《数据结构课程设计》

课程名称：数据结构

题　　目：重庆师范大学校园导航系统

学　　院：计算机与信息科学学院

专业年级：计算机科学与技术（3+2）2 班

小组组长：杨澳

小组成员：陶劲松、张永海、秦艺铨、徐桥

指导教师：张万里　职称：讲师

2022 年 1 月 5 日

目录

重庆师范大学校园导航系统

一、 设计目的/需求分析

1、设计目的：

现在的大学占地面积越来越大，建筑物越来越多，功能越来越多样，校内的道路也是纵横交错，校园导航系统可以帮助用户更加快速的了解学校的路线，建筑布局及建筑物的基本信息等(用户主要是新生、家长、教职工、外来参观人员等)，在帮助用户了解校园路线、实现导航的功能的基础上，校园导航系统还录入了学校各个地点的相关信息，以供使用者更方便快捷的找到目的地。

随着科学技术的不断发展，计算机科学日渐成熟，其强大的功能已为人们所深刻认识，它已进入人类社会的各个领域并发挥着越来越重要的作用。采用计算机进行校园导航已成为衡量校园数字化的重要标志。校园导航效率的好坏对于来校参观的客人和学校管理者来说都至关重要，在很大程度上影响着校园的数字化建设和学校的影响力。因此，本次课程设计研究的校园导航系统具有一定的使用价值和现实意义。

2、需求分析：

该校园导航的使用者分为游客和管理者，功能如下：

（1） 游客：

① 针对游客的校园导航使用说明

② 查看校园全景图，附有校园地点的所有路线情况

③ 输入任意地点，查询该地点的信息，包括地点介绍和相关路线

④ 输入任意两地点，输出两点之间的所有简单路线

⑤ 输入任意两地点，输出两点之间的一条中转次数最少的最短路线

⑥ 输入任意两地点，输出两点之间的一条带权路径最短的最有路线

⑦ 输入任意地点，输出从该点出发的最佳布网方案

（2） 管理员：

① 使用管理员功能首先需要登录，登陆成功方可使用导航管理员功能

② 可以在地图中添加新地点

③ 可以在地图中添加新路线

④ 可以在地图中撤销旧路线

⑤ 注册新的管理员帐号

二、 问题描述

以我校为例，设计一个校园导航系统，主要为来访的客人提供信息查询。系统有两类登陆账号，一类是游客，使用该系统方便校内路线查询；一类是管理员，可以使用该系统查询校内路线，可对校园景点路线可编辑。

遇到的问题：

1. 读取文件之后显示乱码（解决方法：TXT 文档保存编码格式为 ANSI）

2. 最佳布网方案算法问题未完全解决

3. 其他模块算法的选择及构建问题

三、基本要求

1、导航系统的信息包括：地点，路线，最短行程查询，由近到远显示地点，从当前地点出发到其他所有地点的路线；管理员页面，可以增加删除地点的信息，以及地点间距离的信息。

2、系统能实现的操作功能如下：

（1）显示所有地点路线： 显示学校的各个地点，以及每个地点之间的距离信息

（2）最短距离查询： 输入你现在所在位置，并且输入你想要去的地方，程序会计算出两地之间的最短距离，并且给出线路图

（3）查询附近地点： 输入当前地点，可以显示出距离该地点由近及远的其他的地点信息

（4）查询道路上的地点： 输入当前道路，可以显示从这个点出发由近及远的所有地点的路径图。

四、概要设计

1、 算法思路

用 C++进行编写，使用了邻接矩阵和邻接表，弗洛伊德算法，深度优先和广度优先计算距离，还有包括整个框架的构思，图、栈、队列等等的运用，最终构成了整个程序。

2、 工作分配

（1）框架设计: 由杨澳进行构思，同时参考其他人的意见。

（2）系统设计: 由徐桥进行框架的编写。

（3）程序设计: 由张永海和杨澳进行各项功能的编辑。

（4）程序调试：由秦艺铨进行程序的调试与调整。

（5）文档制作：由陶劲松完成文档的编辑工作。

3、程序模块

1.导航使用说明：描述该导航应该如何使用，具有什么功能。

2.校园平面简图：输出一张有校园所有地点的平面图，可以直观的看出校园地点的分布。这是利用了每个地点所存储的坐标，通过对矩阵元素的遍历，输出了各地点的具体方位。并且在平面图下面以这样的形式附有校园两地点相连的所有路线信息：起点<--->终点。

3.查看地点信息：任意输入一个地点序号，输出该地点的介绍，以及所有与该地点连通的路线及其距离。

4.查询简单路径：任意输入两个地点，输出两地点间所有的简单路径。利用图的深度搜索遍历，用栈将经过的地点序号存起来，然后每条路线的地点逐个输出，最后得到所有简单路径。

5.查询最短路径：任意输入两个地点，输出两地点间一条中转次数最少的路线。利用图的广度搜索遍历，用队列将要遍历的地点存起来，通过对队列的操作得到中转次数最少的路线。

6.查询最优路径：任意输入两个地点，输出两地点间一条带权路径最短的路线。利用迪杰斯特拉算法，通过对 dist 和 path 的操作得出最终的最短路线。

7.最佳布网方案：任意输入一个地点，输出从该点出发的最佳布网方案。这是利用了最小生成树的思想，运用了 Prim 算法的思想。

8.添加新地点：输入新地点的名称和坐标，通过对文件增删改的操作，将新地点存储到文件里，就生成了一张新地图。

9.添加新路线：输入新路线的起点.终点 .距离，通过对文件增删改的操作，将新路线存储到文件里，就生成了一条新路线。

10.撤销旧路线：输入要撤销路线的起点.终点.距离，通过对文件增删改的操作，将改动后的路线信息存储到文件里。

11.管理员的登录：管理员可在此输入帐号和密码进行登录，在输入密码时也可以选择是否隐藏密码，如果输入帐号密码不对应，则可重新输入，若错误三次则自动退出系统。通过对用户输入的帐号密码和“admin.txt”文件里保存的帐号密码进行比较，如果相同则可登录。

12.管理员的注册：管理员可在此输入帐号和密码进行注册，密码需要输入两遍，相同则注册成功。通过对“admin.txt”文件增删改的操作，将新帐号和密码存储到文件里，就生成

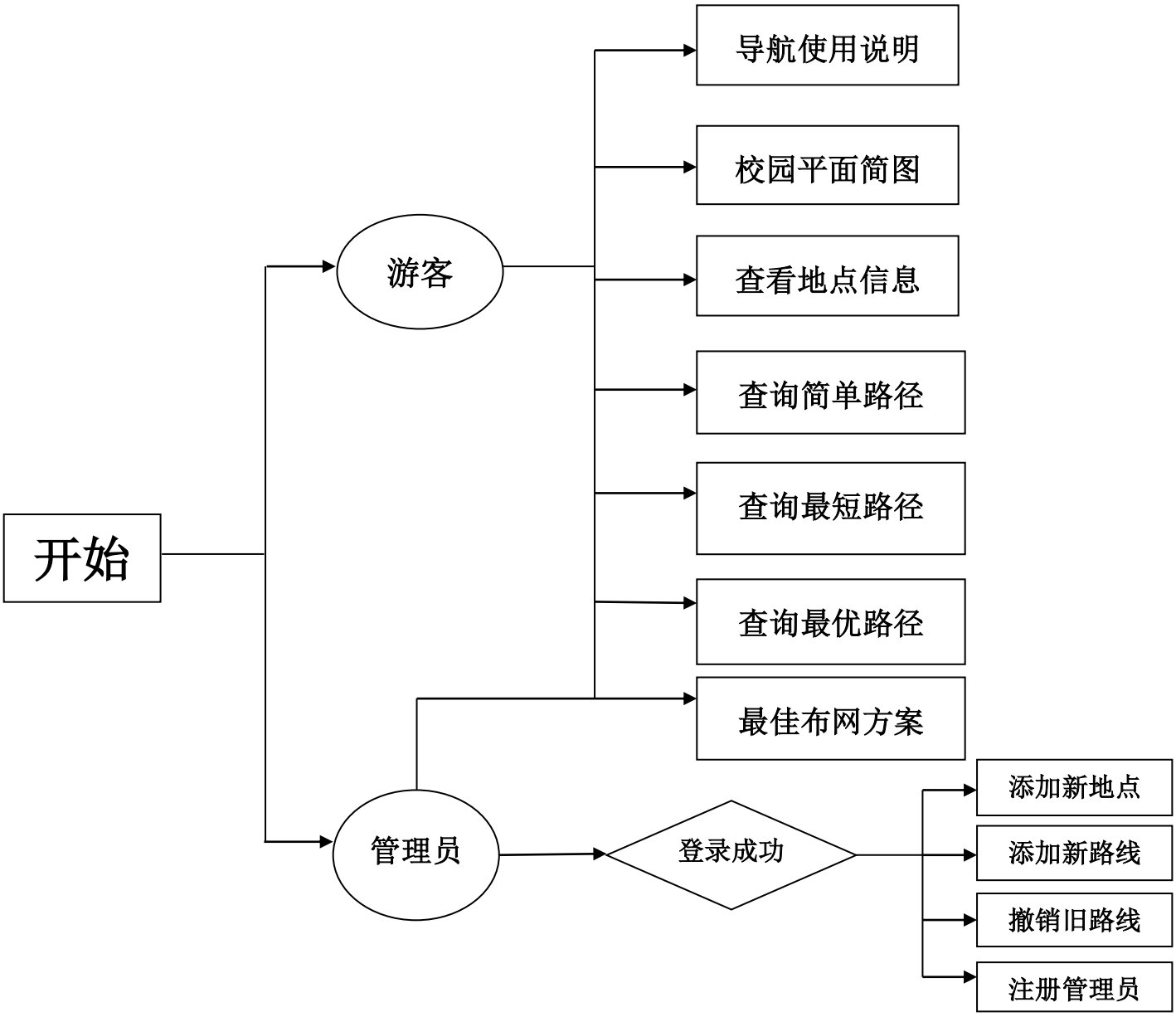
了一个新的管理员帐号。

13.队列的操作：有队列的判空.队列的初始化.出队.入队函数。

14.创建无向图：采用邻接矩阵的结构，通过对“路线信息”文件和“地点介绍”文件的读取，将信息存储到邻接矩阵中，然后创建出带权无向图。

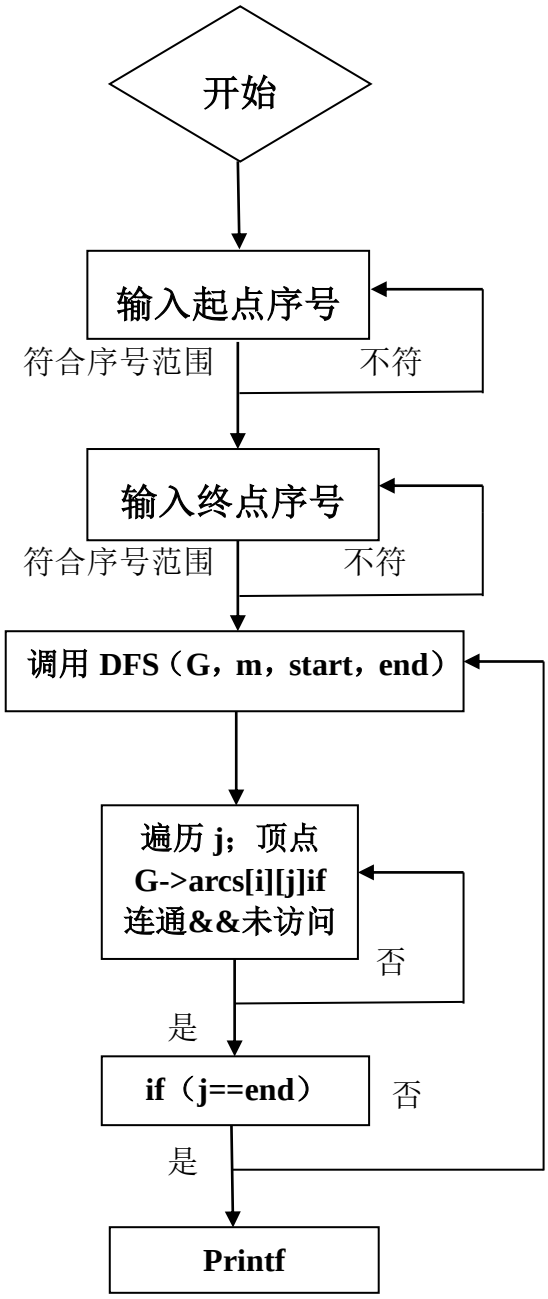
15.游客和管理员菜单：游客和管理员可在其对应的菜单页面进行功能选择。

4、功能模块图

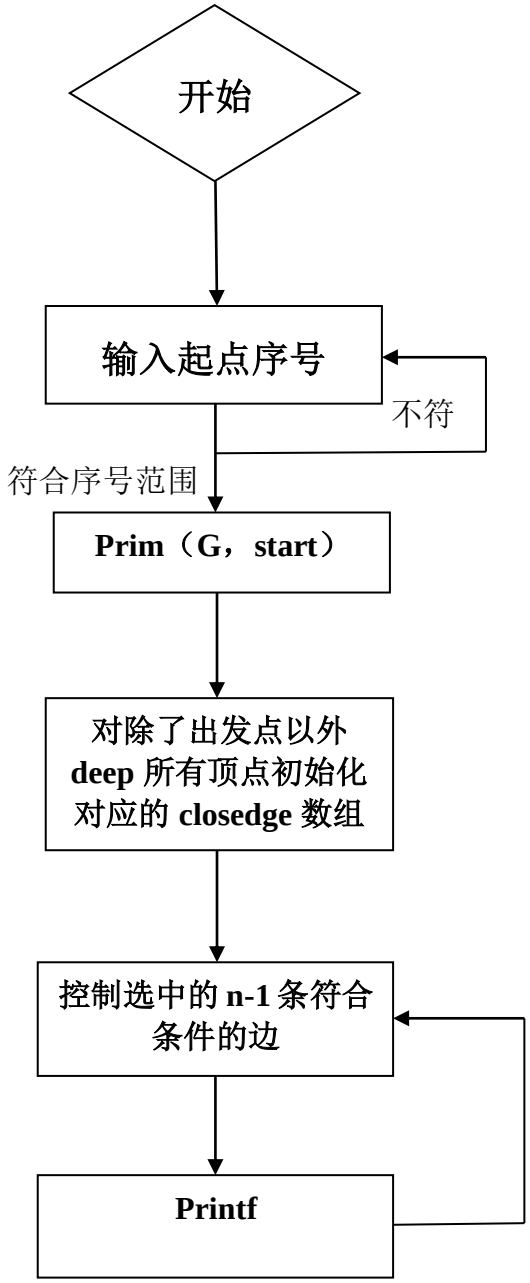

开始
游客
管理员
导航使用说明
校园平面简图
查看地点信息
查询简单路径
查询最短路径
查询最优路径
最佳布网方案
登录成功
添加新地点
添加新路线
撤销旧路线
注册管理员

5、总体函数调用关系图

（1） 所有简单路线函数：SearchDFS（）



（2） 最佳布网方案函数：MiniSpanTree（）



五、主程序

1. 数据结构类型的选择：邻接矩阵

校园导航.c

```
13  int visited[INFINITY];
14  int stack[INFINITY];
15  int count;
16
17  struct user //保存管理员帐号密码的结构体
18  {
19      char id[20];  //管理员帐号
20      char passwd[20];  //管理员密码
21      struct user *next;
22  };
23  typedef struct //保存地点信息的结构体
24  {
25      int No;     //校园地点序号
26      char name[20];     //校园地点名
27      char description[200];  //地点描述
28  }Vextype;
29  typedef struct //邻接矩阵
30  {
31      int arcs[MAXVEX][MAXVEX];    //边集
32      Vextype vex[MAXVEX];         //顶点集
33      int vexnum;                  //顶点数目
34      int arcnum;                  //边数目
35  }AdjMatrix;
36  typedef struct //坐标矩阵
37  {
38      int point; //该点是否有校园地点
39      char name[20]; //该点校园地点名
40      int No; //该点校园地点序号
41  }SchoolMap;
```

图 5.1　邻接矩阵

2. 创建带权无向图：

校园导航.c

```
166 }
167
168 //采用邻接矩阵创建无向图
169 int Create(AdjMatrix *G,SchoolMap M[MAXL][MAXC])
170 {
171     int i,j,weight,m,n;
172
173     FILE *fp1;
174     fp1=fopen("路线信息.txt","r");
175
176     //从"路线信息.txt"文件中读取校园图的景点数目和路线数目
177     fscanf(fp1, "%d %d", &G->vexnum, &G->arcnum);
178
179     //初始化邻接矩阵
180     for(i = 1; i <= G->vexnum; i++)
181         for(j = 1; j<= G->vexnum; j++) {
182             G->arcs[i][j] = INFINITY;
183     }
184     //读取"路线信息.txt"文件中两点序号及距离，并赋值给邻接矩阵
185     while(fscanf(fp1,"%d %d %d",&i,&j,&weight) != EOF) {
186             G->arcs[i][j] = weight;
187             G->arcs[j][i] = weight;
188     }
189     fclose(fp1);
190
191     FILE *fp2;
192     fp2 = fopen("地点介绍.txt", "rt");
193
194     //从"地点介绍.txt"文件中读取校园图中的景点名及描述
```

图 5.2　创建带权无向图

```c
校园导航.c
181         for(j = 1; j<= G->vexnum; j++) {
182             G->arcs[i][j] = INFINITY;
183     }
184     //读取"路线信息.txt"文件中两点序号及距离，并赋值给邻接矩阵
185     while(fscanf(fp1,"%d %d %d",&i,&j,&weight) != EOF) {
186             G->arcs[i][j] = weight;
187             G->arcs[j][i] = weight;
188     }
189     fclose(fp1);
190
191     FILE *fp2;
192     fp2 = fopen("地点介绍.txt", "rt");
193
194     //从"地点介绍.txt"文件中读取校园图中的景点名及描述
195     for(i = 1; i <= G->vexnum; i++) {
196         G->vex[i].No = i;
197         fscanf(fp2, "%s %d %d %s", G->vex[i].name,&m,&n,G->vex[i].description);
198         M[m][n].point = 1;
199         M[m][n].No = i;
200         strcpy(M[m][n].name,G->vex[i].name);
201     }
202     fclose(fp2);
203     return 1;
204 }
205
```

图 5.3　创建带权无向图

3. 查询所有简单路径：深度搜索遍历 DFS 算法：

```c
校园导航.c
292
293 void DFS(AdjMatrix *G, int m, int i, int end)
294 {
295     int j,k;
296     for(j = 1; j <= G->vexnum; j++){
297         if(G->arcs[i][j] != INFINITY && visited[j] == 0) {
298             visited[j] = 1;
299             if(j == end) {
300                 count++;
301                 printf("★ %d.",count);
302                 for(k = 1; k < m; k++) {
303                     printf("%s->", G->vex[stack[k]].name);
304                 }
305                 printf("%s\n", G->vex[end].name);
306                 visited[j] = 0;
307             }
308             else {
309                 stack[m] = j;
310                 m++;
311                 DFS(G, m, j, end);
312                 m--;
313                 visited[j] = 0;
314             }
315         }
316     }
317 }
```

图 5.4　深度搜索遍历 DFS 算法

4. 查询中转次数最少路径：广度搜索遍历 BFS 算法：

```c
校园导航.c
358 }
359
360 void BFS(AdjMatrix *G, int start, int end)
361 {
362     int vis[INFINITY];
363     int i, num;
364     int w, v;
365     LinkQueue *Q;
366
367     Q=(LinkQueue*)malloc(sizeof(LinkQueue));
368     if(start == end)
369         return;
370     memset(vis, 0, INFINITY);
371     vis[start] = 1;
372     InitQueue(Q);
373     EnterQueue(Q, start);
374     while(Q->front != Q->fear){
375         DeleteQueue(Q, &v);
376         num = v;
377         for(i = 1;i <= G->vexnum; i++){
378             if(G->arcs[num][i] != INFINITY) {
379                 w = i;  //求出当前节点的第一个邻接点（求出序号）
380                 while(w != -1){
381                     if(vis[w] == 0){
382                         if(w == end){
383                             BFS(G, start, num);
384                             printf("%s->",G->vex[num].name);
385                             return;
```

图 5.5　广度搜索遍历 BFS 算法

```c
    EnterQueue(Q, start);
    while(Q->front != Q->fear){
        DeleteQueue(Q, &v);
        num = v;
        for(i = 1;i <= G->vexnum; i++){
            if(G->arcs[num][i] != INFINITY) {
                w = i;  //求出当前节点的第一个邻接点（求出序号）
                while(w != -1){
                    if(vis[w] == 0){
                        if(w == end){
                            BFS(G, start, num);
                            printf("%s->",G->vex[num].name);
                            return;
                        }
                        vis[w] = 1;
                        EnterQueue(Q, w);
                        w = NextAdjVertex(G, w, v);
                        //w是求的得第一个邻接点，v是相对w下一个邻接点(求出下一个邻接点的序号)
                    }
                    break;
                }
            }
        }
    }
}
```

图 5.6　广度搜索遍历 BFS 算法

5.　查询带权路径最短：Dijkstra 算法：

```c

void Dijkstra(AdjMatrix *G, int start, int end, int dist[], int path[][MAXVEX])
{
    int mindist, i, j, k, t = 1;
    for(i = 1; i <= G->vexnum; i++) {
        dist[i] = G->arcs[start][i];  //对dist数组初始化
        if(G->arcs[start][i] != INFINITY)
            path[i][1] = start;  //如果该弧存在，则path[i][1]为源点
    }
    path[start][0] = 1; //start加入到S中
    for(i = 2; i <= G->vexnum; i++) {  //寻找各条最短路径
        mindist = INFINITY;
        for(j = 1; j <= G->vexnum; j++)
            if(!path[j][0] && dist[j] < mindist) {
                k = j;
                mindist = dist[j];
            }

            if(mindist == INFINITY)
                return ;
            path[k][0] = 1;   //找到最短路径，将该点加入到S集合中
            for(j = 1; j <= G->vexnum; j++) {  //修改路径
                if(!path[j][0] && G->arcs[k][j] < INFINITY && dist[k]+G->arcs[k][j] < dist[j]) {
                    dist[j] = dist[k] + G->arcs[k][j];
                    t = 1;
                    while(path[k][t] != 0) {
                        path[j][t] = path[k][t];
                        t++;
                    }
```

图 5.7　Dijkstra 算法

```
校园导航.c
445             if(mindist == INFINITY)
446                 return ;
447             path[k][0] = 1;   //找到最短路径，将该点加入到S集合中
448             for(j = 1; j <= G->vexnum; j++) {  //修改路径
449                 if(!path[j][0] && G->arcs[k][j] < INFINITY && dist[k]+G->arcs[k][j] < dist[j]) {
450                     dist[j] = dist[k] + G->arcs[k][j];
451                     t = 1;
452                     while(path[k][t] != 0) {
453                         path[j][t] = path[k][t];
454                         t++;
455                     }
456                     path[j][t] = k;
457                     path[j][t+1] = 0;
458                 }
459             }
460     }
461
462     for(i = 1; i <= G->vexnum; i++)
463         if(i == end)
464             break;
465     printf("\n    ★ ★ ★ %s--->%s的最短路线为: 从%s",G->vex[start].name,G->vex[end].name,G->vex[start].name);
466     for(j = 2; path[i][j] != 0; j++) {
467         printf("->%s",G->vex[path[i][j]].name);
468     }
469     printf("->%s, 距离为%d m\n",G->vex[end].name,dist[i]);
470     printf("\n\t\t\t\t\t按任意键返回...");
471     getch();
472 }
```

图 5.8　Dijkstra 算法

6.　最佳布网方案：最小生成树 Prim 算法：

```
校园导航.c
493
494 void Prim(AdjMatrix *G, int start)
495 {
496     struct
497     {
498         int adjvex;
499         int lowcost;
500     }closedge[MAXVEX];
501     int i, e, k, m, min;
502     closedge[start].lowcost = 0;
503
504     //对除了出发点以外deep所有顶点初始化对应的closedge数组
505     for(i = 1; i <= G->vexnum; i++) {
506         if(i != start) {
507             closedge[i].adjvex = start;
508             closedge[i].lowcost = G->arcs[start][i];
509         }
510     }
511
512     for(e = 1; e <= G->vexnum-1; e++)  //控制选中的n-1条符合条件的边
513     {
514         //选择权值最小的边
515         min = INFINITY;
516         for(k = 1; k <= G->vexnum; k++) {
517             if(closedge[k].lowcost != 0 && closedge[k].lowcost < min) {
518                 m = k;
519                 min = closedge[k].lowcost;
520             }
521         }
```

图 5.9　最小生成树 Prim 算法

```
校园导航.c
508             closedge[i].lowcost = G->arcs[start][i];
509         }
510     }
511
512     for(e = 1; e <= G->vexnum-1; e++)  //控制选中的n-1条符合条件的边
513     {
514         //选择权值最小的边
515         min = INFINITY;
516         for(k = 1; k <= G->vexnum; k++) {
517             if(closedge[k].lowcost != 0 && closedge[k].lowcost < min) {
518                 m = k;
519                 min = closedge[k].lowcost;
520             }
521         }
522         printf("\t\t\t\t\t从%s---%s:%d m\n", G->vex[closedge[m].adjvex].name,G->vex[m].name,closedge[m].lowcost);
523         closedge[m].lowcost = 0;
524
525         for(i = 1; i <= G->vexnum; i++) {
526             if(i != m && G->arcs[m][i] < closedge[i].lowcost) {
527                 closedge[i].lowcost = G->arcs[m][i];
528                 closedge[i].adjvex = m;
529             }
530         }
531     }
532     printf("\n\t\t\t\t\t按任意键返回...");
533     getch();
534 }
535
```

图 5.10　最小生成树 Prim 算法

六、运行截图

图 6.1　运行登录界面

图 6.2　导航游客使用说明

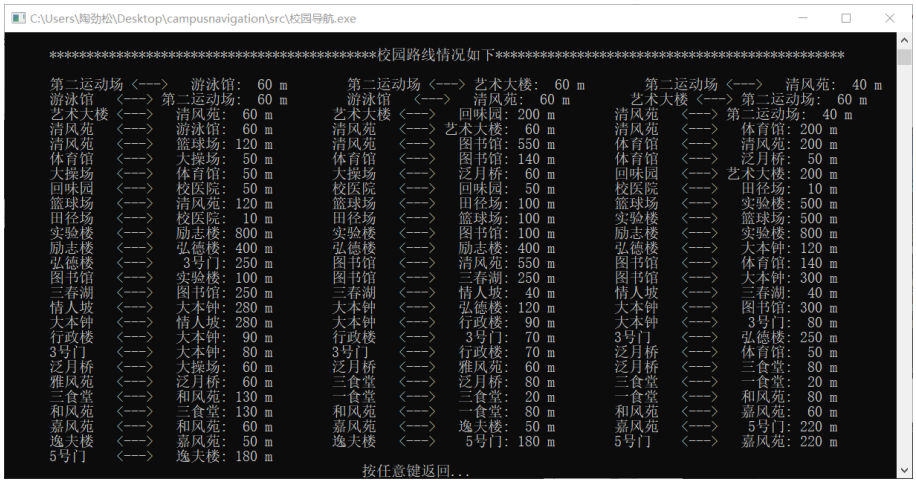


图 6.3　校园平面简图

图 6.4　进入查看地点信息界面

图 6.5　查看地点信息

图 6.6　进入查询简单路径界面

图 6.7　查询简单路径界面

图 6.8　进入中转次数最少的路径

图 6.9　查询中转次数最少的路径

图 6.10　查询带权路径最短

图 6.11　最佳布网方案

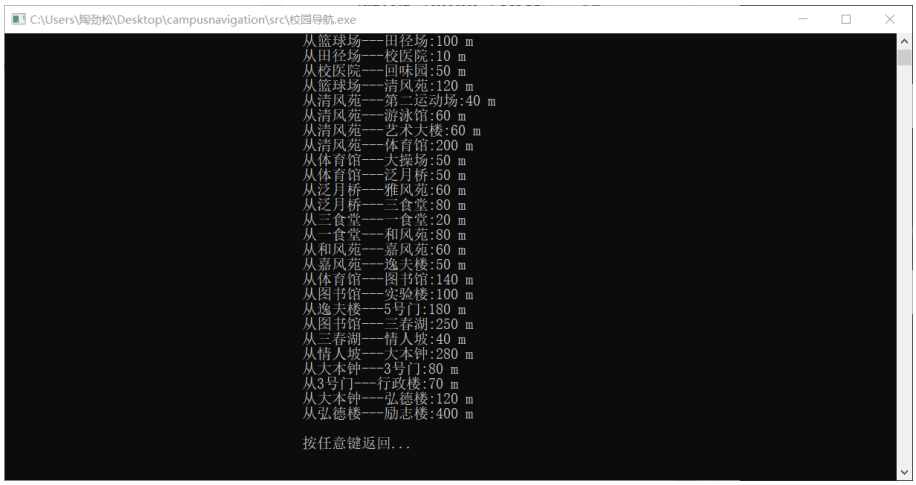


图 6.12　最佳布网方案

图 6.13　管理登录输入账户

图 6.14　输入密码

图 6.15　管理员界面

图 6.16　添加新地点

图 6.17　添加地点四号门

图 6.18　添加新路线

图 6.19　添加路线的数据

图 6.20　撤销新路线

图 6.21　管理员注册

七、总结

学完数据结构这本书后，我学会了多种存储数据方式，如使用线性表、栈和列队、串等等数据结构。在学习线性表的时候，刚开始学的顺序表和单链表时难度不是很大，但在学到循环链表和双向链表时就感觉有点难，算法上复杂了一点。经过反复的摸索和查询大体上学会了线性表的使用，但是算法依然只停留在理解上，要独立自主编写一个算法仍然有很大的难度。紧接着进入第三章的学习栈和队列，在对栈定义理解上没有什么难度，栈和线性表有类似的地方，但是栈有其自身的特点，在学习栈的时候难度在于栈的应用上，如在迷宫求解和栈与递归的实现上有一定的难度。在后来的队列学习上没有什么特别大的难度。接下来串与数组和广义表的学习与前两章的学习情况基本上相同，大多数定义都还能理解，但是在算法的学习上还是有一定的难度的。但在第六章的学习上就较为简单了，这一张结合图形的理解来学习，形象生动图形表示给学习这一章节带来了巨大的帮助，其次关于树的学习在上学期学的离散数学课本中也有所体现，并且有很多知识是通用的，所以也给这一章的学习提供

了辅助作用。这本数据结构给我的总体感受是过于理论化缺少实践操作，有提供具体的实验操作步骤，所以在实验时有一定的难度。学完这本书以后总体的感觉就是在算法的学习上有点难，但是大多数数据结构的定义还是比较容易理解的；总而言之，学完数据结构以后熟悉掌握了各种数据结构的定义，能看懂一些基本算法并知道它们的功能，但在一些新算法的编写上还是有一些困难，需要更多的上机实践来锻炼算法的编辑能力。