

6. 图

深度优先搜索 算法

悔相道之不察兮，延伫乎吾将反
回朕车以复路兮，及行迷之未远

邓俊辉

deng@tsinghua.edu.cn

Depth-First Search

❖ DFS(s) // 始自顶点 s 的深度优先搜索

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

❖ 若此时图中尚有顶点未被访问 // 何时出现这一情况？

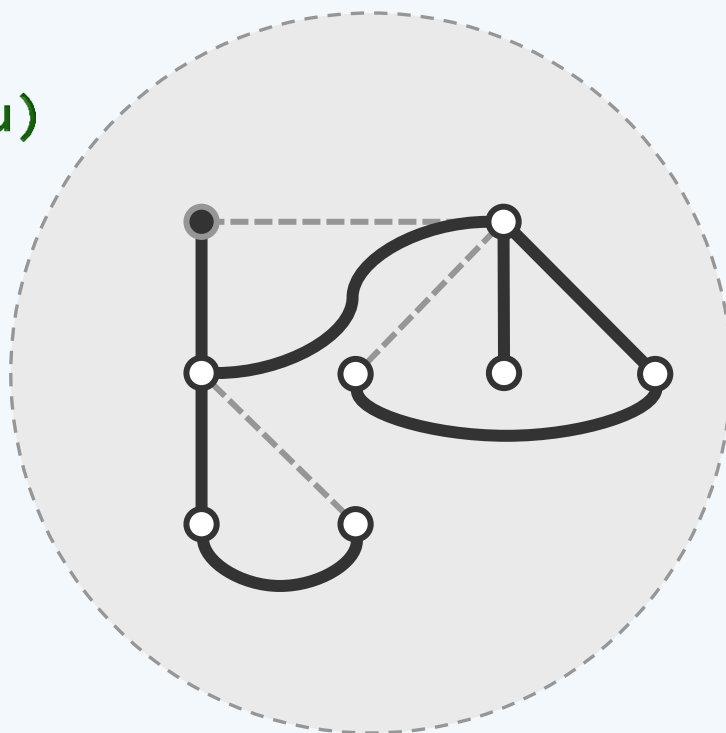
任取这样的顶点作起始点

重复上述过程

直至所有顶点都被访问到

❖ 等效于树的先序遍历

事实上，DFS也的确会构造出原图的一棵支撑树 (DFS tree)



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
```

```
}
```

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

            type(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

            type(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

            type(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS; break;

    } //switch
```