**判断无向图中是否存在环(附代码)**

**代码正确性分析：**

利用DFS判断无向图是否有环的原理：在深度优先遍历中，若一个顶点指向一个已经被访问过且不是父节点的点则该无向图存在环。

**算法过程：**将已经被访问过的点设置状态为visited[i]=1,并且将与之相邻的点的父节点设置为该点，当访问下一个点时，如果该点指向一个已经被访问过且不是该节点父节点的点则存在环，否则不存在环；

**证明：**假设存在一个节点指向一个已经被访问过且不是该节点的父节点的节点，但不存在环。设该节点为**x，**被指向的已经被访问的节点为**y，**则visited[**y**]=1，该节点的父节点为**z,**则prenode[**x**]=**z，**且**y!=z**。既然y节点已经被访问过，则必定存在一条从访问源点到y点的路径，又因为**x**也被访问过，则必定存在一条从访问源点到**x**点的路径且该路径必定包含节点**z**，由此知道必定存在一条由**x**到**y**且经过点**z**的路径，因为现在又有一条从**x**指向**y**的边且不经过点**z**，则由**x**到**y**必定存在两条边也就是构成了一个环。所以该假设是错误的。

**时间复杂度分析**：第一个for循环将visited[]和prenode[]数组初始化，时间复杂度为O(n),n为节点的最大数量。在while和for嵌套的循环中，外层遍历的时间是O(n),内层遍历时间是每个相邻节点的数量。对于邻接数组的描述方式，整个循环遍历的时间是O(n+e)。综上整个过程时间复杂度为O(n+e);

**空间复杂度分析：**在整个过程中，用到了一个存储所有节点的访问数组、存储所有节点的父节点的数组，和一个存储度为节点的度的栈。三个线性表的最大大小是存储所有的节点，所以空间复杂度为O(n),这里n是节点数量。

**代码如下**：

bool circle()

{

int \*prenode = new int[MaxNode];

int visited[MaxNode], s\_top;

for (int i = 0; i < MaxNode; i++)

{

prenode[i] = -1;

visited[i] = 0;

}

stack<int>s;

s.push(0);

while (!s.empty())

{

s\_top = s.top();

visited[s\_top] = 1;

s.pop();

for (int i = 0; i < a[s\_top].size(); i++)

{

if (!visited[a[s\_top][i].next])

{

visited[a[s\_top][i].next] = 1;

s.push(a[s\_top][i].next);

prenode[a[s\_top][i].next] = s\_top;

}

else if (a[s\_top][i].next != prenode[s\_top])

return true;

}

}

return false;

}