

学生学号	0121710880414	实验课成绩	
------	---------------	-------	--

# 武汉理工大学 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	夏红霞
学 生 姓 名	穆逸诚
学生专业班级	软件 1704 班

2018    --    2019    学 年    第   二   学 期

## 实验教学管理基本规范

实验是培养学生动手能力、分析解决问题能力的重要环节；实验报告是反映实验教学水平与质量的重要依据。为加强实验过程管理，改革实验成绩考核方法，改善实验教学效果，提高学生质量，特制定实验教学管理基本规范。

- 1、本规范适用于理工科类专业实验课程，文、经、管、计算机类实验课程可根据具体情况参照执行或暂不执行。
- 2、每门实验课程一般会包括许多实验项目，除非常简单的验证演示性实验项目可以不写实验报告外，其他实验项目均应按本格式完成实验报告。
- 3、实验报告应由实验预习、实验过程、结果分析三大部分组成。每部分均在实验成绩中占一定比例。各部分成绩的观测点、考核目标、所占比例可参考附表执行。各专业也可以根据具体情况，调整考核内容和评分标准。
- 4、学生必须在完成实验预习内容的前提下进行实验。教师要在实验过程中抽查学生预习情况，在学生离开实验室前，检查学生实验操作和记录情况，并在实验报告第二部分教师签字栏签名，以确保实验记录的真实性。
- 5、教师应及时评阅学生的实验报告并给出各实验项目成绩，完整保存实验报告。在完成所有实验项目后，教师应按学生姓名将批改好的各实验项目实验报告装订成册，构成该实验课程总报告，按班级交课程承担单位（实验中心或实验室）保管存档。
- 6、实验课程成绩按其类型采取百分制或优、良、中、及格和不及格五级评定。

**附表：实验考核参考内容及标准**

	观测点	考核目标	成绩组成
实验预习	1. 预习报告 2. 提问 3. 对于设计型实验，着重考查设计方案的科学性、可行性和创新性	对实验目的和基本原理的认识程度，对实验方案的设计能力	20%
实验过程	1. 是否按时参加实验 2. 对实验过程的熟悉程度 3. 对基本操作的规范程度 4. 对突发事件的应急处理能力 5. 实验原始记录的完整程度 6. 同学之间的团结协作精神	着重考查学生的实验态度、基本操作技能；严谨的治学态度、团结协作精神	30%
结果分析	1. 所分析结果是否用原始记录数据 2. 计算结果是否正确 3. 实验结果分析是否合理 4. 对于综合实验，各项内容之间是否有分析、比较与判断等	考查学生对实验数据处理和现象分析的能力；对专业知识的综合应用能力；事实求实的精神	50%

实验课程名称： 算法设计与分析实验

实验项目名称	图与景区信息管理系统实践			实验成绩	
实 验 者	穆逸诚	专业班级	软件 1704	组 别	
同 组 者				实验日期	2019 年 5 月 29 日

## 第一部分：实验分析与设计

### 一、实验目的和要求

实验目的：

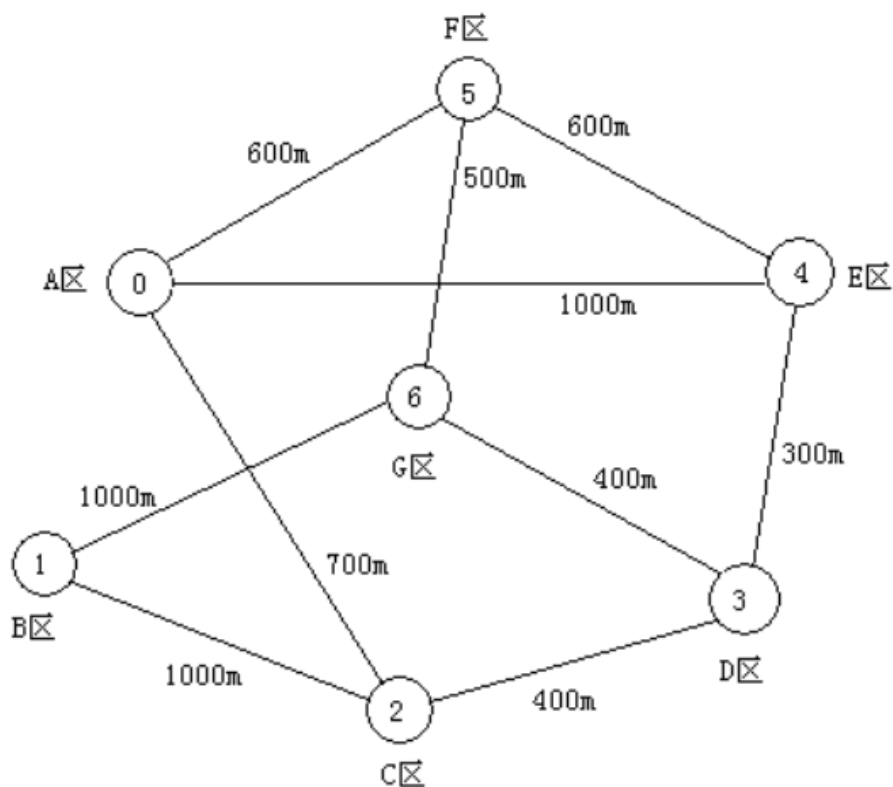
- (1) 掌握图的定义和图的存储结构；
- (2) 掌握图的创建方法；
- (3) 掌握图的两种遍历方法；
- (4) 理解迪杰斯特拉（Dijkstra）算法；
- (5) 理解最小生成树的概念和普利姆（Prim）算法；
- (6) 掌握文件操作；
- (7) 使用 C++ 语言，利用图的数据结构，开发景区信息管理系统。

实验要求：

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) 这些道路都是可以双向行驶的（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。

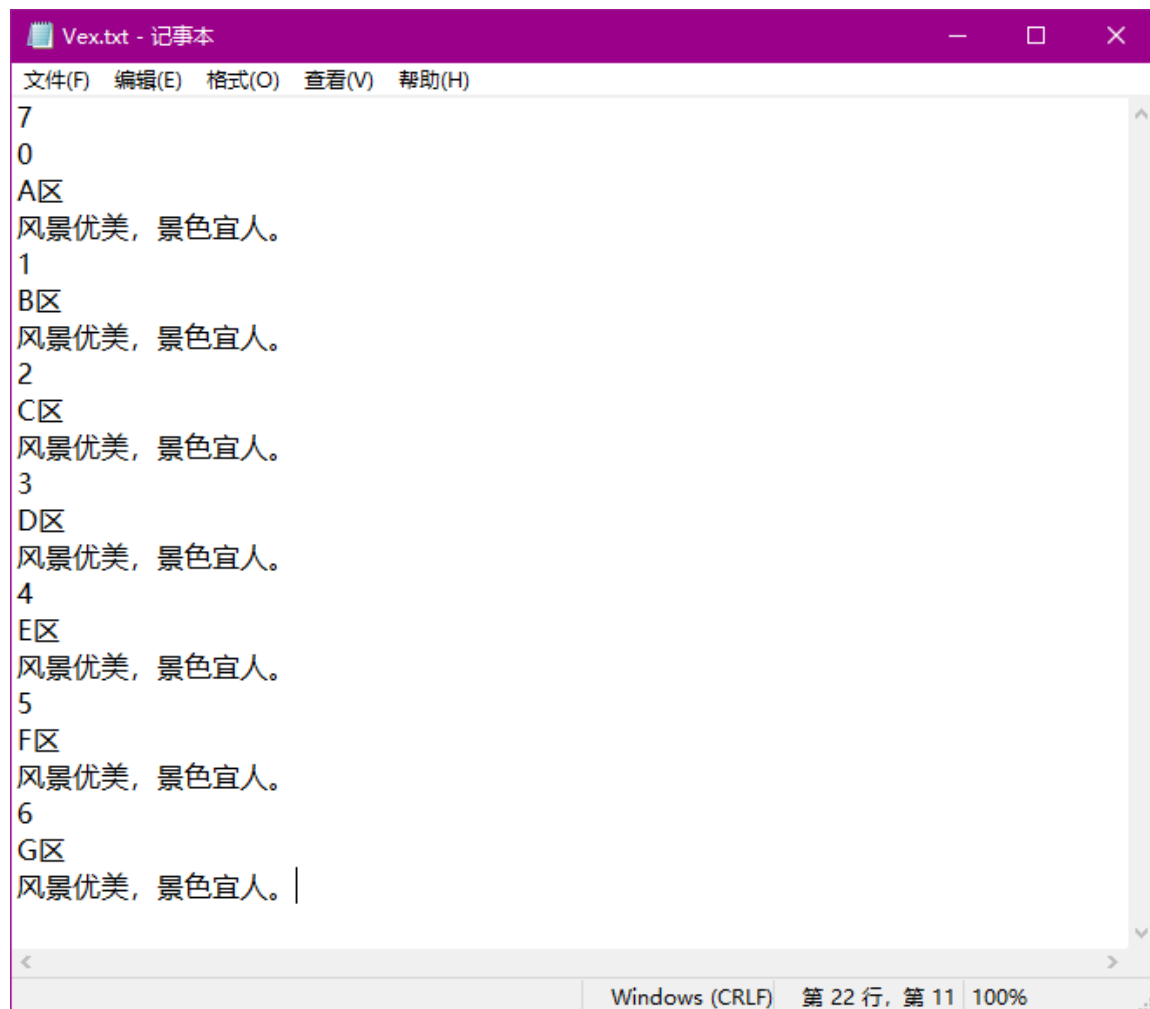


## 二、实验分析与设计

### (1) 文件组织形式

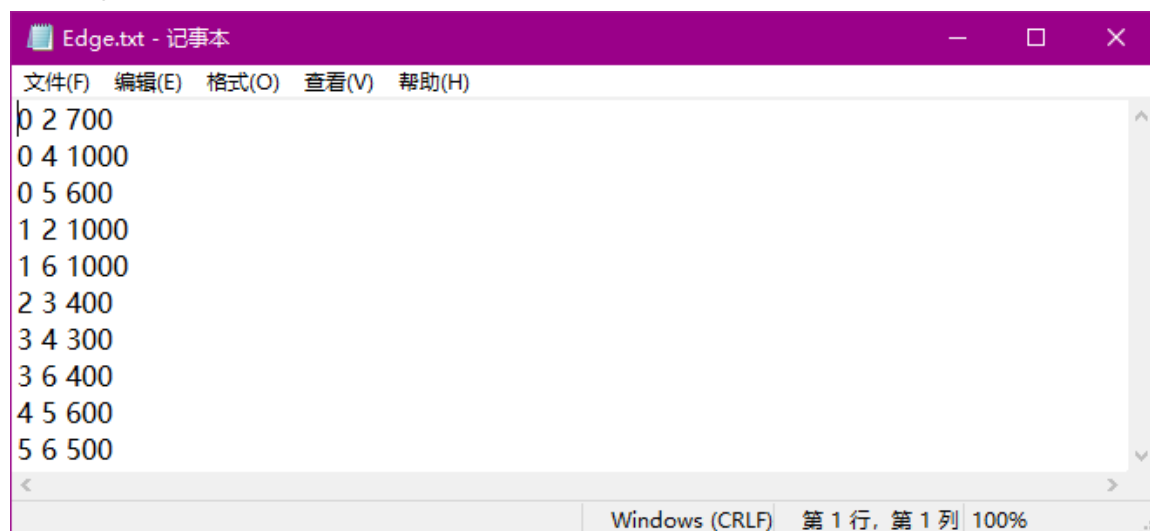
景区的数据包含景点信息和景点之间的道路信息。分别由两个文本文件存储。Vex.txt 文件用来存储景点信息；Edge.txt 文件用来存储道路信息。

Vex.txt:



```
Vex.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
7
0
A区
风景优美, 景色宜人。
1
B区
风景优美, 景色宜人。
2
C区
风景优美, 景色宜人。
3
D区
风景优美, 景色宜人。
4
E区
风景优美, 景色宜人。
5
F区
风景优美, 景色宜人。
6
G区
风景优美, 景色宜人。|
Windows (CRLF) 第 22 行, 第 11 100%
```

Edge.txt:



```
Edge.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0 2 700
0 4 1000
0 5 600
1 2 1000
1 6 1000
2 3 400
3 4 300
3 6 400
4 5 600
5 6 500
Windows (CRLF) 第 1 行, 第 1 列 100%
```

## (2) 数据结构设计

本次实验主要围绕图的存储、遍历等操作展开，因而使用的是邻接矩阵的存储结构，遍历算法涉及到三种，旅游景点导航功能使用的是深度优先搜索算法（DPS），搜索最短路径功能使用的是迪杰斯特拉（Dijkstra）算法、铺设电路规划功能运用到了普利姆（Prim）算法等

## (3) 头文件函数展示

### ①Graph.h

```
#pragma once
#ifndef GRAPH_H
#define GRAPH_H
#define OK 1
#define ERROR 0
//定义景点（图的顶点）
struct Vex {
    int num;           //景点的编号
    char name[20];     //景点的名称
    char desc[1024];   //景点的描述
};
//定义道路信息（图的边）
struct Edge {
    int vex1;          //边的第一个顶点
    int vex2;          //边的第二个顶点
    int weight;        //权值，即两个相邻景点的距离
};
//定义图
struct Graph {
    int m_aAdjMatrix[20][20]; //采用邻接矩阵的形式储存
    Vex m_aVexs[20];          //顶点数组
    int m_nVexNum;             //景点的数目
};
//搜索路径
typedef struct Path {
    int vexs[20];              //保存的一条完整的路径
    Path *next;                //下一条路径
}*PathList;
//初始化图结构
int Init();
//将定点添加到数组中
int InsertVex(Vex sVex);
//将边保存到邻接矩阵中
int InsertEdge(Edge sEdge);
//查询指定顶点信息
Vex GetVex(int nVex);
//查询与指定顶点相连的边
int FindEdge(int nVex, Edge aEdge[]);
```

```

//获取当前顶点数
int GetVexmun();
//实现图的深度优先搜索遍历
void DFS(int nVex, bool bVisited[], int &nIndex, PathList &pList);
//深度优先遍历
void DFSTraverse(int nVex, PathList &pList);
//通过 Dijkstra 算法求得 nVexStart 到 nVexEnd 的最短路径
int FindShortPath(int nVexStart, int nVexEnd, Edge aPath[]);
//构建最小生成树
void FindMinTree(Edge aPath[]);
#endif

```

## ②Tourism.h

```

#pragma once
#ifndef TOURISM_H
#define TOURISM_H
#define OK 1
#define ERROR 0
//读取文件，创建景区景点图
int CreateGraph();
//查询指定景点信息，显示到相邻景点的距离
int GetSPotInfo();
//得到景点导航图路线，并显示
void TravelPath();
//通过调用函数查询两个景点之间的最短路径和距离
void FindShortPath(void);
//查询铺设电路规划图
void DesigePath(void);
#endif

```

### （4）核心算法展示

#### ①深度优先搜索

```

void DFS(int nVex, bool bVisited[], int &nIndex, PathList &pList) {
    bVisited[nVex] = true;           //改为已访问
    pList->vexs[nIndex++] = nVex;    //访问顶点 nVex 并赋值给链表，然后索引值自加
    //判断所有的顶点是否都已经被访问过
    int v_num = 0;
    for (int i = 0; i < m_Graph.m_nVexNum; i++) {
        //如果当前 i 节点被访问过，则 V-Num 自加
        if (bVisited[i])
            v_num++;
    }
    if (v_num == m_Graph.m_nVexNum) {
        //创建一个新链表，将当前的 pList 中的数据保存起来
    }
}

```

```

    pList->next = new Path;
    for (int i = 0; i < m_Graph.m_nVexNum; i++) {
        pList->next->vexs[i] = pList->vexs[i];
    }
    pList = pList->next;          //pList 指针继续往下移动，寻找下一条路径
    pList->next = NULL;          //next 赋值为空
}
//并没有全部访问，则进行寻找下一个相邻节点的操作
else {
    for (int i = 0; i < m_Graph.m_nVexNum; i++) {
        //如果 i 是 nVex 的邻接点 并且未被访问
        if (!bVisited[i] && m_Graph.m_aAdjMatrix[nVex][i] > 0) {
            DFS(i, bVisited, nIndex, pList);          //递归调用 DFS
            bVisited[i] = false;                      //改为未访问，回退
            nIndex--;                                  //索引值-1
        }
    }
}
}
}

```

## ②寻找最短路径

```

int FindShortPath(int nVexStart, int nVexEnd, Edge aPath[]) {
    int nShortPath[20][20];          //保存最短路径，其中行表示终点，列表示从起点到终点的
    //最短路径的每一步
    int nShortDistance[20];          //保存最短距离，保存从起点到任一顶点的最短距离
    bool aVisited[20];              //判断某顶点是否已经加入到最短路径中
    int v;                          //在下面的循环中，表示每一次找到的可以加入集合
    //的顶点，即已经找到了从起点到该顶点的最短路径
    //初始化工作
    for (v = 0; v < m_Graph.m_nVexNum; v++) {
        aVisited[v] = false;
        if (m_Graph.m_aAdjMatrix[nVexStart][v] != 0) {
            //初始化该顶点到其他顶点的最短距离，默认为两顶点间的距离
            nShortDistance[v] = m_Graph.m_aAdjMatrix[nVexStart][v];
        }
        else {
            //如果顶点 v 和 nVexStart 不相连，则最短距离设置为最大值
            nShortDistance[v] = 0x7FFFFFFF;
        }
        nShortPath[v][0] = nVexStart;    //起始点为 nVexStart
        //初始化最短路径
        for (int w = 1; w < m_Graph.m_nVexNum; w++) {
            nShortPath[v][w] = -1;
        }
    }
}

```

```

    }
    //初始化, 将 nVexStart 顶点加入到集合中
    aVisited[nVexStart] = true;
    int min;           //暂存路径的最小值
    for (int i = 1; i < m_Graph.m_nVexNum; i++) {
        min = 0x7FFFFFFF;
        bool bAdd = false;    //判断是否还有顶点可以加入集合
        for (int w = 0; w < m_Graph.m_nVexNum; w++) {
            if (!aVisited[w] && nShortDistance[w] < min) {
                v = w;           //w 顶点距离 nVexStart 顶点最近
                min = nShortDistance[w];    //w 到 nVexStart 的最短距离为 min
                bAdd = true;
            }
        }
        //若果没有顶点可以加入到集合, 则跳出循环
        if (!bAdd) break;
        aVisited[v] = true;    //将 w 顶点加入到集合
        nShortPath[v][i] = v;
        for (int w = 0; w < m_Graph.m_nVexNum; w++) {
            //将 w 作为过渡顶点计算 nVexStart 通过 w 到每个顶点的距离
            if (!aVisited[w] && (min + m_Graph.m_aAdjMatrix[v][w] < nShortDistance[w])
                && (m_Graph.m_aAdjMatrix[v][w] > 0)) {
                //更新当前最短路径及距离
                nShortDistance[w] = min + m_Graph.m_aAdjMatrix[v][w];
                for (int i = 0; i < m_Graph.m_nVexNum; i++) { //如果通过 w 达到顶点 i 的
                    //距离比较短, 则将 w 的最短路径复制给 i
                    nShortPath[w][i] = nShortPath[v][i];
                }
            }
        }
    }
}

```

### ③最小生成树

```

void FindMinTree(Edge aPath[]) {
    bool aVisited[20] = { false };    //判断某顶点是否在最小生成树中, true 表示已经添加
    //到了最小生成树中
    aVisited[0] = true;                //从 0 号顶点开始, 加入到集合中
    int min;
    int nVex1, nVex2;
    for (int k = 0; k < m_Graph.m_nVexNum - 1; k++) {
        min = 0x7FFFFFFF;
        for (int i = 0; i < m_Graph.m_nVexNum; i++) {
            //从集合中取一个顶点
            if (aVisited[i]) {

```



```

        for (int j = 0; j < m_Graph.m_nVexNum; j++) {
            //从不在集合中的顶点 中取出一个顶点
            if (!aVisited[j]) {

                if((m_Graph.m_aAdjMatrix[i][j] < min) && m_Graph.m_aAdjMatrix[i][j] != 0) {
                    nVex1 = i;
                    nVex2 = j;
                    //找出最短边
                    min = m_Graph.m_aAdjMatrix[i][j];
                }
            }
        }
    }
    //保存最短边的两个顶点
    aPath[k].vex1 = nVex1;
    aPath[k].vex2 = nVex2;
    aPath[k].weight = m_Graph.m_aAdjMatrix[nVex1][nVex2];
    //将两个顶点加入集合
    aVisited[nVex1] = true;
    aVisited[nVex2] = true;
}
}

```

### 三、主要仪器设备及耗材

1. PC 机
2. 开发环境：VS2017

## 第二部分：实验调试与结果分析

一、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

### 1. 调试方法描述

- ① 输入 C++ 程序，并保存；
- ② 编译 C++ 程序，找出程序的语法错误并改正；
- ③ 输入测试数据（除每区域普遍值外包括一些特殊临界值），运行 C++ 程序，若有错，查找并修改程序的逻辑错误；
- ④ 重复②-③步，直到得到正确的运行结果

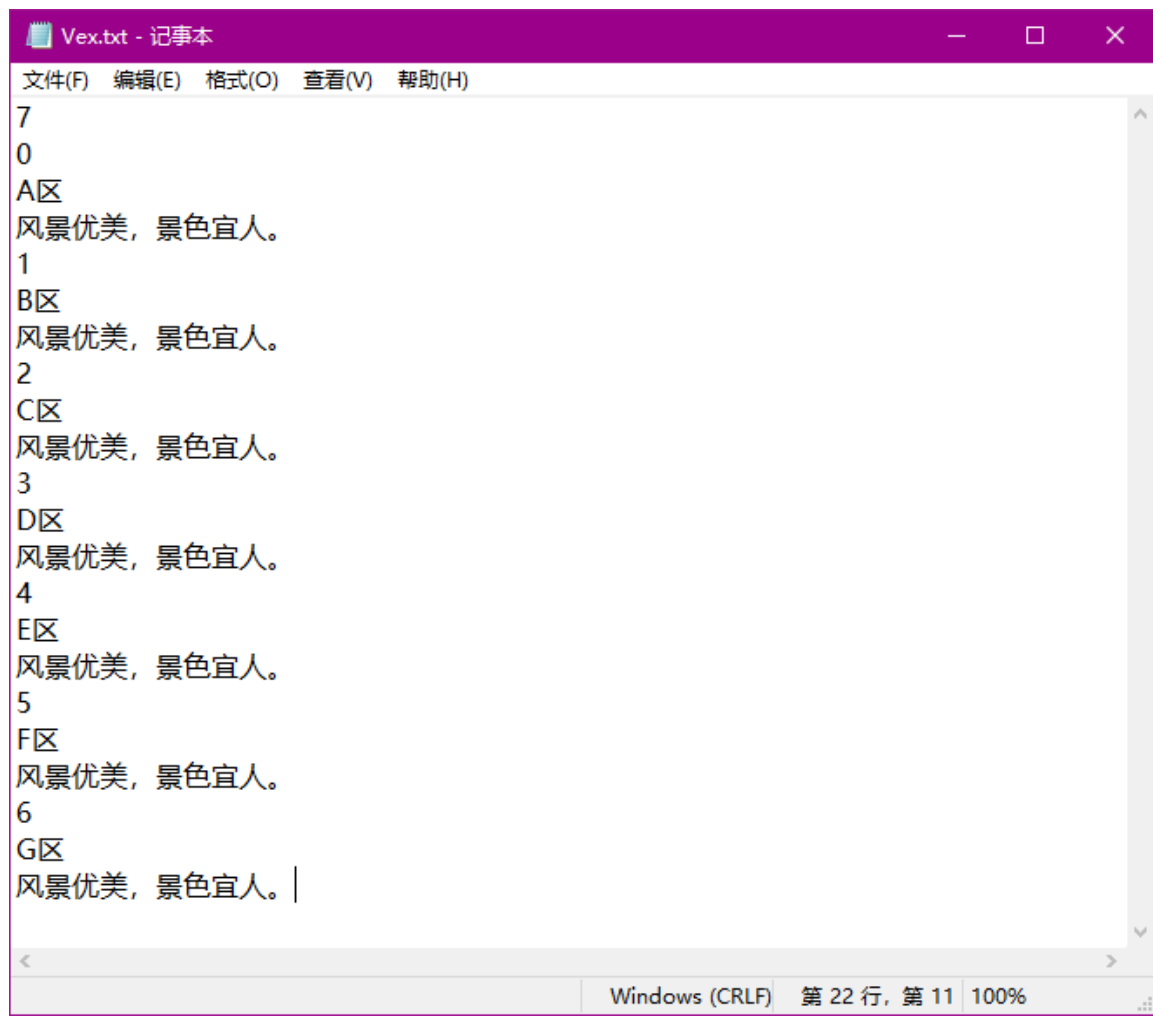
### 2. 实验输入/输出数据记录

—>★具体实验操作截图如下：

- （1）编辑 Vex.txt 文件，用于存储景点信息

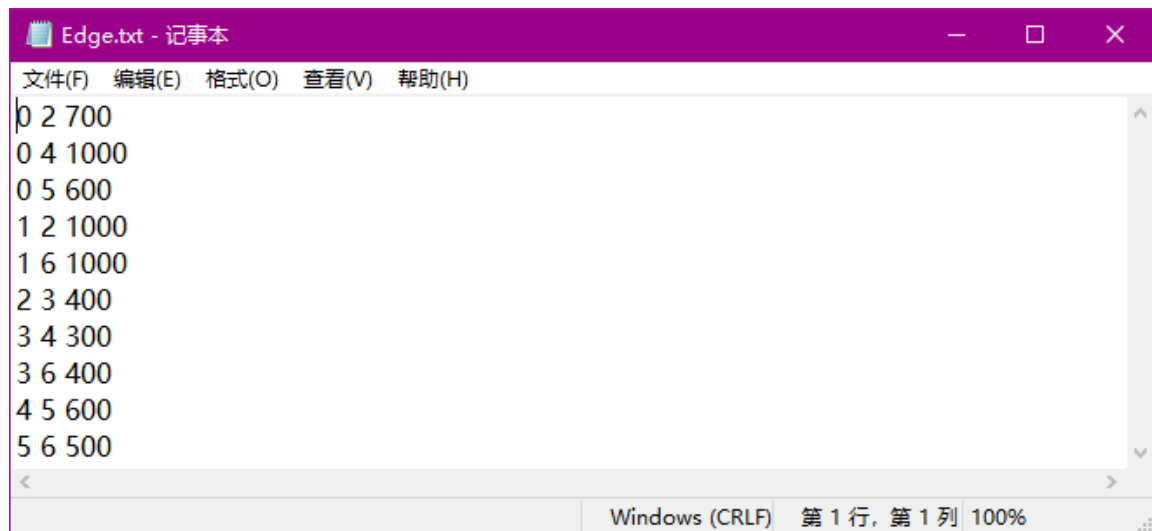
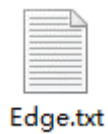


Vex.txt



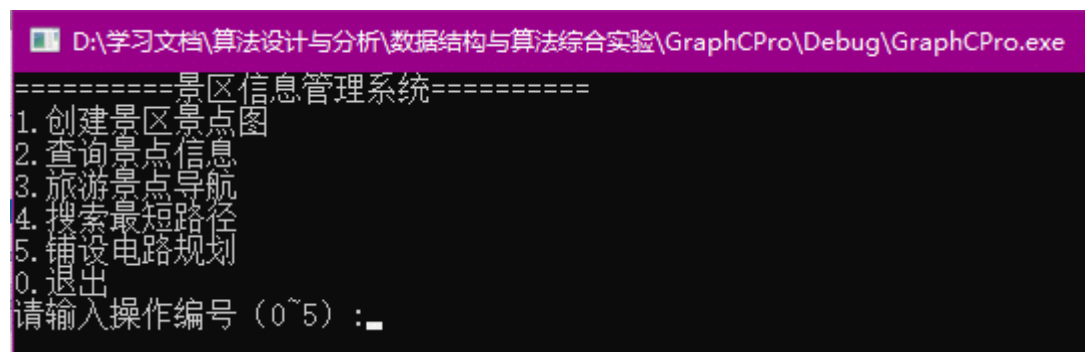
```
7
0
A区
风景优美, 景色宜人。
1
B区
风景优美, 景色宜人。
2
C区
风景优美, 景色宜人。
3
D区
风景优美, 景色宜人。
4
E区
风景优美, 景色宜人。
5
F区
风景优美, 景色宜人。
6
G区
风景优美, 景色宜人。|
```

(2) 编辑 Edge.txt 文件，用于存储道路信息



```
Edge.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0 2 700
0 4 1000
0 5 600
1 2 1000
1 6 1000
2 3 400
3 4 300
3 6 400
4 5 600
5 6 500
Windows (CRLF) 第 1 行, 第 1 列 100%
```

(3) 程序主界面



```
D:\学习文档\算法设计与分析\数据结构与算法综合实验\GraphCPro\Debug\GraphCPro.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
请输入操作编号 (0~5) : _
```

(4) 创建景区景点图

```
D:\学习文档\算法设计与分析\数据结构与算法综合实验\GraphCPro\Debug\GraphCPro.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
请输入操作编号 (0~5) :1
=====创建景点景区图=====
景区数目: 7
-----顶点-----
编号    景区名称    景区描述
0        A        风景优美, 景色宜人。
1        B        风景优美, 景色宜人。
2        C        风景优美, 景色宜人。
3        D        风景优美, 景色宜人。
4        E        风景优美, 景色宜人。
5        F        风景优美, 景色宜人。
6        G        风景优美, 景色宜人。
-----边-----
<0, 2>    700
<0, 4>    1000
<0, 5>    600
<1, 2>    1000
<1, 6>    1000
<2, 3>    400
<3, 4>    300
<3, 6>    400
<4, 5>    600
<5, 6>    500

=====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
请输入操作编号 (0~5) :_
```

(5) 查询景点信息

```
D:\学习文档\算法设计与分析\数据结构与算法综合实验\GraphCPro\Debug\GraphCPro.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
请输入操作编号 (0~5) : 2
=====查询景点信息=====
当前景区个数: 7
编号    景点名称
0        A区
1        B区
2        C区
3        D区
4        E区
5        F区
6        G区

请输入想要查询的景点编号: 0
景点名称    景点描述
A区          风景优美, 景色宜人。
-----周边景区-----
周边景区数目: 3
A区->C区    700m
A区->E区    1000m
A区->F区    600m

=====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
请输入操作编号 (0~5) :
```

(6) 旅游景点导航

```
D:\学习文档\算法设计与分析\数据结构与算法综合实验\GraphCPro\Debug\GraphCPro.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
请输入操作编号 (0~5) :3
=====旅游景点导航=====
0-A区
1-B区
2-C区
3-D区
4-E区
5-F区
6-G区
请输入想要起始点编号: 3
供参考路线:
路线1: D区->C区->A区->E区->F区->G区->B区
路线2: D区->C区->B区->G区->F区->A区->E区
路线3: D区->C区->B区->G区->F区->E区->A区
路线4: D区->E区->A区->C区->B区->G区->F区
路线5: D区->E区->A区->F区->G区->B区->C区
路线6: D区->E区->F区->A区->C区->B区->G区
路线7: D区->E区->F区->G区->B区->C区->A区
路线8: D区->G区->B区->C区->A区->E区->F区
路线9: D区->G区->B区->C区->A区->F区->E区
路线10: D区->G区->F区->E区->A区->C区->B区
=====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
请输入操作编号 (0~5) :_
```

(7) 搜索最短路径

```
D:\学习文档\算法设计与分析\数据结构与算法综合实验\GraphCPro\Debug\GraphCPro.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
请输入操作编号 (0~5) :4
=====搜索最短路径=====
0-A区
1-B区
2-C区
3-D区
4-E区
5-F区
6-G区
请输入起点的编号: 2
请输入终点的编号: 6
最短路径为: C区->D区->G区
最短距离为: 800

=====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
请输入操作编号 (0~5) :_
```

### (8) 铺设电路规划

```
D:\学习文档\算法设计与分析\数据结构与算法综合实验\GraphCPro\Debug\GraphCPro.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
请输入操作编号 (0~5): 5
=====铺设电路规划=====
在以下两个景点之间铺设电路:
A区-F区    600m
F区-G区    500m
G区-D区    400m
D区-E区    300m
D区-C区    400m
C区-B区    1000m
铺设电路的总长度是: 3200m
=====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
请输入操作编号 (0~5):
```

### 3. 实验过程发现的问题

(1) 在进行深度优先算法设计时, 需要递归调用 DFS 函数, 递归结束时, 要更改顶点访问状态, 才可得到所有遍历的路线顶点, 并使用链表保存所有路径;

(2) 在进行 Dijkstra 算法与 Prim 算法的编写后, 我发现二者有如下区别:

①Prim 是计算最小生成树的算法, 比如为 N 个村庄修路, 怎么修花销最少; Dijkstra 是计算最短路径的算法, 比如从 a 村庄走到其他任意村庄的距离;

②Prim 算法中有一个统计总 len 的变量, 每次都要把到下一点的距离加到 len 中; Dijkstra 算法中却没有, 只需要把到下一点的距离加到 dist[] 数组中即可;

③Prim 算法的更新操作更新的 dist[] 是已访问集合到未访问集合中各点的距离; Dijkstra 算法的更新操作更新的 dist[] 是源点到未访问集合中各点的距离。



### 第三部分 实验小结、建议及体会

本次实验主要围绕图与景区信息管理系统实践展开，实验分为三步进行，难度适中，旨在掌握图的定义和图的存储结构、图的创建方法、遍历算法，并理解迪杰斯特拉（Dijkstra）算法、最小生成树的概念和普利姆（Prim）算法，同时掌握文件操作。实验需使用 C++ 语言，利用图的数据结构，开发景区信息管理系统。图是一种重要的数据结构，是表示物件与物件之间的关系的数学对象，是图论的基本研究对象。一个不带权图中若两点不相邻，邻接矩阵相应位置为 0，对带权图(网)，相应位置为 $\infty$ 。

在实验过程中，我成功创建了图的邻接矩阵存储结构，并利用深度优先深度算法、Dijkstra 算法与 Prim 算法分别实现了旅游景点导航、搜索最短路径与铺设电路规划三个功能，经验证，所有功能均能正常运行。实验也不是一帆风顺，我也遇到了许多问题，如在进行深度优先算法设计时，由于对算法的理解不熟悉，不知算法的具体实现步骤与合适的存储结构等，最终都通过查阅资料得以解决。

通过这次实验，我对图的定义和图的存储结构、图的创建方法、遍历算法有了更加深入的了解，并对迪杰斯特拉（Dijkstra）算法、最小生成树的概念和普利姆（Prim）算法有了更多的认识，收获颇丰。

教师签字\_\_\_\_\_