```
                    Path[i][j]=k;
                }
    }
```

弗洛伊德算法的时间复杂度分析：

由算法代码可知，本算法的主要部分是一个三层循环，取最内层循环的操作作为基本操作，则基本操作执行次数为 n^3，因此时间复杂度为 $O(n^3)$。

7.6 拓扑排序

7.6.1 AOV 网

活动在顶点上的网（Activity On Vertex network，AOV）是一种可以形象地反映出整个工程中各个活动之间的先后关系的有向图。图 7-25 所示为制造一个产品的 AOV 网。制造该产品需要 3 个环节，第一个环节获得原材料，第二个环节生产出 3 个部件，第三个环节由 3 个部件组装成成品。在原材料没有准备好之前不能生产部件，在 3 个部件全部被生产出来之前不能组装成成品，这样一个工程各个活动之间的先后次序关系就可以用一个有向图来表示，称为 AOV 网。

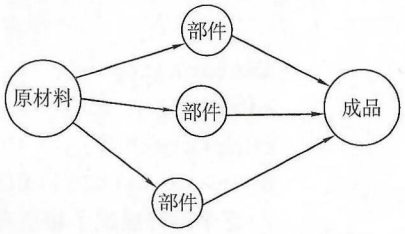

图 7-25 反映一个产品生产过程先后次序的 AOV 网

考试中，只要知道 AOV 网是一种以顶点表示活动、以边表示活动的先后次序且没有回路的有向图即可。因为 AOV 网有实际意义，所以出现回路就代表一项活动以自己为前提，这显然违背实际。

7.6.2 拓扑排序核心算法

对一个有向无环图 G 进行拓扑排序，是将 G 中所有顶点排成一个线性序列，使得图中任意一对顶点 u 和 v，若存在由 u 到 v 的路径，则在拓扑排序序列中一定是 u 出现在 v 的前边。

在一个有向图中找到一个拓扑排序序列的过程如下：

1）从有向图中选择一个没有前驱（入度为 0）的顶点输出；

2）删除 1）中的顶点，并且删除从该顶点发出的全部边；

3）重复上述两步，直到剩余的图中不存在没有前驱的顶点为止。

以邻接表为存储结构，怎样实现拓扑排序的算法呢？因为上述步骤中提到要选取入度为 0 的结点并将其输出，要对邻接表表头结构定义进行修改，可加上一个统计结点入度的计数器，修改如下：

```
typedef struct
{
    char  data;
    int count;                    //此句为新增部分，count 来统计顶点当前的入度
    ArcNode *firstarc;
}VNode;
```

假设图的邻接表已经生成，并且各个顶点的入度都已经记录在 count 中，在本算法中要设置一个栈，用来记录当前图中入度为 0 的顶点，还要设置一个计数器 n，用来记录已经输出的顶点个数。

算法开始时置 n 为 0，扫描所有顶点，将入度为 0 的顶点入栈。然后在栈不空的时候循环执行：出栈，将出栈顶点输出，执行++n，并且将由此顶点引出的边所指向的顶点的入度都减 1，并且将入度变为 0 的顶点入栈；出栈，…，栈空时循环退出，排序结束。循环退出后判断 n 是否等于图中的顶点个数。如果相等则返回 1，拓扑排序成功；否则返回 0，拓扑排序失败。

由此可以写出以下算法代码：

```
int TopSort(AGraph *G)
{
    int i,j,n=0;
    int stack[maxSize],top=-1;      //定义并初始化栈
    ArcNode *p;
    /*这个循环将图中入度为0的顶点入栈*/
    for(i=0;i<G->n;++i)             //图中顶点从0开始编号
        if(G->adjlist[i].count==0)
            stack[++top]=i;
        /*关键操作开始*/
    while(top!=-1)
    {
        i=stack[top--];             //顶点出栈
        ++n;                        //计数器加1，统计当前顶点
        cout<<i<<" ";               //输出当前顶点
        p=G->adjlist[i].firstarc;
        /*这个循环实现了将所有由此顶点引出的边所指向的顶点的入度都减少1，并将这个过程中入度变为0的顶点入栈*/
        while(p!=NULL)
        {
            j=p->adjvex;
            --(G->adjlist[j].count);
            if(G->adjlist[j].count==0)
                stack[++top]=j;
            p=p->nextarc;
        }
    }
    /*关键操作结束*/
    if(n==G->n)
        return 1;
    else
        return 0;
}
```


注意：拓扑排序序列可能不唯一。从算法过程中可以看出，在选择入度为0的顶点输出的时候，对顶点没有其他要求，只需入度为0即可。当前步骤中有多个入度为0的顶点时，可以任选一个输出，这就造成了拓扑排序序列不唯一。

若AOV网中考查各顶点的出度并以下列步骤进行排序，则将这种排序称为逆拓扑排序，输出的结果称为逆拓扑有序序列。

1）在网中选择一个没有后继的顶点（出度为0）输出；

2）在网中删除该顶点，并删除所有到达该顶点的边；

3）重复上述两步，直到AOV网中已无出度为0的顶点为止。

注意：当有向图中无环的时候，还可以采用深度优先搜索遍历的方法进行拓扑排序。由于图中无环，当由图中某顶点出发进行深度优先搜索遍历时，最先退出算法的顶点即为出度为0的顶点，它是拓扑有

序序列中的最后一个顶点。因此，按照 DFS 算法的先后次序记录下的顶点序列即为逆向的拓扑有序序列。

关于这个注意的内容，有不少同学通过微信反映理解上有困难，下面就其中两句话详细解释一下。

1. “最先退出算法的顶点即为出度为 0 的顶点。”

2. “按照 DFS 算法的先后次序记录下的顶点序列。”

第 1 句话中退出算法是指所遍历的顶点退出当前系统栈。

第 2 句话中按照 DFS 算法先后次序并不是指最终遍历结果序列，而是顶点退出系统栈的顺序。

例如，图{A->B，A->C，B->D，C->D}的一种深度优先遍历进出栈过程为：

A 入栈（栈中元素为 A）；

B 入栈（栈中元素为 AB）；

D 入栈（栈中元素为 ABD）；

D 出栈（栈中元素为 AB，D 为第一个出栈元素）；

B 出栈（栈中元素为 A，B 为第二个出栈元素）；

C 入栈（栈中元素为 AC）；

C 出栈（栈中元素为 A，C 为第三个出栈元素）；

A 出栈（栈空，A 为第四个出栈元素）。因此各个元素出栈先后序列为 DBCA，为拓扑序列 ACBD 的逆拓扑序列。

7.6.3 例题选讲

【例 7-6】 分析本节中拓扑排序算法的时间复杂度。

分析：

本算法主体部分为一个单层循环和一个双重循环。单层循环执行次数为 n。对于双重循环，直接根据循环条件分析循环执行的次数比较困难，故换个角度分析。首先看进栈操作，因为在无环情况下，每个结点恰好进栈一次，故进栈操作执行次数为 n；其次分析入度减 1 操作，在无环情况下，当排序结束时，每个边恰好被逻辑删除一次，故入度减 1 操作的执行次数为 e。因此，本算法中基本操作执行次数为 n+n+e，因此时间复杂度为 O(n+e)。

答：拓扑排序算法的时间复杂度为 O(n+e)，其中 n 为图中顶点个数，e 为图中边的条数。

【例 7-7】 求图 7-26 所示的一个拓扑有序序列，要求写出求解步骤。

解：

① 由图 7-26 可知，入度为 0 的顶点为顶点 0。输出 0 并删除其出度，得一新图。

② 由新图可知，入度为 0 的顶点为顶点 1、3。输出 1 并删除其出度，得一新图。

③ 由新图可知，入度为 0 的顶点为顶点 2、3。输出 3 并删除其出度，得一新图。

④ 由新图可知，入度为 0 的顶点为顶点 2。输出 2 并删除其出度，得一新图。

⑤ 由新图可知，入度为 0 的顶点为顶点 5。输出 5 并删除其出度，得一新图。

⑥ 由新图可知，入度为 0 的顶点为顶点 4。输出 4 并删除其出度，得一新图。

⑦ 由新图可知，入度为 0 的顶点为顶点 6。输出 6 并删除其出度，顶点全部输出，拓扑排序过程结束。

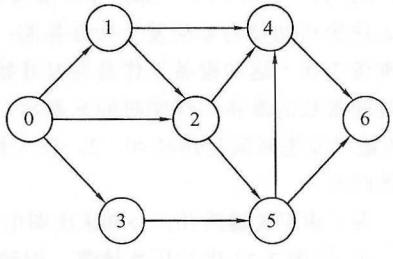

图 7-26 例 7-7 图

由以上步骤得拓扑有序序列为 0，1，3，2，5，4，6。

说明：对于拓扑排序这一部分，重点掌握手工进行拓扑排序的过程（本节开始时讲的 3 步操作）。对于拓扑排序算法代码无须像最短路径算法代码那样当成模板来记忆使用，只需能够根据手工排序过程写出适合自己的代码即可。