实 训 报 告

实训题目：地理信息管理系统

学　　院：计算机科学与信息学院

专　　业：计算机科学与技术

班　　级：计科 151

学　　号：1500170082

学生姓名：刘禾子

指导教师：刘长云

2017 年 7 月 8 日

一、实训目的及要求

数据结构是计算机课程的一门重要的基础课，它 的教学要求大致有三个重要方面：其一就是让学生学会分析研究计算机加工的数据对象的特性，以便为数据选择适当的物理结构和逻辑结构；其二，根据结构，选择 适当的算法，并初步掌握算法的时间分析和空间分析；其三，学习复杂的程序设计。本综合实训利用 Visual Studio 集成编程环境为实践工具，通过上机实践培养学生分析具体问题、解决实际问题的能力，训练和培养学生的数据抽象能力和程序设计的能力。

数据结构是一门实践性较强的课程，以养学生的数据抽象能力和程序设计的能力为目的。在实训时应注重培养学生的实际操作能力。本综合实训安排了 16 学时的实验课时，选做一个综合性较高的设计题目。具体要求如下：

1. 学习和理解该实训题目的基本理论和方法；
2. 掌握每个实验的实现步骤和关键技术；
3. 准备好实验所需要的资源和文档；
4. 上机实现程序，得到通过调试的正确程序。
5. 根据每个实验的不同要求，完成实验报告的 word 文档。

二、实训环境

Windows 10

Visual Studio 2013

三、实训内容

1. 问题描述

地理信息系统（GIS）在军事、城市管理、交通、林业、水利等众多行业有着重要的应 用。一个地理信息系统需要管理空间信息和属性信息。空间信息是地图中点、线、面的位置 和拓扑关系，如建筑坐标、道路路径。属性信息是点、线、面的其它属性，如城市名称、道 路拥堵程度、林地植被类型。 你的任务是实现一个简易的 GIS 原型系统，能够存储若干个点的坐标和名称，以及它们 之间的道路联通情况。

2. 设计要求

（1） 能够存储和修改若干个地点的经纬度、高程和名称。

int AddSpot(string name,float longitude,float latitude, altitude)

void UpdateSpot(int spotId,float longitude,float latitude, altitude)

void UpdateSpotName(int spotId,string newName)

void DeleteSpot(int spotId) void FindSpot(string name)

（2） 能够存储和修改两个地点的道路联通情况，交通耗时。

int AddRoad(string name1,string name2,float cost)

void UpdateRoad(string name1,string name2,float cost)

void DeleteRoad(int edgeId)

（3） 能够查询任意两个点的最短路径。GetShortestPath(string name1,string name2)

（4） 要通过异常抛出和捕获机制来处理异常情况，如删除的点编号不存在。

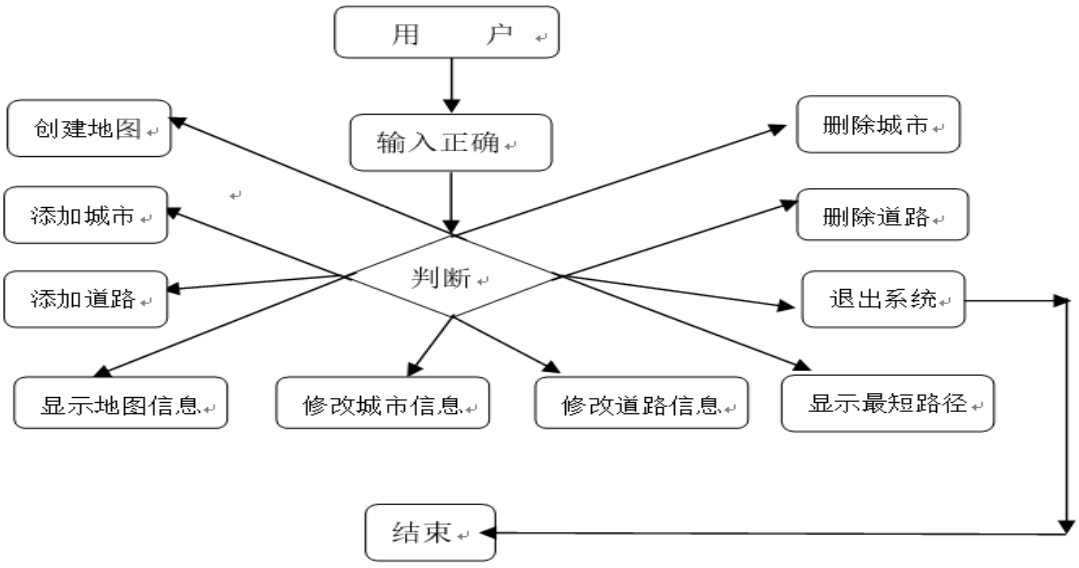
（5） 实现文件存储和载入。（可选内容）

四、算法描述及实训步骤

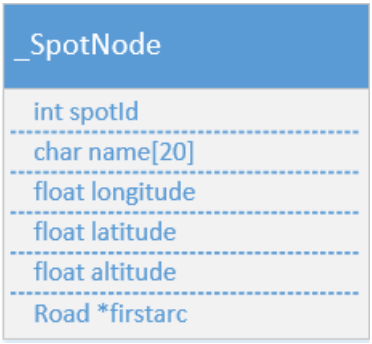
本项目主要是对图的操作，核心算法是迪杰斯特拉最短路径算法，图以邻接链表形式存储。

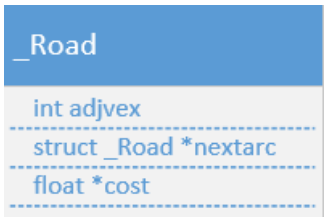
主要功能有：创建地图、添加城市、添加道路、修改城市信息、修改道路信息、删除城市、删除道路、显示地图信息、查看最短路径。

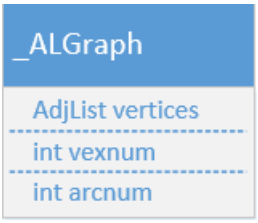
附加功能：实现文件读取地图信息。



基于定义的三个结构体进行操作：城市顶点，道路（边），图结构：



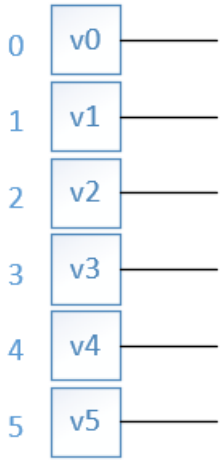


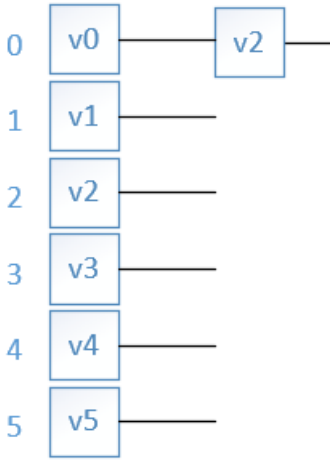


城市结点　　道路结点　　地图结构

创建图方法描述：

根据用户所输入的点的数量构建一个_SpotNode 类的数组并把 firstarc 都置为 NULL 得到：





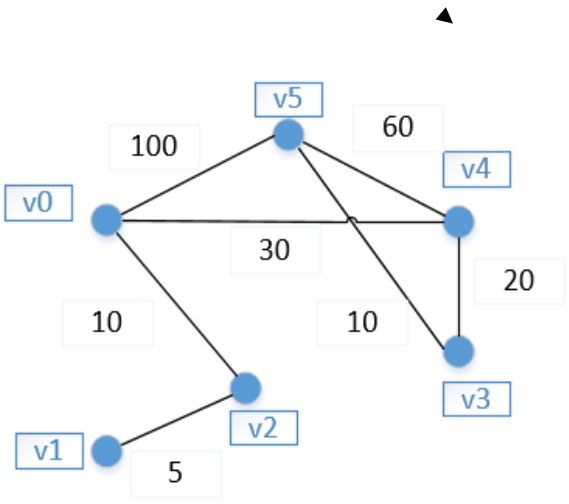
再根据用户所输入边的个数对各点进行操作，因为是无向图，所以在插入过程中分别插入<v,w>和<w,v>，每次都插入在表头，例如插入 v0-v2 这条边，先生成如上图 2 的一个结点链让其相邻链指向空，再将 v0 第一条链接入到新生成的结点链上得到图 3，依次类推可得到完整的图结构。

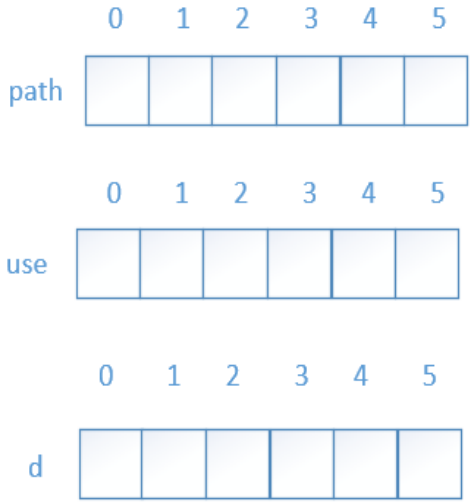
迪杰斯特拉算法描述：

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。

算法过程：





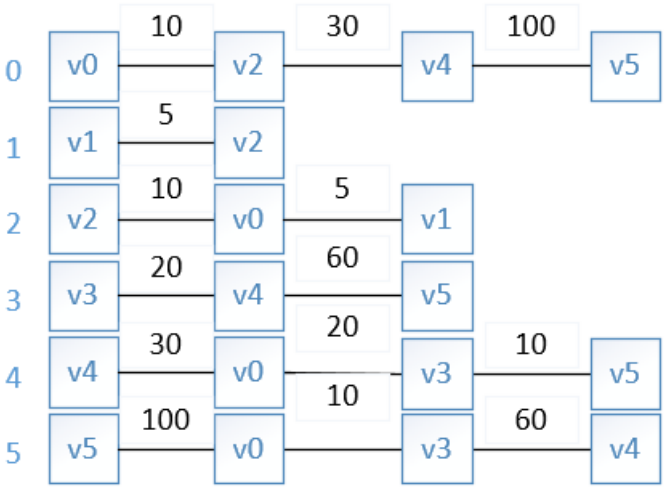
定义三个数组：

int d[MAX_VERTEX_NUM];//记录 v 到各顶点的最小路径

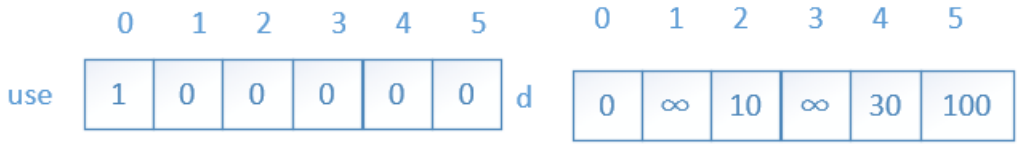
int use[MAX_VERTEX_NUM];//记录各节点是否已求得最短路径，0 表示未求得，1 表示已求得

int path[MAX_VERTEX_NUM];//记录 v0 到各个顶点最小路径的中间节点

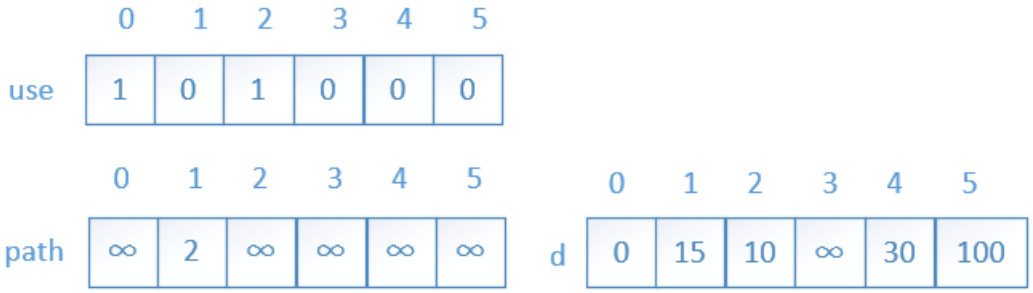
图的存储结构：



假设要求得 v0-v5 的最短路径，首先求得 v0 可直接到达的顶点的各个耗时并存进 d[ ]数组，并把 v0 点标记为已访问。

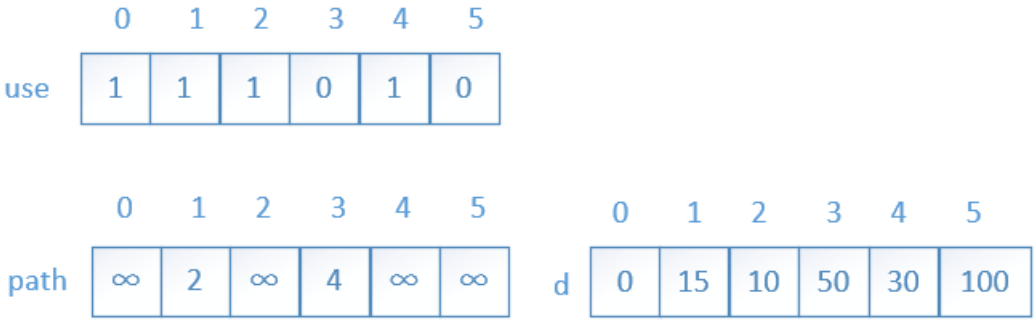


再在 d 数组中寻找到最小值的单元 2 10 标记为已访问用 k 记录下标值 2，并以 v2 为顶点查找相邻且并没有被访问的结点，判断 d[2]+(v2-v1)是否小于 d[0]，小于将 10+5 的结果存入 d[0]，并将 k 存入 path[1]中，此时各数组情况如图：



继续取得 d[ ]数组中最小值而且没有被访问过的点，取到 1 号单元 15，此时 k=1,并标记 use[1]=1,以 v0 为顶点查找相邻且未被访问的点，没有找到，则此时只改变了 use[ ]，如下图：

继续取得 d[ ]数组中最小值而且没有被访问过的点，取到 4 号单元 30，此时 k=4,并标记 use[4]=1,以 v4 为顶点查找相邻且未被访问的点，找到 v4-v3=20,判断 20+d[4]=50 是否小于 d[3],小于记 d[3]=50,path[3]=k=4,此时各数组情况如图：



继续以 v4 为顶点查找相邻且未被访问的点，找到 v4-v5=60，此时 k=4，判断 60+d[4]=90 是否小于 d[5]=100，小于记 d[5]=90，path[5]=k=4，此时 use[ ]不变，另外两个数组如下图：

	0	1	2	3	4	5
path	∞	2	∞	4	∞	4

	0	1	2	3	4	5
d	0	15	10	50	30	90

继续取得 d[ ]数组中最小值而且没有被访问过的点，取到 3 号单元 50，此时 k=3，并标记 use[3]=1，以 v3 为顶点查找相邻且未被访问的点，找到 v3-v5=10，判断 10+d[3]=60 是否小于 d[5]=90，小于记 d[5]=60，path[5]=k=3，此时各数组情况如图：

	0	1	2	3	4	5
use	1	1	1	1	1	0

	0	1	2	3	4	5
path	∞	2	∞	4	∞	3

	0	1	2	3	4	5
d	0	15	10	50	30	60

继续取 d[]数组中最小值且没有被访问过的点，取到 5 号单元 60，此时 k=5，并标记 use[5]=1，以 v3 为顶点查找相邻且未被访问的点，与 v5 相邻的点均被访问过，且所有顶点均被访问过，不作任何操作，程序运行结束，得到各数组情况：

	0	1	2	3	4	5
use	1	1	1	1	1	1

	0	1	2	3	4	5
path	∞	2	∞	4	∞	3

	0	1	2	3	4	5
d	0	15	10	50	30	60

得 v0-v5 最短耗时为 d[5]=60，路径为依次取出 path[5]的内容直到取到∞，即 path[5]=3，得到 v3->v5，再以 3 为下标取到 path[3]=4，即得到 v4->v3->v5，再以 4 为下标取到 path[4]=∞，说明已经到达起点，即得到最短路径 v0-> v4->v3->v5。

五、总结及心得体会

通过此次实训我进一步加强了自己的编程能力，在细节上还有待提高，特别是在编码过程中重现了很多可解决的错误，如数组指针越界，没有很好地掌握数组和指针

的运用方法，又如未知参数没有运用好，重复多次使用循环变量，没有赋好初值，出现一些细小的错误，再如对图的邻接链表的操作不够熟练，但经过反复自己实操才更深入地理解了链表的使用以及它的方便之处，也不会造成大程度的空间浪费，在操作过程中因为输出最短路径是倒过来的一条路径，通过引入另一个数组变量实现它的反转使得最终输出的是一条正序的路径，从而增强用户体验。再而就是通过实现读取地图 txt 文件又复习了一遍 C 语言对文件的读写操作，大大减少了重复输入的麻烦。

总结来说，此次实训我受益颇深，通过查阅资料，从简单的结构体定义到迪杰斯特拉的整个算法核心部分，我都有了一定的了解，并能够区分弗洛伊德算法和迪杰斯特拉算法的区别以及用邻接矩阵存储和邻接表存储图的优缺点。

六、实训结果

```
E:\Software\Visual Studio 2010\项目组\地理信息管理系统\Debug\地理信息管理系统.exe
                欢迎登陆地理信息管理系统

            1. 创建地图          6. 显示地图信息
            2. 添加城市          7. 修改城市信息
            3. 添加道路          8. 修改道路信息
            4. 删除城市          9. 查看最短路径
            5. 删除道路          0. 退出系统
           10. 用文件读取地图
=======================================================
1
请输入地图的城市数，道路数（以空格为间隔）：
6 7
请输入6个城市的编号、名字(20个字符以内）、经度、纬度以及高程：
0 上海 121 31 16
1 北京 116 39 52
2 广东 113 23 47
3 贵阳 106 26 1277
4 福州 119 26 10
5 大庆 125 46 149
请按顺序输入每条道路的耗时以及所连接的城市编号（以空格作为间隔）：
10 0 2
30 0 4
100 0 5
5 1 2
20 3 4
10 3 5
60 4 5
-------------------------------------------------------
```

```
6个城市：
编号   名称   经度   纬度   高程          编号   名称   经度   纬度   高程

0    上海    121    31    16              1    北京    116    39    52
2    广东    113    23    47              3    贵阳    106    26    1277
4    福州    119    26    10              5    大庆    125    46    149
7条道路：
城市    城市    耗时          城市    城市    耗时

上海---大庆:  100            上海---福州:  30
上海---广东:  10             北京---广东:  5
贵阳---大庆:  10             贵阳---福州:  20
福州---大庆:  60
```

查看最短路径：

```
==========================================================
9
请输入要查询最短路径的两个城市编号（以空格为间隔）：
0 5
上海-- 大庆的最短路径是：
上海-->福州-->贵阳-->大庆    耗时：60
----------------------------------------------------------
```

map.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
7 7
0 上海 121 31 16
1 北京 116 39 52
2 广东 113 23 47
3 贵阳 106 26 1277
4 福州 119 26 10
5 大庆 125 46 149
10 习水 103 33 800
10 0 2
30 0 4
100 0 5
5 1 2
20 3 4
10 3 5
60 4 5
```

用文件读取地图信息：

E:\Software\Visual Studio 2010\项目组\地理信息管理系统\Debug\地理信

```
==========================================================
                欢迎登陆地理信息管理系统

            1. 创建地图          6. 显示地图信息
            2. 添加城市          7. 修改城市信息
            3. 添加道路          8. 修改道路信息
            4. 删除城市          9. 查看最短路径
            5. 删除道路          0. 退出系统
           10. 用文件读取地图
==========================================================
10
```

```
7个城市：
编号   名称   经度   纬度   高程          编号   名称   经度   纬度   高程

0    上海    121    31    16             1    北京    116    39    52
2    广东    113    23    47             3    贵阳    106    26    1277
4    福州    119    26    10             5    大庆    125    46    149
10    习水    103    33    800
7条道路：
城市    城市    耗时          城市    城市    耗时

上海---大庆：  100           上海---福州：  30
上海---广东：  10            北京---广东：  5
贵阳---大庆：  10            贵阳---福州：  20
福州---大庆：  60
----------------------------------------------------------
```

七、源代码：

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<iostream>
#define MAX_VERTEX_NUM 10  //最多所拥有的点数
#define MAX_NAME 20            //名称字符的最大值
const int inf = 0x3f3f3f3f;//无限大的值，表示两路不可通
using namespace std;
//采用邻接表存储图结构
typedef struct _Road{
    int adjvex;//该弧指向的顶点的位置
    struct _Road *nextarc;//指向下一条弧的指针
    float *cost;//路程耗时
}Road;
typedef struct _SpotNode{
    int spotId;//点编号
    char name[20];//名称
    float longitude;//经度
    float latitude;//纬度
    float altitude;//高程
    Road *firstarc;//指向第一条依附该顶点的弧的指针
}SpotNode, AdjList[MAX_VERTEX_NUM];
typedef struct _ALGraph
{
    AdjList vertices; //邻接链表
    int vexnum, arcnum;//图当前的顶点数和弧数
}ALGraph;
//点操作函数声明
void AddSpot(ALGraph &G);
void UpdateSpot(ALGraph &G);
void DeleteSpot(ALGraph &G);
int FindSpot(ALGraph G,int u);//u为输入的所需要定位的城市编号
//边操作函数声明
void AddRoad(ALGraph &G);
void UpdateRoad(ALGraph &G);
void DeleteRoad(ALGraph &G);
//迪杰斯特拉算法函数声明
void GetShortestPath(ALGraph &G);
//文件读取地图信息函数声明
void LoadMapByFile(ALGraph &G);
/**********函数具体实现方法**********/
//返回G中spotId为u的点的位置
```

```
int FindSpot(ALGraph G, int u){
    int i;
    for (i = 0; i < G.vexnum; i++)
    {
        if (u == G.vertices[i].spotId)
            return i;
    }
    return -1;//找不到返回-1
}
//用文件加载地图信息
void LoadMapByFile(ALGraph &G){
    int i, j;
    Road *p;
    float w;//权值
    int va, vb;//用于输入边所连接的两个点在邻接表中的存储位置
    FILE *fp;
    fp = fopen("map.txt", "r");
    while (!feof(fp))
    {
        fscanf(fp, "%d %d", &G.vexnum, &G.arcnum);
        for (int i = 0; i < G.vexnum; i++)
        {
            fscanf(fp, "%d %s %f %f %f\n", &G.vertices[i].spotId, &G.vertices[i].name,
                &G.vertices[i].longitude, &G.vertices[i].latitude,&G.vertices[i].altitude);
            G.vertices[i].firstarc = NULL;
        }
        for (int k = 0; k < G.arcnum; k++)
        {
            fscanf(fp, "%f %d %d\n", &w, &va, &vb);
            i = FindSpot(G, va);//弧头
            j = FindSpot(G, vb);//弧尾
            //连接<i,j>
            p = (Road *)malloc(sizeof(Road));
            p->adjvex = j;
            p->cost = (float *)malloc(sizeof(float));
            *(p->cost) = w;
            p->nextarc = G.vertices[i].firstarc;//插在表头
            G.vertices[i].firstarc = p;
            //连接<j,i>
            p = (Road*)malloc(sizeof(Road));
            p->adjvex = i;
            p->cost = (float *)malloc(sizeof(float));
            *(p->cost) = w;
            p->nextarc = G.vertices[j].firstarc;//插在表头
```

```
            G.vertices[j].firstarc = p;
        }
    }
    fclose(fp);
}
//创建图
void CreateGraph(ALGraph &G){
    int i, j, k;
    float w;//权值
    int va,vb;//用于输入边所连接的两个点在邻接表中的存储位置
    Road *p;
    cout << "请输入地图的城市数，道路数（以空格为间隔）：" << endl;
    cin >> G.vexnum >> G.arcnum;
    cout << "请输入" << G.vexnum << "个城市的编号"<<"、名字("<<MAX_NAME<<"个字符以内）、经度、
纬度以及高程：" << endl;
    for (i = 0; i < G.vexnum; i++)//构造顶点
    {
        cin >> G.vertices[i].spotId >>
G.vertices[i].name>>G.vertices[i].longitude>>G.vertices[i].latitude>>G.vertices[i].altitude;
        G.vertices[i].firstarc = NULL;//第一条链指向空
    }
    cout << "请按顺序输入每条道路的耗时以及所连接的城市编号（以空格作为间隔）:" << endl;
    for (k = 0; k < G.arcnum; k++)//构造表结点链表
    {
        cin >> w >> va >> vb;
        i = FindSpot(G, va);//弧头
        j = FindSpot(G, vb);//弧尾
        //连接<i,j>
        p = (Road *)malloc(sizeof(Road));
        p->adjvex = j;
        p->cost = (float *)malloc(sizeof(float));
        *(p->cost) = w;
        p->nextarc = G.vertices[i].firstarc;//插在表头
        G.vertices[i].firstarc = p;
        //连接<j,i>
        p = (Road*)malloc(sizeof(Road));
        p->adjvex = i;
        p->cost = (float *)malloc(sizeof(float));
        *(p->cost) = w;
        p->nextarc = G.vertices[j].firstarc;//插在表头
        G.vertices[j].firstarc = p;
    }
}
//修改点信息
```

```
void UpdateSpot(ALGraph &G){
    int spotId;
    cout << "请输入你要修改的城市的编号：" << endl;
    cin >> spotId;
    int i;
    i = FindSpot(G, spotId);
    if (i>-1){
        cout << "请输入新的名称，经度，纬度，高程（以空格间隔）：" << endl;
        cin >> G.vertices[i].name >> G.vertices[i].longitude >> G.vertices[i].latitude >>
G.vertices[i].altitude;
    }
}
//插入点v
void AddSpot(ALGraph &G){
    int id;
    cout << "请输入要添加城市的编号：" << endl;
    cin >> id;
    for ( int i = 0; i < G.vexnum; i++)
    {
        if (id == G.vertices[i].spotId){
            cout << "城市编号重复，请重新输入！" << endl;
            return;
        }
    }
    cout << "请输入要添加城市的名称，经度，纬度，高程（以空格间隔）：" << endl;
    cin >> G.vertices[G.vexnum].name>> G.vertices[G.vexnum].longitude
        >> G.vertices[G.vexnum].latitude>> G.vertices[G.vexnum].altitude;
    G.vertices[G.vexnum].spotId = id;
    G.vertices[G.vexnum].firstarc = NULL;
    G.vexnum++;//图的顶点数加1
}
//删除点
void DeleteSpot(ALGraph &G){
    int spotId;
    cout << "请输入要删除城市的编号：" << endl;
    cin >> spotId;
    int i, j;
    Road *p, *q;
    j = FindSpot(G, spotId);//j是定点v的序号
    if (j < 0){
        cout << "未找到该城市，删除失败！" << endl;
        return;//v不是图G的顶点
    }
    p = G.vertices[j].firstarc;//删除以v为出度的弧或边
```

```
    while (p)
    {
        q = p;
        p = p->nextarc;
        free(q->cost);
        free(q);
        G.arcnum--;//弧或边数减一
    }
    G.vexnum--;//顶点数减一
    for (i = j; i < G.vexnum; i++)
        G.vertices[i] = G.vertices[i + 1];//顶点v后面的顶点前移
    for (i = 0; i < G.vexnum; i++)//删除以v为入度的弧或边且必要时修改表结点的顶点位置
    {
        p = G.vertices[i].firstarc;//指向第1条弧或边
        while (p)
        {
            if (p->adjvex == j){
                if (p == G.vertices[i].firstarc)//待删除结点是第1个结点
                {
                    G.vertices[i].firstarc = p->nextarc;
                    free(p->cost);
                    free(p);
                    p = G.vertices[i].firstarc;
                }
                else
                {
                    q->nextarc = p->nextarc;
                    free(p->cost);
                    free(p);
                    p = p->nextarc;
                }
            }
            else
            {
                if (p->adjvex > j)
                    p->adjvex--;//修改表结点的顶点位置值
                q = p;
                p = p->nextarc;
            }
        }
    }
}
//添加道路
void AddRoad(ALGraph &G){
```

```
    int v;
    int w;
    cout << "请问你想在哪两个城市搭建道路（输入城市编号，以空格为间隔）" << endl;
    cin >> v >> w;
    Road *p;
    int i, j;
    float w1;
    i = FindSpot(G, v);
    j = FindSpot(G, w);
    if (i < 0 || j < 0){
        cout << "未找到相关城市，添加道路失败！" << endl;
        return;
    }
    G.arcnum++;
    cout << "请输入道路 " << G.vertices[i].name << "--" << G.vertices[j].name << "的耗时：" <<
endl;
    cin >> w1;
    p = (Road *)malloc(sizeof(Road));
    p->adjvex = j;
    p->cost = (float *)malloc(sizeof(float));
    *(p->cost) = w1;
    p->nextarc = G.vertices[i].firstarc;//插在表头
    G.vertices[i].firstarc = p;
    //无向，生产另一个表结点
    p = (Road *)malloc(sizeof(Road));
    p->adjvex = i;
    p->cost = (float *)malloc(sizeof(float));
    *(p->cost) = w1;
    p->nextarc = G.vertices[j].firstarc;//插在表头
    G.vertices[j].firstarc = p;
}
//删除道路
void DeleteRoad(ALGraph &G){
    int v;
    int w;
    cout << "请输入你要删除的道路所连接的两个城市编号（以空格间隔）：" << endl;
    cin >> v >> w;
    Road *p, *q;
    int i, j;
    i = FindSpot(G, v);//i是定点的序号
    j = FindSpot(G, w);//j是定点的序号
    if (i < 0 || j < 0 || i == j){
        cout << "未找到相关道路信息，删除失败！" << endl;
        return;
```

```
    }
    p = G.vertices[i].firstarc;//p指向顶点v的第一条弧尾
    while (p&&p->adjvex != j)//p不空且所指的弧不是待删除<v,w>
    {
        q = p;
        p = p->nextarc;//p指向下一条弧
    }
    if (p&&p->adjvex == j)//找到
    {
        if (p == G.vertices[i].firstarc)//p是指第1条弧
            G.vertices[i].firstarc = p->nextarc;
        else
            q->nextarc = p->nextarc;//指向下一条弧
        free(p->cost);
        free(p);//释放此结点
        G.arcnum--;//弧或边数减1
    }
    //无向，删除对称弧<w,v>
    p = G.vertices[j].firstarc;
    while (p&&p->adjvex != i)
    {
        q = p;
        p = p->nextarc;
    }
    if (p&&p->adjvex == i)
    {
        if (p == G.vertices[j].firstarc)
            G.vertices[j].firstarc = p->nextarc;
        else
            q->nextarc = p->nextarc;
        free(p->cost);
        free(p);
    }
}
//更新道路耗时
void UpdateRoad(ALGraph &G){
    int v,w;//用于输入两个城市的编号
    float w1;
    cout << "请输入你要修改的道路所连接的两个城市编号（以空格间隔）：" << endl;
    cin >> v >> w;
    Road *p, *q;
    int i, j;
    i = FindSpot(G, v);//i是定点的在邻接表内的存储位置
    j = FindSpot(G, w);//j是定点的在邻接表内的存储位置
```

```
    if (i < 0 || j < 0 || i == j){
        cout << "未找到相关道路信息，修改失败！" << endl;
        return;
    }
    cout << "请输入道路" << G.vertices[i].name << "--" << G.vertices[j].name << "新的耗时：" <<
endl;
    cin >> w1;
    p = G.vertices[i].firstarc;//p指向顶点v的第一条弧尾
    while (p&&p->adjvex != j)//p不空且所指的弧不是待修改<v,w>
    {
        p = p->nextarc;//p指向下一条弧
    }
    if (p&&p->adjvex == j){
        *(p->cost) = w1;
    }//找到
    //无向图还要更新另一对称边的耗时
    p = G.vertices[j].firstarc;
    while (p&&p->adjvex != i)
    {
        p = p->nextarc;
    }
    if (p&&p->adjvex == j){
        *(p->cost) = w1;
    }//找到
}
//显示图
void Display(ALGraph G){
    int i;
    int count = 0;//用于换行
    Road *p = NULL;
    cout << G.vexnum << "个城市：" << endl;
    cout << "编号" << "  " << "名称" << "  " << "经度" << "  " << "纬度" << "  " << "高程        "
         << "编号" << "  " << "名称" << "  " << "经度" << "  " << "纬度" << "  " << "高程        "
<< endl;;
    for (i = 0; i < G.vexnum; i++)
    {
        if (i%2==0)
        {
            cout << endl;
        }
        cout << G.vertices[i].spotId << "    " << G.vertices[i].name <<"    "
             << G.vertices[i].longitude <<"   "<< G.vertices[i].latitude<<"    "
             << G.vertices[i].altitude<<"              ";
    }
```

```
    cout << endl;
    cout << G.arcnum << "条道路：" << endl;
    cout << "城市" << "    " << "城市" << "    " << "耗时" << "           "
         << "城市" << "    " << "城市" << "    " << "耗时" << "           " << endl;
    for (i = 0; i < G.vexnum; i++)
    {
        p = G.vertices[i].firstarc;
        while (p)
        {
            if (i < p->adjvex){//使得输出的结果不重复例如输出v0-v3
                if (count%2==0)//就不会再输出v3-v0
                {
                    cout << endl;
                }
                count++;
                cout << G.vertices[i].name << "---" << G.vertices[p->adjvex].name;
                cout << ":" <<"  "<<*(p->cost)<<"             ";
            }
            p = p->nextarc;
        }
    }
    cout << endl;
}
//得到任意两点的最短路径
void GetShortestPath(ALGraph &G){
    int d[MAX_VERTEX_NUM];//记录v到各顶点的最小路径
    int use[MAX_VERTEX_NUM];//记录各节点是否已求得最短路径，0表示未求得，1表示已求得
    int path[MAX_VERTEX_NUM];//记录v0到各个顶点最小路径的中间节点
    int i,j;//循环变量
    int source, sink;
    cout << "请输入要查询最短路径的两个城市编号（以空格为间隔）：" << endl;
    cin >> source >> sink;
    int a, b;
    a = FindSpot(G, source);//a是起点的序号
    b = FindSpot(G, sink);//b是终点的序号
    if (a < 0 || b < 0 || a == b)
        return;
    Road *p,*q;
    p = G.vertices[source].firstarc;
    for (i = 0; i < MAX_VERTEX_NUM; i++)
    {
        d[i] = inf;
    }
    for (i = 0; i < MAX_VERTEX_NUM; i++)
```

```
{
    use[i] = 0;
}
for (p; p != NULL; p = p->nextarc)
{
    for (i = 0; i < G.vexnum; i++)
    {
        if (p->adjvex == i)
        {
            d[i] = *(p->cost);
        }
    }
}
use[source] = 1;
for ( i = 1; i < G.vexnum; i++)
{
    int k, min = inf;//k为中间点，min为最小路径
    for ( j = 0; j < G.vexnum; j++)
    {
        if (!use[j]&&d[j]<min)
        {
            min = d[j];
            k = j;
        }
    }
    use[k] = 1;
    q = G.vertices[k].firstarc;
    while (q)
    {
        for (int w = q->adjvex; w != -1 && q != NULL;)
        {
            w = q->adjvex;
            if (!use[w] && min + *(q->cost) < d[w]){
                d[w] = min + *(q->cost);
                path[w] = k;
            }
            q = q->nextarc;
        }
    }
}
cout << G.vertices[source].name << "-- " << G.vertices[sink].name << "的最短路径是： " << endl;
cout << G.vertices[source].name << "-->";
int location = sink;
int path_out[MAX_VERTEX_NUM];//用于正序输出最短路径
```

```
    int l = G.vexnum-1;
    while(path[sink]>0)
    {
        path_out[l--] = path[sink];
        sink = path[sink];
    }
    for ( i = 0; i < G.vexnum; i++)
    {
        if (path_out[i]>0)
        {
            cout << G.vertices[path_out[i]].name << "-->";
        }
    }
    cout << G.vertices[location].name << "    " << "耗时：" << d[location] << endl;
}
//显示主菜单
void Menu(){
    cout << "=================================================" << endl;
    cout << "              欢迎登陆地理信息管理系统             " << endl;
    cout << "                                                 " << endl;
    cout << "          1.创建地图        6.显示地图信息       " << endl;
    cout << "          2.添加城市        7.修改城市信息       " << endl;
    cout << "          3.添加道路        8.修改道路信息       " << endl;
    cout << "          4.删除城市        9.查看最短路径       " << endl;
    cout << "          5.删除道路        0.退出系统           " << endl;
    cout << "         10.用文件读取地图                       " << endl;
    cout << "=================================================" << endl;
}
//主函数
int main(void){
    int i;
    ALGraph G;
    while (1)
    {
        Menu();
        cin >> i;
        switch (i)
        {
        case 0:return 0;//退出系统
        case 1:CreateGraph(G);
            break;
        case 2:AddSpot(G);
            break;
        case 3:AddRoad(G);
```

```
                break;
            case 4:DeleteSpot(G);
                break;
            case 5:DeleteRoad(G);
                break;
            case 6:Display(G);
                break;
            case 7:UpdateSpot(G);
                break;
            case 8:UpdateRoad(G);
                break;
            case 9:GetShortestPath(G);
                break;
            case 10:LoadMapByFile(G);
                break;
            default:
                break;
            }
        }
        return 0;
}
```