DFS递归非递归

邻接矩阵版本背诵

```
void DFS(int v, MGraph G){
    cout<<"This is "<<v<<endl;
    marked[v] = 1;
    for(int i=0;i<G.n;i++){
        if(G.edge[v][i]!=0 && marked[i] == 0){//存在边，且未被访问过 进行递归搜索
            DFS(i,G);
        }
    }
}

//DFS非递归 统一入栈前访问该元素
void DFSUn(int v,MGraph G){
    cout<< "This is "<< v <<endl;
    marked[v] = 1;
    stack<int> s;
    s.push(v);
    while(!s.empty()){
        int temp = s.top();
        for(int i = 0;i < G.n; i++){
            if(G.edge[temp][i] !=0 && marked[i]==0){//v和xx之间有边，且未被访问过
                cout<< "This is "<< i <<endl;
                marked[i] = 1;
                s.push(i);
                temp = i; i = 0;//将i置为当前要进行深度优先遍历的点。同时循环重新从0开始
            }
        }
        s.pop();
    }
}
```

邻接表版背诵

```
//DFS递归版
void DFS(int v, ALGraph al){
    cout<< v <<endl;
    marked[v] = 1;
    ArcNode* arc = al.vertex[v].firstedge;
    while( arc != NULL ){//找一条未被访问过的边
        if(marked[arc->no] == 0){
            DFS(arc->no, al);
        }
        arc = arc->next;
    }
}
//DFS非递归版
```

```
void DFSUn(int v, ALGraph al){
    cout<<v<<endl;
    marked[v] = 1;
    ArcNode* arc = NULL;
    stack<int> s; s.push(v);
    while(!s.empty()){
        arc = al.vertex[s.top()].firstedge;//得到该点的第一个邻接边
        while(arc != NULL && marked[arc->no]==1){
            arc = arc->next;
        }
        if(arc==NULL){//不能在进行深度搜索了 找到头了
            s.pop();
        }else{
            cout<<arc->no<<endl;
            marked[arc->no] = 1;
            s.push(arc->no);
        }

    }
}
```

邻接矩阵完整可运行代码

```
#include<iostream>
#include<algorithm>
#include<stack>
using namespace std;

#define N 8
//访问数组
int marked[N] = {0};
//定义邻接矩阵
struct MGraph{
    int vertex[N];
    int edge[N][N];
    int e,n;
};


void init(MGraph &G,int vertex[],int edge[N][N]){
    for(int i=0;i<G.n;i++){
        G.vertex[i] = vertex[i];
    }
    for(int i=0;i<G.n;i++){
        for(int j=0;j<G.n;j++){
            G.edge[i][j] = edge[i][j];
        }
    }
}

void DFS(int v, MGraph G){
    cout<<"This is "<<v<<endl;
    marked[v] = 1;
    for(int i=0;i<G.n;i++){
        if(G.edge[v][i]!=0 && marked[i] == 0){//存在边，且未被访问过 进行递归搜索
            DFS(i,G);
        }
    }
}

//DFS非递归 统一入栈前访问该元素
void DFSUn(int v,MGraph G){
    cout<< "This is "<< v <<endl;
    marked[v] = 1;
    stack<int> s;
    s.push(v);
    while(!s.empty()){
        int temp = s.top();
        for(int i = 0;i < G.n; i++){
            if(G.edge[temp][i] !=0 && marked[i]==0){//v和xx之间有边，且未被访问过
                cout<< "This is "<< i <<endl;
                marked[i] = 1;
                s.push(i);
                temp = i; i = 0;//将i置为当前要进行深度优先遍历的点。同时循环重新从0开始
            }
        }
        s.pop();
    }
}

int main(){
    int vertex[N] = {0,1,2,3,4,5,6,7};
    int edge[N][N] = {
        {0,1,0,1,1,0,0,0},
        {1,0,0,1,0,0,1,0},
        {0,0,0,0,0,0,1,0},
        {1,1,0,0,0,0,0,0},
        {1,0,0,0,0,0,0,1},
        {0,0,0,0,0,0,0,1},
        {0,1,1,0,0,0,0,1},
        {0,0,0,0,1,1,1,0}
    };
    MGraph G;
    G.n = N;
    init(G,vertex,edge);
    //DFS(0,G);
    DFSUn(0,G);
    return 0;
}
```

邻接表完整可运行代码

```
#include<iostream>
#include<algorithm>
#include<stack>
using namespace std;
```

```
#define N 8
//访问数组
int marked[N] = {0};

//定义邻接边表
struct ArcNode{
    int no;//该边指向的顶点
    ArcNode *next;
};
//定义顶点表
struct VNode{
    int no;//顶点编号
    ArcNode *firstedge;
};
//复合结构体
struct ALGraph{
    VNode vertex[N];
    int e,n;
};

void init(ALGraph &al, int vertex[], int edge[N][N]){
    for(int i = 0; i < al.n; i++){
        al.vertex[i].no = vertex[i];//声明好邻接点
        al.vertex[i].firstedge = NULL;
    }
    //给边赋值
    for(int i=0;i<al.n;i++){
        ArcNode* temp = NULL;
        for(int j=0;j<al.n;j++){
            if(al.vertex[i].firstedge == NULL && edge[i][j] != 0){
                temp = (ArcNode*)malloc(sizeof(ArcNode));
                temp->no = j; temp->next = NULL;
                al.vertex[i].firstedge = temp;
            }else if(temp!=NULL && edge[i][j] != 0){
                ArcNode* arc = (ArcNode*)malloc(sizeof(ArcNode));
                arc->no = j; arc->next = NULL;
                temp->next = arc;
                temp = temp->next;
            }
        }
    }
    //测试邻接表是否建立好了
    /*
    for(int i=0;i<al.n;i++){
        ArcNode* node = al.vertex[i].firstedge;
        while(node!=NULL){
            cout<<node->no;
            node = node->next;
        }
        cout<<""<<endl;
    }*/
}

//DFS递归版
void DFS(int v, ALGraph al){
    cout<< v <<endl;
    marked[v] = 1;
    ArcNode* arc = al.vertex[v].firstedge;
    while( arc != NULL ){//找一条未被访问过的边
        if(marked[arc->no] == 0){
            DFS(arc->no, al);
        }
        arc = arc->next;
    }
}

//DFS非递归版
void DFSUn(int v, ALGraph al){
    cout<<v<<endl;
    marked[v] = 1;
    ArcNode* arc = NULL;
    stack<int> s; s.push(v);
    while(!s.empty()){
        arc = al.vertex[s.top()].firstedge;//得到该点的第一个邻接边
        while(arc != NULL && marked[arc->no]==1){
            arc = arc->next;
        }
        if(arc==NULL){//不能在进行深度搜索了 找到头了
            s.pop();
        }else{
            cout<<arc->no<<endl;
            marked[arc->no] = 1;
            s.push(arc->no);
        }

    }
}

int main(){
    int vertex[N] = {0,1,2,3,4,5,6,7};
    int edge[N][N] = {
        {0,1,0,1,1,0,0,0},
        {1,0,0,1,0,0,1,0},
        {0,0,0,0,0,0,1,0},
        {1,1,0,0,0,0,0,0},
        {1,0,0,0,0,0,0,1},
        {0,0,0,0,0,0,0,1},
        {0,1,1,0,0,0,0,1},
        {0,0,0,0,1,1,1,0}
    };
    ALGraph G;
    G.n = N;
    init(G,vertex,edge);
    DFS(0,G);
    //DFSUn(0,G);
    return 0;
}
```