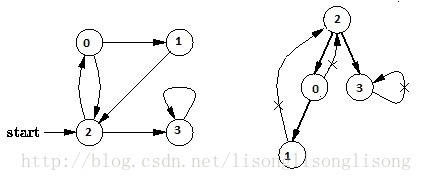
**深度优先搜索**

**一、深度优先搜索（DFS）的算法思想**

深度优先搜索算法所遵循的搜索策略是尽可能“深”地搜索一个图。它的基本思想就是：首先访问图中某一起始顶点v，然后由v出发，访问与v邻接且未被访问的任一顶点w1，再访问与w1邻接且未被访问的任一顶点w2，……重复上述过程。当不能再继续向下访问时，依次退回到最近被访问的顶点，若它还有邻接顶点未被访问过，则从该点开始继续上述搜索过程，直到图中所有顶点均被访问过为止。



如上图所示，从顶点2开始深度优先遍历图，结果为：2，0，1，3。

## **二、DFS算法实现**

和[广度优先搜索](http://blog.csdn.net/lisonglisonglisong/article/details/36704573)一样，为了防止顶点被多次访问，需要使用一个访问标记数组来标记顶点是否已经被访问过。

这里使用邻接表表示图。对于一个****有向图****，****假设从给定顶点可以访问到图的所有其他顶点****，则DFS递归算法的C++代码实现：

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

#define MAX 105

typedef struct ARCNode //边节点

{

int adjvex; //这条边指向的顶点的位置

struct ARCNode \* nextarc; //下一个边节点

}ARCNode;

typedef struct Vnode //顶点信息

{

char data; //存放着数据

ARCNode \*first; //与它连接的第一条边的信息

}Vnode,Adjlist[MAX]; //一个顶点 很多个顶点

typedef struct

{

Adjlist vertices; //这是一个顶点数组

int vexnum,arcnum; //顶点的数目，边的数目

}ALGraph;

int LocateVex(ALGraph T,char data)

{

for(int i=0;i<T.vexnum;i++)

{ if(T.vertices[i].data==data)

return i; //找到了就返回这个顶点数据的索引

}

return -1; //-1表示没有找到

}

void CreateUDG(ALGraph &G)

{

cin>>G.vexnum>>G.arcnum; //输入总顶点数目，总边数目

for(int i=0;i<G.vexnum;i++)

{

cin>>G.vertices[i].data;

G.vertices[i].first=NULL; //一开始指向空地方

}

for(int k=0;k<G.arcnum;k++)

{

char a,b; int i,j; cin>>a>>b; //输入顶点

i=LocateVex(G,a);

j=LocateVex(G,b);

ARCNode \* p1=new ARCNode;

ARCNode \* p2=new ARCNode;

p1->adjvex=j; p1->nextarc=G.vertices[i].first; //头插法

G.vertices[i].first=p1; //图是无向的，因此是相互的

p2->adjvex=i;

p2->nextarc=G.vertices[j].first;

G.vertices[j].first=p2;

}

} //这个函数的作用是返回第一个邻接边顶点在数组中的索引

int FirstAdjvex(ALGraph &T,int v)

{

return T.vertices[v].first->adjvex;

} //返回邻接边w的下一个邻接边的索引，不存在的话则返回-1，作为标志

int NextAdjVex(ALGraph &T,int v,int w)

{

ARCNode \*p;

p=T.vertices[v].first;

while(p->adjvex!=w)

{

p=p->nextarc;

}

p=p->nextarc;

if(p!=NULL) return p->adjvex;

else return -1;

} //是图的变量

//无论是哪一种方法，都避免不了设置访问标志，即需要一个数组来存储访问记录

bool visited[MAX]; //顶点遍历标记

void DFS(ALGraph &T,int v) //从第v个顶点出发递归地深度优先遍历图 连通图

{

cout<<T.vertices[v].data<<" ";

visited[v]=true;

for(int w=FirstAdjvex(T,v);w>=0;w=NextAdjVex(T,v,w)) //FirstAdjvex检查所有的邻节点

{

if(!visited[w]) DFS(T,w);

}

}

void DFSTraverse(ALGraph &T) //深度优先遍历非连通图

{

for(int i=0;i<T.vexnum;i++)

visited[i]=false;

for(int i=0;i<T.vexnum;i++)

if(!visited[i]) DFS(T,i);

}

int main()

{

ALGraph T;

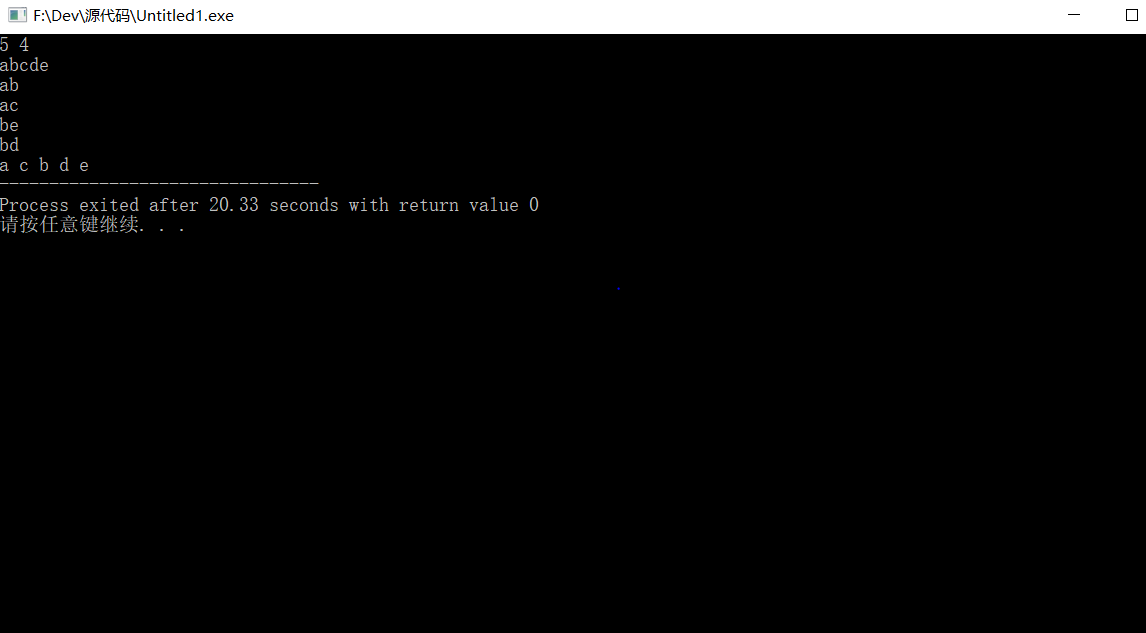
CreateUDG(T);

DFS(T,0); //遍历连通图结果，需要指出从哪个顶点开始遍历，这里注意，是顶点的索引 //DFSTraverse(T); //遍历非连通图连通图

return 0;

}

运行结果：



## **三、DFS算法性能分析**

1 . ****空间复杂度****

DFS算法是一个递归算法，需要借助一个递归工作栈，故它的空间复杂度为O(|V|)。

2 . ****时间复杂度****

当以****邻接表****存储时，时间复杂度为****O(|V|+|E|)****。

当以****邻接矩阵****存储时，时间复杂度为****O(|V|^2)****。