可以用来检测连通图，虽然有时候非连通图也可以全遍历出来（就要多遍历多个点）
这个连通图完全适用，非联通图可能遍历不全

C

```c
#include<stdio.h>
#include <malloc.h>

/** 以邻接表形式创建有向图为例，DFS（深度优先搜索）：这里是邻接表构建的有向图为例，实际上邻接表构建的无向图也是完全适用的
 *
 * 0 | A -> 1
 * 1 | B -> 2 -> 3 -> 4
 * 2 | C
 * 3 | D
 * 4 | E -> 5
 * 5 | F -> 0
 * */

#define MaxVertex 6

typedef char E;

typedef struct Node{ //结点与头结点分别来创建，普通结点记录邻接顶点信息
    int nextVertex; //下一个顶点的序号(终点)
    struct Node * next;
} * Node;

struct HeadNode{ //头结点记录元素(A,B,C,D)
    E element; //顶点元素
    struct Node * next;
};

typedef struct AdjacencyGraph{
    int vertexCount; //顶点数
    int edgeCount; //边数
    struct HeadNode vertex[MaxVertex]; //头结点数组
} * Graph;

Graph Create(){
    Graph graph = malloc(sizeof(struct AdjacencyGraph));
    graph->vertexCount = graph->edgeCount = 0;
    return graph;
}

void AddVertex(Graph graph, E element){//添加顶点
    if(graph->vertexCount >= MaxVertex) return; //还是先判断满了吗
    graph->vertex[graph->vertexCount].element = element; //没满就把元素给插入到头结点数组里面去
    graph->vertex[graph->vertexCount++].next = NULL; //并将next置空,之后顶点数加1
}
```

```
void AddEdge(Graph graph, int start, int end){//添加边
    Node node = graph->vertex[start].next;//先取出起点后面的那个结点
    Node newNode = malloc(sizeof(struct Node));//创建一个新结点
    newNode->next = NULL;
    newNode->nextVertex = end;// 将终点信息储存在newNode里面

    if(!node){ //如果头节点的下一个为空，说明没有结点了，就直接连上去
        graph->vertex[start].next = newNode;
    }else{//否则说明当前顶点已经连接了至少一个其他顶点了，有可能会出现已经连接过的情况，所以说要特别处理一下
        do{
            if(node->nextVertex == end)//如果要连的边已经连上了，直接结束就可以了（先释放内存）
                return free(newNode);
            if(node->next) node = node->next;//否则继续往后面遍历
            else break; //如果没有下一个了，那就找到最后一个结点了，直接结束
        }while(1);
        node->next = newNode;//把新结点插入到最后一个结点里面
    }
    graph->edgeCount++;//边数加1
}

void PrintGraph(Graph graph){//打印邻接表
    for (int i = 0; i < graph->vertexCount; ++i) {
        printf("%d | %c", i, graph->vertex[i].element);
        Node node = graph->vertex[i].next;
        while (node) {
            printf(" -> %d", node->nextVertex);
            node = node->next;
        }
        putchar('\n');
    }
}

/**
* @param startVertex 起点顶点下标
* @param targetVertex 目标顶点下标
* @param visited 已到达过的顶点数组
* @Return 找到为1，没有找到为0
**/
_Bool Dfs(Graph graph, int startVertex, int targetVertex, int * visited){//找目标顶点的DFS
    visited[startVertex] = 1;
    printf("%c -> ", graph->vertex[startVertex].element);
    if(startVertex == targetVertex) return 1;   //如果当前顶点就是要找的顶点，直接返回
    Node node = graph->vertex[startVertex].next;
    while (node) {
        if(!visited[node->nextVertex])
            if(Dfs(graph, node->nextVertex, targetVertex, visited))//如果查找成功，直接返回1，不用再看其他分支了
                return 1;
        node = node->next;
    }
    return 0;
}
```

```
void DFS(Graph graph, int startVertex, int * visited){//深度优先搜索，如果不是连通图，可能有些结点到不了
    visited[startVertex] = 1;   //走过之后一定记得mark一下
    printf("%c -> ", graph->vertex[startVertex].element);   //打印当前顶点值
    Node node = graph->vertex[startVertex].next;   //遍历当前顶点所有的分支
    while (node) {
        if(!visited[node->nextVertex])   //如果已经到过（有可能是走其他分支到过，或是回头路）那就不继续了
            DFS(graph, node->nextVertex,  visited);  //没到过就继续往下走，这里将startVertex设定为对于分支的下一个顶点，按照同样的方式去寻找
        node = node->next;
    }
}


int main(){
    Graph graph = Create();
    for(int c = 'A'; c <= 'F'; c++){
        AddVertex(graph,(char)c);
    }
    AddEdge(graph, 0, 1);   //A -> B
    AddEdge(graph, 1, 2);   //B -> C
    AddEdge(graph, 1, 3);   //B -> D
    AddEdge(graph, 1, 4);   //B -> E
    AddEdge(graph, 4, 5);   //E -> F
    AddEdge(graph, 5, 0);   //F -> A
    PrintGraph(graph);

    int arr1[graph->vertexCount];
    for(int i = 0; i < graph->vertexCount; i++){
        arr1[i] = 0;
    }
    int arr2[graph->vertexCount];
    for(int i = 0; i < graph->vertexCount; i++){
        arr2[i] = 0;
    }

    DFS(graph, 4, arr1);
    printf("\n");
    printf("\n%d",Dfs(graph, 0, 5, arr2));



}
```