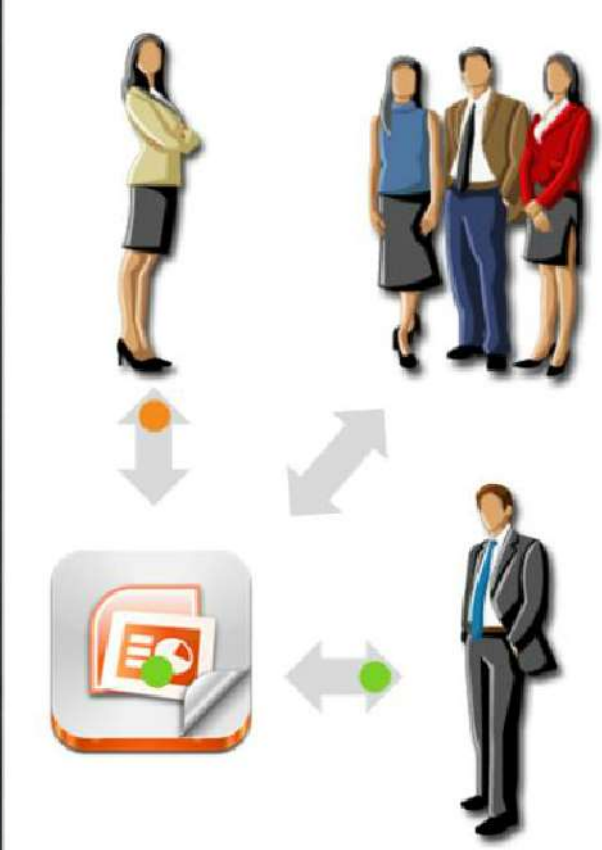
景区信息管理系统

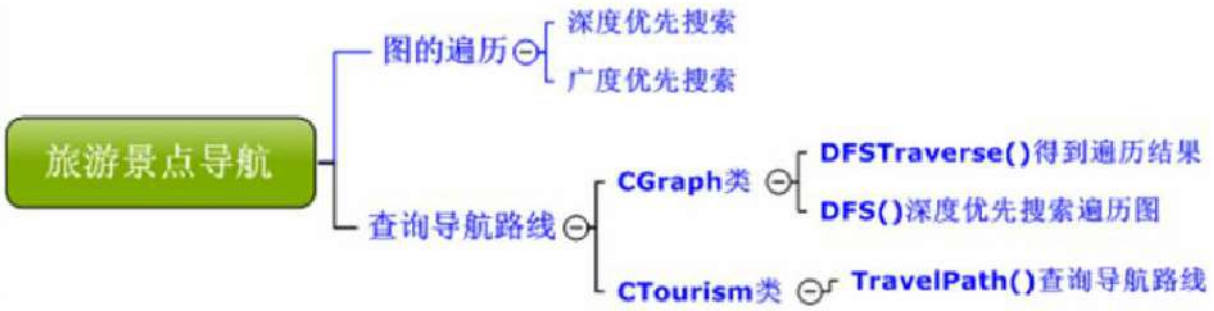


景区信息管理系统

旅游景点导航

主要内容

- 功能简介
- 设计思路
- 技术分析
- 实现
- 小结


旅游景点导航
图的遍历
深度优先搜索
广度优先搜索
查询导航路线
CGraph类
DFSTraverse()得到遍历结果
DFS()深度优先搜索遍历图
CTourism类
TravelPath()查询导航路线

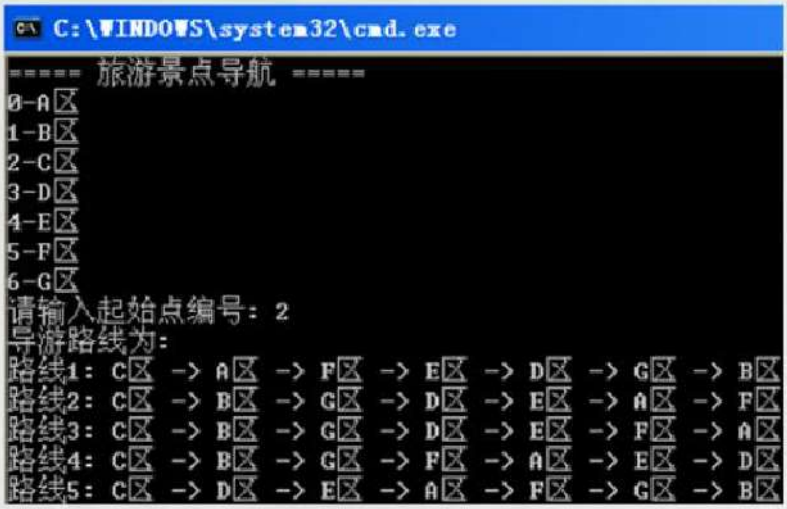
功能简介

在“创建图和查询景点”的基础上，开发旅游景点导航功能。

输入：起始景点的编号

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。

输出：将查询到的旅游路线显示到控制台中。



设计思路

在“创建图和查询景点”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：

(1) 深度优先搜索

(2) 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。

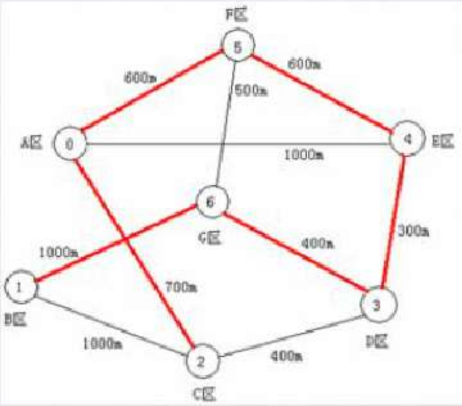
4、访问结束后，将顶点的访问状态改为未访问，访问深度减1，以便于生成其他访问路线。

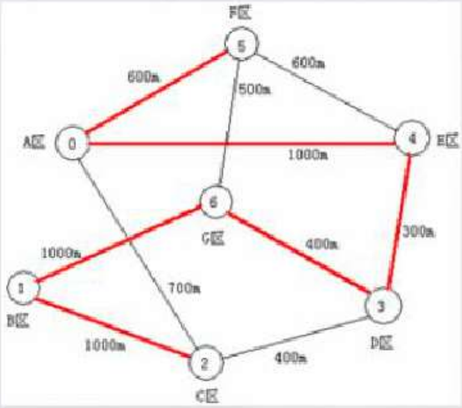
定义链表PathList来保存所有路径。

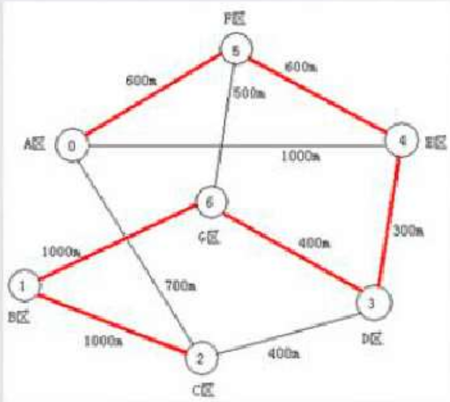
```
typedef struct Path
{
        int vexs[20]; // 保存一条路径
        Path *next;// 下一条路径
} * PathList;
```

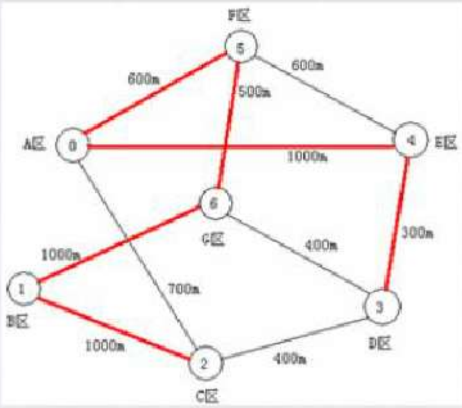
设计思路

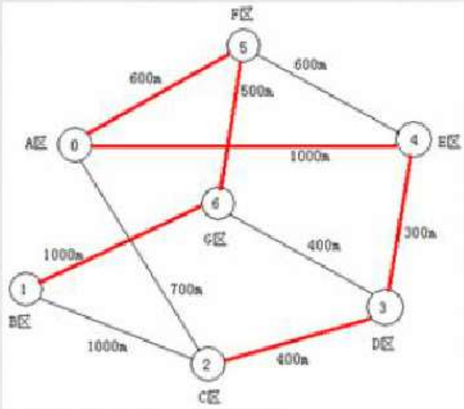
根据改进后的深度优先搜索算法，从图中的顶点2开始遍历，得到5种遍历结果：











3、类的设计

CGraph类

增加成员函数：

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

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

CTourism类

增加成员函数：

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。

图的遍历：

（1）深度优先搜索遍历

（2）广度优先搜索遍历

1、如何通过图的遍历得到旅游景点导航路线？

2、如何得到多条导航路线？

编程实现

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。

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)

步骤四：编译和运行。

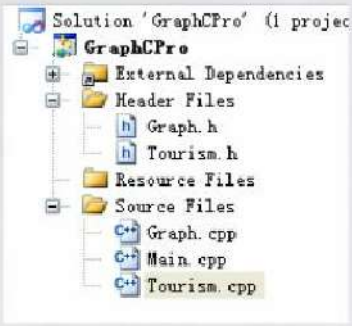
实现：导入工程

步骤一：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。



实现：遍历景区景点图(一条路线)

步骤二：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CGraph类中添加**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[nIndex++] = 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
                }
        }
 }
```

实现：遍历景区景点图(一条路线)

(2) 在CGraph类中添加**DFSTraverse()**方法，通过调用DFS()函数，进行图的深度优先遍历。

```
void CGraph::DFSTravrse(int nVex, PathList *pList)
{
        int nIndex = 0;
        bool aVisted[MAX_VERTEX_NUM] = {false};
        DFS(nVex, aVisted, nIndex, pList);
}
```

实现：遍历景区景点图(一条路线)

(3) 在CTourism类中添加**TravelPath()**方法，通过调用m_Graph.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
}
```

实现：遍历景区景点图(一条路线)

(4) 编译和运行

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：

实现：优化遍历算法(多条路线)

步骤三：优化遍历算法(多条路线)

改进DFS算法：

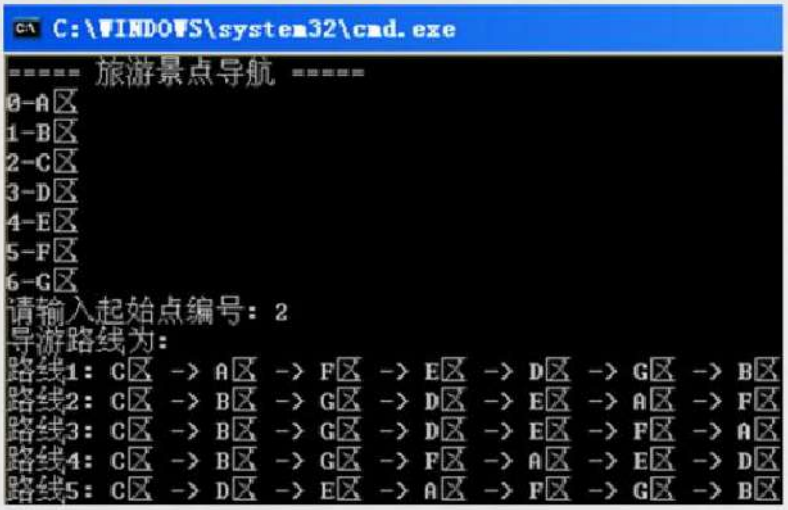
(1) 若所有顶点都被访问过，就保存一条路径。

(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 aPath[], 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(...;...;...)// 搜索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
                        }
                }
        }
 }
```

实现：编译和运行

步骤四：编译和运行



小结

1、图的遍历，分为广度优先搜索(BFS)、深度优先搜索(DFS)两种算法。
2、递归调用DFS函数，递归结束时，更改顶点访问状态，即可得到所有遍历的路线顶点。
3、使用链表保存所有路径。