学生学号	0121610870126	实验课成绩	

武漢理工大學

学 生 实 验 报 告 书

实验课程名称	数据结构与算法综合实验
开课学院	计算机科学与技术学院
指导教师姓名	李晓红
学生姓名	严伟滔
学生专业班级	软件工程 1604

2017 -- 2018 学年 第 2 学期

实验课程名称：数据结构与算法综合实验

实验项目名称	图与景区信息管理系统			报告成绩	
实验者	严伟滔	专业班级	软件 1602	组别	
同组者				完成日期	2018 年 5 月 20 日

第一部分：实验分析与设计（可加页）

一、 实验目的和内容

1.实验目的

图结构的学习包括概念、定义、操作和编程。

通过“景区信息管理系统”的编程实践，学习图的遍历、Dijkstra 算法、最小生成树算法、Prim 算法和他们的编程应用。

开发和学习“景区信息管理系统”，达到如下目标：

(1)掌握图的定义和图的存储结构

(2)掌握图的创建方法

(3)掌握图的两种遍历方法

(4)理解迪杰斯特拉（Dijkstra）算法

(5)理解最小生成树的概念和普里姆（Prim）算法

(6)掌握文件操作

(7)使用 C++语言，利用图的数据结构，开发景区信息管理系统。

2.实验内容

1）业务背景

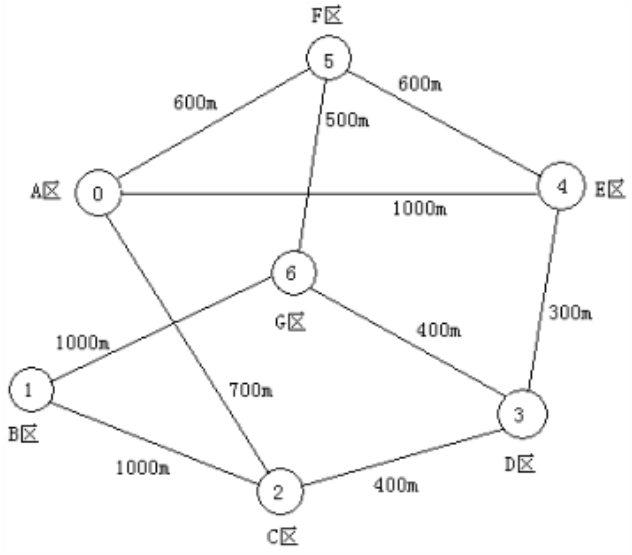
现有一个景区，景区里面有若干个景点，景点之间满足以下条件：

(1)某些景点之间铺设了道路（相邻）

(2)这些道路都可以双向行驶的（无向图）

(3)从任意一个景点出发都可以游览整个景区（连通图）

开发景区信息管理系统，对景区的信息进行管理。使用图的数据结构来保存景区景点信息，为用户提供创建图、查询景点信息、旅游景点导航、搜索最短路径、铺设电路规划等功能。



2)景点数据

景区的数据包含景点信息和景点之间的道路信息。分别由两个文本文件存储。Vex.txt 文件用来存储景点信息；Edge.txt 文件用来存储道路信息。

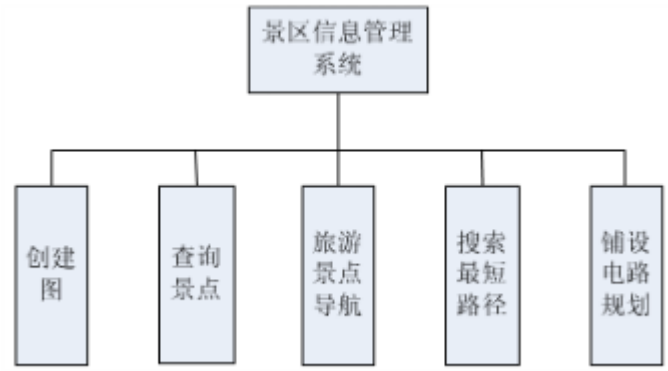
1.景点信息：景点编号、名字和介绍

编号	*名字*	*介绍*
0	*A 区*	…
1	*B 区*	…
2	*C 区*	…
3	*D 区*	…
4	*E 区*	…
5	*F 区*	
6	*G 区*	…

2.道路信息：景点 1，景点 2，两个景点之间的距离

景点 1	*景点 2*	*距离（m）*
A	*C*	*700*
A	*E*	*1000*
A	*F*	*600*
B	*C*	*1000*
B	*G*	*1000*
C	*D*	*400*
D	*E*	*300*
D	*G*	*400*
E	*F*	*600*
F	*G*	*5 0*

3）系统功能



1.创建图

输入：从 Vex.txt 文件中读取景点信息，从 Edge.txt 文件中读取道路信息。

处理：根据读取的景点信息创建景区景点图。

输出：

创建成功，一次输出：

（1）顶点数目

（2）顶点编号

（3）顶点名字

（4）边两端的顶点编号

（5）边的权值

创建失败，输出失败的提示信息。

2.查询景点

输入：想要查询的景点的编号

处理：根据输入的景点编号，查询该景点及相邻景点的信息。

输出：

（1）景点名字

（2）景点介绍

（3）相邻景点的名字

（4）到达相邻景区的路径长度

3.旅游景点导航

输入：起始景点的编号

处理：使用深度优先搜索（DFS）算法,查询以该景点为起点，无回路游览整个景区的路线。

输出：所有符合要求的导航路线。

4.搜索最短路径

输入：

（1）起始景点的编号

（2）终点的编号

处理：使用迪杰斯特拉算法，求得从起始景点到终点之间的最短路径，计算路径总长度。

输出：

（1）最短路线

（2）路径总长度

5.铺设电路规划

输入：景区的所有景点信息和道路信息

处理：根据景区景点图使用普里姆（Prim）算法构造最小生成树，设计出一套铺设线路最短，但能满足每个景点都能通电的方案。

输出：

（1）需要铺设电路的道路

（2）每条道路铺设电路的长度

（3）铺设电路的总长度

二、 分析与设计

1.数据结构的设计

(使用 C++STL 中的 string 代替 char 数组存储字符串)

顶点结构体:

```
struct Vex
{
    int num;//编号
    std::string name;//名称
    std::string desc;//介绍描述
};
```

边结构体:

```
struct Edge
{
    int vex1;// 顶点 1
    int vex2;// 顶点 2
    int weight;//权重(边的长度)
};
```

图类:

```
class CGraph
{
private:
    int m_aAdjMatrix[20][20];//邻接矩阵
    Vex m_aVexs[20];//顶点数组
    int m_nVexNum;//顶点数目
public:
    void init();
    void dfs(int n,int vis[],int path[]);//DFS 算法遍历所有节点
    int findSSSP(int i, int j,std::vector<int>& vis);//查找两点之间的最短路径
    bool InsertVex(Vex sVex);//插入顶点
    int FindEdge(int v, std::vector<Edge> &aEdge);//查找顶点 v 周围有几条边
```

```
        bool InsertEdge(Edge sEdge);//插入边
        void printVex();
        void printEdge();
        Vex GetVex(int v);
        int FindEdge(int v, Edge aEdge[]);
        int getVexNum();
        int DesignPath();
    };
```

2.核心算法设计

DFS 算法遍历图:

①.从图中某个顶点 v0 出发，首先访问 v0;

②.访问结点 v0 的第一个邻接点，以这个邻接点 vt 作为一个新节点，访问 vt 所有邻接点。直到以 vt 出发的所有节点都被访问到，回溯到 v0 的下一个未被访问过的邻接点，以这个邻结点为新节点，重复上述步骤。直到图中所有与 v0 相通的所有节点都被访问到。

③.若此时图中仍有未被访问的结点，则另选图中的一个未被访问的顶点作为起始点。重复深度优先搜索过程，直到图中的所有节点均被访问过。

Dijkstra 算法:

①.先取一点 v[0]作为起始点，初始化 dis[i],d[i]的值为 v[0]到其余点 v[i]的距离 w[0][i]，如果直接相邻初始化为权值，否则初始化为无限大；

②.将 v[0]标记，vis[0] = 1(vis 一开始初始化为 0)；

③.找寻与 v[0]相邻的最近点 v[k]，将 v[k]点记录下来，v[k]与 v[0]的距离记为 min；

④.把 v[k]标记，vis[k]=1；

⑤.查询并比较，让 dis[j]与 min+w[k][j]进行比较，判断是直接 v[0]连接 v[j]短，还是经过 v[k]连接 v[j]更短，即 dis[j]=MIN(dis[j],min+w[k][j])；

⑥.继续重复步骤③与步骤⑤，知道找出所有点为止。

最小生成树:

在连通网的所有生成树中，所有边的代价和最小的生成树，称为最小生成树。

kruskal 算法的基本思想：

1.首先将 G 的 n 个顶点看成 n 个孤立的连通分支（n 个孤立点）并将所有的边按权从小大排序。

2.按照边权值递增顺序，如果加入边后存在圈则这条边不加，直到形成连通图

三、主要仪器设备及耗材

1. 安装了 Windows XP 或 Windows 7 或其它版本的 Windows 操作系统的 PC 机 1 台
2. PC 机系统上安装了 Microsoft Visual Studio 开发环境

第二部分：实验过程和结果（可加页）

一、 实现说明

DFS 算法:

```
/*n 为本次访问的顶点,数组 vis 为所有顶点的访问情况,path 为构建路径
基于递归实现*/
void CGraph::dfs(int n, int vis[], int path[])
{
    if (n == m_nVexNum)
    {
        cout << m_aVexs[path[0]].name;
        for (int i = 1; i < m_nVexNum;  ++i)
        {
            cout << "->" << m_aVexs[path[i]].name;
        }
        cout << endl;
        return;
    }
    else
    {
        for (int i = 0; i < m_nVexNum;  ++i)
        {
            if (vis[i] == 0&&m_aAdjMatrix[path[n-1]][i]!=0)//首次访问且连通
            {
                vis[i] = 1;
                path[n]  = i;
                dfs(n + 1, vis, path);
                vis[i] = 0;
            }
        }
    }
}
```

Dijkstra 算法:

```
/*数组 d 为存储各个节点到 i 的最短距离
 *i,n 分别为起始节点和节点数量
 *fa 是存储第 i 条边的前一条边用于输出
 *g[i][j]表示节点 i 的第 j 条边
```

```
 */
void dijkstra(int d[],int i, int n,Vex v[], Edge fa[], vector<vector<Edge>> g)
{
    priority_queue<HeapNode> Q;
    for (int k = 0; k < n; ++k) d[k] = INF;
    d[i] = 0;
    int *done = new int[n + 1];
    memset(done, 0, sizeof(int)*(n+1));
    HeapNode temp;
    temp.d = 0;
    temp.u = i;
    Q.push(temp);
    while (!Q.empty())
    {
        HeapNode x = Q.top(); Q.pop();
        int u = x.u;
        if (done[u])continue;
        done[u] = 1;
        for (int k = 0; k < g[u].size(); ++k)
        {
            Edge &e = g[u][k];
            if (d[e.vex2] > d[u] + e.weight)
            {
                d[e.vex2] = d[u] + e.weight;
                fa[e.vex2] = g[u][k];
                HeapNode t;
                t.u = e.vex2;
                t.d = d[e.vex2];
                Q.push(t);
            }
        }
    }
    delete[] done;
}
```

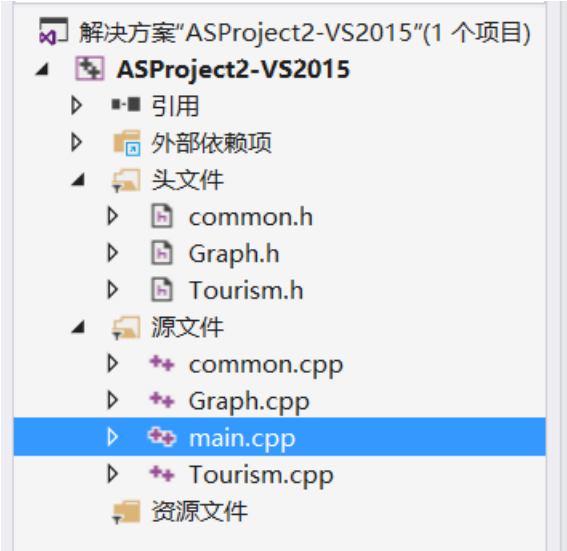
最小生成树算法:

```
/*运用 Kruskal 算法,将图裁边后变为树,使得这棵树所有*/
int CGraph::DesignPath()
{
    int m_num = this->m_nVexNum;
    int root[30];
    initTree(root, m_num);
    //start at Vex zero
    set<int>Vn;
    int ret = 0;
```

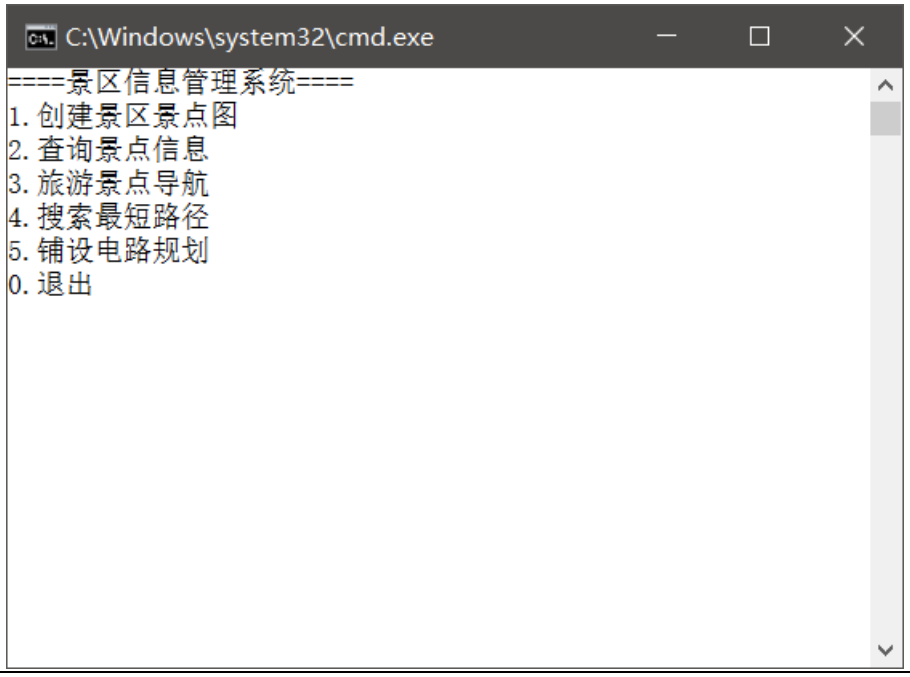
```
        Edge temp[30];
        int size= 0;
        Edge e;
        int min = INF;
        //挑选第一条边
        size=this->FindEdge(0, temp);
        for (int i = 0; i < size; ++i)
        {
            if (temp[i].weight < min)
            {
                e = temp[i];
                min = e.weight;
            }
        }
        Vn.insert(e.vex1);
        Vn.insert(e.vex2);
        //merge
        mergeNode(root,e.vex1, e.vex2);
        ret += e.weight;
        //start
        cout << "铺设计划如下:" << endl;
        cout << this->m_aVexs[e.vex1].name << " - " << this->m_aVexs[e.vex2].name
    << "      " << e.weight <<"m"<< endl;
        set<int>::iterator iter;
        for (int i = 0; i < m_num - 2; ++i)
        {
            min = INF;
            for (iter = Vn.begin(); iter != Vn.end(); ++iter)
            {
                int vex1 = *iter;
                int fv1 = find(root, vex1);
                size = this->FindEdge(*iter, temp);
                for (int j = 0; j < size; ++j)
                {
                    int vex2 = temp[j].vex2;
                    int weight = temp[j].weight;
                    int fv2 = find(root, vex2);
                    if (fv1 != fv2 && weight < min)
                    {
                        min = weight;
                        e = temp[j];
                    }
                }
            }
            //merge
```

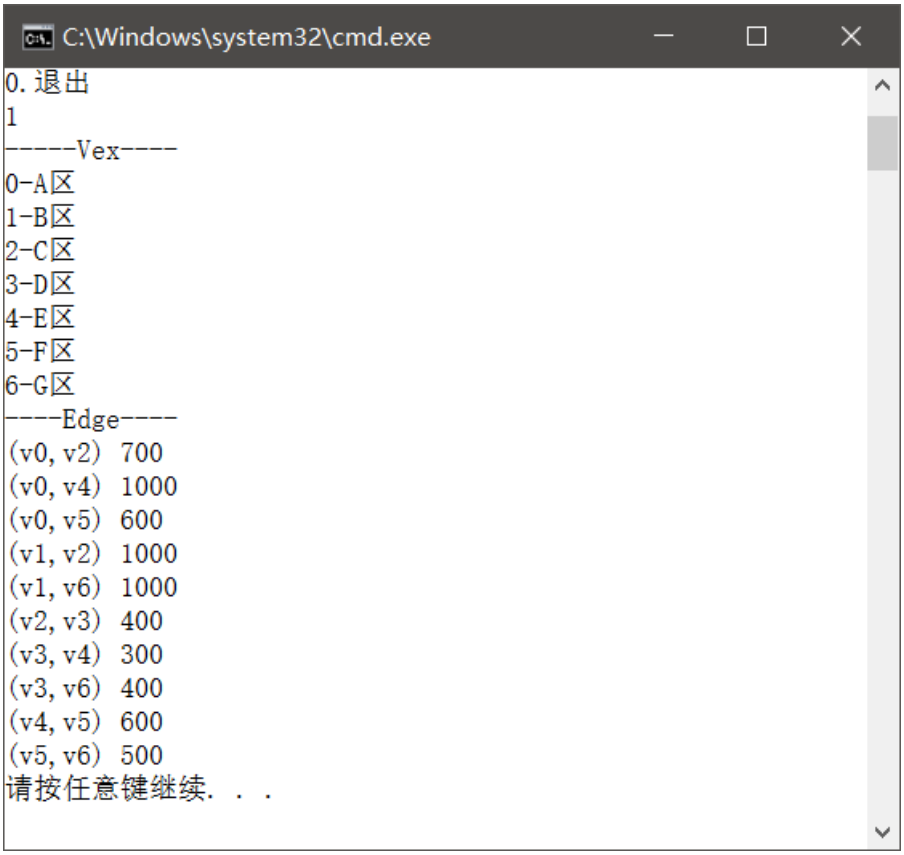
```
            mergeNode(root, e.vex1, e.vex2);
            //cout
            cout << this->m_aVexs[e.vex1].name  << " - "
    << this->m_aVexs[e.vex2].name  << "   " << e.weight <<"m"<< endl;
            Vn.insert(e.vex2);
            ret += e.weight;
        }
        cout << "铺设电路的最短长度为:" << ret << endl;
        return 0;
    }
```

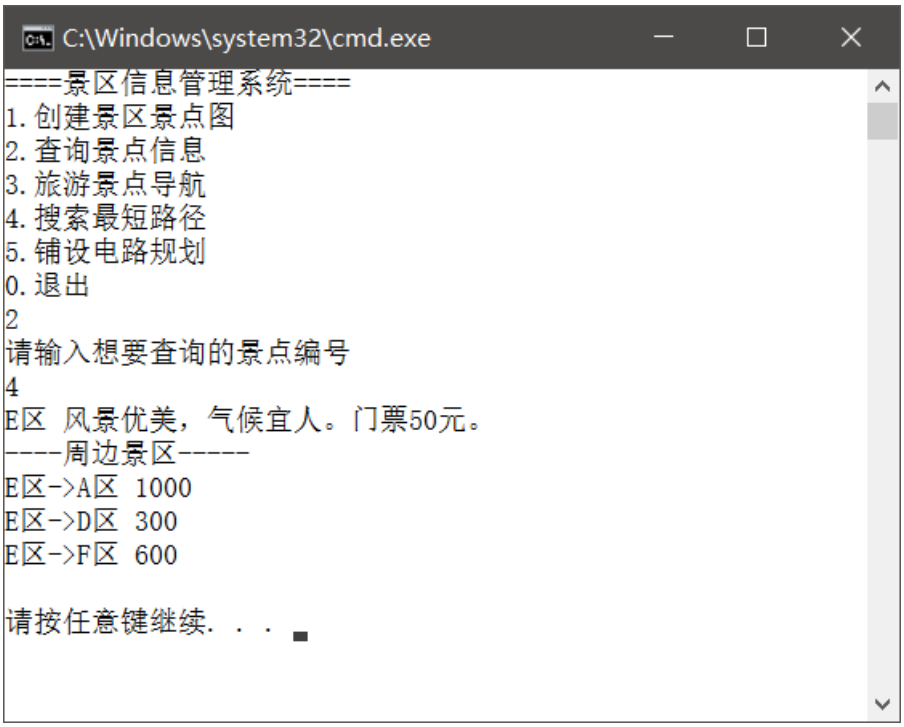
二、 源代码

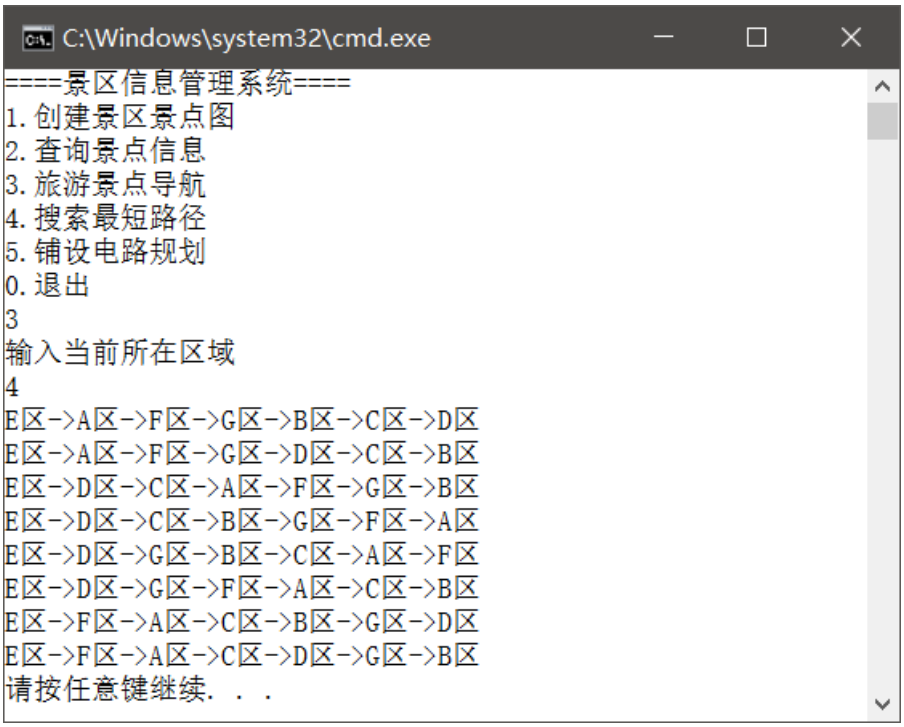


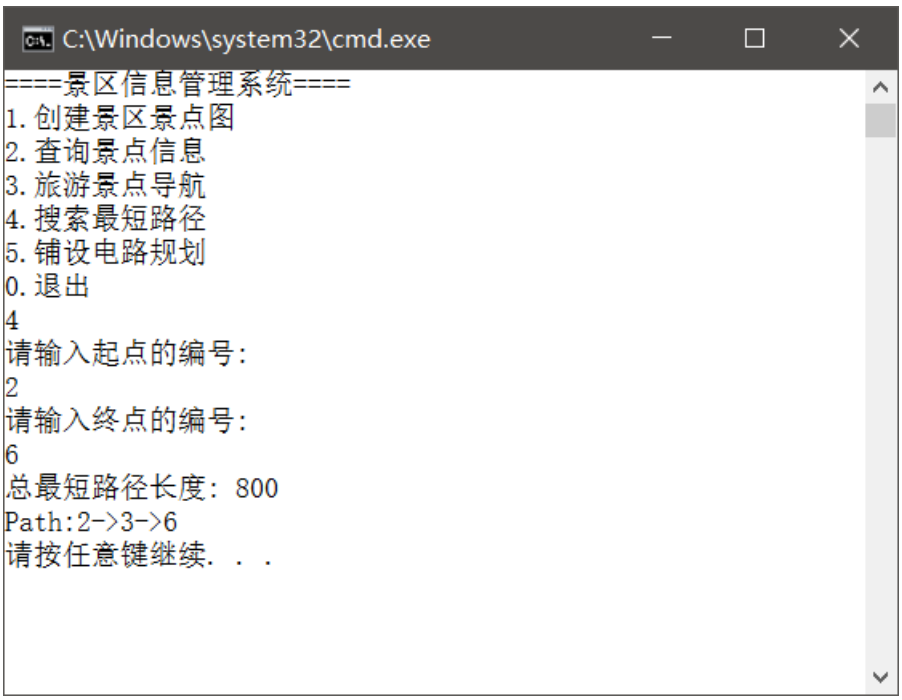
三、 调试说明（调试手段、过程及结果分析）

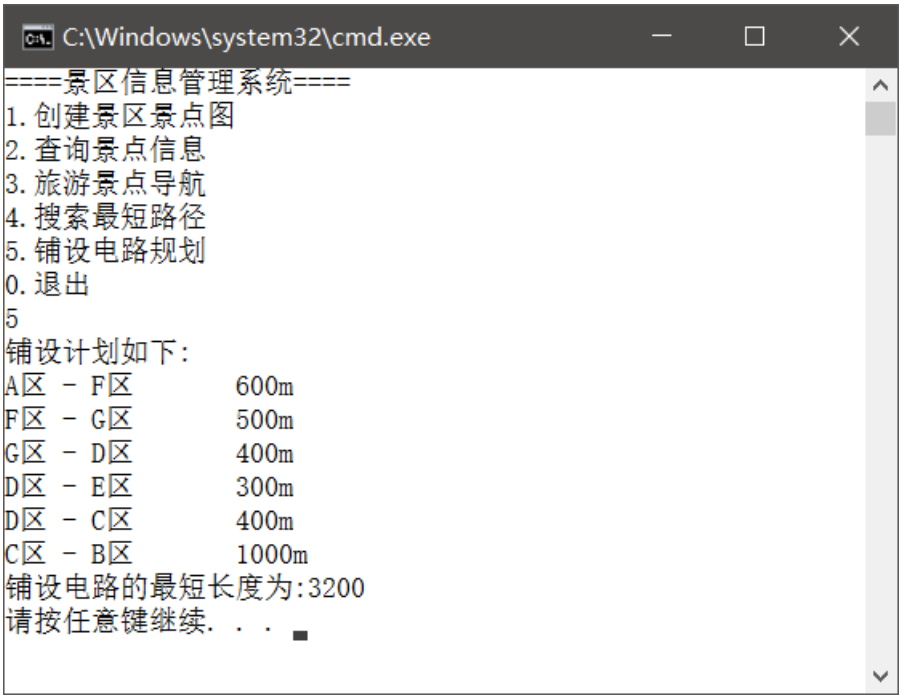



C:\Windows\system32\cmd.exe
0.退出
1
-----Vex----
0-A区
1-B区
2-C区
3-D区
4-E区
5-F区
6-G区
----Edge----
(v0,v2) 700
(v0,v4) 1000
(v0,v5) 600
(v1,v2) 1000
(v1,v6) 1000
(v2,v3) 400
(v3,v4) 300
(v3,v6) 400
(v4,v5) 600
(v5,v6) 500
请按任意键继续. . .


C:\Windows\system32\cmd.exe
====景区信息管理系统====
1.创建景区景点图
2.查询景点信息
3.旅游景点导航
4.搜索最短路径
5.铺设电路规划
0.退出
2
请输入想要查询的景点编号
4
E区 风景优美，气候宜人。门票50元。
----周边景区-----
E区->A区 1000
E区->D区 300
E区->F区 600
请按任意键继续. . .


C:\Windows\system32\cmd.exe
====景区信息管理系统====
1. 创建景区景点图
2. 查询景点信息
3. 旅游景点导航
4. 搜索最短路径
5. 铺设电路规划
0. 退出
3
输入当前所在区域
4
E区->A区->F区->G区->B区->C区->D区
E区->A区->F区->G区->D区->C区->B区
E区->D区->C区->A区->F区->G区->B区
E区->D区->C区->B区->G区->F区->A区
E区->D区->G区->B区->C区->A区->F区
E区->D区->G区->F区->A区->C区->B区
E区->F区->A区->C区->B区->G区->D区
E区->F区->A区->C区->D区->G区->B区
请按任意键继续. . .


C:\Windows\system32\cmd.exe
====景区信息管理系统====
1. 创建景区景点图
2. 查询景点信息
3. 旅游景点导航
4. 搜索最短路径
5. 铺设电路规划
0. 退出
4
请输入起点的编号:
2
请输入终点的编号:
6
总最短路径长度: 800
Path:2->3->6
请按任意键继续. . .



第三部分：实验小结、收获与体会

1.数据结构中树与图的应用很重要 ,DFS, Dijkstra,Kruskal,算法有着十分重要的意义,通过本次实验学习到了数据结构与算法的很多知识,收获颇丰

2.善于运用 this,在 get/set 类方法中遇到局部变量与类成员变量同名时可通过 this 来解决冲突问题

3.面向对象的思想在编程中有着重要的运用,实际是计算机发展过程中对大自然各种现象的模拟,深入理解对编程的学习很有帮助

4.编程过程中,对于边界条件需要仔细考虑,比如顶点数,距离,边数,方便以及提示用户输入有误,这些都是很重要的细节,在产品设计中也有着很重要的运用,做到良好的人机交互性.

5.通过本次数据结构与算法综合实验,构建了旅游景点路径规划以及导航,并且实现了搜索最短路径,各点之间电路铺设,分别设计并编程实现了 DFS, Dijkstra,Kruskal, 算法.

附录:

工程代码如下:

common.h

```
#ifndef _COMMON_H_
#define _COMMON_H_

#include<string>
```

```
#include"Graph.h"
const int MAX_VERTEX_NUM = 20;
//定义未联通路
const int INF = 999999;
const std::string VexFileName = "Vex.txt";
const std::string EdgeFileName = "Edge.txt";
void initTree(int root[], int n);
int find(int root[], int i);
void mergeNode(int root[], int i, int j);

#endif // !_COMMON_H
```

Graph.h

```
#ifndef _GRAPH_H_
#define _GRAPH_H_

#include<string>
#include<vector>

struct Vex
{
    int num;
    std::string name;
    std::string desc;
};

struct Edge
{
    int vex1;
    int vex2;
    int weight;
};

class CGraph
{
private:
    int m_aAdjMatrix[20][20];
    Vex m_aVexs[20];
    int m_nVexNum;
public:
    void init();
    void dfs(int n, int vis[], int path[]);
    int findSSSP(int i, int j, std::vector<int>& vis);
    bool InsertVex(Vex sVex);
```

```
    int FindEdge(int v, std::vector<Edge> &aEdge);
    bool InsertEdge(Edge sEdge);
    void printVex();
    void printEdge();
    Vex GetVex(int v);
    int FindEdge(int v, Edge aEdge[]);
    int getVexNum();
    int DesignPath();
};

#endif // !_GRAPH_H_
```

Tourism.h

```
#ifndef _TOURISM_H_
#define _TOURISM_H_

#include<vector>

#include"Graph.h"

struct HeapNode {
    int d, u;
    bool operator<(const HeapNode& rhs)const {
        return d < rhs.d;
    }
};

void createGraph();
void print();
void GetSpotInfo();
void dijkstra(int d[], int i, int n, Vex v[], Edge fa[], std::vector<std::vector<Edge>> g);
void FindShortPath(int i, int j);
void TravelPath();
void designPath();

#endif // !_TOURISM_H_
```

common.cpp

```
#include <iostream>
#include "common.h"
#include "Graph.h"
#include "Tourism.h"
```

```
using namespace std;
void initTree(int root[],int n)
{
    for (int i = 0; i < n; ++i)
    {
        root[i] = i;
    }
}
int find(int root[], int i)
{
    int r = root[i];
    while (root[r] != r)r = root[r];
    int j, k;
    j = i;
    while (j != r)
    {
        k = root[j];
        root[j] = r;
        j = k;
    }
    return r;
}
void mergeNode(int root[],int i, int j)
{
    int fi = find(root,i);
    int fj = find(root,j);
    if(fi!=fj)
    root[j] = fi;
}
```

Graph.cpp

```
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
#include <cstring>
#include <set>
#include <queue>

#include "common.h"
#include "Graph.h"
#include "Tourism.h"
using namespace std;
```

```
void CGraph::init()
{
    this->m_nVexNum = 0;
    memset(this->m_aAdjMatrix, 0, sizeof(this->m_aAdjMatrix));
}

void CGraph::dfs(int n, int vis[], int path[])
{
    if (n == m_nVexNum)
    {
        cout << m_aVexs[path[0]].name;
        for (int i = 1; i < m_nVexNum; ++i)
        {
            cout << "->" << m_aVexs[path[i]].name;
        }
        cout << endl;
        return;
    }
    else
    {
        for (int i = 0; i < m_nVexNum; ++i)
        {
            if (vis[i] == 0&&m_aAdjMatrix[path[n-1]][i]!=0)//首次访问且连通
            {
                vis[i] = 1;
                path[n] = i;
                dfs(n + 1, vis, path);
                vis[i] = 0;
            }
        }
    }
}


int CGraph::getVexNum()
{
    return m_nVexNum;
}
int CGraph::DesignPath()
{
    int m_num = this->m_nVexNum;
    int root[30];
    initTree(root, m_num);
    //start at Vex zero
    set<int>Vn;
```

```
    int ret = 0;
    Edge temp[30];
    int size= 0;
    Edge e;
    int min = INF;
    //挑选第一条边
    size=this->FindEdge(0, temp);
    for (int i = 0; i < size; ++i)
    {
        if (temp[i].weight < min)
        {
            e = temp[i];
            min = e.weight;
        }
    }
    Vn.insert(e.vex1);
    Vn.insert(e.vex2);
    //merge
    mergeNode(root,e.vex1, e.vex2);
    ret += e.weight;
    //start
    cout << "铺设计划如下:" << endl;
    cout << this->m_aVexs[e.vex1].name << " - " << this->m_aVexs[e.vex2].name << "  " <<
e.weight <<"m"<< endl;
    set<int>::iterator iter;
    for (int i = 0; i < m_num - 2; ++i)
    {
        min = INF;
        for (iter = Vn.begin(); iter != Vn.end(); ++iter)
        {
            int vex1 = *iter;
            int fv1 = find(root, vex1);
            size = this->FindEdge(*iter, temp);
            for (int j = 0; j < size; ++j)
            {
                int vex2 = temp[j].vex2;
                int weight = temp[j].weight;
                int fv2 = find(root, vex2);
                if (fv1 != fv2 && weight < min)
                {
                    min = weight;
                    e = temp[j];
                }
            }
        }
```

```
            //merge
            mergeNode(root, e.vex1, e.vex2);
            //cout
            cout << this->m_aVexs[e.vex1].name << " - " << this->m_aVexs[e.vex2].name << "  "
<< e.weight <<"m"<< endl;
            Vn.insert(e.vex2);
            ret += e.weight;
        }
        cout << "铺设电路的最短长度为:" << ret << endl;
        return 0;
}
//vis存储从源点到目标点之间最短路的访问路径
int CGraph::findSSSP(int i, int j,vector<int>& vis)
{
        Edge *fa = new Edge[m_nVexNum+1];
        int *d = new int[m_nVexNum];
        //void dijkstra(int i, int n,Vex v[], Edge fa[], vector<Edge> g[])
        //vector<Edge>* g = new vector<Edge>[m_nVexNum];
        vector<vector<Edge>> g;
        int num = m_nVexNum;
        while (num--)
        {
            vector<Edge> e;
            g.push_back(e);
        }
        for (int i = 0; i < m_nVexNum; ++i)
        {
            this->FindEdge(i, g[i]);
        }
        dijkstra(d,i, m_nVexNum, this->m_aVexs, fa, g);
        vis.clear();
        int p = j;
        while (p != i)
        {
            vis.push_back(p);
            Edge temp= fa[p];
            p = temp.vex1;
        }
        reverse(vis.begin(), vis.end());
        delete[] fa;
        int ret=d[j];
        delete[] d;
        return ret;
}
bool CGraph::InsertVex(Vex sVex)
```

```
{
    if (m_nVexNum == MAX_VERTEX_NUM)
    {
        cout << "InsertVex:顶点超出最大限制!" << endl;
        return false;
    }
    this->m_aVexs[this->m_nVexNum++] = sVex;
    return true;
}

bool CGraph::InsertEdge(Edge sEdge)
{
    if (sEdge.vex1 < 0 || sEdge.vex1 >= m_nVexNum || sEdge.vex2 < 0 || sEdge.vex2 >= m_nVexNum)
    {
        cout << "InsertEdge:下标越界" << endl;
        return false;
    }
    m_aAdjMatrix[sEdge.vex1][sEdge.vex2] = sEdge.weight;
    m_aAdjMatrix[sEdge.vex2][sEdge.vex1] = sEdge.weight;
    return true;
}

void CGraph::printVex()
{
    cout << "-----Vex----" << endl;
    for (int i = 0; i < m_nVexNum; ++i)
    {
        cout << m_aVexs[i].num << "-" << m_aVexs[i].name << endl;
    }
}
void CGraph::printEdge()
{
    cout << "----Edge----" << endl;
    for (int i = 0; i < m_nVexNum; ++i)
    {
        for (int j = i+1; j < m_nVexNum; ++j)
        {
            if (m_aAdjMatrix[i][j] != 0)
            {
                printf("(v%d,v%d) %d\n", i, j, m_aAdjMatrix[i][j]);
            }
        }
    }
}
Vex CGraph::GetVex(int v)
```

```
{
    return m_aVexs[v];
}

int CGraph::FindEdge(int v, Edge aEdge[])
{
    int k = 0;
    Edge edge;
    edge.vex1 = v;
    for (int i = 0; i < m_nVexNum; ++i)
    {
        if (m_aAdjMatrix[v][i] != 0)
        {
            edge.vex2 = i;
            edge.weight = m_aAdjMatrix[v][i];
            aEdge[k++] = edge;
        }
    }
    return k;
}
int CGraph::FindEdge(int v, vector<Edge>& aEdge)
{
    int k = 0;
    Edge edge;
    edge.vex1 = v;
    for (int i = 0; i < m_nVexNum; ++i)
    {
        if (m_aAdjMatrix[v][i] != 0)
        {
            edge.vex2 = i;
            edge.weight = m_aAdjMatrix[v][i];
            aEdge.push_back(edge);
        }
    }
    return aEdge.size();
}
```

Tourism.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <queue>
```

```
#include "Tourism.h"
#include "Graph.h"
#include "common.h"

using namespace std;
CGraph m_Graph;
void createGraph()
{
    m_Graph.init();
    ifstream fin(VexFileName);
    int count;
    fin >> count;
    Vex vex;
    for (int i = 0; i < count; ++i)
    {
        fin >> vex.num >> vex.name >> vex.desc;
        m_Graph.InsertVex(vex);
    }
    fin.close();
    fin.open(EdgeFileName);
    Edge edge;
    while (fin >> edge.vex1 >> edge.vex2 >> edge.weight)
    {
        m_Graph.InsertEdge(edge);
    }
    return;
}
void print()
{
    m_Graph.printVex();
    m_Graph.printEdge();
}
void GetSpotInfo()
{
    cout << "请输入想要查询的景点编号" << endl;
    int i;
    cin >> i;
    Vex vex = m_Graph.GetVex(i);
    cout << vex.name << " " << vex.desc << endl;
    cout << "----周边景区-----" << endl;
    Edge aEdge[20];
    int k=m_Graph.FindEdge(i, aEdge);
    for (int j = 0; j < k; ++j)
    {
        Vex temp = m_Graph.GetVex(aEdge[j].vex2);
```

```
            cout << vex.name << "->" << temp.name<< " " << aEdge[j].weight << endl;
        }
        cout << endl;
        return;
}
void FindShortPath(int i, int j)
{
        vector<int>vis;
        vis.clear();
        int d=m_Graph.findSSSP(i, j,vis);
        cout <<"总最短路径长度: "<< d << endl;
        cout << "Path:" << i;
        for (int k = 0; k < vis.size(); ++k)
        {
            cout << "->" << vis[k];
        }
        cout << endl;
}
/*数组d为存储各个节点到i的最短距离
 *i,n分别为起始节点和节点数量
 *fa是存储第i条边的前一条边用于输出
 *g[i][j]表示节点i的第j条边
 */
void dijkstra(int d[],int i, int n,Vex v[], Edge fa[], vector<vector<Edge>> g)
{
        priority_queue<HeapNode> Q;
        for (int k = 0; k < n; ++k) d[k] = INF;
        d[i] = 0;
        int *done = new int[n + 1];
        memset(done, 0, sizeof(int)*(n+1));
        HeapNode temp;
        temp.d = 0;
        temp.u = i;
        Q.push(temp);
        while (!Q.empty())
        {
            HeapNode x = Q.top(); Q.pop();
            int u = x.u;
            if (done[u])continue;
            done[u] = 1;
            for (int k = 0; k < g[u].size(); ++k)
            {
                Edge &e = g[u][k];
                if (d[e.vex2] > d[u] + e.weight)
                {
```

```
                    d[e.vex2] = d[u] + e.weight;
                    fa[e.vex2] = g[u][k];
                    HeapNode t;
                    t.u = e.vex2;
                    t.d = d[e.vex2];
                    Q.push(t);
                }
            }
        }
        delete[] done;
}
void TravelPath()
{
    int i;
    cout << "输入当前所在区域" << endl;
    cin >> i;
    int *vis = new int[21];
    int *path = new int[21];
    path[0] = i;
    memset(vis, 0, sizeof(int) * 21);
    vis[i] = 1;
    m_Graph.dfs(1,vis,path);
    delete[] vis;
    delete[] path;
    return ;
}

void designPath()
{
    m_Graph.DesignPath();
}
```

main.cpp

```
#include <cstdio>
#include <cstdlib>
#include <iostream>

#include "common.h"
#include "Graph.h"
#include "Tourism.h"

using namespace std;

void showMenu()
```

```
{
    cout << "====景区信息管理系统====" << endl;
    cout << "1.创建景区景点图" << endl;
    cout << "2.查询景点信息" << endl;
    cout << "3.旅游景点导航" << endl;
    cout << "4.搜索最短路径" << endl;
    cout << "5.铺设电路规划" << endl;
    cout << "0.退出" << endl;
    return;
}
int main()
{
    int selectCode=1;
    while (selectCode)
    {
        system("cls");
        showMenu();
        cin >> selectCode;
        switch (selectCode)
        {
        case 0: {
            break;
        }
        case 1: {
            createGraph();
            print();
            system("pause");
            break;
        }
        case 2: {
            GetSpotInfo();
            system("pause");
            break;
        }
        case 3: {
            TravelPath();
            system("pause");
            break;
        }
        case 4: {
            int i, j;
            cout << "请输入起点的编号: " << endl;
            cin >> i;
            cout << "请输入终点的编号: " << endl;
            cin >> j;
```

```
            FindShortPath(i, j);
            system("pause"); break;

        }case 5: {
            designPath();
            system("pause");
            break;
        }
        default: {
            cout << "输入错误，请重新输入" << endl;
            break;
        }
        }
    }
    cout << "欢迎下次使用!" << endl;
    return 0;
}
```