**深度优先遍历算法**(1)邻接矩阵的深度优先遍历算法：  
**void AdjMWGraph::Depth(int v,int visited[])  
 { cout<<""\n 顶点"<<v+1"权值："<<Vertices[v];  
   visited[v]=1;  
 for(int col=0;col<numV;col++)  
  { if(Edge[v][col]==0||Edge[v][col]==MaxWeight)continue;  
    if(!visited[col])Delpth(col,visited);  
  }  
}**  
　　故用邻接矩阵表示图时，搜索一个顶点的所有邻接点需花费O(n)时间，则从n个顶点出发搜索的时间应为O(n2)，即DFS算法的时间复杂度是（n2）。  
  
(2)邻接链表的深度优先遍历算法：  
**Void AdjTWGraph::DepthFirst(int v, int visited[])  
 { int vj; Edge \*p;  
  cout<<Vertices[v].data<<" ";  
  visited[v]=1;  
  p=Vertices[v].adj;     //取v的邻接边结点  
  while(p!=NULL)  
  { vj=p->dest;       //取v的邻接点编号  
   if(visited[vj]==0)DepthFirst(vj, visited);  
   p=p->next;       //取下一个邻接边结点  
  }  
 }**  
　　使用邻接链表来表示图时，其 DFS 算法的时间复杂度为 O(n+e) ，此处 e 为无向图中边的数目或有向图中弧的数目。

**广度优先遍历算法**(1)邻接矩阵的广度优先遍历算法：  
**void AdjMWGraph::Depth(int v,int visited[])  
 {  sqQueue<  int>q;        //定义队列queue  
   q.EnQueue(v);        //顶点v入队列  
   while(!q.IsEmpty())        //当队列非空时循环  
       {  v=q.DeQueue();        //出队列得到队头顶点u  
          cout<<endl<<"顶点"<<v+1<<"权值："<<Vertices[col];          //访问顶点v  
          visited[col]=1;//标记顶点v已访问  
          for(int col=0;col<numV;col++)  
          if(Edge[v][col]>0;&&Edge[v][col]<MaxWeight&&visite[col]==0)  
          q.EnQueue(col);  
   }  
   cout<<endl<<"end!"<<"endl;  
 };**  
　　邻接矩阵表示图时，搜索一个顶点的所有邻接点需花费O(n)时间，则从n个顶点出发搜索的时间应为O(n2)，即BFS算法的时间复杂度是（n2）。  
  
(2)邻接链表的深度优先遍历算法：  
**void AdjTWGraph::BroadFirst(int v, int visited[])  
 {   int vj;   Edge \*p;   SqQueue <int> Q; Q.EnQueue(v);  
    while(!Q.IsEmpty())               //队列不空，循环  
      {   v=Q.DeQueue();         //出队并且访问顶点v  
     cout<<Vertices[v].data<<" ";   visited[v]=1;  
      p=Vertices[v].adj;             //取v的邻接边结点  
    while(p!=NULL)   { vj=p->dest;     //取v的邻接点编号vj  
     if(visited[vj]==0) Q.EnQueue(vj);             //vj未访问，入队  
       p=p->next;             //取下一个邻接边结点  
    }  
   }  
  }**  
　　使用邻接链表来表示图时，其 BFS 算法的时间复杂度为 O(n+e) ，此处 e 为无向图中边的数目或有向图中弧的数目。

附代码：  
#include<iostream>  
#include<queue>  
#include<stack>  
#include<stdlib.h>  
#define MAX 100  
using namespace std;  
   
typedef struct  
{  
    int edges[MAX][MAX];    //邻接矩阵  
    int n;                  //顶点数  
    int e;                  //边数  
}MGraph;  
   
bool visited[MAX];          //标记顶点是否被访问过  
   
void creatMGraph(MGraph &G)    //用引用作参数  
{  
    int i,j;  
    int s,t;                 //存储顶点编号  
    int v;                   //存储边的权值  
    for(i=0;i<G.n;i++)       //初始化  
    {  
        for(j=0;j<G.n;j++)  
        {  
            G.edges[i][j]=0;  
        }  
        visited[i]=false;  
    }  
    for(i=0;i<G.e;i++)      //对矩阵相邻的边赋权值  
    {  
        scanf("%d %d %d",&s,&t,&v);   //输入边的顶点编号以及权值  
        G.edges[s][t]=v;  
    }  
}  
   
void DFS(MGraph G,int v)      //深度优先搜索  
{  
    int i;  
    printf("%d ",v);          //访问结点v  
    visited[v]=true;  
    for(i=0;i<G.n;i++)       //访问与v相邻的未被访问过的结点  
    {  
        if(G.edges[v][i]!=0&&visited[i]==false)  
        {  
            DFS(G,i);  
        }  
    }  
}  
   
void DFS1(MGraph G,int v)   //非递归实现，注意理解！！  
{  
    stack<int> s;  
    printf("%d ",v);        //访问初始结点  
    visited[v]=true;  
    s.push(v);              //入栈  
    while(!s.empty())  
    {  
        int i,j;  
        i=s.top();          //取栈顶顶点  
        for(j=0;j<G.n;j++)  //访问与顶点i相邻的顶点  
        {  
            if(G.edges[i][j]!=0&&visited[j]==false)  
            {  
                printf("%d ",j);     //访问  
                visited[j]=true;  
                s.push(j);           //访问完后入栈  
                break;               //找到一个相邻未访问的顶点，访问之后则跳出循环  
            }  
        }  
        if(j==G.n)                   //如果与i相邻的顶点都被访问过，则将顶点i出栈  
            s.pop();  
    }  
}  
   
void BFS(MGraph G,int v)      //广度优先搜索  
{  
    queue<int> Q;             //STL模板中的queue  
    printf("%d ",v);  
    visited[v]=true;  
    Q.push(v);  
    while(!Q.empty())   
    {  
        int i,j;  
        i=Q.front();         //取队首顶点  
        Q.pop();  
        for(j=0;j<G.n;j++)   //广度遍历  
        {  
            if(G.edges[i][j]!=0&&visited[j]==false)  
            {  
                printf("%d ",j);  
                visited[j]=true;  
                Q.push(j);  
            }  
        }  
    }  
}  
   
int main(void)  
{  
    int n,e;    //建立的图的顶点数和边数  
    while(scanf("%d %d",&n,&e)==2&&n>0)  
    {  
        MGraph G;  
        G.n=n;  
        G.e=e;  
        creatMGraph(G);  
        DFS(G,0);  
        printf("\n");  
    //  DFS1(G,0);  
    //  printf("\n");  
    //  BFS(G,0);  
    //  printf("\n");  
    }  
    return 0;  
}