# 课程编号：C0801207040

数据结构与算法

课程设计报告



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **姓名** | **贾敬哲** | **学号** | | **20175276** |
| **班级** | **软件工程1701** | **指导教师** | | **费园园** |
| **实验名称** | **数据结构与算法课程设计** | | | |
| **开设学期** | **2018-2019学年第1学期** | | | |
| **开设时间** | **第20周——第21周** | | | |
| **报告日期** | **2019.1.23** | | | |
| **评定成绩** |  | | **评定人** |  |
| **评定日期** | **2019-1-25** |

**东北大学软件学院**

第一章 系统分析

描述系统背景，系统的功能需求，分析系统可能的解决方案，介绍自己的主要工作等内容。

* 1. 系统背景

随着科技发展，老式的旅游管理不再受欢迎，并且老式的旅游管理有很多限制，随着国内游客数量的增加，游客要求的提示，我们需要给游客和管理者都提供方便快捷的工具。

所以研究和开发高质量、高性能的景区信息管理系统变得尤为重要，可以提高景区的管理水平，解放了繁重的体力劳动和脑力劳动，极大的改善了用户服务的质量，提高了景区的信誉。

1.2系统的功能需求

现有某景区需要开发一个景区信息管理系统，具体问题有：建立一个主程序应用菜单选项，方便管理员和游客执行他们想要的功能。为了能够适应不同景区的需求，系统需要能够创建景点分布图和输出景点分布图，对于创建，可以逐条添加景点信息和路径信息；对于输出，需要输出景点分布图的邻接矩阵和景点分布图。而且，系统需要能够制订旅游景点导游线路策略，以便于游客游览所有景点，即生成一个导游线路图；同时，在导游线路图中，还需要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特定的景点。所以两个景点之间的最短路径和最短距离很重要，系统需要提供输出两个景点间的最短路径和最短距离的功能。系统同样需要提供景点搜索和景点排序的功能，游客通过输入关键字来了解相关景点以便计划好游览地点和顺序；而景点排序可以通过景点的欢迎度或与景点相邻的边数来排序，让游客对各个景点有个直观的感受。最后，景区中还有固定容量的停车场，系统需要对到达车辆和离开车辆进行管理；当车辆到达时，如果停车场中还有剩余位置，则直接进入并开始计时，否则在便道等待；当车辆离开时，在它之后进入的车辆必须先退出车场为它让路，待该辆车开出大门外，其它车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。

1.3解决方案分析

通过对需求进行分析，功能性需求如下：

1. 主程序应用菜单，便于用户操作和选择。
2. 输出景点分布图，适应不同景区的需求。利用python的networkx第三方库进行可视化绘图
3. 输出导游路线图及图中的回路，便于游客游览所有景点，并能够根据回路优化导游 路线图，可以使用普里姆算法（Prim算法）实现。
4. 输出两个景点之间的最短路径和距离，为一些不愿按线路走的游客提供信息服务， 可以使用迪杰斯特拉算法（heap优化）或者弗洛伊德算法（spfa）实现。
5. 景点查找和排序，便于游客提前计划游览地点和顺序，可以使用KMP匹配算法、冒 泡排序算法、堆排序以及快速排序算法。
6. 停车场管理系统，能够对车场进行初始化、车辆添加和车辆离开功能。使用栈模拟 停车场和暂时停车场，使用队列模拟候车道。
7. 管理员操作，能够对图进行增删改查以及发布通告的操作。

1.4工作内容

使用更优化的算法，实现本工程的全部功能

第二章 系统设计

描述系统中自己的数据结构与算法的设计方案。

**2.1总体设计**

**2.1.1系统初始化：**

系统从文本文件中读取景点信息和路信息，采用带权无向图的邻接链表的数据结构存储景区信息。文件中包含图中路径的起点、终点和路程。

**2.1.2数据结构：**

景点结构：

**public class** Attractions {  
 **private** String **spotName**;*//景点名字* **private int cost**;*//用于实现堆优化* **private** String **introductionToAttractions**;  
 **private int attractionsWelcome**;  
 **private** Boolean **seatingArea**;  
 **private** Boolean **publicToilet**;  
 **private** ArrayList<Edge> **edgeArrayList** = **new** ArrayList<>();*//用于实现邻接表*

}

景点之间边的结构：

**public class** Edge{  
 **public** String **to**;  
 **public int cost**;

}

汽车类的结构：

**public class** zanlind{String **numberOfCar** = **""**;  
 Date **ar\_time** ;

}

管理员结构：

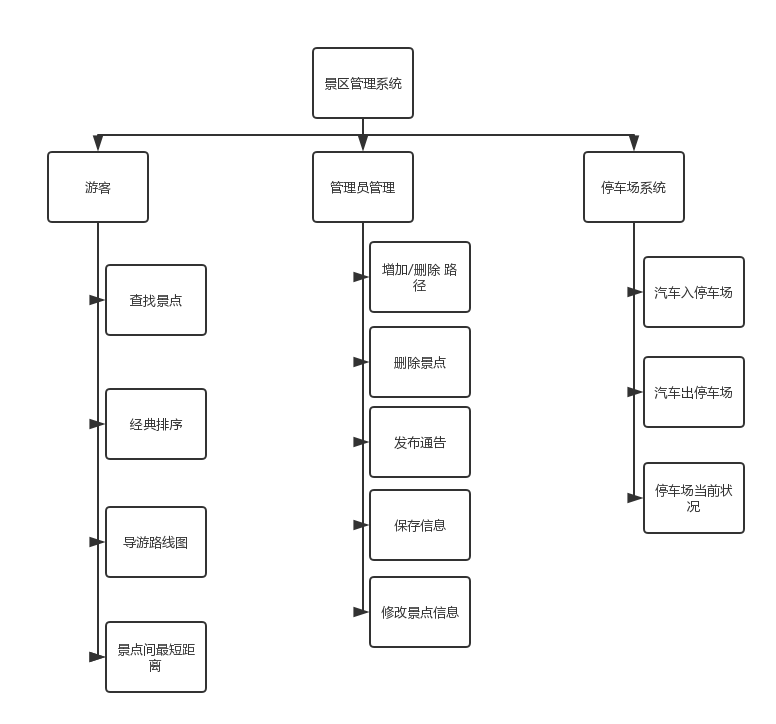
**public class** administrator {  
**private** String **admin\_Name** = **new** String();  
 **private** String **admin\_PWD** = **new** String();

}

模拟的栈:

模拟队列:

**2.1.3系统结构：**



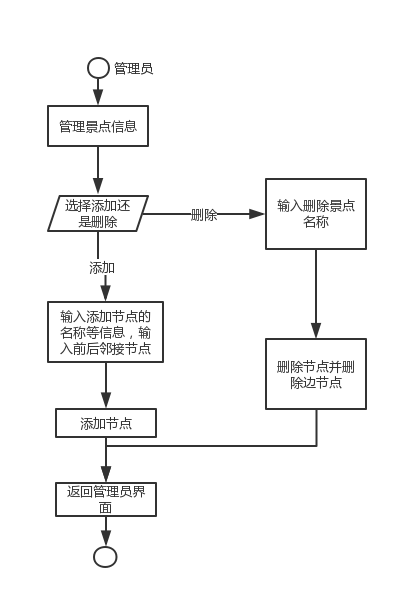
在针对需求分析后，景区管理系统可以分为管理员模块和游客模块：

（1）管理员功能

1.登录：实现管理员使用账户名和密码登录系统

2.公告发布：在景点信息管理选择菜单中添加公告。

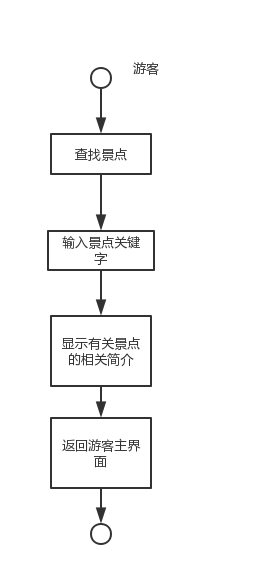
3.管理景点：实现景点及景点信息的增加，删除和修改的功能，景点的删除需要实现路的同时删除。



4.管理道路：实现道路的插入和删除的功能，维护各景点之间路的信息。

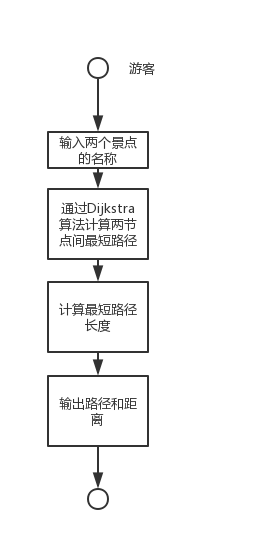
（2）游客功能

1.查询：根据用户输入的关键字进行景点的查找，关键字可以在景点名称也可以在景点介绍中。查找成功则返回景点的相关简介，如果查找不成功请给予正确提示。

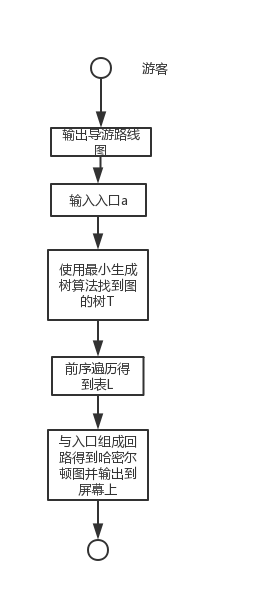


2.排序：可以根据不同的需求进行排序，用户可以选择，如按景点欢迎度或者景点的岔路数对景点进行排序，排序算法可以采用冒泡、快速、插入排序等。排序后的结果按照顺序显示在屏幕上。

3.景点之间最短路径：输出游客选择的两个景点间的最短旅游路径，按点与点的方式输出，并输出最短路径的距离。采用堆优化的Dijkstra算法或者spfa算法。

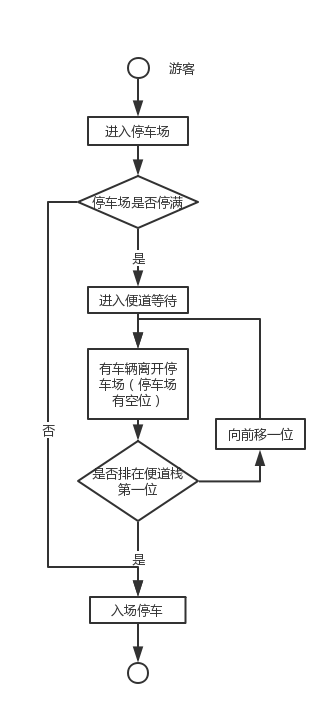


4.输出导游路线图：建立一个旅游路线图，以最小的耗费遍历所有的景点，并从出发点回来。采用最小哈密尔顿回路来解决，采用最小生成树算法。并将导游路线图输出。



（3）停车场功能：

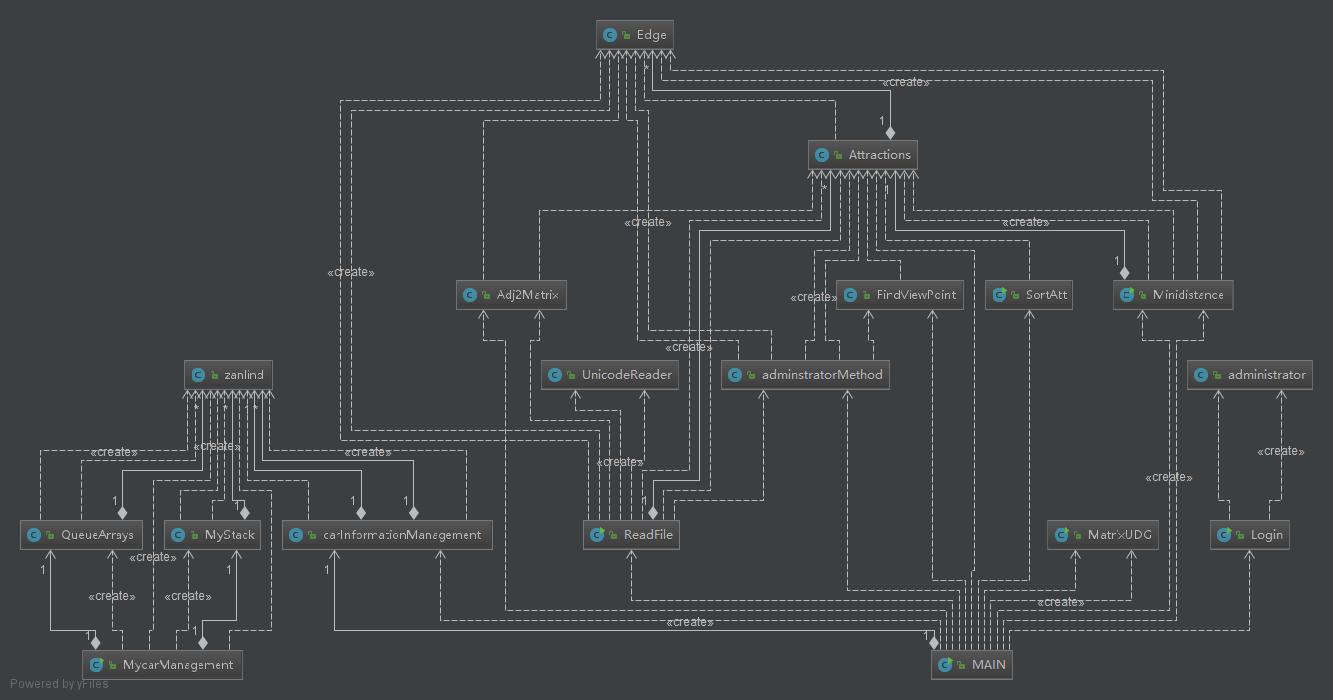
输出停车场车辆进出的信息：提供车辆进车场，出车场以及检查停车场信息的功能。进车场时提供车牌信息，停车时间，停车位置。如果车场已满，显示等待在变道的位置。出场时显示车牌以及停留时间和缴费信息。



第三章 系统实现

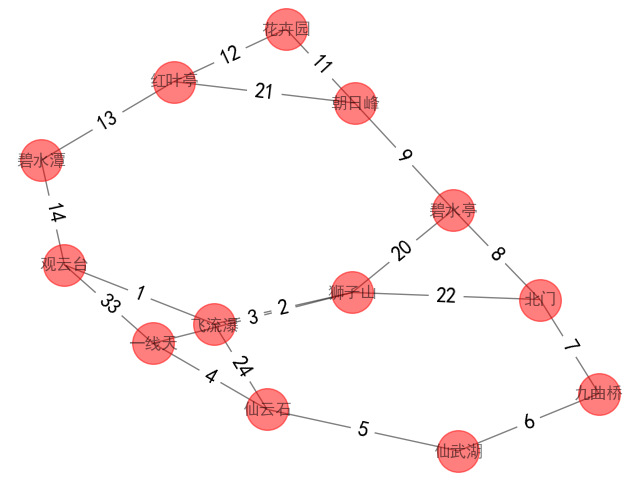
针对自己所设计的方案，描述具体的实现开发内容。

**3.1 UML图如图所示：**



**3.2python绘制可视化的景点分布图：**

利用python的networkx（网络处理库）和matplotlib（绘图库）进行读取文件及可视化的绘图。结果如图所示：



**3.3程序设计：**

3.3.1停车场模块：

按照实验要求的停车场功能，数据结构主要使用了栈和队列。主要是两个栈和一个队列，其中一个是停车场栈一个是临时停车场的栈还有一个是便道的队列。

1. 车辆到达时，将车辆对象添加入栈（这个栈重新写的，同时储存了一个时间信息，new Date），并在每一次判断是否有重复以及栈是否满了。

关键代码：

**public void** addCar(String numberOfCar) {  
 *//如果停车场不满 直接加入停车场中* **if** (*parkingLot*.size() < ***MAXSIZE***) {  
 *parkingLot*.push(**new** zanlind(numberOfCar));  
 }  
 *//如果停车场满了 则加入到便道中* **else** {  
 *sidewalk*.add(**new** zanlind(numberOfCar));  
 }  
}

汽车退出停车场时，需要先依据车牌号找出该车所在位置。用自定义栈实现，首先需要将停车场栈的数据移入临时栈空间，再移出如果车牌号相同，防止其加入停车场栈，若车辆在便道中，情况类似。关键代码：

**while** (!*myparkingLot*.isEmpty()) {  
 *mytmpPark*.push(*myparkingLot*.pop());  
}  
**while** (!*mytmpPark*.isEmpty()) {  
 zanlind tmp = *mytmpPark*.pop();  
 **if** (tmp.**numberOfCar**.equals(numberOfCar)) {  
 **continue**;  
 }  
 *myparkingLot*.push(tmp);  
}  
  
**while** (!*mysidewalk*.isEmpty()) {  
 zanlind zan = *mysidewalk*.outQueue();  
 **if** (zan.**numberOfCar**.equals(numberOfCar)) {  
 **resOfCost** = 0;  
 **continue**;  
 }  
 *tmpsidewalk*.inQueue(zan);  
}  
**while** (!*tmpsidewalk*.isEmpty()){  
 zanlind zan = *tmpsidewalk*.outQueue();  
 *mysidewalk*.inQueue(zan);  
}

若用jdk的栈和队列实现，只采用过滤器即可。

关键代码：

**if** (*parkingLot*.removeIf(zanlind -> numberOfCar.equals(zanlind.getNumberOfCar()))) {  
 zanlind tmpCar = *sidewalk*.poll();  
 *parkingLot*.push(tmpCar);  
} **else** {  
 *//尝试删除便道中车牌号为numberOfCar的汽车* **if** (*sidewalk*.removeIf(zanlind -> numberOfCar.equals(zanlind.getNumberOfCar()))) {  
 **resOfCost** = 0;  
 } **else** {  
  
 **return**;  
 }  
}

1. 景点的排序

主要采用的堆排序的方法：将初始待排序关键字序列(R1,R2....Rn)构建成大顶堆，此堆为初始的无序区；将堆顶元素R[1]与最后一个元素R[n]交换，此时得到新的无序区(R1,R2,......Rn-1)和新的有序区(Rn)；由于交换后新的堆顶R[1]可能违反堆的性质，因此需要对当前无序区(R1,R2,......Rn-1)调整为新堆，然后再次将R[1]与无序区最后一个元素交换，得到新的无序区(R1,R2....Rn-2)和新的有序区(Rn-1,Rn)。不断重复此过程直到有序区的元素个数为n-1，则整个排序过程完成。

关键代码：

**public static void** HeapSort(easyView []arr){  
 *//1.构建小顶堆* **for**(**int** i=arr.**length**/2-1;i>=0;i--){  
 *//从第一个非叶子结点从下至上，从右至左调整结构  
 adjustHeap*(arr,i,arr.**length**);  
 }  
 *//2.调整堆结构+交换堆顶元素与末尾元素* **for**(**int** j=arr.**length**-1;j>0;j--){  
 *swap*(arr,0,j);*//将堆顶元素与末尾元素进行交换  
 adjustHeap*(arr,0,j);*//重新对堆进行调整* }  
}**public static void** adjustHeap(easyView []arr,**int** i,**int** length){  
 easyView temp = arr[i];*//先取出当前元素i* **for**(**int** k=i\*2+1;k<length;k=k\*2+1){*//从i结点的左子结点开始，也就是2i+1处开始* **if**(k+1<length && arr[k].**population**<arr[k+1].**population**){*//如果左子结点小于右子结点，k指向右子结点* k++;  
 }  
 **if**(arr[k].**population** >temp.**population**){*//如果子节点大于父节点，将子节点值赋给父节点（不用进行交换）* arr[i] = arr[k];  
 i = k;  
 }**else**{  
 **break**;  
 }  
 }  
 arr[i] = temp;*//将temp值放到最终的位置*}

同时实现了其他排序算法：快速排序，直接插入排序，选择排序，折半插入排序，希尔排序，冒泡排序

1. 景点的最短路径：

利用优先队列实现dijkstra算法。Dijkstra算法是典型的最短路径算法，用于计算一个节点到其他所有节点的最短路径。设G=(V,E)是一个带权有向图，把图中顶点集合V分成两组，第一组为已求出最短路径的顶点集合（用S表示，初始时S中只有一个源点，以后每求得一条最短路径 , 就将加入到集合S中，直到全部顶点都加入到S中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用U表示），按最短路径长度的递增次序依次把第二组的顶点加入S中。在加入的过程中，总保持从源点v到S中各顶点的最短路径长度不大于从源点v到U中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S中的顶点的距离就是从v到此顶点的最短路径长度，U中的顶点的距离，是从v到此顶点只包括S中的顶点为中间顶点的当前最短路径长度。

堆优化的主要思想就是使用一个优先队列（就是每次弹出的元素一定是整个队列中最小的元素）来代替最近距离的查找，用邻接表代替邻接矩阵，这样可以大幅度节约时间开销。

关键代码：

**public void** dijkstra(String start,ArrayList<Attractions> AdjacencyList){  
 **for** (Attractions a:AdjacencyList  
 ) {  
 **judge**.put(a.getSpotName(),**false**);*//初始化HashMap* **len**.put(a.getSpotName(),***MAXN***);  
 }  
 **while** (!**que**.isEmpty()) **que**.poll();*//初始化堆  
  
 //起点大小为0* **len**.remove(start);  
 **len**.put(start,0);  
 *//将起点加入堆中* **que**.add(findOfAdj(start,AdjacencyList));  
 Attractions tmp = **null**;  
  
 **while** (!**que**.isEmpty()){  
 *//当队列（堆）不为空时* tmp = **que**.poll();  
 String tmpName = tmp.getSpotName();  
 **if**(**judge**.get(tmpName)) **continue**;  
 *//设置节点已经访问* **judge**.remove(tmpName);  
 **judge**.put(tmpName,**true**);  
  
 **for** (Edge edges:tmp.getEdgeArrayList()  
 ) {  
 String next = edges.**to**;  
 **int** cost =edges.**cost**;  
 **if**(!**judge**.get(next) && **len**.get(next)>**len**.get(tmpName)+cost){  
 **len**.remove(next);  
 **len**.put(next,**len**.get(tmpName)+cost);  
 *//松弛操作* Attractions tmpA = findOfAdj(next,AdjacencyList);  
 tmpA.setCost(**len**.get(tmpName)+cost);  
 **que**.add(tmpA);  
 **path**.put(next,tmpName);  
 }  
 }  
 }  
}

同时实现了spfa算法：SPFA(Shortest Path Faster Algorithm)算法是求单源最短路径的一种算法，它是Bellman-ford的队列优化，它是一种十分高效的最短路算法。数组d记录每个结点的最短路径估计值，用邻接表来存储图G。我们采取的方法是动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点u，并且用u点当前的最短路径估计值对离开u点所指向的结点v进行松弛操作，如果v点的最短路径估计值有所调整，且v点不在当前的队列中，就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止。

关键代码：

**public void** SPFA(String start ,ArrayList<Attractions>AdjacencyList){  
 **for** (Attractions a:AdjacencyList  
 ) {  
 **len**.put(a.getSpotName(),***MAXN***);  
 **judge**.put(a.getSpotName(),**false**);*//初始化HashMap* }  
 Queue<String> que = **new** LinkedList<String>();  
  
 *//起点大小为0* **len**.remove(start);  
 **len**.put(start,0);  
 que.offer(start);  
  
 **while** (!que.isEmpty()){  
 String tmp = que.poll();  
 Attractions tmpAtt = findOfAdj(tmp,AdjacencyList);  
 **judge**.remove(tmp);  
 **judge**.put(tmp,**false**);*//从queue中退出* **for** (Edge edges:tmpAtt.getEdgeArrayList()){  
 String to = edges.**to**;  
 **if** (**len**.get(to) == ***MAXN***||**len**.get(to)>(**len**.get(tmp)+edges.**cost**)){  
 **len**.remove(to);  
 **len**.put(to,**len**.get(tmp)+edges.**cost**);*//松弛(RELAX)操作* **if** (!**judge**.get(to)){  
 **judge**.remove(to);  
 **judge**.put(to,**true**);  
 que.offer(to);  
 }  
 }  
 }  
 }  
}

1. 最小生成树算法prim算法：从某个顶点开始，假设v0，此时v0属于最小生成树结点中的一个元素，该集合假设u，剩下的V-v0为待判定的点，此时选取u中的顶点到V-v0中顶点的一个路径最小的边，并且将其中非u中的顶点加入到u中，循环直到u中的顶点包含图所有的顶点为止。

关键代码：

**public void** primNo(**int** start) {  
 **int** num = **mVexs**.**length**; **int** index=0; String[] prims = **new** String[num]; **int**[] weights = **new int**[num]; prims[index++] = **mVexs**[start];  
**for** (**int** i = 0; i < num; i++ )  
 weights[i] = **mMatrix**[start][i];  
 weights[start] = 0;  
 **for** (**int** i = 0; i < num; i++) {  
 **if**(start == i)  
 **continue**;  
 **int** j = 0;  
 **int** k = 0;  
 **int** min = ***INF***;  
 **while** (j < num) {  
 *// 若weights[j]=0，意味着"第j个节点已经被排序过"(或者说已经加入了最小生成树中)。* **if** (weights[j] != 0 && weights[j] < min) {  
 min = weights[j];  
 k = j;  
 }  
 j++;  
 } prims[index++] = **mVexs**[k];  
 weights[k] = 0;  
 **for** (j = 0 ; j < num; j++) {  
 *// 当第j个节点没有被处理，并且需要更新时才被更新。* **if** (weights[j] != 0 && **mMatrix**[k][j] < weights[j])  
 weights[j] = **mMatrix**[k][j];  
 }  
 }  
 *// 计算最小生成树的权值* **int** sum = 0;  
 **for** (**int** i = 1; i < index; i++) {  
 **int** min = ***INF***;  
 *// 获取prims[i]在mMatrix中的位置* **int** n = getPosition(prims[i]);  
 *// 在vexs[0...i]中，找出到j的权值最小的顶点。* **for** (**int** j = 0; j < i; j++) {  
 **int** m = getPosition(prims[j]);  
 **if** (**mMatrix**[m][n]<min)  
 min = **mMatrix**[m][n];  
 }  
 sum += min;  
 }}

1. TSP问题实现：

利用python实现了

模拟退火算法：模拟退火算法来源于固体退火原理，是一种基于概率的算法，将固体加温至充分高，再让其徐徐冷却，加温时，固体内部[粒子](https://baike.baidu.com/item/%E7%B2%92%E5%AD%90/81757)随[温升](https://baike.baidu.com/item/%E6%B8%A9%E5%8D%87/10468133)变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到[平衡态](https://baike.baidu.com/item/%E5%B9%B3%E8%A1%A1%E6%80%81/8965512)，最后在[常温](https://baike.baidu.com/item/%E5%B8%B8%E6%B8%A9/1144996)时达到基态，内能减为最小。

禁忌算法：

禁忌（Tabu Search）算法是一种亚启发式(meta-heuristic)随机搜索算法，它从一个初始可行解出发，选择一系列的特定搜索方向（移动）作为试探，选择实现让特定的目标函数值变化最多的移动。为了避免陷入局部最优解，TS搜索中采用了一种灵活的“记忆”技术，对已经进行的优化过程进行记录和选择，指导下一步的搜索方向，这就是Tabu表的建立。

第四章 系统测试

描述针对所开发的系统，如何展开测试的内容。列出在给定开发算法上的具体的实验分析内容，并分析实验结果。

测试方法：

采用选取一定输入keyword的测试

4.1测试用例：

1.对输出旅游旅游路线图的测试：

（1）输入空的起点

（2）输入相同的节点和终点

2.对最短路径的测试：

（1）输入任意一个起始点和终点

（2）输入一个不存在的节点名和终点

3.对停车场的测试：

（1）输入车辆进入的车牌号码

（2）输入以及存在的车牌号码

（3）输入超过停车场数量的车辆个数

（4）输入停车场内第一个离开车辆信息

（5）初始化填入过大的数字

4.对景点管理的测试：

（1）随意在地图上点击新景点位置

（2）输入一个新景点信息

（3）删除一个连着边的景点

（4）删除不存在的景点

5.对路的管理的测试：

（1）输入一个新的边的信息

（2）删除一条不存在的边的信息。

4.2测试结果：

1.对输出旅游旅游路线图的测试结果：

（1）提示不能为空

（2）提示起点终点不能相同

2.对最短路径的测试：

（1）给出一条路径

（2）提示起始点和终点不能为空

3.对停车场的测试：

（1）显示车牌号和进入时间

（2）显示停车场已经有这辆车了

（3）大于停车场数量，停入便道中

（4）动态显示车辆离开过程

（5）提示停车场大小不能超过10

4.对景点管理的测试：

（1）显示新的景点样式

（2）地图上出现新的景点和名称

（3）景点和边都被删除

（4）提示没有这个节点

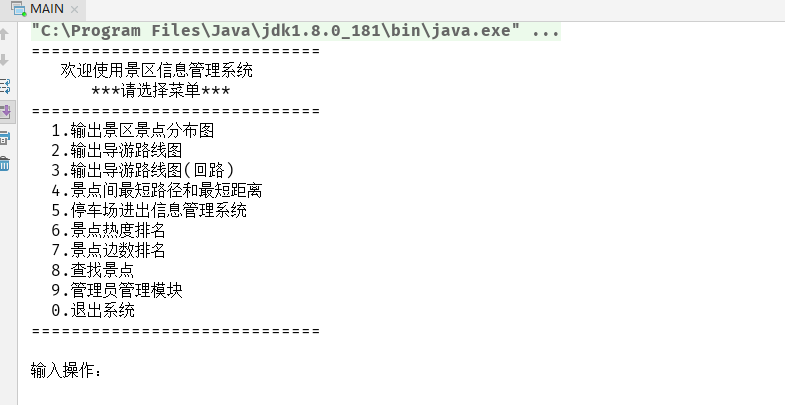
测试时遇到的主要问题：

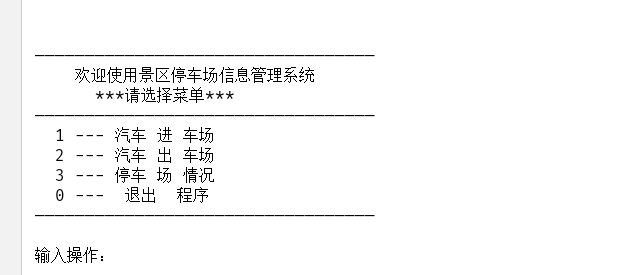
1. Java在读取utf-8的txt文件时，会读取到bom解决bom问题，使用了UnicodeReader类
2. Java中在遍历时，不允许对ArrayList进行修改需要首先进行tmp变量记录

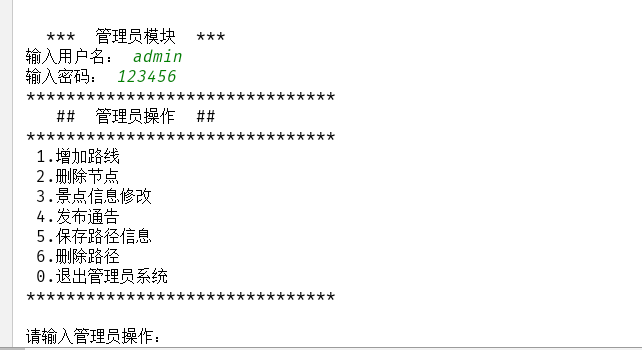
第五章 结论

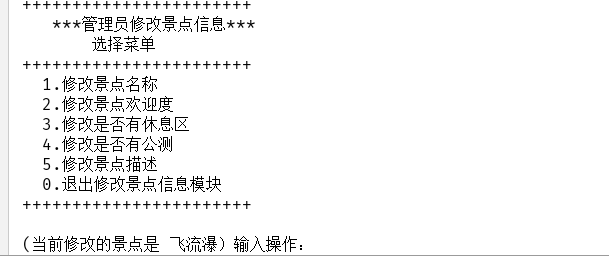
给出本次实验的结论。

运行结果：









参考文献

列出实验中所查阅的相关参考文献。

[1]BOM <https://blog.csdn.net/change_any_time/article/details/79572370>

[2] Java ConcurrentModificationException异常原因和解决方法 <https://www.cnblogs.com/dolphin0520/p/3933551.html>

[3]Dijkstra堆优化算法<https://www.cnblogs.com/cain-/p/7505684.html>

[4]spfa算法<http://www.cnblogs.com/shadowland/p/5870640.html>

[5]模拟退火算法<https://baike.baidu.com/item/%E6%A8%A1%E6%8B%9F%E9%80%80%E7%81%AB%E7%AE%97%E6%B3%95/355508?fr=aladdin>

[6]禁忌算法<https://baike.baidu.com/item/%E7%A6%81%E5%BF%8C%E6%90%9C%E7%B4%A2%E7%AE%97%E6%B3%95/6436980?fr=aladdin>

**附录：**

**《数据结构课程设计》实验成绩评定表**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **评价内容** | **具 体 要 求** | **分值** | **得分** |
|
| **平时表现** | **课程设计过程中，无缺勤、迟到、早退现象，学习态度积极。能够主动查阅文献，积极分析系统中数据结构与算法的多种可能的设计方案，并认真地对所选择方案进行实现、测试、分析与总结。** | **20** |  |
| **分析与解决问题的能力** | **能够理解复杂数据结构及算法的设计思路和基本原理；能够应用所学数据结构与算法等相关知识和技能去解决实验系统中要求的各个题目；设计或实现思路有独特见解。** | **20** |  |
| **实验结果与工作量** | **能够按实验要求完成系统的开发与测试，并达到实验要求的预期结果；能够认真记录实验数据，并对实验结果分析准确，归纳总结充分；工作量饱满。** | **20** |  |
| **报告质量** | **实验报告文字通顺、格式规范，体例符合要求；报告内容充实、正确，实验目的归纳合理到位。** | **40** |  |
| **总 分** | | |  |