任务 3 操作步骤

1.1 实验目标

1、掌握图的两种遍历方法和应用。
2、基于已经实现“景区信息管理系统”功能，采用迭代开发，使用 C++语言和深度优先搜索算法实现“旅游景点导航”功能开发。

1.2 实验任务

在“创建图和查询景点”工程的基础上，为景区信息管理系统增加“旅游景点导航”的功能。

(1) 输入

起始景点的编号。

(2) 处理

从起始景点开始，遍历景区所有的景点，记录所有无重复的路径。

(3) 输出

将查询到的旅游路线显示到控制台中。

图 Error! No text of specified style in document.-1 图的遍历

1.3 分析和设计

在“创建图和查询景点” 工程的基础上进行迭代开发。

1、 算法设计

旅游景点导航实际上就是从某一顶点出发，搜索出一条能够游览完所有景点的路径，其中搜索的过程就是图的遍历。

图的遍历方式有两种为深度优先搜索和广度优先搜索，这里采用深度优先搜索的方式遍历图。

深度优先搜索(Depth First Search)

从顶点 v0 出发：

(1) 访问 v0;

(2) 依次访问 v0 的邻接点 v1，v1 的邻接点 v2，v2 的邻接点 v3……直到所有顶点都被访问过。

2、旅游景点导航

从一个景点出发游览整个景区时，路线可能不止一条，因此需要在深度优先搜索（DFS）的算法上进行改进，用来得到多条路线。

改进思路：

(1) 定义数组 bool aVisited [20]保存图中顶点的访问状态。

(2) 定义整数 int nIndex 记录图的访问深度。

(3) 若所有顶点都被访问过，就保存一条路径。

(4) 访问结束后，将顶点的访问状态改为未访问，访问深度减 1，以便于生成其他访问路线。

定义链表 PathList 来保存所有路径。

```
Typedef struct Path
{
      int vexs[20];  //保存一条路径
      Path *next;  //下一条路径
}*PathList;
```

根据改进后的深度优先搜索算法，从图中的顶点 2 开始遍历，得到 5 种遍历结果

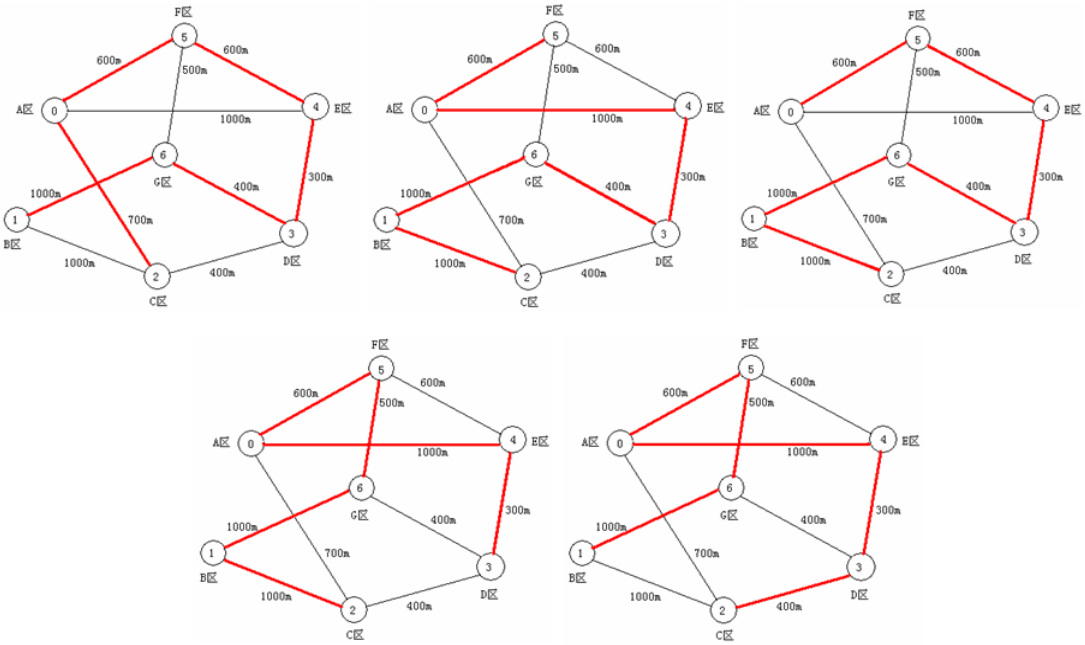

图 Error! No text of specified style in document.-2 5 种遍历结果

3、算法的设计

(1) Graph.cpp 文件

增加函数：

1)void DFS(int nVex, bool bVisted[], int &nIndex, PathList &pList)

输入参数：int nVex，顶点编号。

输入参数：bVisted[]，bool 类型的数组，用来记录某个顶点是否被遍历过。

输入参数：int &nIndex，记录遍历的深度。

输出参数：PathList &pList，遍历得到的结果。

功能：使用深度优先搜索算法遍历图。

2)void DFSTraverse(int nVex, PathList &pList)。

输入参数：int nVex，顶点编号。

输出参数：PathList &pList，遍历得到的结果。

功能：通过调用 DFS()函数，得到深度优先搜索遍历结果。

(2) Tourism.cpp 文件

增加函数：

void TravelPath()

输入：void

输出：void

功能：通过调用 DFSTraverse()函数，实现旅游景点导航功能，将查询到的景点导航路线显示在界面上。

1.4 编码实现

为景区信息管理系统增加旅游景点导航功能。

(1) 使用深度优先搜索算法实现图的遍历，得到一条导航路线。

(2) 改进深度优先搜索算法，用来得到多条导航路线。

在实现过程中采用迭代思路，具体实现步骤如下：

步骤一：导入工程。

步骤二：遍历景区景点图(一条路线)。

步骤三：优化遍历算法(多条路线)。

1.4.1 导入工程

在实现了创建景区景点图和查询景点信息的功能之后，在原有工程的基础上新增旅游景点导航功能。

(1) 打开 Microsoft Visual Studio 2010 开发工具。

(2) 导入“GraphCPro”工程。

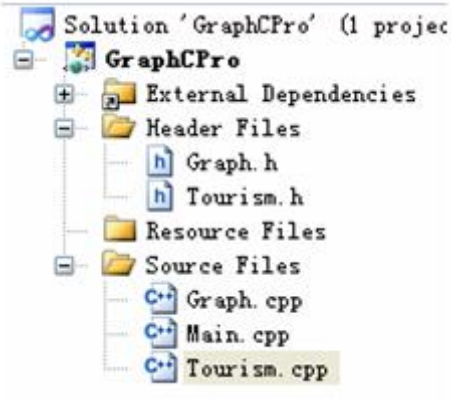

图 Error! No text of specified style in document.**-3 Tourism.cpp**

1.4.2 遍历景区景点图(一条路线)

采用深度优先搜索方式遍历图。对于同一顶点上的多个邻接点，按照它们的存储顺序来访问。

(1) 在 Graph.cpp 文件中添加 **DFS()**方法，用来实现图的深度优先搜索遍历。

```
void DFS(int nVex,bool bVisited[],int &nIndex,PathList &pList)
{
        aVisited[nVex] = true;   // 改为已访问
        pList->vexs[nIdex++] = nVex;   //访问顶点 nVex
        for(…)   //搜索 v 的所有邻接点
        {
                if(i 是 nVex 的邻接点&&！bVisited[i])
                {
                        DFS(I,aVisited,nIndex,pList);   //递归调用 DFS
                }
```

```
            }
}
```

(2) 在 Graph.cpp 文件中添加 DFSTraverse()方法，通过调用 DFS()函数，进行图的深度优先遍历。

```
void Cgraph::DFSTraverse(int nVex,PathList *pList)
{
        int nIndex = 0;
        bool aVisited[MAX_VERTEX_NUM] = {false};
        DFS(nVex,aVisited,nIndex,pList);
}
```

(3) 在 Tourism.cpp 文件中添加 TravelPath()方法，通过调用 DFSTraverse()函数，得到景点导航路线，并显示在界面上。

```
void CTourism::TravelPath(void)
{
      //输入景点编号
      //遍历景区景点图
      //输出遍历结果
}
```

(4) 编译运行

在 main()函数 case 3 语句中调用 TravelPath()函数，进入旅游景点导航功能。

编译运行，得到结果如下。

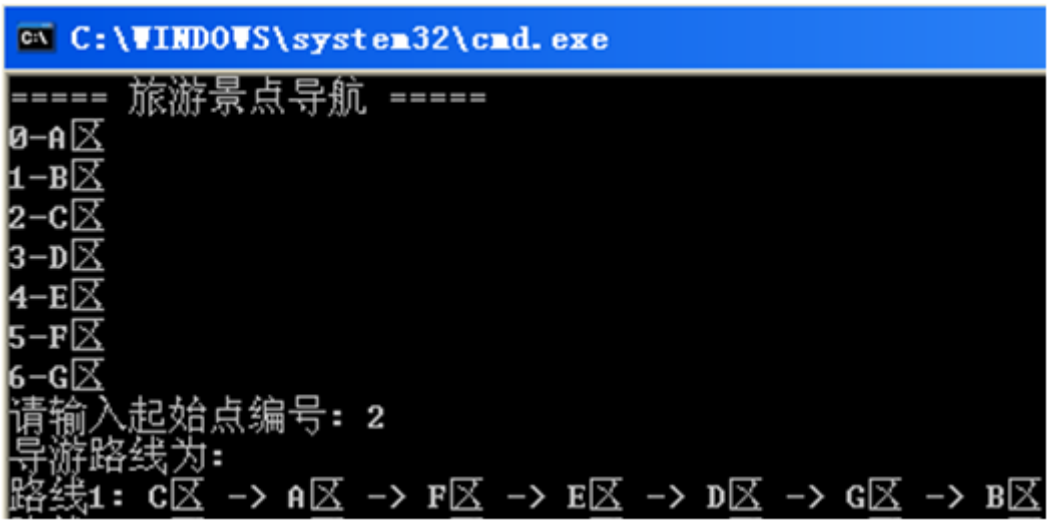

图 Error! No text of specified style in document.**-4 导航一条路线**

1.4.3 优化遍历算法(多条路线)

改进 DFS 算法：

(1) 若所有顶点都被访问过，就保存一条路径。

(2) 访问结束后，将顶点的访问状态改为未访问，访问深度减 1。

```
void DFS(int v,bool bVisited[],int aPth[],int index)
{
      bVisited[v] = true;   // 改为已访问
      aPath[index++] = v;   // 访问顶点 v
      if(所有的顶点都被访问过)
      {
      // 1、保存一条路径
      }
      else
      {
            for(…;…;…;)   // 搜素 v 的所有邻接点
            {
                  if(w 是 v 的邻接点  &&  ！ bVisited[w])
                  {
                        DFS(w,bVisited,aPath,index); // 递归调用 DFS
                        bVisited[w] = false;   // 2、改为未访问
                        index--;   // 索引值减 1
                        }
                  }
             }
}
```

1.5 调试和运行

(1) 按 Ctrl+F5，编译运行程序。

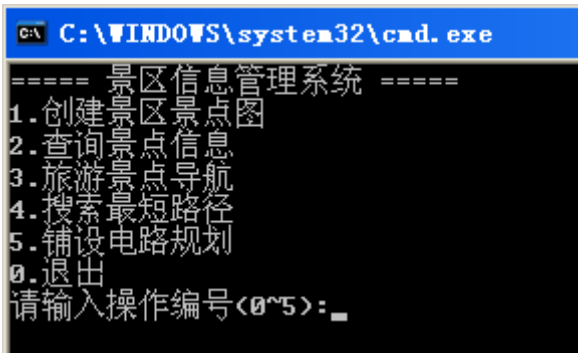

图 Error! No text of specified style in document.**-5 主菜单**

(2) 在主菜单中输入 3，则进行对应“旅游景点导航”操作。

在 main()函数 case 3 语句中调用 TravelPath ()函数，进入旅游景点导航功能。编译运行，得到结果：

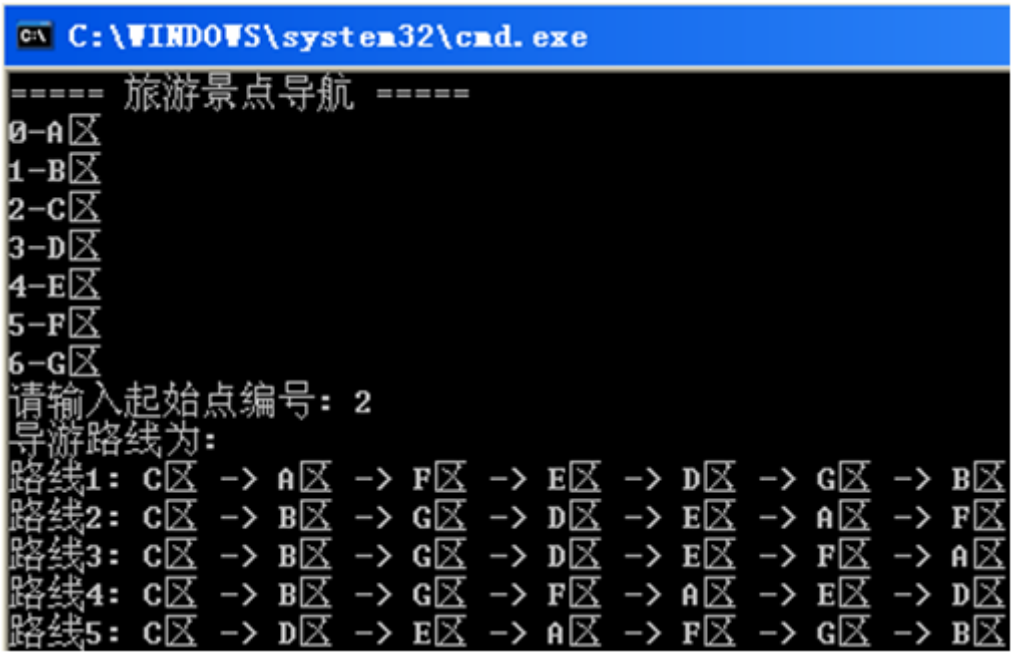

图 Error! No text of specified style in document.**-6 旅游景点导航**