[vexNum];

System.***out***.println("请分别输入景区的各个顶点:");

**for** (**int** i = 0; i < vexNum; i++) //构造顶点向量

vexs[i] = (T) in.next();

//初始化邻接矩阵

arcs = **new** **int**[vexNum][vexNum];

**for** (**int** i = 0; i < vexNum; i++)

**for** (**int** j = 0; j < vexNum; j++) {

**if**(i==j) **this**.arcs[i][j]=0;

**else** **this**.arcs[i][j]=***MAX***;

}

//输入边信息

**try** {

System.***out***.println("请输入各个边的两个顶点及其权值:");

**for** (**int** k = 0; k < arcNum; k++) {

**int** i = locateVex((T) in.next()); //顶点

**int** j = locateVex((T) in.next()); //顶点

arcs[i][j] = in.nextInt(); //权值

arcs[j][i] = arcs[i][j];

} //结束

}

**catch**(Exception e) {

System.***out***.println("您输入出现错误！输入样例[A B 20]");

}

}

**public** **int** getVexNum() {

**return** vexNum;

}

**public** **int**[][] getArcs() {

**return** arcs;

}

**public** T getVertex(**int** i) {

**return** vexs[i];

}

**public** **void** set(**int** i,**int** j,**int** w) { //i行j列复制为值为w

**this**.arcs[i][j]=w;

}

**private** **int** locateVex(T vex) {

//遍历顶点数组

**for** (**int** i = 0; i < vexNum; i++)

**if** (vexs[i].equals(vex))

**return** i;

**return** -1;

}

**public** **void** find() { //查找景点

Scanner in = **new** Scanner(System.***in***);

**try** {

System.***out***.println("请输入需要查找的景点名称:");

T vex = (T)in.nextLine();

**int** k = locateVex(vex);

**for**(**int** i=0;i<getVexNum();i++) {

**if**(getArcs()[k][i]>0 && getArcs()[k][i]<***MAX***)

System.***out***.println(vex+"与"+vexs[i]+"相连，距离为"+getArcs()[k][i]);

}

}

**catch**(Exception e){

System.***out***.println("输入景点名称有误！");

}

}

**private** **void** insertVex(T x) { //插入顶点

T[] temp = **this**.vexs;

**this**.vexs = (T[])**new** Object[vexNum+1];

**for**(**int** i=0;i<**this**.vexNum+1;i++) {

**if**(i==**this**.vexNum)

**this**.vexs[i] = x;

**else**

**this**.vexs[i] = temp[i];

}

setRowsColumns(**this**.vexNum+1,**this**.vexNum+1);

vexNum++;

}

**private** **void** setRowsColumns(**int** m,**int** n) { //扩充二维数组

**if**(m>0 && n>0) {

**if**(m>**this**.arcs.length || n>**this**.arcs[0].length) {

**int**[][] source = **this**.arcs;

**this**.arcs = **new** **int**[m][n];

**for**(**int** i=0;i<**this**.vexNum;i++)

**for**(**int** j=0;j<**this**.vexNum;j++)

**this**.arcs[i][j] = source[i][j];

}

}

**else** **throw** **new** IllegalArgumentException("扩充的行列不能≤0！");

}

**public** **void** add() {

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("请输入你要增加的新景点名称：");

T name = (T)in.nextLine();

**if**(locateVex(name)==-1) {

insertVex(name);

**for**(**int** i=0;i<vexNum-1;i++) { //初始化新增的行列

arcs[vexNum-1][i]=***MAX***;

arcs[i][vexNum-1]=***MAX***;

}

System.***out***.println("该景点到已有景点的距离以及已有景点的名称（输入-1结束）：");

**try** {

**while**(**true**) {

**int** weight = in.nextInt(); //输入距离

**if**(weight==-1) **break**;

T name1 = (T)in.next(); //输入景点名称

**this**.arcs[vexNum-1][locateVex(name1)] = weight;

**this**.arcs[locateVex(name1)][vexNum-1] = weight;

}

}

**catch**(Exception e) {

System.***out***.println("您输入的距离格式错误或景点名称不存在！输入样例[20 游乐园]");

}

}

**else**

System.***out***.println("该景点已存在！");

}

**public** **void** removeVex() { //删除顶点

**try** {

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("请输入你要删除的景点：");

T name = (T)in.nextLine();

**int** i = locateVex(name); //i存放要删除顶点的坐标

**for**(**int** k=i;k<vexNum-1;k++) { //改变顶点数组vexs[]

vexs[k]=vexs[k+1];

}

**for**(**int** n=0;n<vexNum;n++) //循环统计边减少的数量

**if**(getArcs()[i][n]>0 && getArcs()[i][n]<***MAX***)

arcNum--;

**for**(**int** k=i;k<vexNum-1;k++) //将每行上移

**for**(**int** j=0;j<vexNum;j++)

arcs[k][j]=arcs[k+1][j];

**for**(**int** k=i;k<vexNum-1;k++) //将每列左移

**for**(**int** j=0;j<vexNum;j++)

arcs[j][k]=arcs[j][k+1];

vexNum--; //顶点数减一

}

**catch**(Exception e) {

System.***out***.println("您输入的景点名称有误！");

}

}

//返回vexs[v]在vexs[w]后的后继邻接顶点序号

**protected** **int** next(**int** v,**int** w) **throws** Exception {

**if**(v<0&&v>=vexNum)

**throw** **new** Exception("第"+v+"个顶点不存在！");

//从w+1开始遍历第v行

**for**(**int** j=w+1;j<vexNum;j++)

**if**(arcs[v][j]!=0 && arcs[v][j]<***MAX***)

**return** j;

**return** -1;

}

//从顶点vexs[i]出发的一次深度优先搜索，遍历一个连通分量，visited指定访问标记数组。递归算法

**private** **void** depthfs(**int** i,**boolean**[] visited) **throws** Exception {

System.***out***.print(vexs[i]+"->"); //访问顶点vexs[i]

visited[i] = **true**; //设置访问标记

**int** j = **this**.next(i, -1); //返回vexs[i]的第一个邻接顶点序号

**while**(j!=-1) { //若vexs[i]存在一个邻接顶点vexs[j]

**if**(!visited[j]) //且顶点vex[j]未被访问

depthfs(j,visited); //递归调用

j=**this**.next(i, j); //返回vexs[i]在vexs[j]后的后继邻接顶点序号

}

}

**public** **void** DFS() **throws** Exception {

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("请输入你的出发起点：");

**try** {

T name = (T)in.nextLine();

**int** i = locateVex(name);

**boolean**[] visited = **new** **boolean**[vexNum]; //标记访问数组

**int** j=i;

System.***out***.println("您的推荐游览顺序为：");

**do** {

**if**(!visited[j]) { //若顶点未被访问

**this**.depthfs(j, visited); //从顶点进行一次深度优先搜索

}

j=(j+1)%vexNum; //在其他连通分量中寻找未被访问节点

}**while**(j!=i);

System.***out***.println("到达终点");

}

**catch**(Exception e){

System.***out***.println("您输入的出发起点有误！");

}

}

//返回path路径矩阵中从顶点i到顶点j的一条字符串路径

**private** String toPath(Matrix path,**int** i,**int** j) {

SinglyList<T> pathlink = **new** SinglyList<T>(); //单链表，记录最短路径经过顶点，用于反序

pathlink.insert(0,**this**.getVertex(j)); //单链表插入最短路径终点j

**for**(**int** k=path.getArcs()[i][j];k!=i&&k!=j&&k!=-1;k=path.getArcs()[i][k])

pathlink.insert(0, **this**.getVertex(k)); //单链表头插入经过的顶点，反序

pathlink.insert(0, **this**.getVertex(i)); //最短路径的起点i

**return** pathlink.toString();

}

**public** **void** shortestPath() {

Scanner in = **new** Scanner(System.***in***);

**int** n = vexNum;

Matrix path = **new** Matrix(n),dist = **new** Matrix(n); //最短路径长度矩阵，初值为0

**for**(**int** i=0;i<n;i++) //初始化dist、path矩阵

**for**(**int** j=0;j<n;j++) {

**int** w = **this**.getArcs()[i][j];

dist.set(i,j,w); //dist的初值是图的邻接矩阵

path.set(i, j, (i!=j&&w<***MAX***?i:-1));

}

**for**(**int** k=0;k<n;k++) //以k作为其他路径的中间顶点

**for**(**int** i=0;i<n;i++) //测试每对从i到j路径长度是否更短

**if**(i!=k)

**for**(**int** j=0;j<n;j++)

**if**(j!=k&&j!=i&&dist.getArcs()[i][j]>dist.getArcs()[i][k]+dist.getArcs()[k][j]) { //若更短，则替换

dist.set(i, j, (dist.getArcs()[i][k]+dist.getArcs()[k][j]));

path.set(i, j, path.getArcs()[k][j]);

}

System.***out***.println("请输入起点：");

T name1 = (T)in.nextLine();

System.***out***.println("请输入终点：");

T name2 = (T)in.nextLine();

**int** i = locateVex(name1);

**int** j = locateVex(name2);

System.***out***.println("推荐的路径为：");

System.***out***.println(toPath(path,i,j)+" 距离为"+(dist.getArcs()[i][j]==***MAX***?"∞":dist.getArcs()[i][j]));

//System.out.println("标准邻接矩阵为：");

//dist.toGraph();

}

//prim算法，构造带权无向图的最小生成树，输出最小生成树及各边代价

**public** **void** minSpanTree() {

Triple[] mst = **new** Triple[vexNum-1]; //最小生成树的边合集

**for**(**int** i=0;i<mst.length;i++) //边合集初始化，从顶点0出发构造

mst[i] = **new** Triple(0,i+1,**this**.getArcs()[0][i+1]); //保存从0到其他各个顶点的边

**for**(**int** i=0;i<mst.length;i++) { //选择n-1条边，每趟确定一条权值最小的边

**int** minweight = mst[i].value, min = i; //最小权值及边的下标

**for**(**int** j=i+1;j<mst.length;j++) //在i~n-1范围内寻找最小的边

**if**(mst[j].value<minweight) { //若存在更小权值，则更新最小值变量

minweight = mst[j].value; //最小权值

min = j; //保存当前权值最小的序号

}

//将权值最小的边min交换到第i个元素，表示该边加入TE合集

Triple edge = mst[min];

mst[min] = mst[i];

mst[i] = edge;

//将i+1~n-1的其他边用权值更小的边替换

**int** tv = edge.column; //刚并入TV的顶点

**for**(**int** j=i+1;j<mst.length;j++) {

**int** v = mst[j].column; //原边在V-TV中的终点

**int** weight = **this**.getArcs()[tv][v];

**if**(weight<mst[j].value) //若（tv，v）边比第j条边的权值更小，则替换

mst[j] = **new** Triple(tv,v,weight);

}

}

System.***out***.println("\n景区道路铺设建议：");

**int** mincost = 0;

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

System.***out***.println("["+vexs[mst[i].row]+"]与["+vexs[mst[i].column]+"]相连");

mincost += mst[i].value;

}

System.***out***.println("最短道路总距离为"+mincost+"\n");

}

//输出图的邻接矩阵信息

**public** String toString() {

**for**(**int** i=0;i<getVexNum();i++)

System.***out***.print("\t"+vexs[i]);

System.***out***.println();

**for**(**int** i=0;i<getVexNum();i++) {

System.***out***.print(vexs[i]+"\t");

**for**(**int** j=0;j<getVexNum();j++)

**if**(getArcs()[i][j]==***MAX***)

System.***out***.print("∞\t"); //最大值用字符串“∞”代替

**else**

System.***out***.print(getArcs()[i][j]+"\t");

System.***out***.println();

}

**return** **null**;

}

**public** String toGraph() {

**for**(**int** i=0;i<getVexNum();i++) {

System.***out***.print("[");

**for**(**int** j=0;j<getVexNum();j++)

**if**(j==getVexNum()-1) {

**if**(getArcs()[i][j]==***MAX*** || i==j)

System.***out***.print("0"); //最大值用字符串“∞”代替

**else**

System.***out***.print(getArcs()[i][j]);

}

**else** {

**if**(getArcs()[i][j]==***MAX*** || i==j)

System.***out***.print("0,\t"); //最大值用字符串“∞”代替

**else**

System.***out***.print(getArcs()[i][j]+",\t");

}

**if**(i==getVexNum()-1)

System.***out***.println("]");

**else**

System.***out***.println("],");

}

**return** **null**;

}

}

/////////////////////////////////////

**package** matrixGraph;

**import** java.util.Scanner;

**public** **class** Test {

**private** **static** **final** **int** ***MAX***=0x0000ffff; //最大权重 测试用

**static** **public** **void** frame() {

System.***out***.println("---------------");

System.***out***.println("景区旅游信息管理系统");

System.***out***.println("---------------");

System.***out***.println("1.创建景区景点分布图");

System.***out***.println("2.输出景区景点分布图");

System.***out***.println("3.查询景点");

System.***out***.println("4.增加景点");

System.***out***.println("5.删除景点");

System.***out***.println("6.输出导游线路图(无回路遍历景点)");

System.***out***.println("7.景点之间最短路径");

System.***out***.println("8.输出道路修建规划图");

System.***out***.println("9.生成标准邻接矩阵");

System.***out***.println("0.直接输入测试用图");

System.***out***.println("请选择相应功能！（1~9）");

}

**static** **int** menu() **throws** Exception {

//↓↓测试用↓↓

**int** vexNum = 9;

**int** arcNum = 13;

String[] vexs = {"体育馆","教学楼","食堂","图书馆","学生公寓","计电楼","数理学院","经外楼","学校大门"};

**int**[][] arcs = {{0,60,20,***MAX***,***MAX***,***MAX***,***MAX***,***MAX***,***MAX***},{60,0,***MAX***,15,***MAX***,44,***MAX***,***MAX***,***MAX***},

{20,***MAX***,0,31,55,***MAX***,***MAX***,***MAX***,***MAX***},{***MAX***,15,31,0,***MAX***,20,***MAX***,40,50},

{***MAX***,***MAX***,55,***MAX***,0,10,15,***MAX***,***MAX***},{***MAX***,44,***MAX***,20,10,0,5,***MAX***,***MAX***}

,{***MAX***,***MAX***,***MAX***,***MAX***,15,5,0,***MAX***,6},{***MAX***,***MAX***,***MAX***,40,***MAX***,***MAX***,***MAX***,0,***MAX***}

,{***MAX***,***MAX***,***MAX***,50,***MAX***,***MAX***,6,***MAX***,0}};

//↑↑测试用↑↑

Scanner in = **new** Scanner(System.***in***);

Matrix<String> a = **new** Matrix<String>();

*frame*();

**while**(**true**) {

**int** k = in.nextInt();

**switch**(k) {

**case** 0:

a.createM(vexNum, arcNum, vexs, arcs);

**break**;

**case** 1:

a.createUW();

**break**;

**case** 2:

a.toString();

**break**;

**case** 3:

a.find();

**break**;

**case** 4:

a.add();

**break**;

**case** 5:

a.removeVex();

**break**;

**case** 6:

a.DFS();

**break**;

**case** 7:

a.shortestPath();

**break**;

**case** 8:

a.minSpanTree();

**break**;

**case** 9:

a.toGraph();

**break**;

**default**:

System.***out***.println("你输入的数字范围出错！");

**break**;

}

System.***out***.println("\n该功能完成，请继续输入下一个功能序号：（1~9）\n");

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

*menu*();

}

}

////////////////////////////////////////////////

**Python代码：**

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

fig = plt.figure(figsize=(4,4))

ax = fig.add\_subplot(111)

G = nx.Graph()

Matrix = np.array(

[

[0, 60, 20, 0, 0, 0, 0, 0, 0],

[60, 0, 0, 15, 0, 44, 0, 0, 0],

[20, 0, 0, 31, 55, 0, 0, 0, 0],

[0, 15, 31, 0, 0, 20, 0, 40, 50],

[0, 0, 55, 0, 0, 10, 15, 0, 0],

[0, 44, 0, 20, 10, 0, 5, 0, 0],

[0, 0, 0, 0, 15, 5, 0, 0, 6],

[0, 0, 0, 40, 0, 0, 0, 0, 0],

[0, 0, 0, 50, 0, 0, 6, 0, 0]

]

)

for i in range(len(Matrix)):

for j in range(i,len(Matrix)):

if Matrix[i,j]!=0:

G.add\_node(i)

G.add\_node(j)

G.add\_edge(i,j,weight = Matrix[i][j])

nx.draw(G,with\_labels=True)

plt.show()

print("0:体育馆","1:教学楼","2:食堂","3:图书馆","4:学生公寓","5:计电楼","6:数理学院","7:经外楼","8:学校大门")