

## 6. 图

### (e) 拓扑排序

邓俊辉

[deng@tsinghua.edu.cn](mailto:deng@tsinghua.edu.cn)

# 有向无环图

## ❖ DAG

(Directed Acyclic Graph)

## ❖ 应用

类派生和继承关系图中，是否存在循环定义

操作系统中，相互等待的一组线程可否调度，如何调度

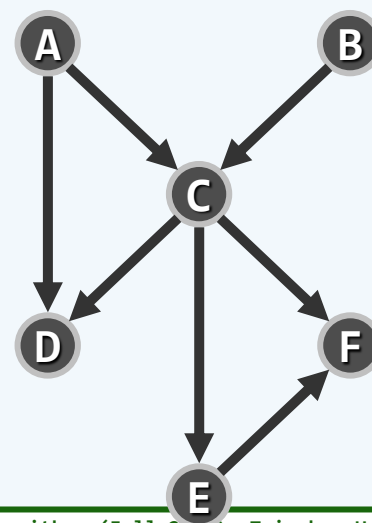
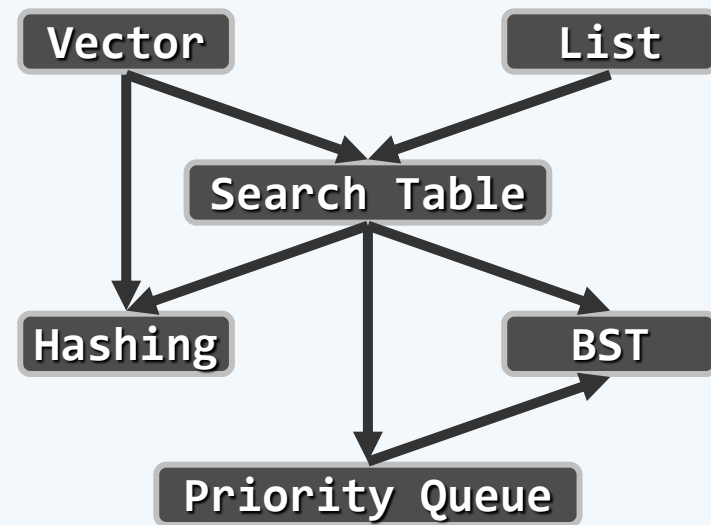
给定一组相互依赖的课程，是否存在可行的培养方案

给定一组相互依赖的知识点，是否存在可行的教学进度方案

项目工程图中，是否存在可串行施工的方案

email系统中，是否存在自动转发或回复的回路

...

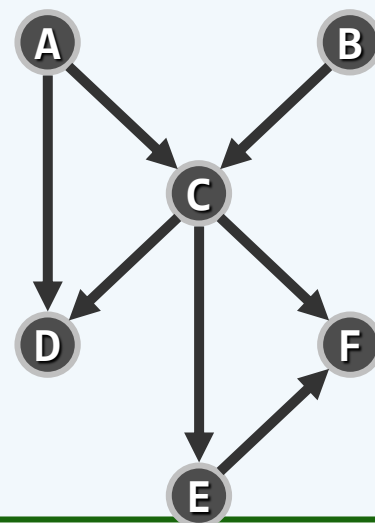
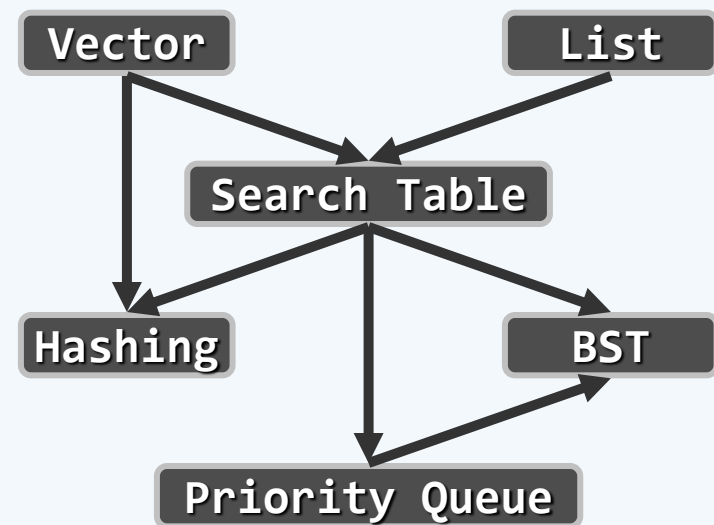
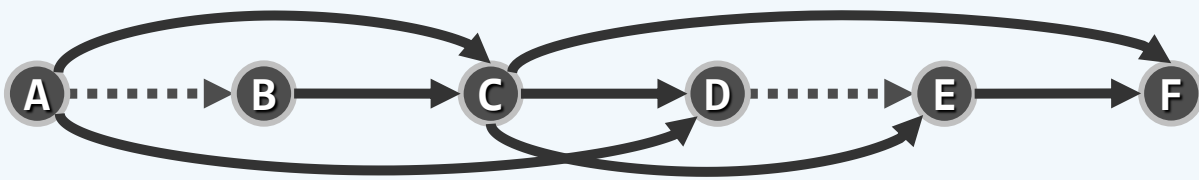


## 拓扑排序

- ❖ 任给有向图G，不一定是DAG
- ❖ 尝试将其中顶点排成一个**线性序列**  
其次序须与原图**相容**，亦即  
每一顶点都不会通过边指向**前驱**顶点

- ❖ 算法要求

若原图存在回路（即并非DAG），检查并报告  
否则，给出一个相容的线性序列



## 存在性

- ❖ 每个DAG对应于一个偏序集；拓扑排序对应于一个全序集

所谓的拓扑排序，即构造一个与指定偏序集相容的全序集

- ❖ 可以拓扑排序的有向图，必定无环 //反之

任何DAG，都存在（至少）一种拓扑排序？是的！ //为什么...

- ❖ 有限偏序集必有极大/极大元素

任何DAG都存在（至少）一种拓扑排序

- ❖ 可归纳证明，并直接导出一个算法...

## 存在性

1. 任何DAG，必有（至少一个）顶点入度为零 //记作 $m$
  2. 若  $\text{DAG} \setminus \{m\}$  存在拓扑排序  $S = \langle u_{k1}, \dots, u_{k(n-1)} \rangle$  //subtraction  
则  $S' = \langle m, u_{k1}, \dots, u_{k(n-1)} \rangle$  即为DAG的拓扑排序 //DAG子图亦为DAG
- ❖ 只要 $m$ 不唯一，拓扑排序也应不唯一 //反之呢？

## 算法A：顺序输出零入度顶点

将所有入度为零的顶点存入栈s，取空队列Q //  $O(n)$

while (!S.empty()) { //  $O(n)$

    Q.enqueue( v = S.pop() ); // 栈顶v转入队列

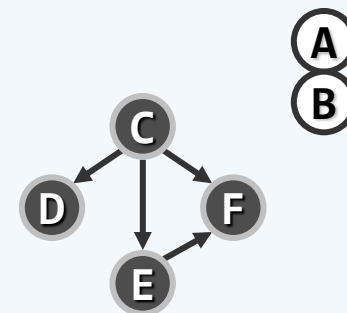
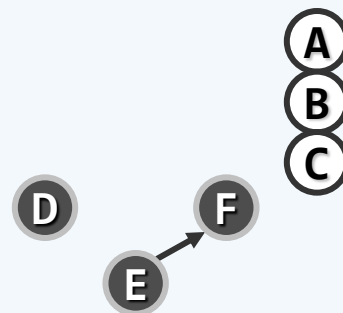
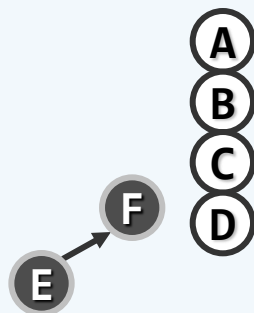
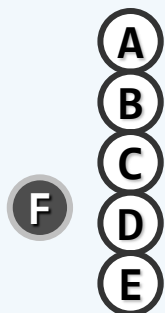
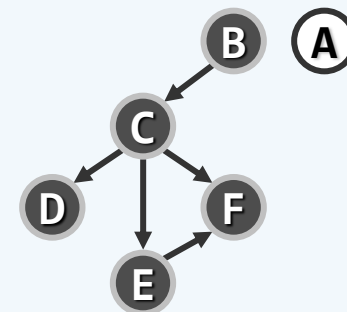
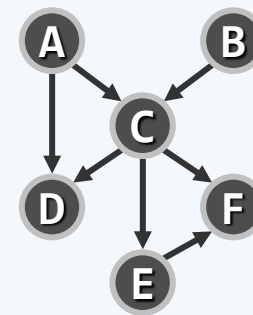
    for each edge(v, u) // v的邻接顶点u若入度仅为1

        if ( inDegree(u) < 2 ) S.push(u); // 则入栈

    G = G \ {v}; // 删除v及其关联边 (邻接顶点入度减1)

} // 总体  $O(n + e)$

return |G| ? Q : "NOT\_DAG"; // 残留的G空，当且仅当原图可拓扑排序



## 算法B：逆序输出零出度顶点

❖ /\* 基于DFS，借助栈S \*/

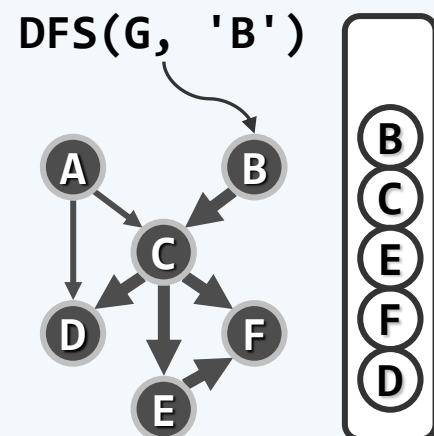
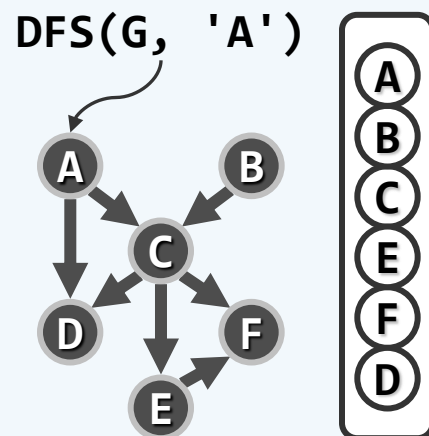
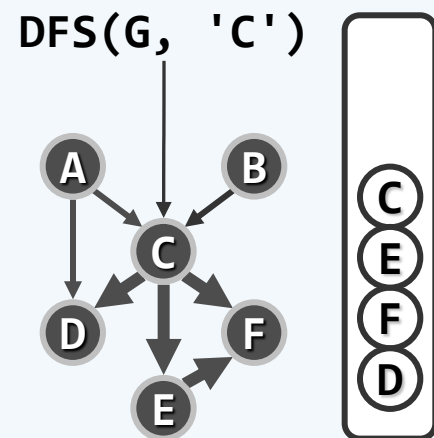
对图G做DFS，其间 //得到组成DFS森林的一系列DFS树

每当有顶点被标记为VISITED，则将其压入S

一旦发现有后向边，则报告非DAG并退出

DFS结束后，顺序弹出S中的各个顶点

❖ 复杂度与DFS相当，也是 $O(n + e)$



## 算法B：实现

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

```
bool Graph<Tv, Te>::TSort(int v, int & clock, Stack<Tv>* S) {
```

```
    dTime(v) = ++clock; status(v) = DISCOVERED; //发现顶点v
```

```
    for ( int u = firstNbr(v); -1 < u; u = nextNbr(v, u) ) //枚举v所有邻居u
```

```
        /* ... 视u的状态，分别处理 ... */
```

```
    status(v) = VISITED; S->push( vertex(v) ); //顶点被标记为VISITED时入栈
```

```
    return true;
```

```
}
```



## 算法B：实现

```
❖ for ( int u = firstNbr(v); -1 < u; u = nextNbr(v, u) ) //枚举v所有邻居u
    switch ( status(u) ) { //并视u的状态分别处理
        case UNDISCOVERED:
            parent(u) = v; status(v, u) = TREE; //树边(v, u)
            if ( !TSort(u, clock, S) ) return false; break; //从顶点u处深入
        case DISCOVERED: //一旦发现后向边 ( 非DAG )
            status(v, u) = BACKWARD; return false; //则退出而不再深入
        default: // VISITED (digraphs only)
            status(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS; break;
    }
```