**拓扑排序**

一个无环的有向图被称为**有向无环图（Directed Acycline Graph，DAG）**。有向无环图是描述一个工程、计划、生产、系统等流程的有效工具。一个大工程可分为若干子工程（活动），活动之间通常有一定的约束，例如先做什么活动，在什么活动完成后才可以开始下一个活动。

**用节点表示活动，用弧表示活动之间的优先关系的有向图，被称为AOV网（Activity On Vertex Network）。**

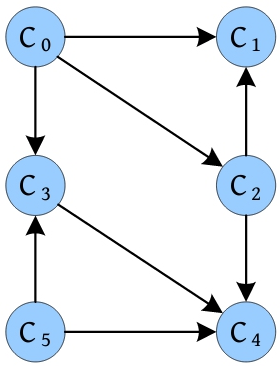
在AOV网中，**若从节点i到节点j存在一条有向路径，则称节点i是节点j的前驱，或者称节点j是节点i的后继**。

若**<i,j>是图中的弧**，则称节点**i是节点j**的**直接前驱**，节点**j是节点i**的**直接后继**。

AOV网中的弧表示了活动之间存在的制约关系。例如，计算机专业的学生必须完成一系列规定的基础课和专业课才能毕业。学生按照怎样的顺序来学习这些课程呢？这个问题可以被看成一个大的工程，其活动就是学习每一门课程。课程的名称与相应编号如下表所示。



如果用节点表示课程，用弧表示先修关系，若课程i是课程j的先修课程，则用弧<i,j>表示，课程之间的关系如下图所示。

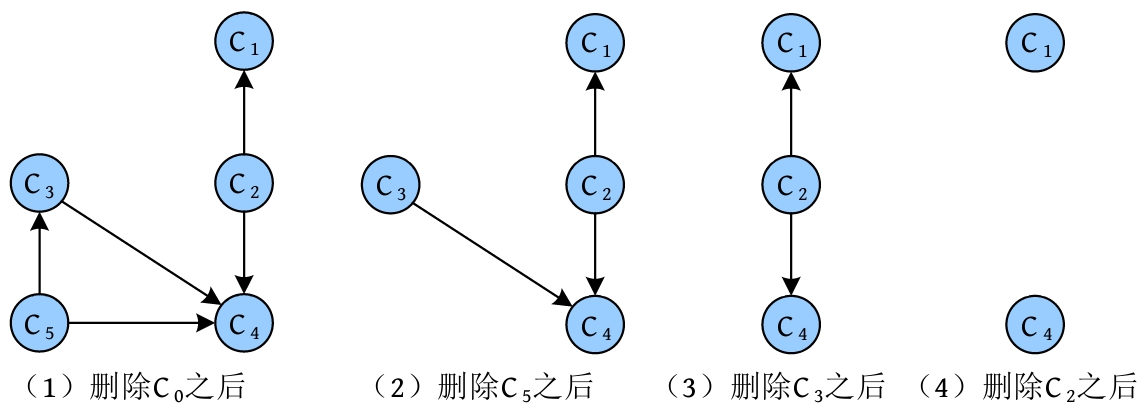


在**AOV网中是不允许有环**的，否则会出现自己是自己的前驱的情况，陷入死循环。**怎么判断在AOV网中是否有环呢？**一种检测的办法是对有向图中的节点进行拓扑排序。**如果AOV网中的所有节点都在拓扑序列中，则在AOV网中必定无环**。

**拓扑排序指将AOV网中的节点排成一个线性序列**，该序列必须满足：**若从节点i到节点j有一条路径，则在该序列中节点i一定在节点j之前**。

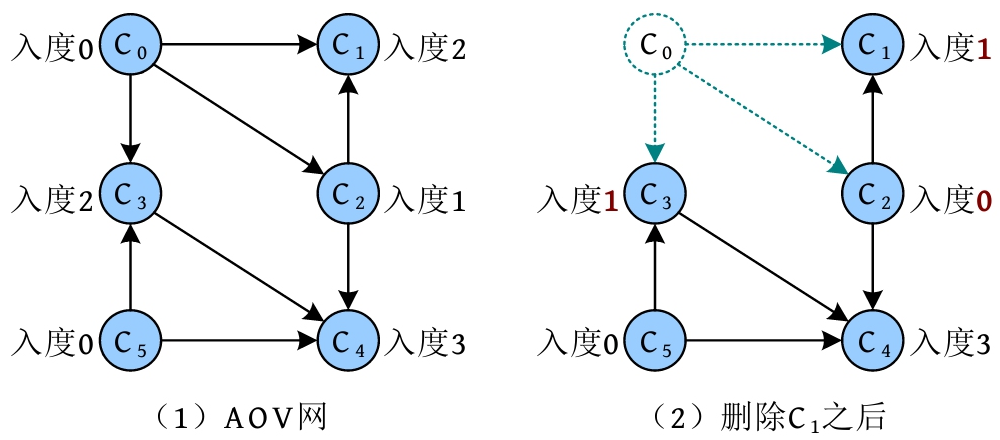
**拓扑排序的基本思想：**①选择一个无前驱的节点并输出；②从图中删除该节点和该节点的所有发出边；③重复步骤1、2，直到不存在无前驱的节点；④如果输出的节点数少于AOV网中的节点数，则说明网中有环，否则输出的序列即拓扑序列。

拓扑排序并不是唯一的，例如在上图中，节点C0和C5都无前驱，先输出哪一个都可以，如果先输出C0，则删除C0及C0的所有发出边。此时C2和C5都无前驱，如果输出C5，则删除C5及C5的所有发出边。此时C2和C3都无前驱，如果输出C3，则删除C3及C3的所有发出边。此时C2无前驱，如果输出C2，则删除C2及C2的所有发出边。此时C1和C4都无前驱，将其输出并删除即可。



拓扑序列为C0、C5、C3、C2、C1、C4。

在上述描述过程中有删除节点和边的操作，实际上，**没必要真的删除节点和边**。**可以将没有前驱的节点（入度为0）暂存到栈中，输出时出栈即表示删除**。**进行边的删除时将其邻接点的入度减1即可**。例如在下图中删除C0的所有发出边，相当于将C3、C2、C1节点的入度减1。



**1. 算法步骤**

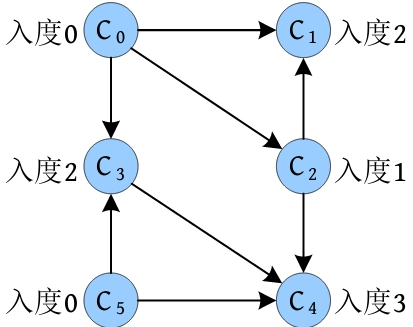
（1）求各节点的入度，将其存入数组indegree[]中，并将入度为0的节点入栈S。

（2）如果栈不空，则重复执行以下操作：①将栈顶元素i出栈并保存到拓扑序列数组topo[]中；②将节点i的所有邻接点入度都减1，如果减1后入度为0，则立即入栈S。

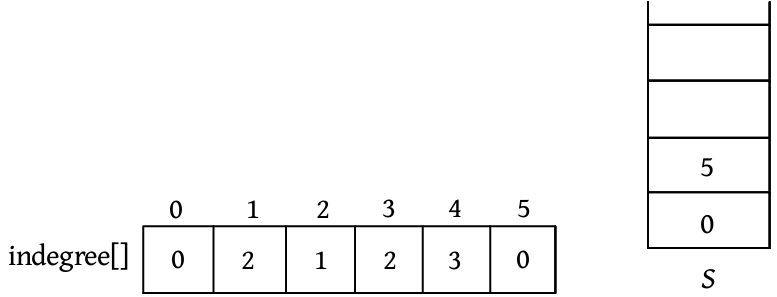
（3）如果输出的节点数少于AOV网中的节点数，则说明网中有环，否则输出拓扑序列。

**2. 图解**

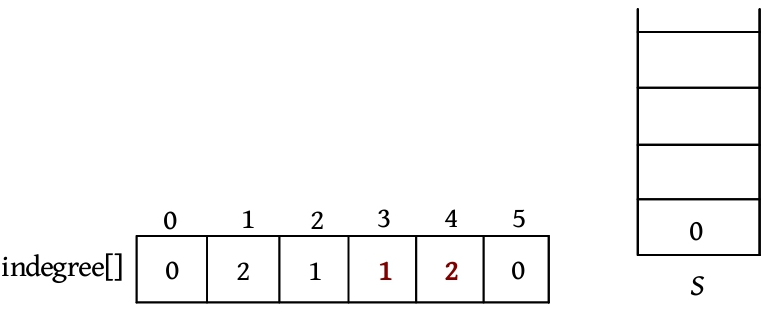
例如，一个AOV网如下图所示，其拓扑排序的过程如下。



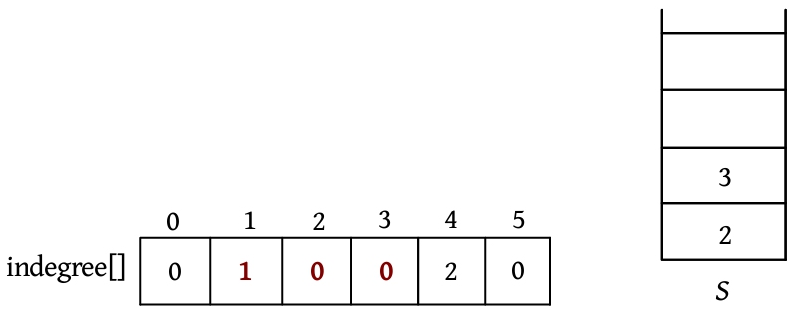
（1）输入边时累加节点的入度并保存到数组indegree[]中，将入度为0的节点入栈S。



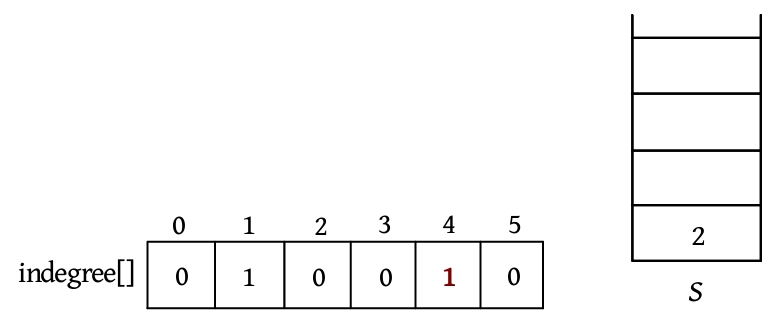
（2）将栈顶元素5出栈并保存到拓扑序列数组topo[]中。将节点5的所有邻接点（C3、C4）入度都减1，如果减1后入度为0，则立即入栈S。



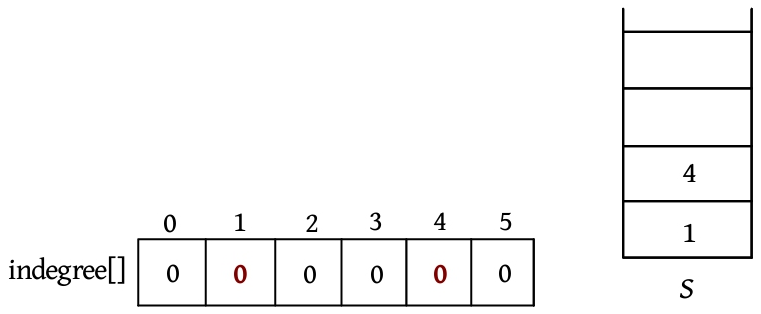
（3）将栈顶元素0出栈并保存到拓扑序列数组topo[]中。将节点0的所有邻接点（C1、C2、C3）入度都减1，如果减1后入度为0，则立即入栈S。



（4）将栈顶元素3出栈并保存到拓扑序列数组topo[]中。将节点3的邻接点C4入度减1，如果减1后入度为0，则立即入栈S。



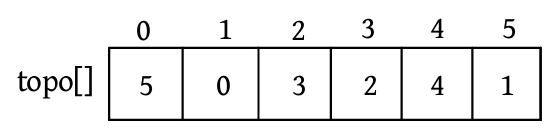
（5）将栈顶元素2出栈并保存到拓扑序列数组topo[]中。将节点2的所有邻接点（C1、C4）入度减1，如果减1后入度为0，则立即入栈S。节点1没有邻接点，什么也不做。



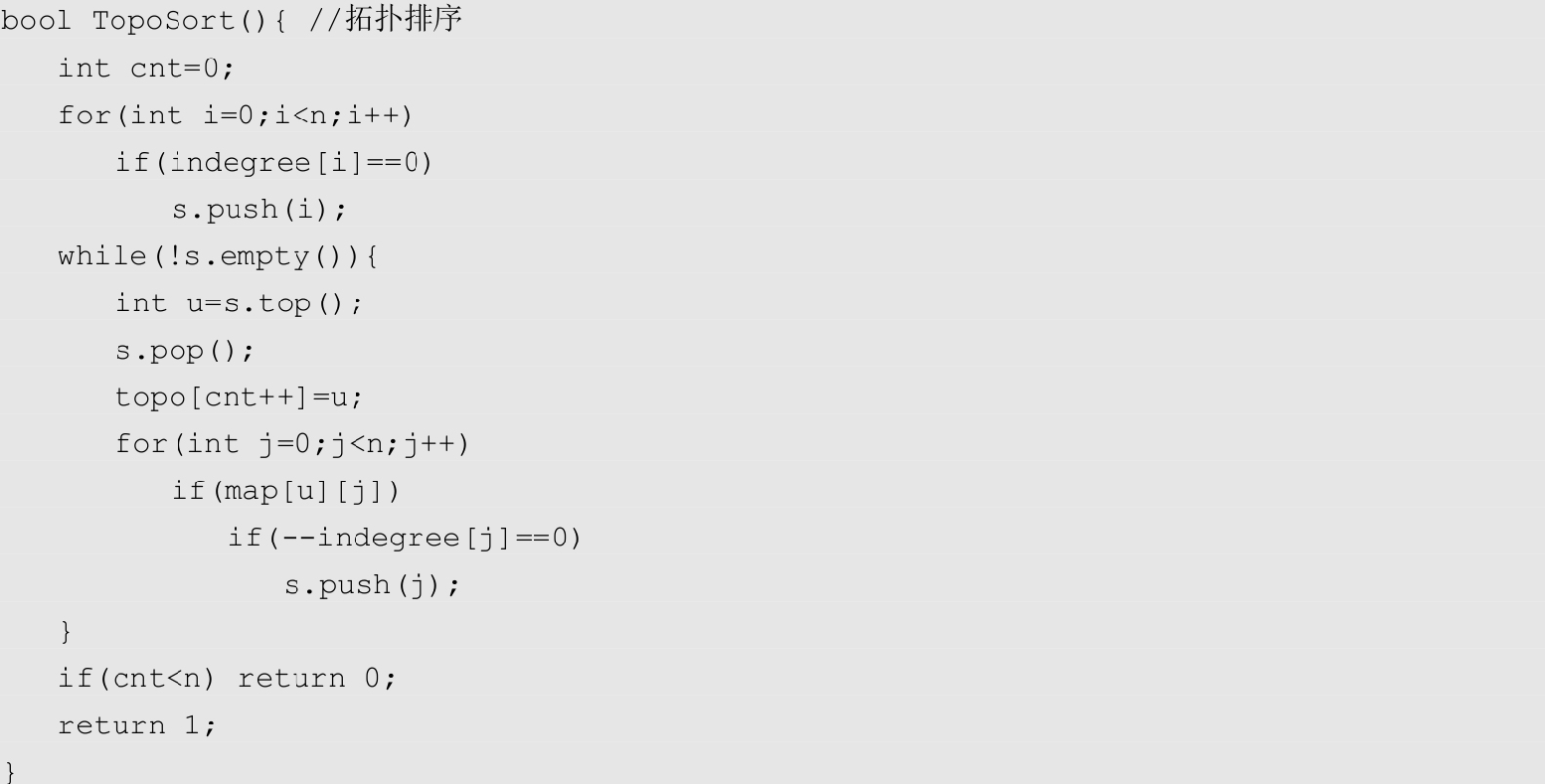
（6）将栈顶元素4出栈并保存到拓扑序列数组topo[]中。节点4没有邻接点。

（7）将栈顶元素1出栈并保存到拓扑序列数组topo[]中。节点1没有邻接点。

（8）栈空，算法停止。输出拓扑排序序列。



**3. 算法实现**

****

**4. 算法分析**

**时间复杂度：**度数为0的节点入栈的时间复杂度为O(n)，在每个节点出栈后都需要将其邻接点入度减1，如果使用邻接矩阵存储，则每次访问邻接点的时间复杂度都为O(n)，总的时间复杂度为O(n2)。采用邻接表或链式前向星存储访问一个节点的所有邻接点，访问次数为该节点的度，总的时间复杂度为O(e)。

**空间复杂度：**辅助空间包括入度数组indegree[]、拓扑序列数组topo[]、栈S，算法的空间复杂度是O(n)。