# 课程编号：B080109010

数据结构课程设计

总结报告



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **姓名** | **亓懿康** | **学号** | | **20144638** |
| **班级** | **软件1403** | **指导教师** | |  |
| **实验名称** | **数据结构课程设计** | | | |
| **开设学期** | **2016-2017第一学期** | | | |
| **开设时间** | **第10周——第12周** | | | |
| **报告日期** | **2016.12.1** | | | |
| **评定成绩** |  | | **评定人** |  |
| **评定日期** | **2016-11-28** |

**东北大学软件学院**

第一章需求分析

1. 问题定义

实现景区信息管理系统，该系统包含的主要功能有：创建与输出导游路线图、计算与打印导游路线图、寻找两个景点之间的最短路径、根据关键词查询定位具体景点并打印景点相关信息、按照热度，岔路口等特性对各个景点进行排序。

另外还有一个子系统——停车场车辆进出信息记录系统，负责记录和处理停车和候车等信息、计算各车辆停车时长和应付的费用。

1. 问题分析

（1）用邻接链表和邻接矩阵的带权无向图来存储景点分布图的信息。通过邻接链表完成创建景区景点分布图，通过邻接矩阵来输出景区景点分布图。

（2）使用深度遍历的方法对景点进行遍历，并输出景点景区旅游信息管理系统中制订旅游景点导游线路策略。

（3）采用Dijkstra算法，使用一个优先级队列来更新每个节点间的距离，来达到求取景点之间的最短路径。

（4）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。排序算法可以采用冒泡、快速、插入排序等。

（5）以栈模拟停车场的相关信息，以队列模拟车场外的便道，通过从终端读取的数据来进行停车场的更新。

1. 研究意义

本次实验涉及到栈、队列、图的邻接链表和邻接矩阵储存形式、树等数据结构以及Dijkstra算法、Prim算法等算法。通过此次实验，了解并掌握数据结构和算法设计的方法，具备初步的独立分析和设计能力。学会了思考选择哪种数据结构和比较所采用的算法的复杂度和优良性。

第二章系统设计

2.1总体设计

**2.1.1设计思想**

这个系统我是用C语言写的，由于C语言没有关于类和对象的语法，所以我的设计思想是把每个子功能分别封装到一个头文件中。这样每个功能块都有对应的程序，最后把这些头文件include到main函数中，在main函数中实现系统。

**2.1.2基本数据结构**

这个程序分别用到了图/栈/队列的数据结构。

图用到了邻接链表和邻接矩阵两种储存结构。

**2.1.3抽象数据类型定义**

①景点信息的数据结构

typedef struct Information{

char introduction[200];

int popularity;

int roadnum;

}Information;

②图邻接链表

typedef struct ArcNode{

int adjvex;

struct ArcNode \*next;

int weight;

}ArcNode;

typedef struct Node{

VertexType name;

ArcNode \*first;

Information info;

}Node,AdjList[MAX\_VERTEX\_NUM];

typedef struct ALGraph{

int vexnum;//顶点数vertex

int arcnum;//边数arc

AdjList vertices;

}ALGraph;

③图邻接矩阵

typedef struct

{

int arcs[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM]; //邻接矩阵

int vexnum, arcnum; //无向网当前顶点数和弧数

VertexType vexs[MAX\_VERTEX\_NUM];//描述顶点的数组

Information info[MAX\_VERTEX\_NUM];//介绍顶点信息的数组

} MGraph;

④车辆信息

typedef struct zanlind{

int number; //汽车车号

int ar\_time; //汽车到达时间

}zanInode;

⑤栈数据结构

typedef struct

{

zanInode car[N];

int top; //栈指针\*/

} SqStack;

⑥队列数据结构

typedef struct

{

int CarNo[MM]; /\*车牌号\*/

int front,rear; /\*队首和队尾指针\*/

} SqQueue;

**2.1.4主要的操作的定义**

①ShowMenu.h里面包含了显示菜单功能。

②DtatStructure.h里面包含了图的数据结构定义以及图的链表矩阵储存形式转化/输出图的邻接矩阵等基本操作

③CreateGraph.h

④DFS.h

⑤Dijkstra.h

⑥Sort.h

⑦Search.h

⑧PrimTree.h

⑨ParkingLot.h

**2.1.5主要功能实现方法**

①void showMenu(); 显示菜单函数。

②int locateVex(ALGraph G, VertexType v); 输入名称定位该顶点为第几个顶点。

③void change(ALGraph &G,MGraph &M); 图的邻接链表储存形式转化为邻接矩阵储存形式。

④void createGraph(ALGraph &G); 创建图的邻接链表储存形式。

⑤void addInformation (ALGraph &G); 对图中顶点加入额外介绍信息。

⑥void DFSTraverse(ALGraph G); 图的深度遍历优先函数。

⑦void CreatTourSortGraph(ALGraph G,MGraph M,ALGraph &G1); 根据图的深度遍历优先创建导游图。

⑧void dijkstra(MGraph G,int v0,int vd); Dijkstra算法求景区两景点的最短路径跟最短距离。

⑨void prim(ALGraph G, int start); Prim算法求图的最小生成树问题，即景区修路所需材耗最小问题。

⑩void searchWord(ALGraph G); 按照关键字搜索图中顶点信息。

⑾void hotSort(ALGraph G);/void roadSort(ALGraph G); 按照热度或者岔路数对图中顶点进行排序。

⑿void parkinglotSystem(); 停车场系统主函数，调用栈或队列的一系列基本函数进行操作。

2.2程序设计

1. **创建图的邻接链表储存形式。**

第一步：先确定图的顶点数与边的个数。

printf("请输入顶点数和弧数：\n");

scanf("%d %d",&(G.vexnum),&(G.arcnum));

第二步：输入各顶点的名字，创建图的邻接链表数组。

printf("请输入各顶点名字：\n");//创建链表数组顶点

for(i=0; i<G.vexnum; i++)

{

scanf("%s",&(G.vertices[i].name));

G.vertices[i].first=NULL;

strcpy(G.vertices[i].info.introduction,"无景区介绍");

G.vertices[i].info.popularity=-1;

G.vertices[i].info.roadnum=-1;

}

第三步：输入各边的顶点及权值，输入到链表中。

for(i=0; i<G.arcnum; i++)

{

printf("\n输入第%d边的两个顶点以及该边的权值：",i+1);

scanf("%s %s %d",&name1,&name2,&weight);

int m=0,n=0;

m=locateVex(G,name1);

n=locateVex(G,name2);

p=(ArcNode\*)malloc(sizeof(ArcNode));//产生一个新的链表结点

p->weight=weight;

p->adjvex=n;

p->next=G.vertices[m].first;

G.vertices[m].first=p;

}

1. **图的深度遍历优先函数。**

第一步：先对顶点进行操作。

void DFS(ALGraph G, int v)

{

visited[v]=true;//true=1访问过

strcpy(vex[k++],G.vertices[v].name);

//printf("%s ",G.vertices[i].name);

p=G.vertices[v].first;

for(p=G.vertices[v].first; p ;p=p->next)

if(!visited[p->adjvex])

{

DFS(G,p->adjvex);

p=G.vertices[v].first;

}

}

第二步：对每个顶点进行遍历。

void DFSTraverse(ALGraph G)

{

int v;

//printf("导游路线为：");

for (v=0; v<G.vexnum; v++)

{

visited[v] = false;

}

for (v=0; v<G.vexnum; v++)

{

if (!visited[v])

{

DFS(G, v);

}

}

printf("\n");

}

1. **图的邻接链表向邻接矩阵储存形式转化。**

第一步：邻接矩阵的顶点数与边个数与邻接矩阵相等。

M.arcnum=G.arcnum;

M.vexnum=G.vexnum;

第二步：把邻接矩阵初始化赋值为最大值。

for(i=0;i<M.vexnum;i++){

for(j=0;j<M.vexnum;j++)

{ M.arcs[i][j]=INF;

}

}

第三步：把临界链表数组的每个链表分别赋值给邻接矩阵。

for(i=0;i<G.vexnum;i++){

strcpy(M.vexs[i],G.vertices[i].name);

strcpy(M.info[i].introduction,G.vertices[i].info.introduction);

M.info[i].popularity=G.vertices[i].info.popularity;

M.info[i].roadnum=G.vertices[i].info.roadnum;

p1=G.vertices[i].first;

while(p1){ //第一个点指向的点依次给邻接矩阵赋值

M.arcs[i][p1->adjvex]=p1->weight;

M.arcs[p1->adjvex][i]=M.arcs[i][p1->adjvex];

p1=p1->next;

}

}

1. **景区图的最短路径与最短距离，即Dijkstra算法。**

第一步：分别创建三个数组。

int dist[MAX\_VERTEX\_NUM],path[MAX\_VERTEX\_NUM];

//距离与路径，path表示这个顶点前的顶点

int s[MAX\_VERTEX\_NUM];

//已求出最短路径的顶点集合,每求得一条最短路径 , 就将其加入到集合S中

第二步：初始化。

for(i=0; i<n; i++){

dist[i]=G.arcs[v0][i];//各顶点到v0的距离设为直接到达距离

s[i]=0;

if(G.arcs[v0][i]<INF)//如果可以直接相连

path[i]=v0;

else

path[i]=-1;

}

s[v0]=1;//先把v0设置为已求出最短距离的顶点

path[v0]=0;//v0到自身的路径为0

第三步：寻找最短距离跟最短路径。

for(i=0; i<n; i++){

mindis=INF;//最短距离先设置为最大

u=-1;

for(j=0; j<n; j++)//找出距离当前顶点直接距离最近的顶点

if(s[j]==0&&dist[j]<mindis){

u=j;

mindis=dist[j];

}

s[u]=1;//以u作为当前顶点开始寻找

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

if(s[j]==0)

if(G.arcs[u][j]<INF&&dist[u]+G.arcs[u][j]<dist[j]){

dist[j]=dist[u]+G.arcs[u][j];

path[j]=u;

}

}

1. **输出道路修建规划图，即Prim最小生成树问题。**

第一步：定义两个数组。

VertexType prims[MAX\_VERTEX\_NUM];

// prim最小树的结果数组

int weights[MAX\_VERTEX\_NUM];

// 顶点间边的权值

第二步：初始化起始顶点。

strcpy(prims[index++], G.vertices[start].name);

// 初始化"顶点的权值数组"，

// 将每个顶点的权值初始化为"第start个顶点"到"该顶点"的权值。

for (i = 0; i < G.vexnum; i++ )

weights[i] = getWeight(G, start, i);

第三步：按Prim算法进行计算最小权值边，并把下一个顶点放到数组prims中。

for (i = 0; i < G.vexnum; i++)

{

// 由于从start开始的，因此不需要再对第start个顶点进行处理。

if(start == i){

continue;}

j = 0;

k = 0;

min = INF;// 在未被加入到最小生成树的顶点中，找出权值最小的顶点。

while (j < G.vexnum)

{

if (weights[j] != 0 && weights[j] < min) {// 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)。

min = weights[j];

k = j;

}

j++;

}// 经过上面的处理后，在未被加入到最小生成树的顶点中，权值最小的顶点是第k个顶点。

strcpy(prims[index++],G.vertices[k].name);// 将第k个顶点加入到最小生成树的结果数组中

weights[k] = 0;// 将"第k个顶点的权值"标记为0，意味着第k个顶点已经已经加入了最小树结果中。

for (j = 0 ; j < G.vexnum; j++){// 当第k个顶点被加入到最小生成树的结果数组中之后，更新其它顶点的权值。

tmp = getWeight(G, k, j);// 获取第k个顶点到第j个顶点的权值

if (weights[j] != 0 && tmp < weights[j])// 当第j个节点没有被处理，并且需要更新时才被更新。

weights[j] = tmp;

}

}

1. **按关键字搜索图中顶点信息。**

第一步：输入要查找的关键字。

printf("请输入要查找的关键字：\n");

scanf("%s",word);

第二步：按顶点顺序查找顶点名称与简介中是否包含关键字。

for(i=0;i<G.vexnum;i++){

searchnode[i].info.popularity=G.vertices[i].info.popularity;

searchnode[i].info.roadnum=G.vertices[i].info.roadnum;

strcpy(searchnode[i].name,G.vertices[i].name);

strcpy(searchnode[i].info.introduction,G.vertices[i].info.introduction);

}

第三步：输出查找到的顶点简介。

for(i=0;i<G.vexnum;i++){

if(strstr(searchnode[i].name,word)!=NULL){

printf("您要查找的景区为：%s\n",searchnode[i].name);

printf("景区简介为：%s\n",searchnode[i].info.introduction);

}

else if(strstr(searchnode[i].info.introduction,word)!=NULL){

printf("您要查找的景区为：%s\n",searchnode[i].name);

printf("景区简介为：%s\n",searchnode[i].info.introduction);

}

}

1. **停车场停车函数。**

fflush(stdin);

printf("请输入车牌号：");

scanf("%d",&no);

fflush(stdin);

printf("请输入车辆进入停车场的时间（整数）：");

scanf("%d",&time);

if (!StackFull(St)) /\*停车场不满\*/

{

Push(St,no,time);

printf("恭喜您停车成功！>>您的车在停车场位置为:%d\n",St->top+1);

}

else /\*停车场满\*/

{

if (!QueueFull(Qu)) /\*候车场不满\*/

{

enQueue(Qu,no);

printf("抱歉停车场已满，已停在候车场！>>您的车在候车场位置为:%d\n",Qu->rear);

}

else

printf("抱歉停车场与候车场均满，无法停车或候车！\n");

}

第三章系统实现与调试

3.1 导游线路图的创建

1.实现过程

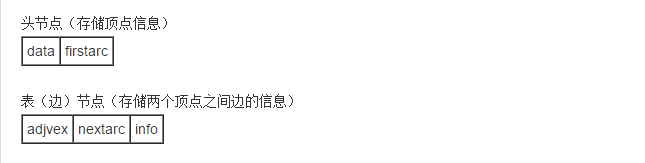
根据实验要求，创建一个存储各个节点信息的结构体struct Information，然后再创建邻接链表数据结构。先根据每个顶点名称创建链表数组，然后根据每条边的两个顶点以及权值添加链表结点。

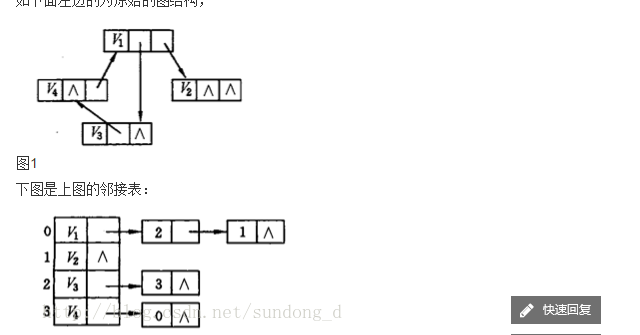
2.遇到的难点

对链表结点的储存形式一开始没有弄清楚。以为ArcNode结构体中的int adjvex应该换为节点的名字，用char表示。这个问题导致后面的编程变的十分麻烦。

3.解决方案

把Node结构体中的VertexType name换为int adjvex。结点ArcNode结构体中不包括节点名字，只需要adjvex变量表明该节点是第几个顶点即可。





3.2输出导游路线图及其图中的回路部分

1.实现过程

通过图的深度遍历优先算法来遍历图中各顶点，再建立导游路线图，实现依次遍历图中各点。

2.遇到的难点

遍历结点序列与导游线路图转换是难点

3.解决方案

设遍历结果为v1→v2→…→vi→vi+1→…→vn

　　对于结点vi和vi+1，如果vi和vi+1存在边，则直接转换。

　　否则，加入边vi→vi-1，如果vi-1和vi+1存在边，则加入边vi-1→vi+1。

　　再否则，加入边vi-1→vi-2，如果vi-2和vi+1存在边，则加入边vi-2→vi+1。

　　如果vi-2和vi+1还不存在边，继续回溯，一定能找到某个整数k（因为景点分布图是连通图），使得vi-k和vi+1存在边，则加入边vi-k→vi+1。在本任务中，转换后的线路图存于数组vex1中。

for (i=0; i<G.vexnum-1; i++)

{

k=0;

Is=true;

while (Is)

{

strcpy(vex1[n++],vex[i+k]);

if(IsEdge(G,M,vex[i+k],vex[i+1]))//如果他们之间有边就直接连接上这条边

Is=false;

else

k--;//如果没有就回溯，直到找到和vex[i+1]有边的

}

}

strcpy(vex1[n],vex[i]);//将最后一个顶点放进vex数组中

for(i=0; i<=n; i++)

{

visited[i]=false;

}

3.3输出两个景点之间最短路径和最短距离

1.实现过程

用Dijkstra算法计算最短路径。首先得到游客输入的起点，然后根据Dijkstra算法，计算出从该起点到其他所有顶点的最短路径，通过计算出起点与其他相邻节点的当前最短距离，对节点进行整体的遍历。最后找到最短的路径。然后再根据游客输入的终点，输出该起点到该终点的最短距离和最短路径。

2.遇到的难点

遇到的难点是我用了一个path数组来标记这个顶点前的顶点，怎么输出顶点就变得困难了。

3.解决方案

用了一个递归的函数来把此顶点前的顶点依次读取出来。

void ppth(int path[],int i,int v0,MGraph G){

int k;

k=path[i];

if(k==v0)

return;

ppth(path,k,v0,G);

printf("%s→",G.vexs[k]);

}

3.4输出道路修建规划图

1.实现过程

输出道路修建规划图也就是求图的最小生成树问题。我用到了Prim算法求图的最小生成树。分别用到了两个数组VertexType prims[MAX\_VERTEX\_NUM]用来记录prim最小树的结果数组；int weights[MAX\_VERTEX\_NUM]用来记录顶点间边的权值。

2.遇到的难点

因为Prim算法与Dijkstra算法大致相同，所以在编写Prim算法时遇到的困难较少。此外我还定义了一个sum整形变量来计算最小生成树的总权值。

3.解决方案

sum = 0;

for (i = 1; i < index; i++)

{

min = INF;

// 获取prims[i]在G中的位置

n = locateVex(G, prims[i]);

// 在vexs[0...i]中，找出到j的权值最小的顶点。

for (j = 0; j < i; j++)

{

m = locateVex(G, prims[j]);

tmp = getWeight(G, m, n);

if (tmp < min)

min = tmp;

}

sum += min;

}

3.5查找及排序

1.实现过程

查找函数我定义了一个数组Node searchnode[MAX\_VERTEX\_NUM]并把图中每个顶点的信息读取到searchnode数组中。第二步再根据关键字查找searchnode数组的当前元素的name和info.introduction变量是否包含关键字，如果包含则输出，如果不包含则查找下个元素。

排序函数用了冒泡排序的方法，对图中顶点的info.popularity跟info.roadnum分别排序，然后按照倒叙输出。

2.遇到的难点

实现冒泡排序是因为顶点信息不止包含info.popularity跟info.roadnum，还有其他信息。我在交换的时候只交换了这两个数据，导致出错。

3.解决方案

把这个元素进行交换，而不只是单单交换元素里的info.popularity跟info.roadnum变量。

for(i=0;i<G.vexnum - 1;i++){

for(j=0;j < G.vexnum - 1 - i;j++){

if(sortn[j].roadnum > sortn[j+1].roadnum){

temp=sortn[j];

sortn[j]=sortn[j+1];

sortn[j+1]=temp;

}

}

}

3.6输出车辆的进出信息

1.实现过程

首先按照要求，创建两个栈，一个队列，然后实现进场出场的逻辑，当车辆要进场时，判断停车场这个栈是否是满的，如果是满的，便放入队列中；如果没满就进行压栈操作，即进入停车场。如果车辆要出场时，我先让这辆车前面的车按顺序压入临时栈中，然后这辆车出场后，先对其进行收费，再将临时栈中的车出栈，回到停车场栈，最后再判断便道上是否有车，如果有车则进行压栈，进入停车场。

2.遇到的难点

栈和队列的实现。停车场中的车出栈时，要把后面进来的车放到另外一个临时栈中，等车开走后，再按原顺序把车放回到停车场中。

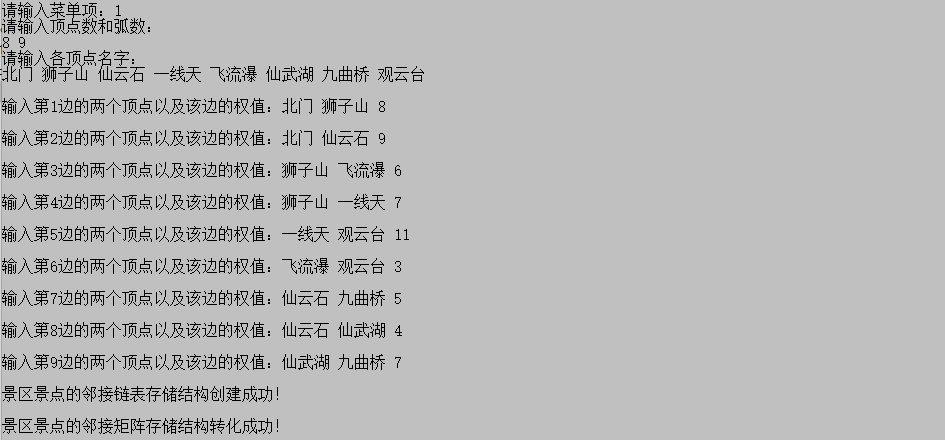
3.解决方案

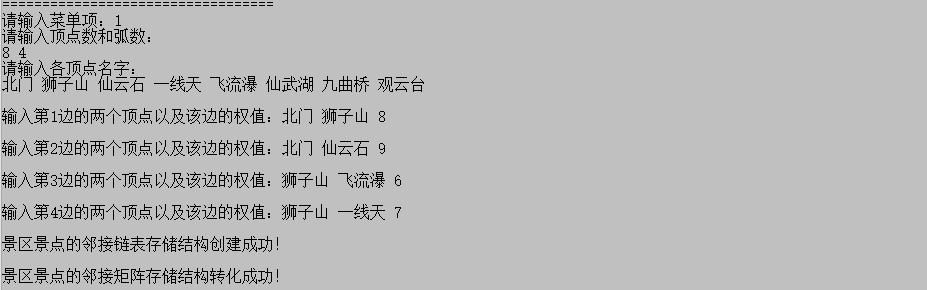
进行栈和队列的判断，是否为空和为满。压栈和出栈，进行栈顶和栈底的++ 和--操作。

第四章系统测试

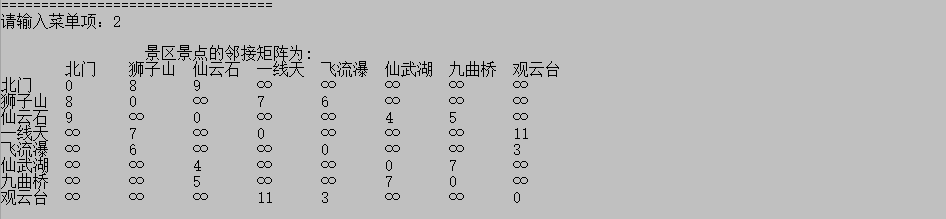
测试方法：输入不同数据判断系统是否可靠

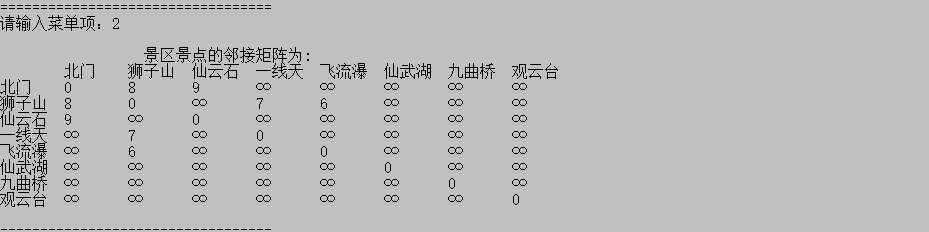
测试用例（应该给出几组具有不同特征的数据进行测试）：





测试结果：





出现的问题及解决办法：

在系统菜单中输入菜单中没有的选项会导致系统崩溃，后添加了

default:

printf("菜单项输入错误，请重新输入!\n");



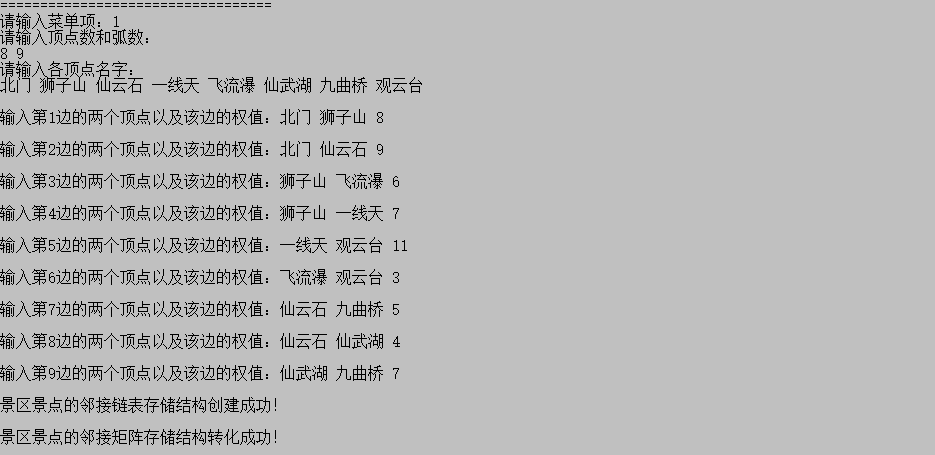
第五章结论

5.1程序的最终实现结果。效果图截图

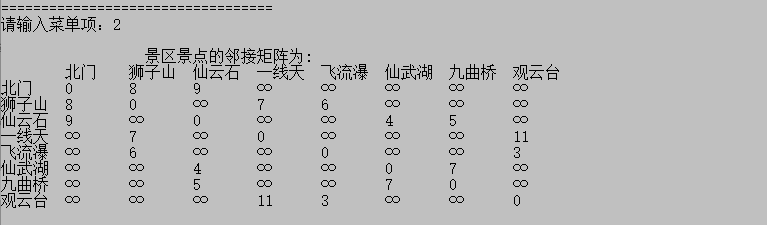
1. 程序主菜单



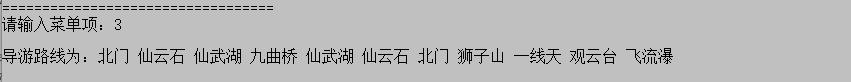
1. 创建景区景点分布图



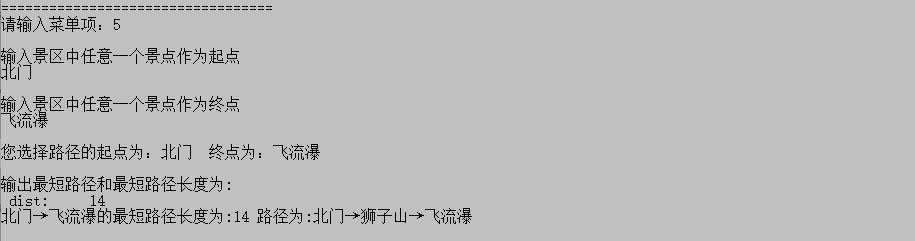
1. 输出景区景点分布图



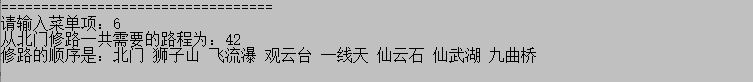
1. 输出导游路线图



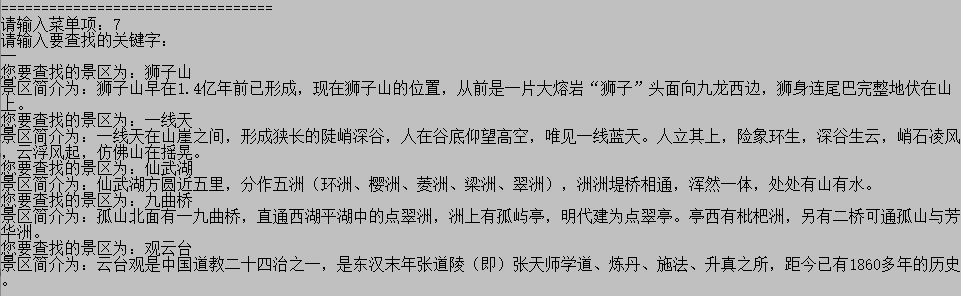
1. 求两个景点间的最短路径和最短距离



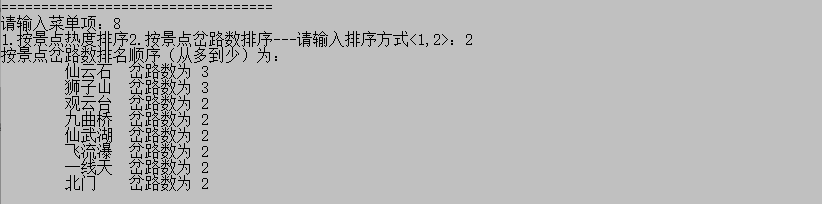
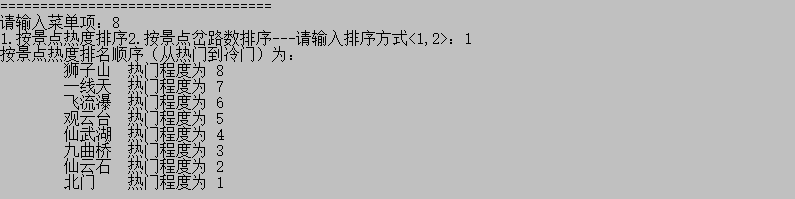
1. 输出道路修建规划图



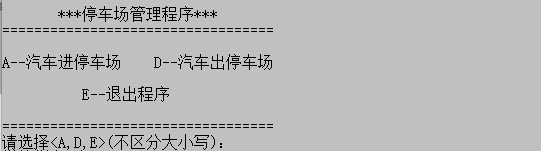
1. 按关键字查找景区信息



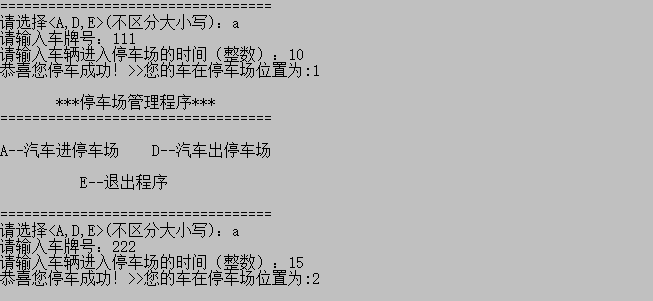
1. 对景区景点排序



1. 停车场系统
2. 主界面



1. 停车



1. 取车



5.2系统遗留问题及可能解决的途径

系统存在健壮性不足的问题，当有些输入出现错误的时候系统存在崩溃的可能。可以通过增加判断语句等来增加系统的健壮性。

参考文献

例：

[1] 数据结构、算法与应用：C++语言描述 [Data Structures,Algorithms,and Applications in C++][M].[机械工业出版社](http://book.jd.com/publish/%E6%9C%BA%E6%A2%B0%E5%B7%A5%E4%B8%9A%E5%87%BA%E7%89%88%E7%A4%BE_1.html)出版时间：2000-01-01.

[2] 数据结构(C语言版) [M].北京: 中国铁道出版社, 2011-08-01.

**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。课设过程未做与课设无关的事情。** | **10** |  |
| **报告质量** | **实验报告格式规范，体例符合要求；报告内容充实、正确，实验目的归纳合理到位，思考题回答准确。** | **40** |  |
| **创新性** | **在实践内容实现上，有自己独立的想法，并在功能实现上有一定的创新性。** | **10** |  |
| **实验内容** | **能够按实验要求合理设计并开发出程序，认真记录实验数据，原理及实验结果分析准确，归纳总结充分。程序代码书写规范，简洁。代码块区别显著，有重要位置的注释内容。** | **40** |  |
| **总 分** | | |  |