L18：拓扑排序

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 掌握AOV网、拓扑排序等概念。
- 掌握拓扑排序的求解方法及相关分析。

问题背景

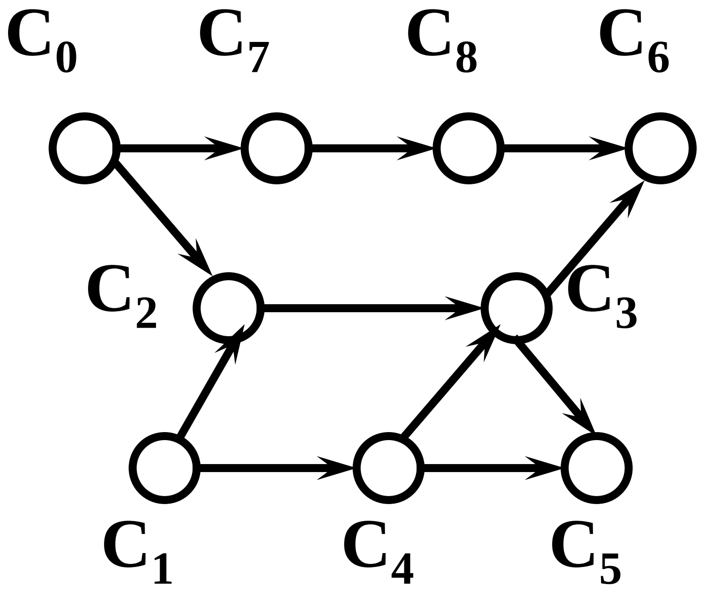
- “工程”或任务
  - ✓ 计划、施工过程、生产流程、程序流程等都可以看作一个“工程”或任务。
- “活动”
  - ✓ 除了很小的工程外，一般都把工程分为若干个叫做“活动”的子任务。
- 活动之间一般会有先后关系
  - ✓ 如果不违反限制完成所有活动，那么整个工程将顺利完成。

例：选课

- 计算机专业的学习可以看作一个工程；
- 每门课程的学习就是整个工程的一项活动；
- 其中有些课程要求先修课程，有些则不要求。这样有的课程之间存在先后关系，有的课程可以并行地学习。
- 任务：安排一种学习次序，学习完成所有课程，并满足课程间的限制关系。

计算机专业必修课程（部分）

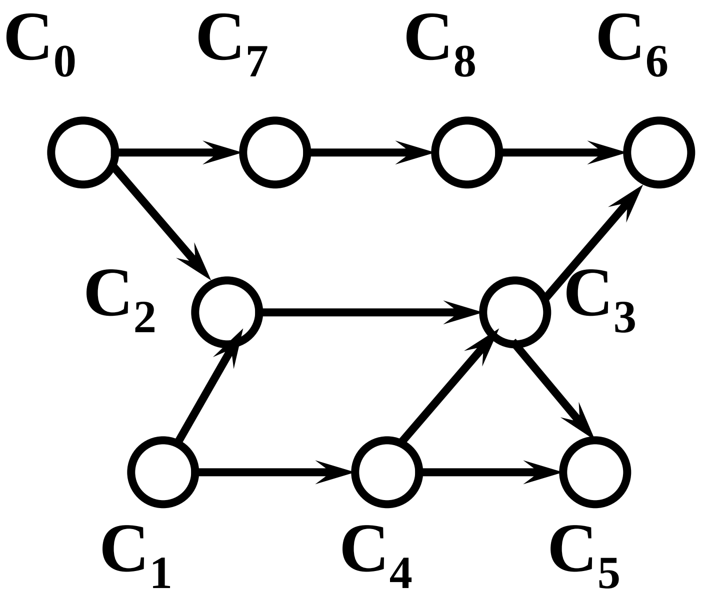
课程代号	课程名称	先修课程
C_0	高等数学	无
C_1	程序设计基础	无
C_2	离散数学	C_0,C_1
C_3	数据结构	C_2,C_4
C_4	程序设计语言	C_1
C_5	编译技术	C_3,C_4
C_6	操作系统	C_3,C_8
C_7	普通物理	C_0
C_8	计算机原理	C_7



概念

- AOV网：用顶点表示活动，用有向边表示活动之间的先后关系，称这样的有向图为AOV网(Activity On Vertex Network)。
- 拓扑序列：AOV网中的所有顶点的一个线性序列，要求：如果存在有向边$<V_i , V_j>$，那么在序列中V_i必位于V_j之前。
- 拓扑排序：构造AOV网的拓扑序列的过程称作拓扑排序。

拓扑序列构造



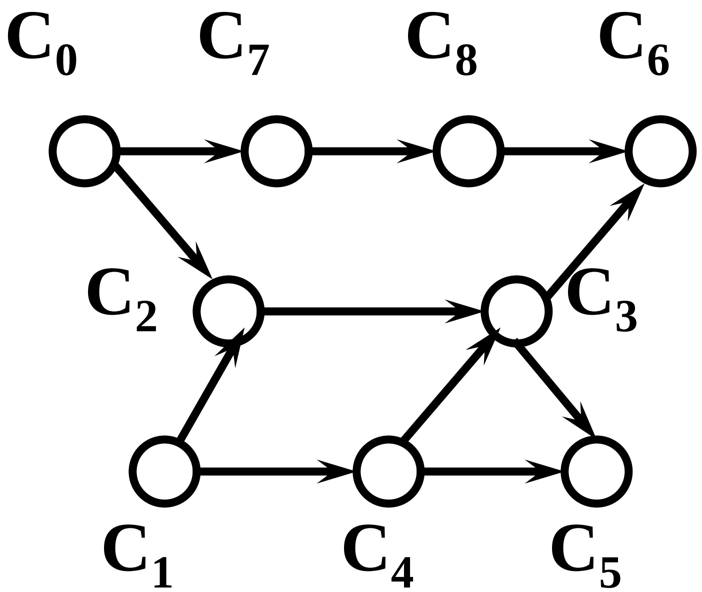
- 一种可能的拓扑序列是：**C0 , C1 , C2 , C4 , C3 , C5 , C7 , C8 , C6**

拓扑序列构造方法

① 从网中选择一个入度为0的顶点输出；

② 从网中删除该顶点及其所有出边；

执行① ②，直至所有顶点已输出，或网中剩余顶点入度均不为0（网中存在回路）。



- C0 , C1 , C2 , C4 , C3 , C5 , C7 , C8 , C6
- C0 , C1 , C4 , C2 , C3 , C5 , C7 , C8 , C6
- C0 , C1 , C7 , C2 , C4 , C3 , C5 , C8 , C6

算法设计

- 图的存储: AOV网用邻接表的形式存储；
- 数组count[ ]: count[i]的值是顶点i的入度；
- 使用一个数据结构，存放入度为0的点
  - ✓ 线性表
  - ✓ 取用顺序无所谓：栈或队列
- 关键：如何处理删除？
  - ✓ 对于拓扑排序而言，减少入度即可

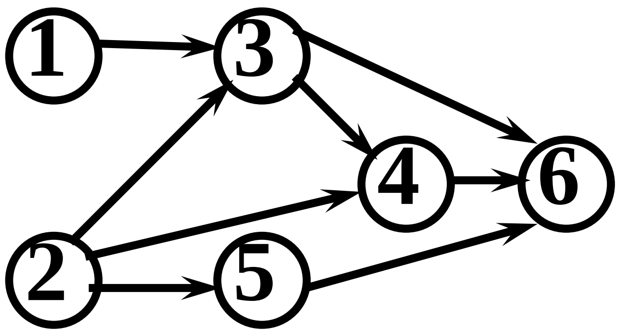
拓扑排序算法——原始版

```
算法TopoOrder( )
 /* 图的拓扑排序算法，n表示顶点数 */
T1[初始化]
    for( i = 1 ; i<= n ; i ++ ) count[i] = 0;
    for( i = 1 ; i<= n ; i ++ )
        for( p = Head[i].adjacent ; p ; p = p->link )
            count[ p->VerAdj ] ++;

   CREATESTACK(S);     //用队列也可以
   for( i = 1 ; i<= n ; i ++ )
       if( count[i] == 0 )  push(S, i);  //入度为0的点入栈
```

T2[拓扑排序]

```
    for( i = 1 ; i <= n ; i++ ){
        if (empty(S)) { cout<<“有回路! ”; RETURN; }

        pop( S, j ).                        //弹出栈顶j *
        cout<<  j ;                         //按要求处理 顶点 j
        for ( p= Head[j] . adjacent ; p ; p = p -> link) {
            k  = p -> VerAdj ;
            count[k] -- ;                   // 顶点k的入度减1
            if (count[k] == 0 ) push( S, k );
        }
 }
```

运行示例

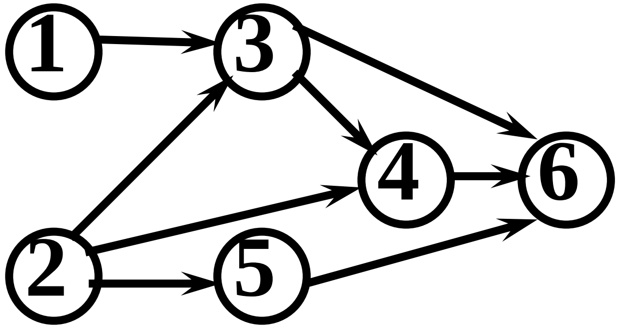


- count数组
- 堆栈S

原始版的改进

- 观察到count数组中存0的位置已无用，恰好可以用这部分空间作为容器，节省空间。即用count数组模拟栈。

- 用入度为0的count[i]空间记录栈元素的下标；
- 用top始终记录栈顶元素的下标。
- 利用变量top和count数组元素的值来模拟堆栈的压入和弹出。



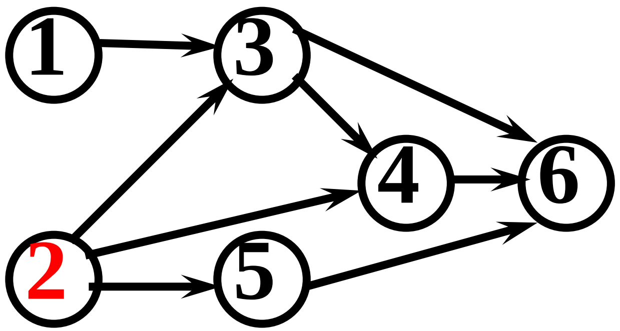
	1	2	3	4	5	6
count	0	0	2	2	1	3

	1	2	3	4	5	6
count	-1	0	2	2	1	3

top

	1	2	3	4	5	6
count	-1	1	2	2	1	3

top



	1	2	3	4	5	6
count	-1	1	2	2	1	3

top

	1	2	3	4	5	6
count	-1	1	1	1	0	3

top

模拟栈的状态

- 初始化： top = -1;
- 栈　空： top == -1
- 入栈： count[i] = top; top = i;
- 出栈： j = top; top = count[top];

拓扑排序算法

算法TopoOrder( )

*/** 图的拓扑排序算法，n表示顶点数 **/*

T1[初始化]

```
for( i = 1 ; i<= n ; i ++ ) count[i] = 0;
for( i = 1 ; i<= n ; i ++ )
    for( p = Head[i].adjacent ; p ; p = p->link )
        count[ p->VerAdj ] ++;
for( i = 1 ; i<= n ; i ++ )
    if( count[i] == 0 )
        count[i] = top , top = i ;
```

T2[拓扑排序]

```
for( i = 1 ; i <= n ; i++ ){
    if ( top == - 1 ) { cout<<“有回路! ”; RETURN; }
    j = top , top = count[top] .   /* 弹出栈顶j */
    cout<<  j ;  //按要求处理 顶点 j
    for ( p= Head[j] . adjacent ; p ; p = p -> link) {
        k  = p -> VerAdj ;
        count[k] -- ;// 顶点k的入度减1
        if (count[k] == 0 ) count[k]=top, top = k;
    }
}
```

相关定理

- 引理6.1： 设图G = (V, E)是有向无环图, V(G)≠Φ, 则G中一定存在入度为零的顶点。
  - 有向无环图（DAG）：非循环环图
- 定理6.2 设G=(V, E)是有向无环图，V(G)={1, 2 ,…, n}, e=|E(G)|. 则算法TopoOrder是正确的且算法的时间复杂性为 O(n+e).

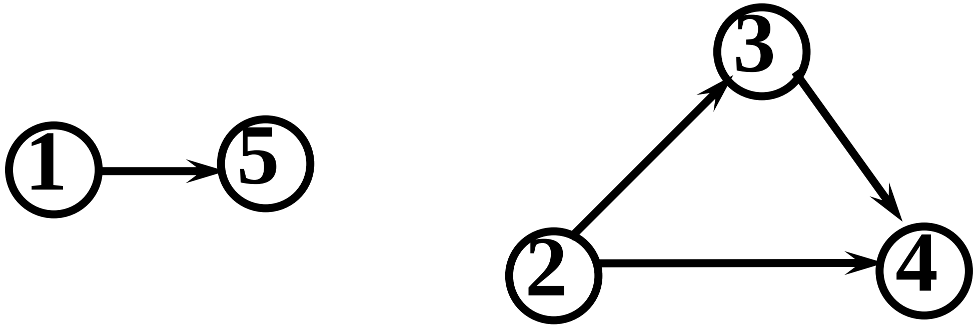
正确性证明

- 正确性证明
  - ✓ 初始化**T1**时，栈不为空（引理**6.1**）
  - ✓ 出栈一个元素相当于删除一个顶点及其所有出边，**T2**时，若**G**不空，栈也不空。输出**n**个顶点结束
  - ✓ 设**<v,w>**是边，则 **v** 一定 排在 **w** 之前。
- 时间效率分析
  - ✓ O(n+e)

拓扑序列的存在性

- 任意图的拓扑序列不一定存在。
  - 例如，存在回路的AOV网就无法找到拓扑序列。因为出现了有向环，则意味着某项活动以自己作为先决条件。
- 有向无环图（DAG）一定存在拓扑序列。
- 拓扑排序与环的关系
  - 有向图中，可拓扑排序 等价于 无环。

拓展: 拓扑序列计数



count = (C(5,2)*1) *(C(3,3)*1) =10

计算：独立的块之间是乘法关系；
　　　块内要枚举每种情况。