

## 06E 深度优先搜索 DFS

#数据结构邓神

### 算法

#### 算法

❖ **DFS(s)** //始自顶点s的深度优先搜索 (Depth-First Search)

访问顶点s

若s尚有未被访问的邻居，则任取其一u，递归执行DFS(u)

否则，返回

### 框架

#### Graph::DFS()

❖ template <typename Tv, typename Te> //顶点类型、边类型

void Graph<Tv, Te>::DFS( int v, int & clock ) {

dTime(v) = ++clock; status(v) = DISCOVERED; //发现当前顶点v

for ( int u = firstNbr(v); -1 < u; u = nextNbr(v, u) ) //枚举v的每一邻居u

/\* ... 视u的状态，分别处理 ... \*/

/\* ... 与BFS不同，含有递归 ... \*/

status(v) = VISITED; fTime(v) = ++clock; //至此，当前顶点v方告访问完毕

```
template <typename Tv, typename Te> void Graph<Tv, Te>::DFS(int s, int clock){
    dTime(v) = ++clock;
    status(v) = DISCOVERED;
    for (int u = firstNbr(v); -1 < u < ; u = nextNbr(v, u)) {
        /** 看 u 状态处理 **/

        /** 与BFS不同这里可能有递归 **/
    }
    status(v) = VISITED;
    fTime = ++clock;
```

```
}
```

## 细节

### Graph::DFS()

```
❖ for ( int u = firstNbr(v); -1 < u; u = nextNbr(v, u) ) //枚举v所有邻居u
    switch ( status(u) ) { //并视其状态分别处理
        case UNDISCOVERED: //u尚未发现，意味着支撑树可在此拓展
            status(v, u) = TREE; parent(u) = v; DFS(u, clock); break; //递归
        case DISCOVERED: //u已被发现但尚未访问完毕，应属被后代指向的祖先
            status(v, u) = BACKWARD; break;
        default: //u已访问完毕 ( VISITED, 有向图 ), 则视承袭关系分为前向边或跨边
            status(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS; break;
    } //switch
```

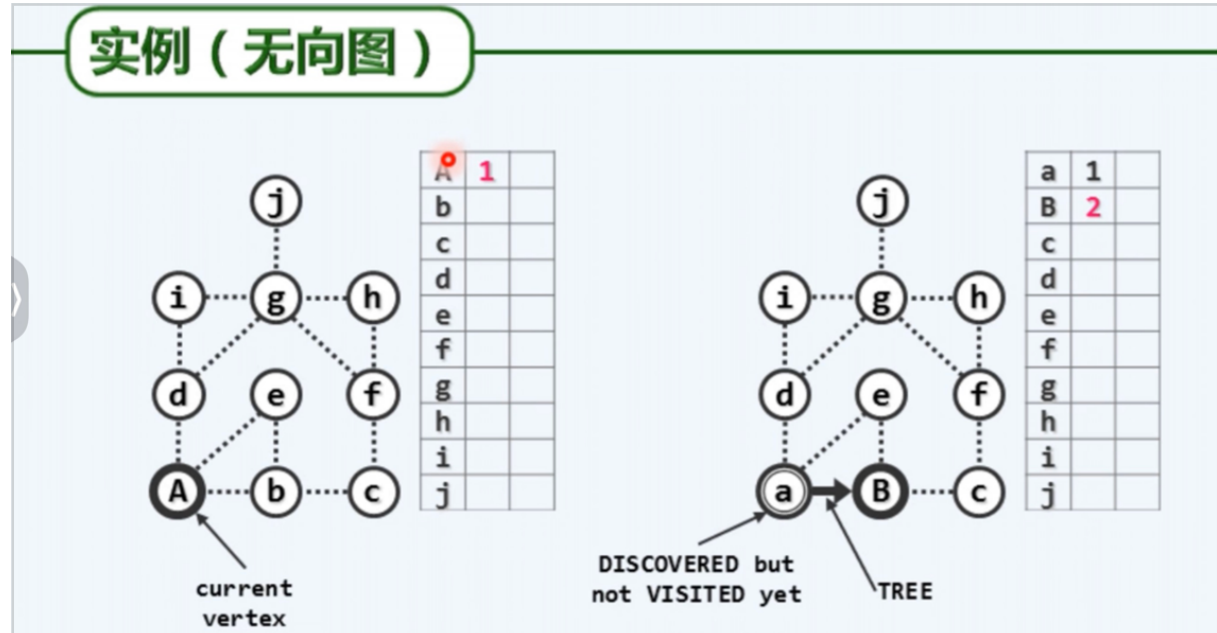
```
template <typename Tv, typename Te> void Graph<Tv, Te>::DFS(int s, int clock){
    dTime(v) = ++clock;
    status(v) = DISCOVERED;
    for (int u = firstNbr(v); -1 < u < ; u = nextNbr(v, u)) {
        switch(status(u)){
            case UNDISCOVERED:{
                status(v, u) = TREE;
                parent(u) = v;
                DFS(u, clock);
                break;
            }
            case DISCOVERED:{
                status(v, u) = BACKWARD;
                break;
            }
            default:{
                status(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS;
                break;
            }
        }
    }
}
```

```

status(v) = VISITED;
fTime = ++clock;
}

```

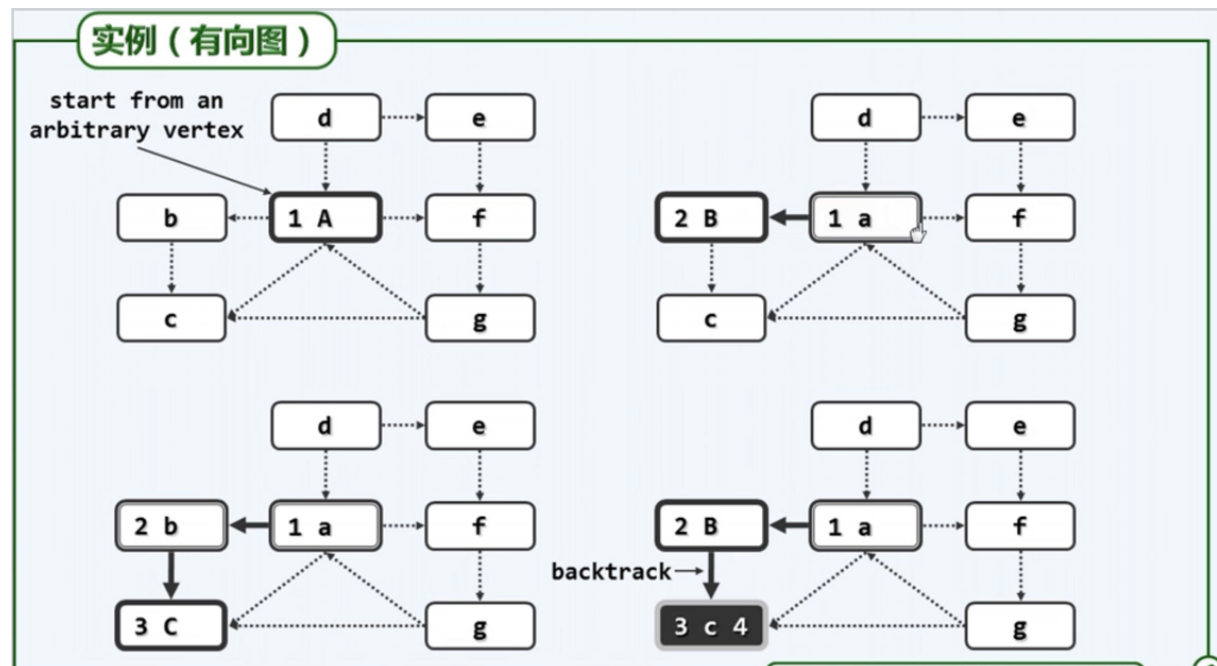
## 无向图



分别为标识符，dTime和fTime

但是这个递归方法如果图很大会有很多的递归层数，那么能否用迭代解决呢？

## 有向图



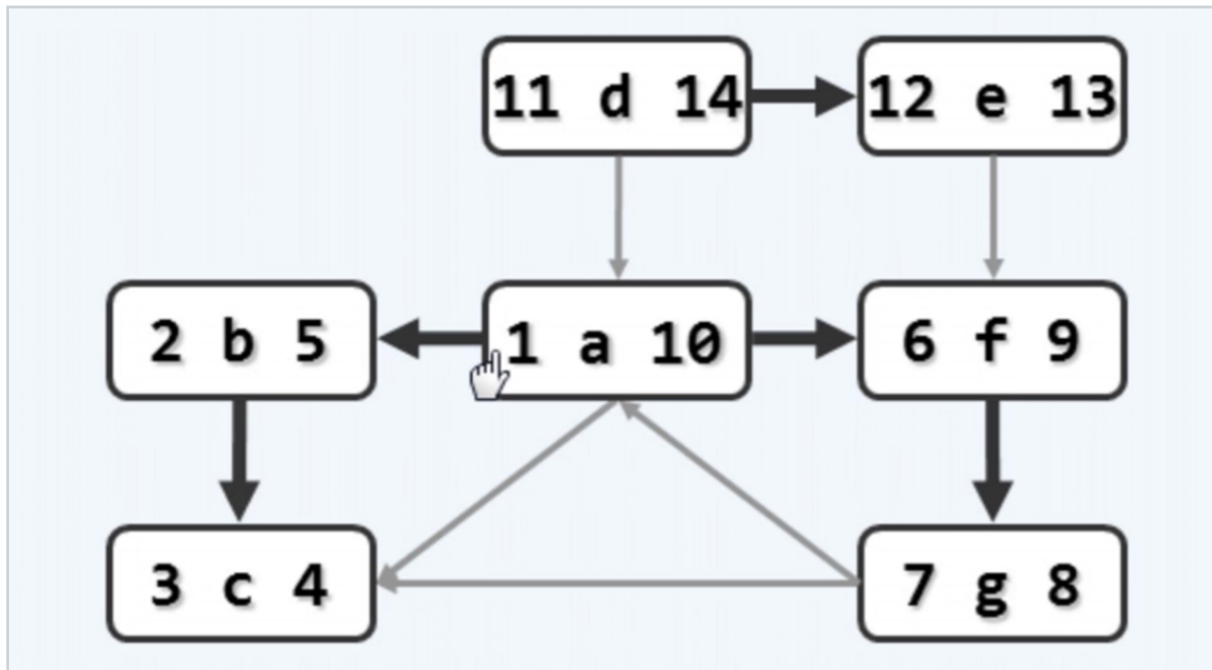
从a出发是无法达到d,e的，a的可达区域为 abcfg

## 多可达域

我们可以类似于广度优先搜索的算法

在包装一层循环来枚举图中所有的节点，保证完成所有的可达域

## 遍历成果



大黑边：构成了遍历森林 是最为重要的  
对于所有小灰边 进行了仔细的分类

其中时间标签的作用是非常巨大的！！

## 嵌套引理

顶点的活动期:  $\text{active}[u] = (\text{dTime}[u], \text{fTime}(u))$

只有存在包含和被包含的关系才存在血缘关系

祖先的活跃期一定包含后代的

后代的活跃期一定被祖先的活跃期包含。

### 括号引理

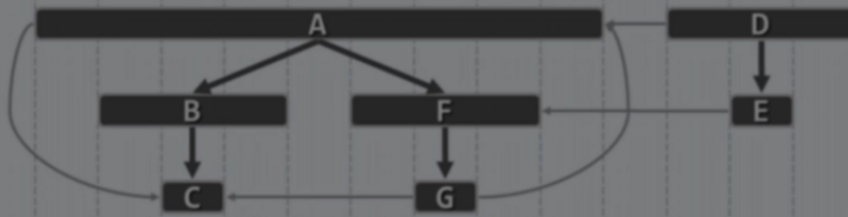
❖ 顶点的活动期:  $\text{active}[u] = (\text{dTime}[u], \text{fTime}[u])$

❖ Parenthesis Lemma: 给定有向图  $G = (V, E)$  及其任一DFS森林, 则

$u$  是  $v$  的后代 iff  $\text{active}[u] \subseteq \text{active}[v]$

$u$  是  $v$  的祖先 iff  $\text{active}[u] \supseteq \text{active}[v]$

$u$  与  $v$  “无关” iff  $\text{active}[u] \cap \text{active}[v] = \emptyset$



上面的图可以很直观的表现

这样的我们就可以在 (很小的常系数 只需要两个比较)  $O(1)$  的时间下确定是否有直系  
的血缘关系