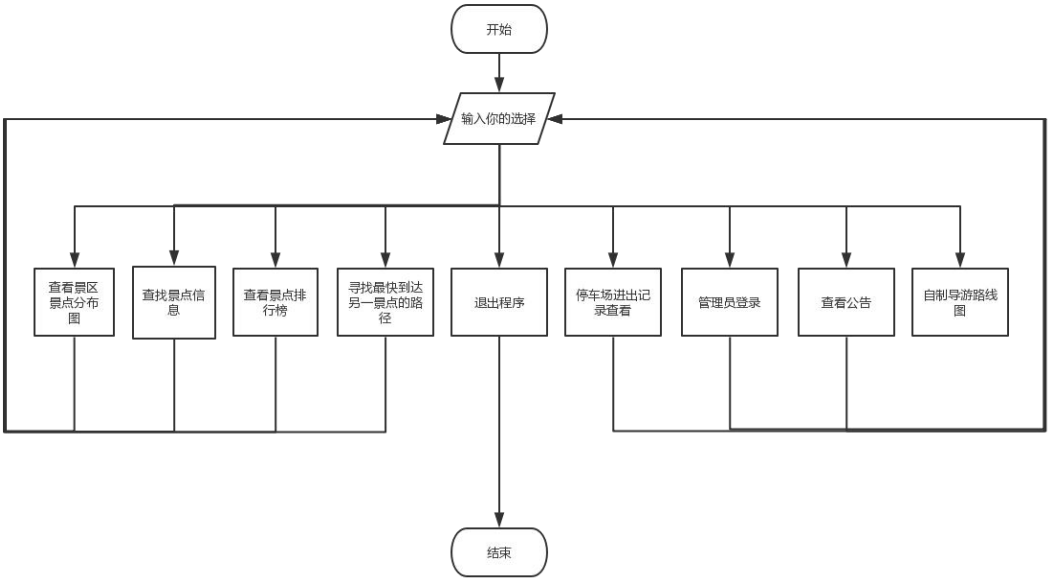
第一章 系统分析

一、 系统背景

本次实验主要目的为，通过使用栈、队列、链表、图等基本的数据结构完成景区信息管理系统，深刻理解图、树、栈、队列等数据结构。系统为大型景区管理系统，共分为八个模块：输出景区景点分布图模块，查找景点模块，景点排序模块，查找最快到达另一个景点的路径模块，输出导游路线图模块，停车场车辆进出记录模块，管理员模块以及查看公告模块。系统用户包括管理员和游客两类，管理员负责管理景区的景点维护；游客可以根据自己的需求对景区进行各种信息查询，及路线规划等

二、 功能需求



本次实验可大致分为三个模块

1. 游客模块

   游客模块包括，游客能够自主选择功能，如查看景区的景点分布图，模糊查找景点的信息，查看景点根据不同的因素的排序，查找游览到另一个景点的最短路径，查看并自制导游线路图，游客能随时查看管理员发布的公告，查看停车场信息。

2. 管理员模块

   管理员模块包括插入删除景点，增加或删除景点的道路，并可以自动发布公告。

3. 系统模块

   系统模块包括菜单界面以及选择操作。

三、 解决方案

1. 游客模块

1.1 查看景区的景点分布图：通过读取文件信息，生成景区的的邻接链表，并由此转换成邻接矩阵，输出邻接矩阵；

1.2 模糊查找景点信息：游客通过输入关键字，或是信息中的关键字进行模糊查找，了解景点的各种信息；

1.3 查看景点排序：使用 Bubble sort 冒泡排序法，对景点的信息进行排序。游客能够了解景点的热度排行榜以及由于岔路口的多少而生成的人流量排行榜；

1.4 查看两个景点间最短路径：游客能够查找到达另一个景点的最快方案。通过 Dijkstra 算法或是 Floyd 算法计算两点之间通路的最短路径来得到游客的到达景点的最短方案。

1.5 查看并自制导游线路图：游客能够选择是游览所有景点并回到原点或是游览所有景点并到达另一个地方。通过 Prim 算法计算得到最小生成树，遍历最小生成树，如果遍历的点与后一个点不相同，使用 Dijkstra 算法到达下一个点，并继续遍历最小生成树，知道所有景点都游览完成，最后返回原点或是想到达的最后的景点。

1.6 随时查看管理员发布的公告：游客能够通过查看管理员发布的公告，随时了解景区景点以及道路的信息。

1.7 查看停车场信息：游客能够随时查看停车场的信息。为实现停车场，能够使用两个栈来模拟停车场，使用队列来模拟在停车场外排队等待进入的车辆。

2. 管理员模块

2.1 管理员插入景点：管理员能够插入景点，并输入该景点的相关信息，如景点名称，与其他可达景点的路线和距离，该景点的介绍，欢迎度等。为放防止景点被孤立，管理员需要输入与该景点连通的路径。系统能够自动生成发布景点的公告。

2.2 管理员删除景点：管理员能够删除一个景点，删除景点的同时，删除与之相连的所有景点。系统能够自动发布景点删除的公告。

2.3 管理员新增道路：管理员能够新增两个景点之间的道路，并自动生成景点相连的公告。

2.4 管理员删除道路：管理员能够删除两个景点之间的通路，并自动生成景点间通路被删除的公告告知游客。

3. 系统模块

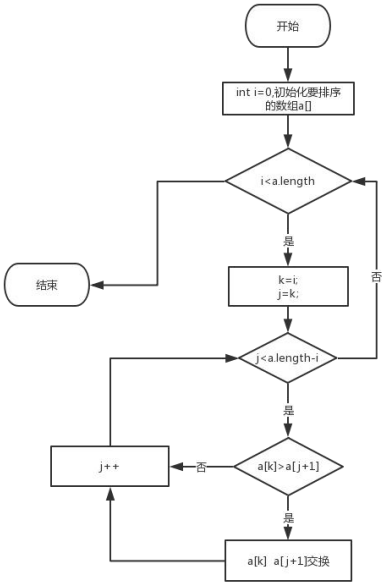
3.1 系统生成菜单：系统初始化时，自动生成 9 个模块。并系统初始化时，会自动从文件中读取景点间的信息。系统退出时，会将所有新更新的信息写回文件中。

第二章 系统设计

一、 重要算法

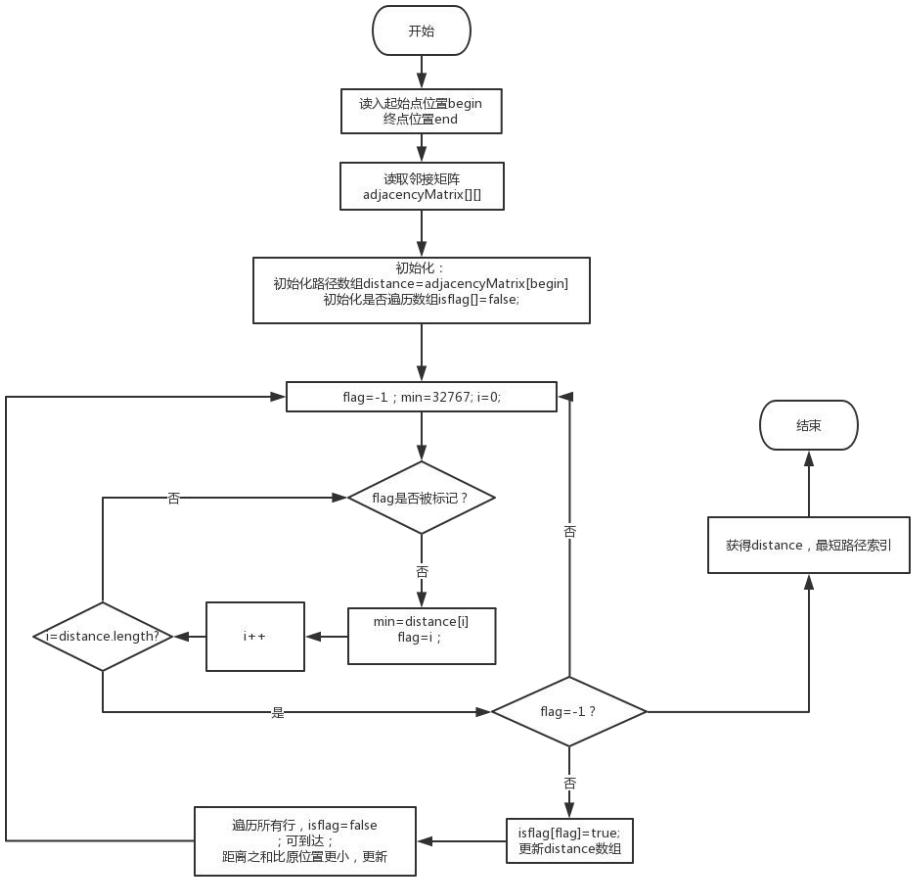
1. 排序法

1.1 冒泡排序法



2. 最短路径算法

2.1 Dijkstra 算法



2.2 Floyd 算法

第 1 步：使用两个集合 S 和 U。初始化 S。 矩阵 S 中顶点 a[i][j]的距离为顶点 i 到顶点 j 的权值；如果 i 和 j 不相邻，则 a[i][j]=∞。实际上，就是将图的原始矩阵复制到 S 中。

第 2 步：以第 1 个顶点为中介点，若 a[i][j] > a[i][0]+a[0][j]，则设 a[i][j]=a[i][0]+a[0][j]。同理，依次将其他顶点作为中介点，并更新 a[i][j]的大小。

2.3 算法区别：

1）迪杰斯特拉算法不能算有负权值的图，弗洛伊德算法能够算有负权值的图，并能够检测回路

2）迪杰斯特拉算法的复杂度为 O(n^2)；弗洛伊德算法的复杂度为 O(n^3)

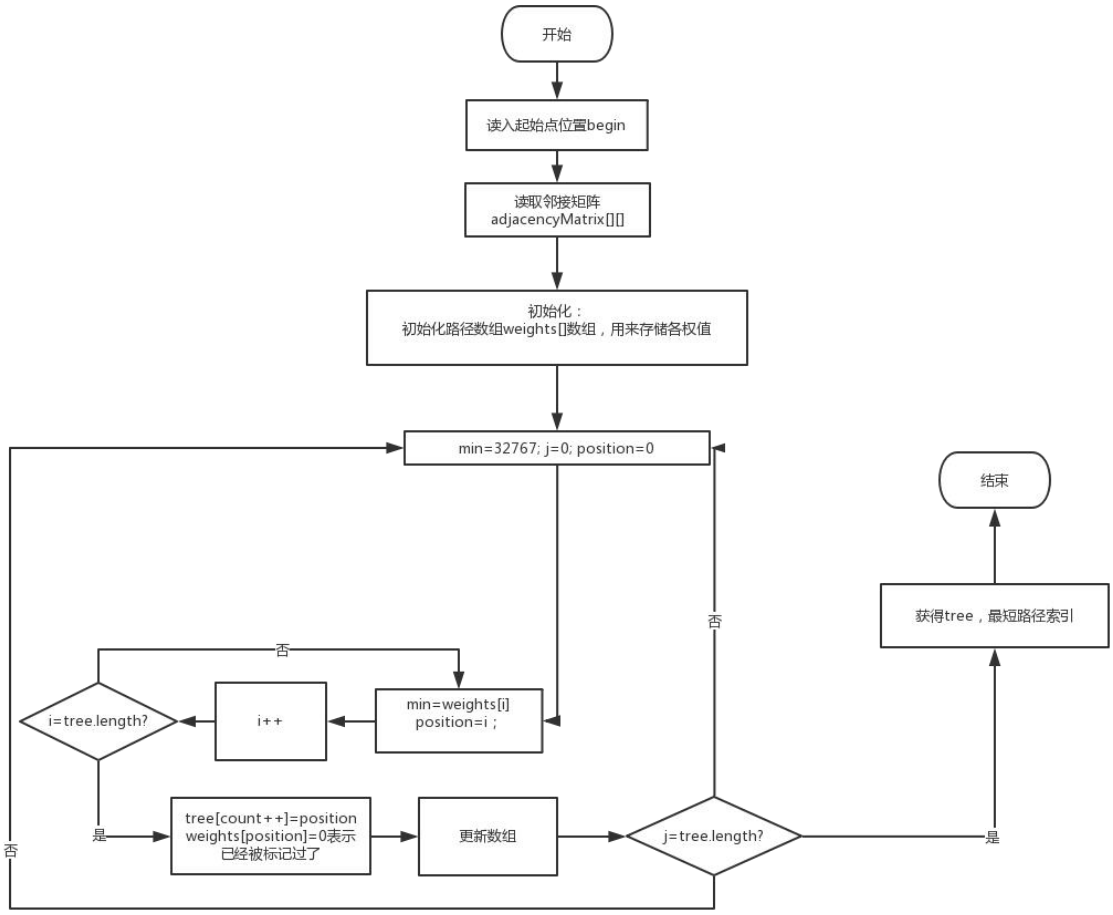
3）迪杰斯特拉算法只能计算该点到另一点的最短路径；弗洛伊德算法能够计算一点到其他个点的最短路径。

3. 最小生成树算法

3.1 Prim 算法

对于图 G 而言，V 是所有顶点的集合；现在，设置两个新的集合 U 和 T，其中 U 用于

存放 G 的最小生成树中的顶点, T 存放 G 的最小生成树中的边。从所有 u∈U, v∈(V-U) (V-U 表示出去 U 的所有顶点)的边中选取权值最小的边(u, v)，将顶点 v 加入集合 U 中，将边(u, v)加入集合 T 中，如此不断重复，直到 U=V 为止，最小生成树构造完毕，这时集合 T 中包含了最小生成树中的所有边。



3.3 算法区别

1）Prim 算法直接查找，多次寻找临边的权重最小值，而 Kruskal 是需要先对权重排序后查找的。

2）Kruskal 在算法效率上比 Prim 快，因为 Kruskal 只需一次对权重的排序就能找到最小生成树，而 Prim 算法需要多次对邻边排序才能找到。

二、 重要数据结构

1. 节点邻接表：每一个类除停车场类，都定义了该数据环境，用来存储节点信息和与该节点连接的路线链表。景点节点属性包括景点名称，景点介绍，景点的欢迎度，有无休息区，有无公厕。

2. 顺序栈：停车场类中，定义两个顺序栈，分别用来存储停车场里的车辆和从停车场让路的车辆，满足先进后出原则。汽车节点包括汽车的车牌号和到达时间，离开时间，进入停车场时间。

3. 队列：停车场类中，定义一个队列，用来存储便道上的车辆，满足先进先出原则。
4. 点的链表：景点节点中包含边的链表。
5. 边类包括边的起始景点，终点景点，两个景点间的距离。
6. 图类：包括了所有的景点，以及各种与图有关系的算法

第三章 系统实现

一、 游客模块

1. 查看景区分布图

1.1. 生成邻接表：初始化邻接表，将景点信息与景点相关的信息从文件中读取出来。对于每个顶点，创建链表用来存储相连的所有边。对于一个图，创建链表来存储图中的所有点。

1.1.1 读入景点信息，初始化图，将所有点加入图

```
while ((spotsInfo = InfoRead.readLine()) != null) {
String a[]=spotsInfo.split("    ");
Vertex vertex=new Vertex(a[0]);
initialGraph.addVertex(vertex);
}
```

1.1.2 读入边的信息，将所有点的连接边加入图。遍历所有点，依次加入

```
for(int i=0;i<initialGraph.getAllVertex().size();i++){if(initialGraph.getAllVertex().get(i).getSpotsName().equals(a[0])) {
        Edge linkedEdge=new Edge(first,second,Integer.parseInt(a[2]));
        initialGraph.getAllVertex().get(i).addEdge(linkedEdge);
    }else if(initialGraph.getAllVertex().get(i).getSpotsName().equals(a[1])){
        Edge linkedEdge=new Edge(second,first,Integer.parseInt(a[2]));
        initialGraph.getAllVertex().get(i).addEdge(linkedEdge);
    }
}
```

2. 查找景点

2.1 关键字查找及模糊关键字查找。使用 contains 做到模糊查找。

```
for (int i = 0; i < initialGraph.getAllVertex().size(); i++) {
```

```
if (spotName.equals(initialGraph.getAllVertex().get(i).getSpotsName())||
        initialGraph.   getAllVertex().get(i).getIntro().contains(spotName)) {
            /*  输出景点信息*/
              break;
        }
        if(i==initialGraph.getAllVertex().size()-1){System.out.println("\t\t\t\t 对
不起，景区没有此景点");}
        }
```

3. 景点排序

3.1 使用冒泡排序法：挨个遍历，复杂度为 O(n^2)

```
for (int i=0;i<sort.length;i++){
    for (int j=1;j<sort.length-i;j++){
        if
(Integer.parseInt(sort[j-1].getWelcomePoint())<Integer.parseInt(sort[j].getWelcom
ePoint())){
            Vertex temp=sort[j-1];
            sort[j-1]=sort[j];
            sort[j]=temp;
        }}}
```

4. 最快到达另一个景点路径

4.1 Dijkstra 算法

```
int adjacencyMatrix[][]=getAdjacencyMatrix(this);//我的邻接矩阵
int distance[]=new int[this.getAllVertex().size()];//用来更新的数组
boolean isflag[]=new boolean[distance.length];//标记该索引位置是否已标记
int shortestPath[]=new int[distance.length];
for (int i=0;i<isflag.length;i++){
    isflag[i]=false;
}
int begin=this.allVertex.indexOf(root);
int end=this.allVertex.indexOf(destinationVertex);
//从第一行开始
```

```
distance = adjacencyMatrix[begin];
for (int i=0;i<distance.length;i++){
    if(i!=begin&&distance[i]<32767) shortestPath[i] = begin;
    else shortestPath[i]=-1;
}
isflag[begin]=true;
while (true){
    //首先找到 distance 中的最短路径
    int min=32767;
    int flag=-1;
    for (int i=0;i<distance.length;i++){
        if (!isflag[i]){//若这行未被标记过，则进行寻找最小值
            if (distance[i]<min){
                min=distance[i];//找到这行中最小的
                flag=i;
            }
        }
    }//找完未更新 flag 说明没有通路
    if (flag==-1){
        break;
    }
    //标记这行
    isflag[flag]=true;
    //更新最短的路径
    for (int i=0;i<distance.length;i++){
        if
(!isflag[i]&&adjacencyMatrix[flag][i]!=32767&&(distance[flag]+adjacencyMatrix
[flag][i])<distance[i]){//未被标记并且是相连的
            distance[i]=distance[flag]+adjacencyMatrix[flag][i];
            shortestPath[i]=flag;
        }
    }

}
```

4.2 Floyd 算法：能够算出该点到其他点的距离

```
int begin=this.allVertex.indexOf(start);
int theEnd=this.allVertex.indexOf(end);
int adjacencyMatrix[][] = getAdjacencyMatrix(this);//我的邻接矩阵
int shortestPath[][] = new int[this.allVertex.size()][this.allVertex.size()];

//将 shortestPath 数组初始化
for (int i = 0; i < shortestPath.length; i++) {
    for (int j = 0; j < shortestPath.length; j++) {
        shortestPath[i][j] = j;
    }
}

for (int k = 0; k < adjacencyMatrix.length; k++) {
    for (int i = 0; i < adjacencyMatrix.length; i++) {
        for (int j = 0; j < adjacencyMatrix.length; j++) {
            int temp;
            if(adjacencyMatrix[i][k]==32767||adjacencyMatrix[k][j]==32767){
                temp=32767;
            }else {
                temp=adjacencyMatrix[i][k]+adjacencyMatrix[k][j];
            }
            if (adjacencyMatrix[i][j] > temp) {
                adjacencyMatrix[i][j] = temp;
                shortestPath[i][j] = shortestPath[i][k];
            }
        }
    }
}
```

5. 景区导游线路图

5.1 基本思路：使用最小生成树+Dijikstra 算法实现。先使用 prim 算法得到最小生成树。遍历最小生成树中的结点，如果前一个点和后一个点不相连，则使

用 Dijkstra 算法将两点相连再继续遍历。当所有点都经过后，使用 Dijkstra 算法将起点或目的地点加到数组最后，形成最终的回路。

5.2 Prim 算法

```
int start=this.getAllVertex().indexOf(root);
int adjacencyMatrix[][]=this.getAdjacencyMatrix(this);
int tree[]=new int[this.allVertex.size()];//用来存储最小生成树
int weights[]=new int[this.allVertex.size()];//用来存储各权值
int count=0;
//进行初始化
if (this.getAllVertex().indexOf(root)!=-1) {
    tree[count++] = this.getAllVertex().indexOf(root);//树的第一个结点为开始的结点

    for (int i = 0; i < this.allVertex.size(); i++) {
        weights[i] = adjacencyMatrix[start][i];//将权值赋为所在行的权值, 即代表开始的结点到各个结点的距离
    }
    for (int j = 0; j < this.allVertex.size(); j++) {
        if (start == j)
            continue;
        int positon = 0;
        int min = 32767;
        //选出权值最小的结点
        for (int i = 0; i < this.allVertex.size(); i++) {
            if (weights[i] < min && weights[i] != 0) {
                min = weights[i];
                positon = i;//获得权值最小的点的位置
            }
        }
        //将获得的点键入结果集中
        tree[count++] = positon;
        //并将标记的位置重置为 0，表明已经被标记过了
        weights[positon] = 0;
        //更新权值数组
```

```
            for (int i = 0; i < this.allVertex.size(); i++) {
                if (weights[i] != 0 && adjacencyMatrix[positon][i] < weights[i]) {
                    weights[i] = adjacencyMatrix[positon][i];
                }
            }

        }
```

6. 停车场进出

6.1 使用两个 stack 和一个 queue 来实现停车场。一个 Stack 来实现停车场，另一个 stack 来存放从 Stack 中暂时退出的车辆。Queue 来实现排队，当停车场满时，将车放到 queue 中，只有当有车离开时，才能从队列中出来进入停车场

6.2 停车

```
if (parkingSlot.isFull()){   //如果停车场满了
    waitingLine.add(car);   //停到过道
}else {
    parkingSlot.push(car);   //否则，停入停车场
    car.setInParkingslotTime(arriveTime);
}
```

6.3 离开停车场

```
while (!parkingSlot.isEmpty()){
    outCar=parkingSlot.pop();//将顶部的元素推出
    if (outCar.getCarNumber().equals(number)){
//若是要离开的车，则停止算法，将原来的车推回停车场
        outCar.setLeaveTime(leaveTime);//设置离开时间
        while (!outCars.isEmpty()){
            VirtualCar backCar=outCars.pop();
            parkingSlot.push(backCar);
        }
        break;
    }else {
        outCars.push(outCar);//将顶部的车推入另一个 Stack
```

```
        }
    }
    //当有车离开时，将等待中的车推入停车场
    if (!waitingLine.empty()){//若等待的队列不为空
        VirtualCar inCar=waitingLine.remove();
        inCar.setInParkingslotTime(leaveTime);
        parkingSlot.push(inCar);
    }
```

7. 查看公告：每次管理员改变景区信息就实例化 Notice 对象，存在数组。

二、管理员模块

1. 插入景点：将景点实例化，并插入图中

```
Vertex newSpot=new Vertex(spotName);
initialGraph.addVertex(newSpot);
```

2. 删除景点：先遍历该点的所有边，将对应景点的连接边删除，再删除此点

```
for (int i=0;i<initialGraph.getAllVertex().size();i++){
    if (initialGraph.getAllVertex().get(i).getSpotsName().equals(spotName)){
//含有此点，删除
        for (int j=0;j<initialGraph.getAllVertex().get(i).getAllEdge().size();j++){
//获得每条与改点相连的边
            //获得连接边下一个结点，删除此边。
            Vertex nextSpot=initialGraph.getAllVertex().get(i).getAllEdge().get(j).getDestinationSpot();
nextSpot.getAllEdge().remove(nextSpot.getLinkedEdge(initialGraph.getAllVertex().get(i)));}
        initialGraph.getAllVertex().remove(initialGraph.getAllVertex().get(i));
        return;
    }
    if(i==initialGraph.getAllVertex().size()-1){
```

```
                System.out.println("输入错误，无此景点");
            }
        }
```

3. 插入道路

   类似于插入景点，实例化后，加入对应点

4. 删除道路

   找到两个点，删除连接边

5. 自动发布公告

   每次管理员进行操作，都自动实例化 Notice 对象，加入公告数组。

```
Notice newNotice=new Notice();
newNotice.setNotice("景区删除道路: "+first+" 与 "+second+" 已不能通行 ");
notices.add(newNotice);
System.out.println("发布公告! ");
```

三、系统模块

1. 读取文件

   使用 FileInputStream 解决读取中文字符出乱码的问题

```
edgeIn = new InputStreamReader(new FileInputStream(edgeFileName), "GBK");
infoIn=new InputStreamReader(new FileInputStream(infoFileName), "GBK");
```

2. 写入文件

   使用 FileOutputStream 解决写入出现乱码的问题。

```
outInfo = new OutputStreamWriter(new FileOutputStream(fileName), "GBK");
```

   当每次系统正常退出时，保存所有文件。

第四章 系统测试

一、 测试方法:

1. 单元测试：将编写的每个模块都进行一次测试，保证模块的正确性
2. 集成测试：将各个模块组成的大模块进行测试

3. 系统测试：系统编写完后，将整体进行

二、 进入系统：输出主菜单

```
        欢迎使用景区管理系统
**********************************
    1. 景区景点分布图
    2. 查找景点
    3. 景点排序榜
    4. 最快到达景点
    5. 景区导游线路图
    6. 停车场记录
    7. 管理员登录
    8. 查看发布的公告
    9. 退出
**********************************
请输入你的选择：
```

三、输出景区景点分布图：读入文件

```
北门 -- 9 -- 狮子山 -- 8 -- 仙云石
狮子山 -- 9 -- 北门 -- 7 -- 一线天 -- 6 -- 飞流瀑
仙云石 -- 8 -- 北门 -- 4 -- 仙武湖 -- 5 -- 九曲桥
观云台 -- 11 -- 一线天 -- 3 -- 飞流瀑 -- 15 -- 红叶亭 -- 16 -- 碧水潭
一线天 -- 7 -- 狮子山 -- 11 -- 观云台 -- 10 -- 花卉园
飞流瀑 -- 6 -- 狮子山 -- 3 -- 观云台
仙武湖 -- 4 -- 仙云石 -- 7 -- 九曲桥 -- 20 -- 碧水亭
九曲桥 -- 5 -- 仙云石 -- 7 -- 仙武湖 -- 20 -- 朝日峰
花卉园 -- 10 -- 一线天 -- 9 -- 红叶亭
红叶亭 -- 15 -- 观云台 -- 9 -- 花卉园 -- 10 -- 朝日峰
碧水亭 -- 20 -- 仙武湖 -- 17 -- 朝日峰
朝日峰 -- 20 -- 九曲桥 -- 10 -- 红叶亭 -- 17 -- 碧水亭
碧水潭 -- 16 -- 观云台
```

	北门	狮子山	仙云石	观云台	一线天	飞流瀑	仙武湖	九曲桥	花卉园	红叶亭	碧水亭	朝日峰	碧水潭
北门	0	9	8	32767	32767	32767	32767	32767	32767	32767	32767	32767	32767
狮子山	9	0	32767	32767	7	6	32767	32767	32767	32767	32767	32767	32767
仙云石	8	32767	0	32767	32767	32767	4	5	32767	32767	32767	32767	32767
观云台	32767	32767	32767	0	11	3	32767	32767	32767	15	32767	32767	16
一线天	32767	7	32767	11	0	32767	32767	32767	10	32767	32767	32767	32767
飞流瀑	32767	6	32767	3	32767	0	32767	32767	32767	32767	32767	32767	32767
仙武湖	32767	32767	4	32767	32767	32767	0	7	32767	32767	20	32767	32767
九曲桥	32767	32767	5	32767	32767	32767	7	0	32767	32767	32767	20	32767
花卉园	32767	32767	32767	32767	10	32767	32767	32767	0	9	32767	32767	32767
红叶亭	32767	32767	32767	15	32767	32767	32767	32767	9	0	32767	10	32767
碧水亭	32767	32767	32767	32767	32767	32767	20	32767	32767	32767	0	17	32767
朝日峰	32767	32767	32767	32767	32767	32767	32767	20	32767	10	17	0	32767
碧水潭	32767	32767	32767	16	32767	32767	32767	32767	32767	32767	32767	32767	0

四、查找景点：输入关键字（可为景点名称也可在景区介绍中模糊搜索）

```
大门
**********************************************
    景点名称： 北门
    景点介绍： 景区大门
    热度： 50
    休息区： 无
    公测： 有
    连接的景点：2
```

五、景点排序：输入 1/2/3（图为输入 1），输出景区欢迎度排行榜

输入

```
请输入你的排序方式：
1. 按欢迎度排序
2. 按人流量排序
3. 返回菜单
```

输出

```
观云台   热度：  100
飞流瀑   热度：  100
仙云石   热度：  90
朝日峰   热度：  90
碧水潭   热度：  90
狮子山   热度：  80
一线天   热度：  80
仙武湖   热度：  70
红叶亭   热度：  70
碧水亭   热度：  70
九曲桥   热度：  60
花卉园   热度：  60
北门   热度：  50
**********
```

六、**最短路径**: 输入你的起点以及你想去的地方, 输出最短路径推荐 (分别为 Dijkstra 算法和 Floyd 算法)

输入

```
请输入你的起点
北门
请输入你的终点
一线天
```

输出

```
Dijkstra
最短路径推荐为：
北门   ->   狮子山   ->   一线天
Floyd
最短路程为：    16
最短路径推荐为：
北门   ->   狮子山   ->   一线天
```

七、 回路

输入

```
为你列出导游推荐路线，请输入你所在的位置，我们为你安排最佳旅游线路
你的起点：
北门
是否回到原点？1.是 2. 否
2
请输入你最后想到达的地方
一线天
```

输出

```
最小生成树为：  北门   仙云石   仙武湖   九曲桥   狮子山   飞流瀑   观云台   一线天   花卉园   红叶亭   朝日峰   碧水潭   碧水亭
北门   ->   仙云石   ->   仙武湖   ->   九曲桥   ->   仙云石   ->   北门   ->   狮子山   ->   飞流瀑   ->   观云台   ->   一线天   ->

花卉园   ->   红叶亭   ->   朝日峰   ->   红叶亭   ->   观云台   ->   碧水潭   ->   碧水潭   ->   观云台   ->   红叶亭   ->   朝日峰   ->   碧水亭   ->   碧水潭   ->   观云台   ->   一线天
```

八、 停车场管理

输入选择

请输入你的选择：

6

1. 管理汽车
2. 查看记录
3. 退出

输出当前状态

*******停车场中车辆******

****已有车辆：4****

辽A00000

辽A00001

辽A00002

辽A00003

输入车牌号及到达时间

输入你的车牌号

辽A00005

输入到达时间

10：10

1. 停车
2. 离开车库
3. 退出

输出各个数据结构的状态

****停车****

车牌号为： 辽A00005

到达时间为：10：10

*******停车场中车辆******

****已有车辆：5****

辽A00000

辽A00001

辽A00002

辽A00003

辽A00005

*******暂停车场中车辆******

****已有车辆：0****

*******过道中车辆******

****已有车辆：0在等待****

九、 管理员维护

选择成功后

一线天

删除成功！

此时

	北门	狮子山	仙云石	观云台	飞流瀑	仙武湖	九曲桥	花卉园	红叶亭	碧水亭	朝日峰	碧水潭
北门	0	9	8	32767	32767	32767	32767	32767	32767	32767	32767	32767
狮子山	9	0	32767	32767	6	32767	32767	32767	32767	32767	32767	32767
仙云石	8	32767	0	32767	32767	4	5	32767	32767	32767	32767	32767
观云台	32767	32767	32767	0	3	32767	32767	32767	15	32767	32767	16
飞流瀑	32767	6	32767	3	0	32767	32767	32767	32767	32767	32767	32767
仙武湖	32767	32767	4	32767	32767	0	7	32767	32767	20	32767	32767
九曲桥	32767	32767	5	32767	32767	7	0	32767	32767	32767	20	32767
花卉园	32767	32767	32767	32767	32767	32767	32767	0	9	32767	32767	32767
红叶亭	32767	32767	32767	15	32767	32767	32767	9	0	32767	10	32767
碧水亭	32767	32767	32767	32767	32767	20	32767	32767	32767	0	17	32767
朝日峰	32767	32767	32767	32767	32767	32767	20	32767	10	17	0	32767
碧水潭	32767	32767	32767	16	32767	32767	32767	32767	32767	32767	32767	0

一线天已消失

十、 查看公告

回到菜单

```
**************************
请输入你的选择：
8
景区删除：一线天
```

可看到公告