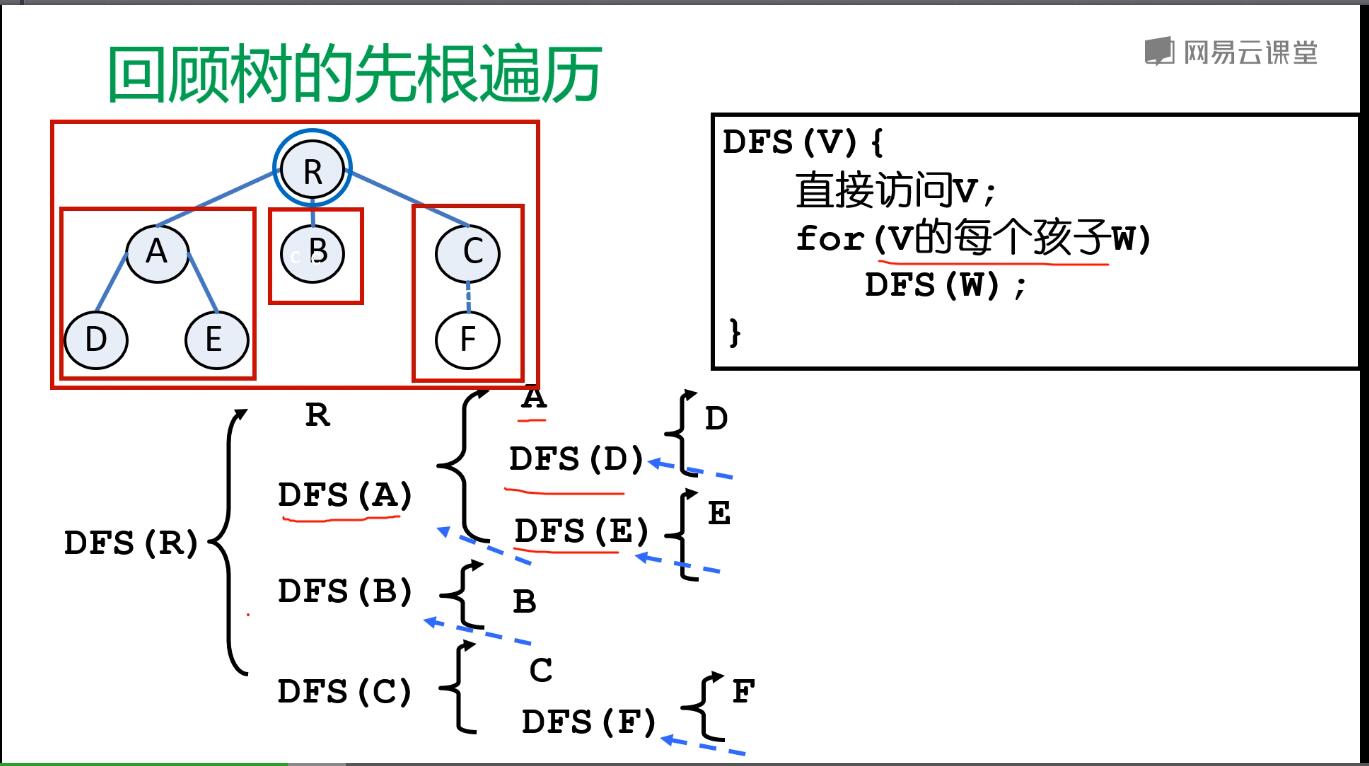
**图的深度遍历**

1. 我们可以参考树的先序遍历



1. 与树的先序遍历稍有不同的是，我们需要一个flag的数组，来判断某个节点是否已经访问呢过了
2. 树的先序遍历的方法来进行图的深度遍历代码

void DFS(void(\*vist)(const T& v)) {

claer\_flag(); //先把所有顶点设为未访问

for (unsigned i = 0; i < vertices.size(); i++) { //从每个顶点开始

if (flag[i] == false) { //如果未访问进行下面的操作

DFS(i, vist); //注意这不是递归啊哈

}

}

}

//这是递归遍历

void DFS(int v, void(\*vist)(const T & v)) {

vist(vertices[v]);

flag[v] = true; //把该顶点设为访问过了

for (unsigned e = 0; e < vertices.size(); e++) {

if (edges[v][e] != inv) { //如果边是相连，则数值是1不会是无效的值，那么我们要递归访问它的相连节点

if (!flag[e]) {

DFS(e, vist);

}

}

}

}

void claer\_flag() {

if (flag.size() != vertices.size()) {

flag.resize(vertices.size(), false);

}

for (int i = 0; i < flag.size(); i++) {

flag[i] = false;

}

}

bool is\_flag(int i) {

return flag[i];

}

};

template <typename T> //输出函数

void vist(const T & vertex) {

cout << vertex << " ";

}

1. 有栈来辅助图的深度遍历代码

//利用栈来辅助呢

void DFS\_(int v, void(\*vist)(const T& v)) {

stack<T> s; //初始化栈

s.push(vertices[v]);

flag[v] = true; //把顶点放进栈，并设置为访问过了

while (!s.empty) {

T e = s.top();

vist(e); //进行访问

s.pop();

for (unsigned w = 0; w < vertices.size(); w++) {

if (edges[v][w] != inv) {

if (!flag[w]) { //一次对其相连的节点进行同样的操作即可

s.push(vertices[w]);

flag[w] = true;

}

}

}

}

}