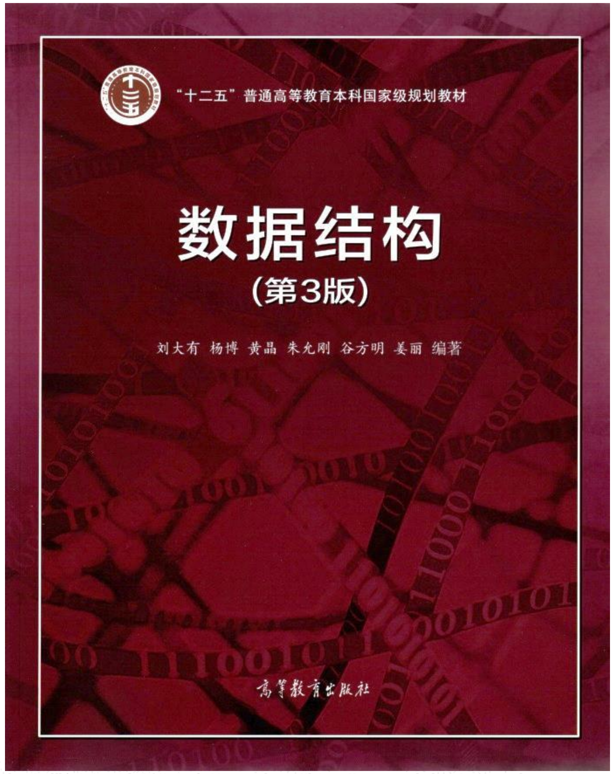
图的拓扑排序与关键路径



- 拓扑排序
- 关键路径

数据之法
结构之美
算法之道

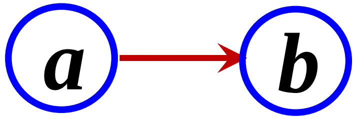
成功者是时间的主人

拓扑排序

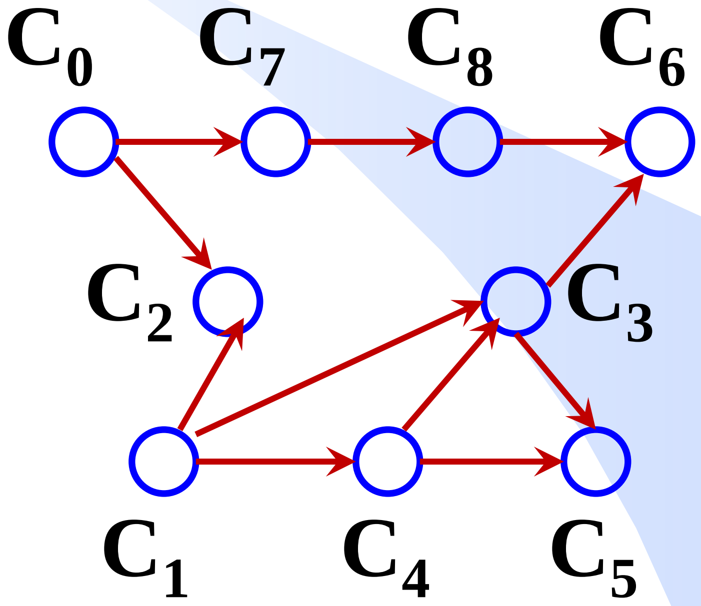
- 一个任务（例如一个工程）通常可以被分解成若干个子任务，要完成整个任务就可以转化为完成所有的子任务。
- 在某些情况下，各子任务之间有序，要求一些子任务必须先于另外一些子任务被完成。
- 各任务之间的先后关系可以用有向图来表示。

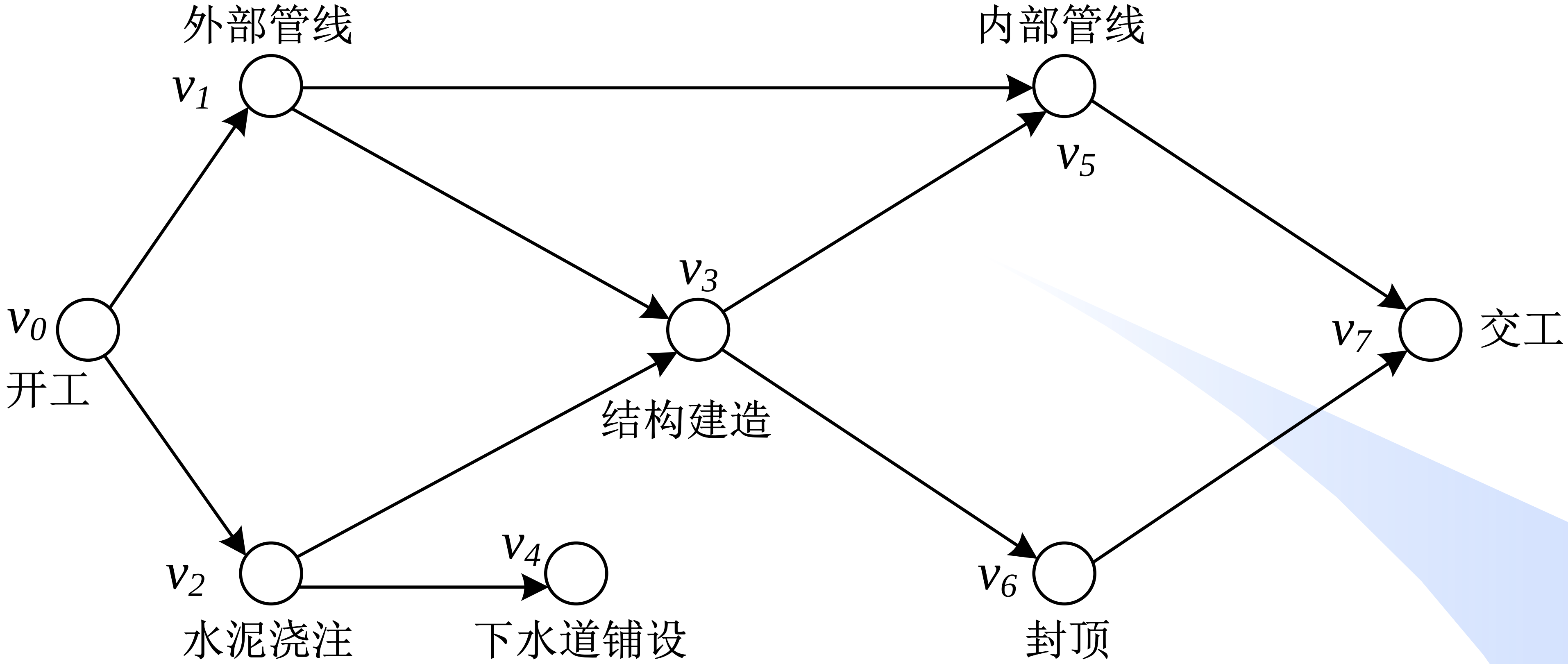
例：计算机专业学生的学习就是一个任务，每一门课程的学习就是整个任务的一个子任务。其中有些课程要求先修课程，有些则不要求。这样在有的课程之间有先后关系，有的课程可以并行地学习。

计算机专业部分课程



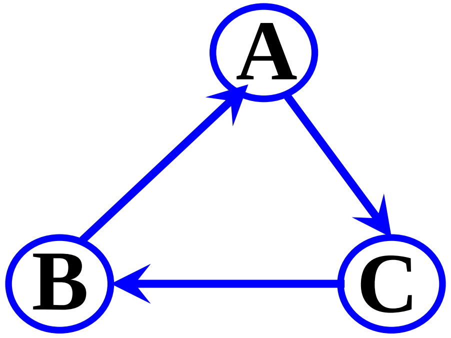
课程代号	课程名称	先修课程
C_0	高等数学	无
C_1	程序设计基础	无
C_2	离散数学	C_0,C_1
C_3	数据结构	C_1,C_4
C_4	C++语言	C_1
C_5	编译原理	C_3,C_4
C_6	操作系统	C_3,C_8
C_7	大学物理	C_0
C_8	计算机原理	C_7





例：建楼工程示意图，图中V_3 必须在V_1和V_2被完成后才能开始；V_4必须在V_2被完成后才能开始；V_5必须在V_1和V_3被完成后才能开始；V_6必须在V_3被完成后才能开始；V_5和V_6最后被完成时才能说整个工程可以交工。

- **AOV网:**在有向图中，**顶点表示活动**（或任务），有向边表示活动（或任务）间的先后关系，称这样的有向图为**AOV网(Activity On Vertex Network)**。
- 在AOV网络中，如果活动V_i必须在活动V_j之前进行，则存在有向边 $V_i \rightarrow V_j$.
- AOV网络中**不能出现有向回路**，即有向环。在AOV网络中如果出现了有向环，则意味着某项活动应以自己作为先决条件。



- **拓扑序列：**就是把AOV网中的所有顶点排成一个线性序列，**若AOV网中存在有向边 $V_i \to V_j$，则在该序列中，V_i 必位于 V_j 之前。**

在拓扑序列中，先进行的任务一定在后进行的任务的前面。按照拓扑序列完成各子任务，就可以顺利完成整个任务。

- **拓扑排序：**构造AOV网的拓扑序列的过程被称为拓扑排序。
- 如果通过拓扑排序能将AOV网的所有顶点都排入一个拓扑序列中，则该AOV网络中必定不会出现有向环；相反，如果不能把所有顶点都排入一个拓扑序列，**则说明AOV网络中存在有向环，**此AOV网络所代表的任务是不可行的。

拓扑排序算法基本步骤：

① 从图中选择一个入度为0的顶点并输出。

② 从图中删除该顶点及该顶点引出的所有边。

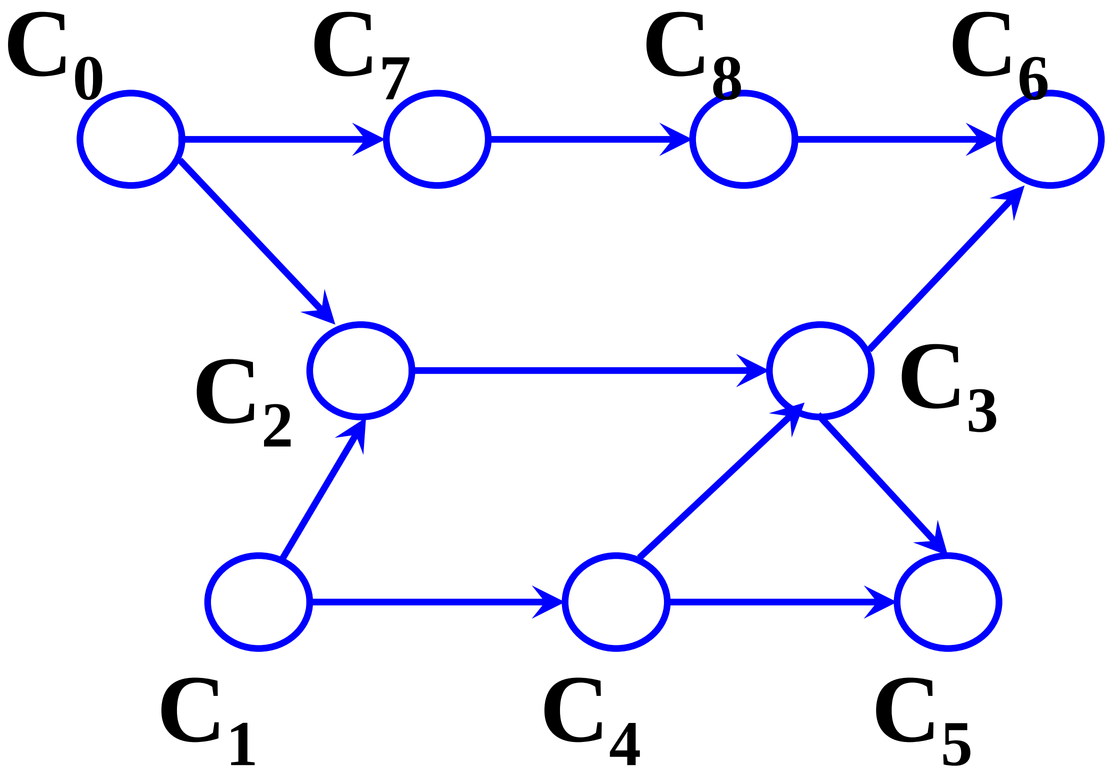
③ 执行①②，直至所有顶点已输出，或图中剩余顶点入度均不为0（说明存在环，无法继续拓扑排序）。

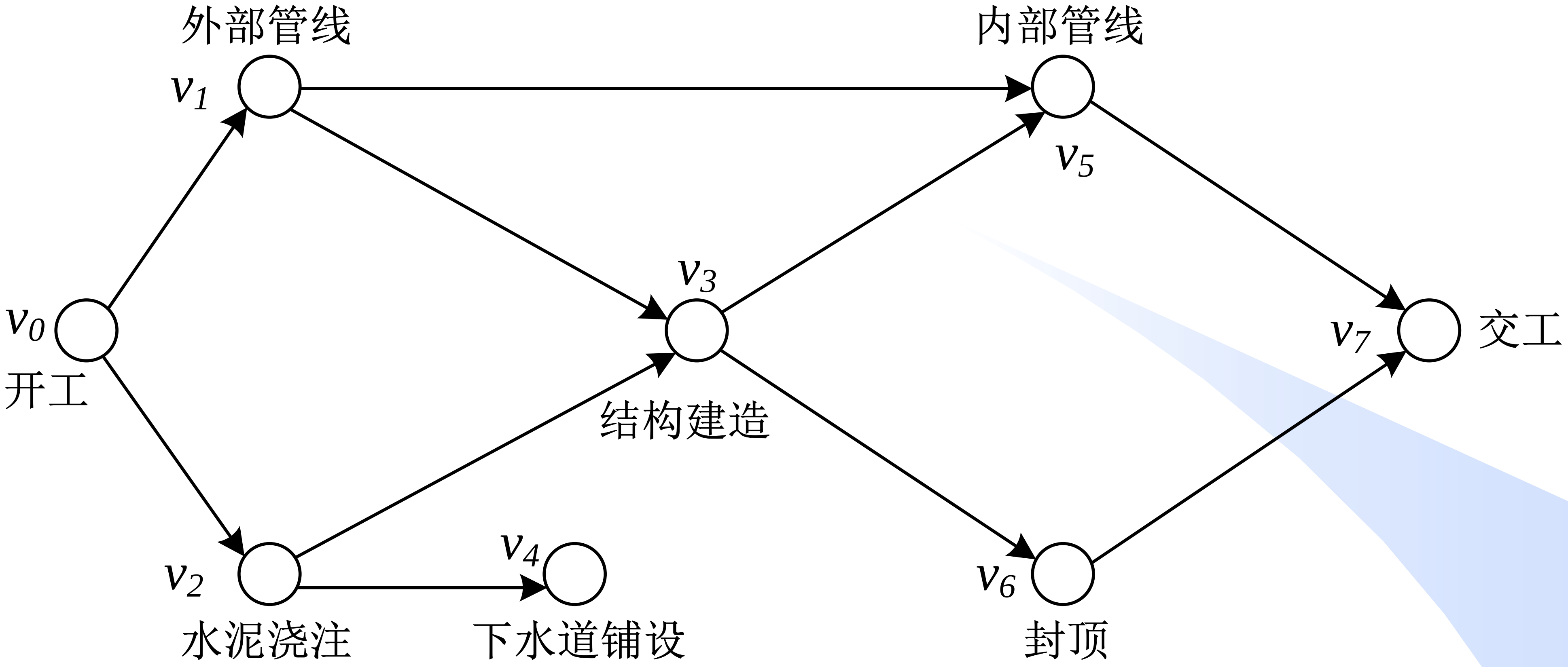
对于任何无环的AOV网，其顶点均可排成拓扑序列，其拓扑序列未必唯一。

例：对下图进行拓扑排序，得到的拓扑序列为

$C_0, C_1, C_2, C_4, C_3, C_5, C_7, C_8, C_6$

或 $C_0, C_7, C_8, C_1, C_4, C_2, C_3, C_6, C_5$ 等



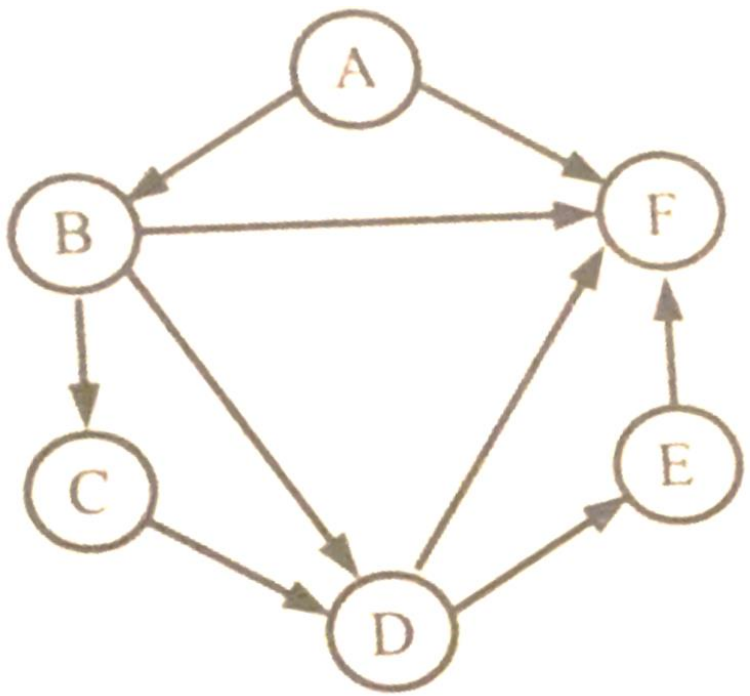


$V_0, V_1, V_2, V_4, V_3, V_5, V_6, V_7$ 和
$V_0, V_2, V_4, V_1, V_3, V_6, V_5, V_7$ 均是上图的拓扑序列。

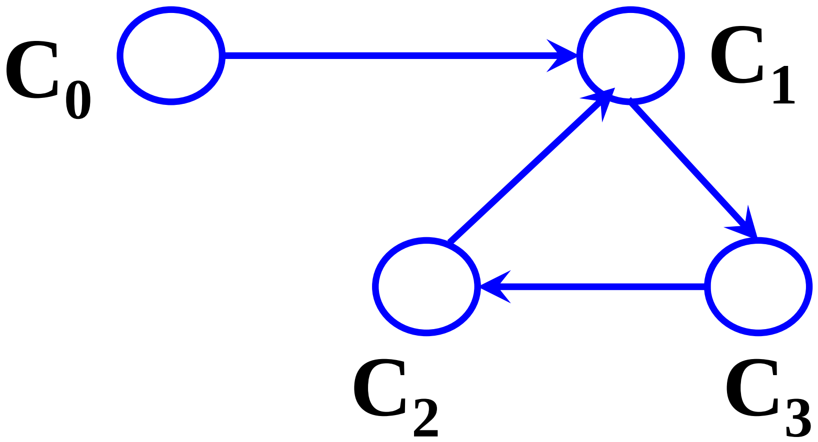
课下思考：

给定如下有向图，该图的拓扑序列的个数为_______【2021年考研题全国卷，2分】

A. 1　　B. 2　　C. 3　　D. 4



拓扑排序判断有向图中是否含有环



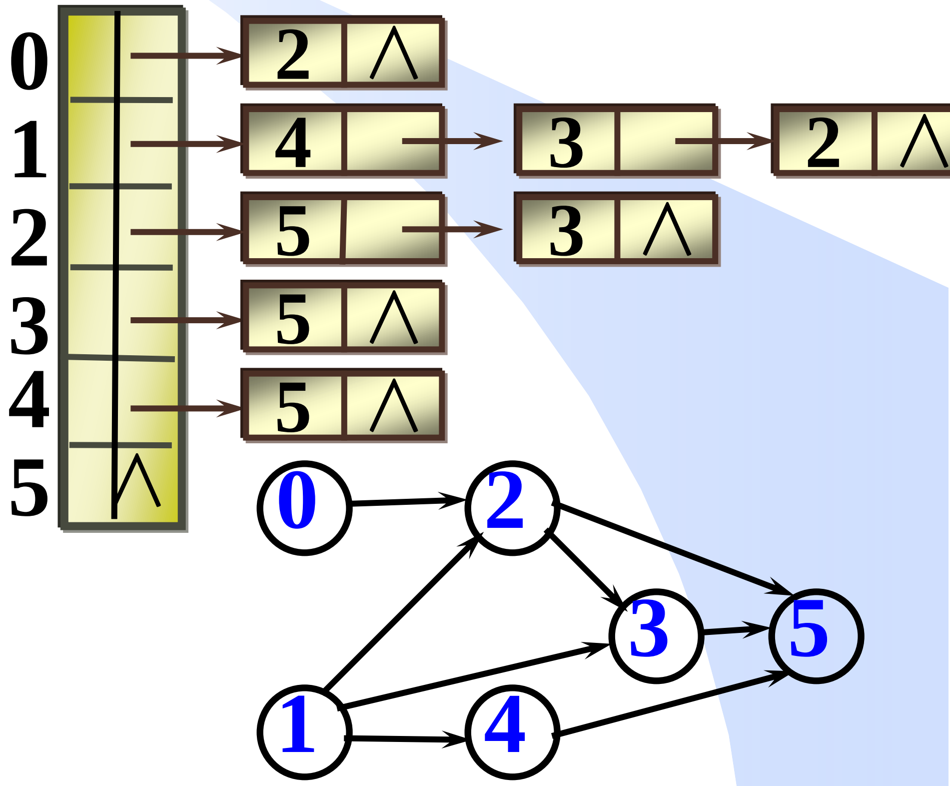
- 假定AOV网以邻接表的形式存储。为实现拓扑排序算法，事先需好两项准备工作：
- 建立一个数组count[ ]，count[*i*]为顶点*i*的入度；
- 建立一个栈，存放入度为0的顶点，每当一个顶点的入度为0，就将其压栈。

回顾：求每个顶点的入度

```
void InDegree(Vertex Head[],int n, int count[]) {
   for(int i=0;i<n;i++) count[i]=0;
   for(int i=0;i<n;i++){//用i扫描每个顶点
      Edge* p=Head[i].adjacent;
      while(p!=NULL){
         int k=p->VerAdj;
         count[k]++;
         p = p->link;
      }
   }
}
```

用p扫描每个顶点对应的边结点（邻接顶点）

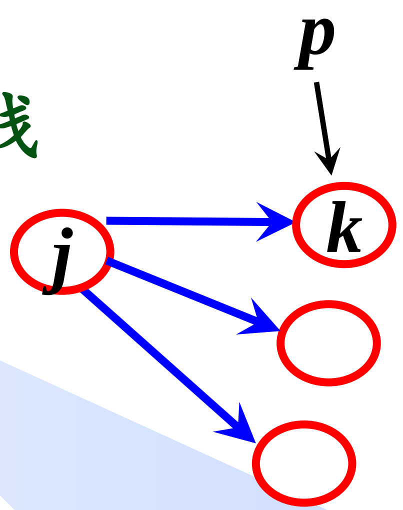
	0	1	2	3	4	5
count	0	0	2	2	1	3




```
for(Edge* p=Head[i].adjacent; p!=NULL; p=p->link )
```

时间复杂度为O(*n*+*e*)

```
bool TopoOrder(Vertex Head[], int n){
    int count[N]; Stack s;
    InDegree(Head, n, count);    //求每个顶点的入度
    for(int i = 0; i < n; i++)   //入度为0的顶点进栈
        if(count[i]==0) s.PUSH(i);
    for(int i = 0; i < n; i++){
        if(s.IsEmpty()) return false;
        //尚未输出n个顶点就没有入度为0的顶点了，说明有环
        int j=s.POP(); printf("%d ",j);//选出1个入度为0的顶点输出
        for(Edge *p=Head[j].adjacent; p!=NULL; p=p->link){
            //删除j和j引出的边，其效果是j的邻接顶点的入度减1
            int k=p->VerAdj;  count[k]--; //顶点k的入度减1
            if(count[k]==0) s.PUSH(k);
        }
    } return true;
}
```

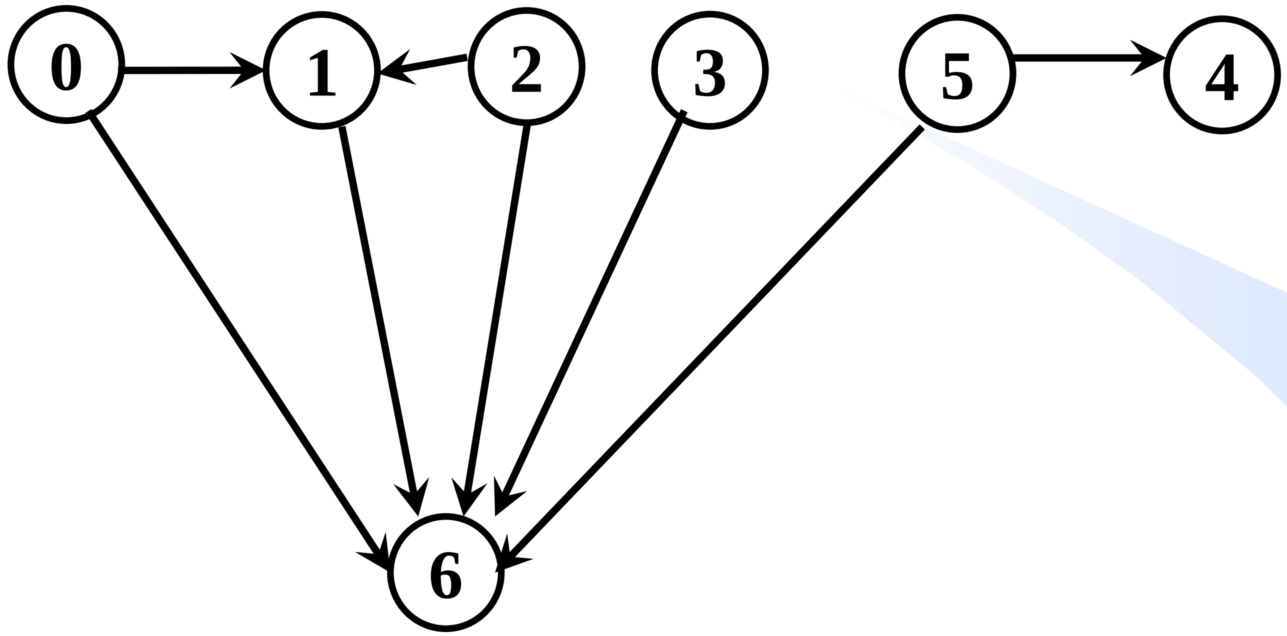


思考：如何判断拓扑序列是否唯一

```
bool TopoOrder(Vertex Head[], int n){
   int count[N];  int stack[N],top=-1;
   InDegree(Head, n, count);//求每个顶点的入度
   for(int i = 0; i < n; i++)//入度为0的顶点进栈
      if(count[i]==0) stack[++top]=i;
   for(int i = 0; i < n; i++){
      if(  top==-1  ) return false; //有环
      int j=stack[top--];
      printf("%d ",j);//选出1个入度为0的顶点输出
      for(Edge *p=Head[j].adjacent; p!=NULL; p=p->link){
         int k=p->VerAdj;  count[k]--; //顶点k的入度减1
         if(count[k]==0) stack[++top]=k;
      }
   } return true;
}
```

最简单写法用数组实现栈

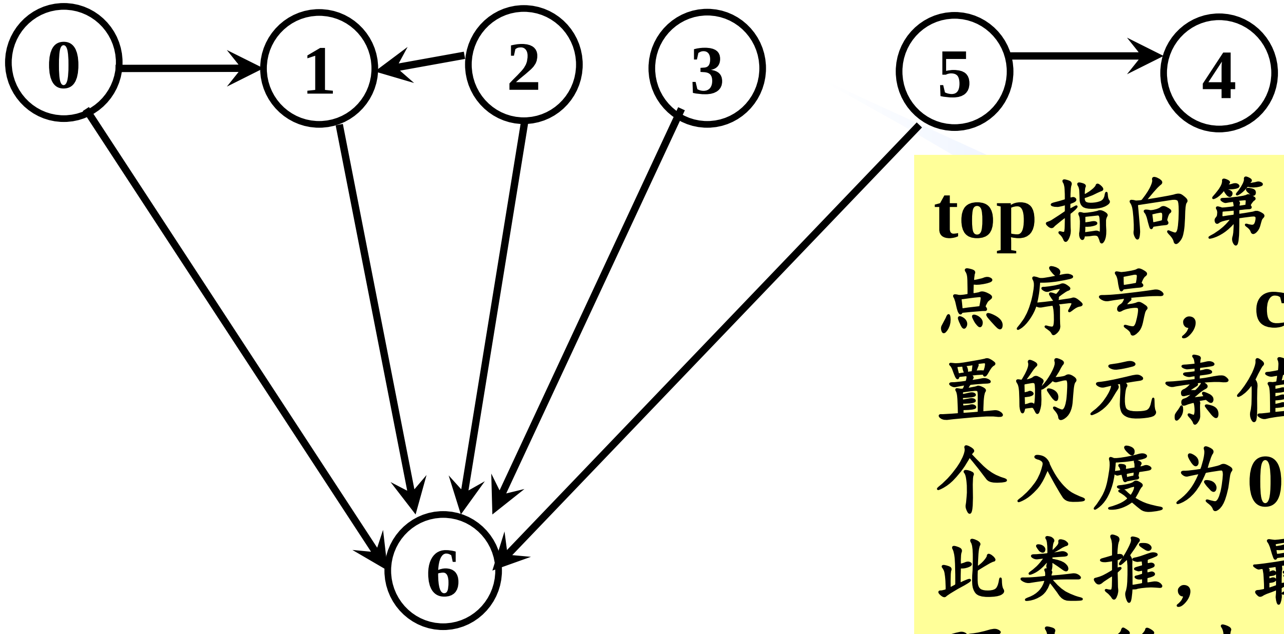
改进

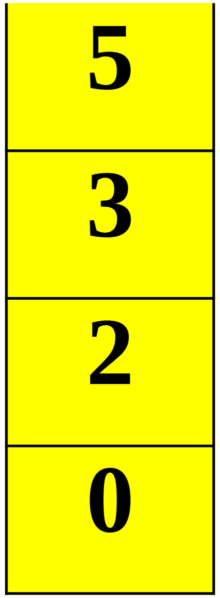


5
3
2
0

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

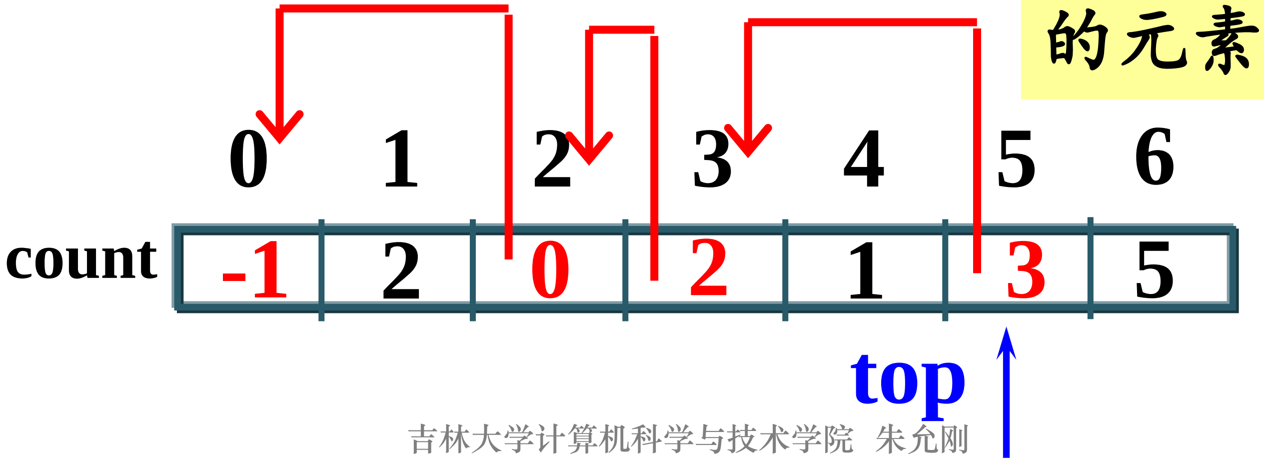
可以用count数组空闲的空间来组织栈





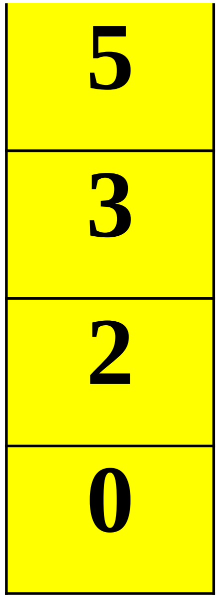
top指向第1个入度为0的顶点序号，count数组中该位置的元素值count[top]是第2个入度为0的顶点序号，依此类推，最后一个入度为0顶点所对应的count数组中的元素值为-1，表示栈底。

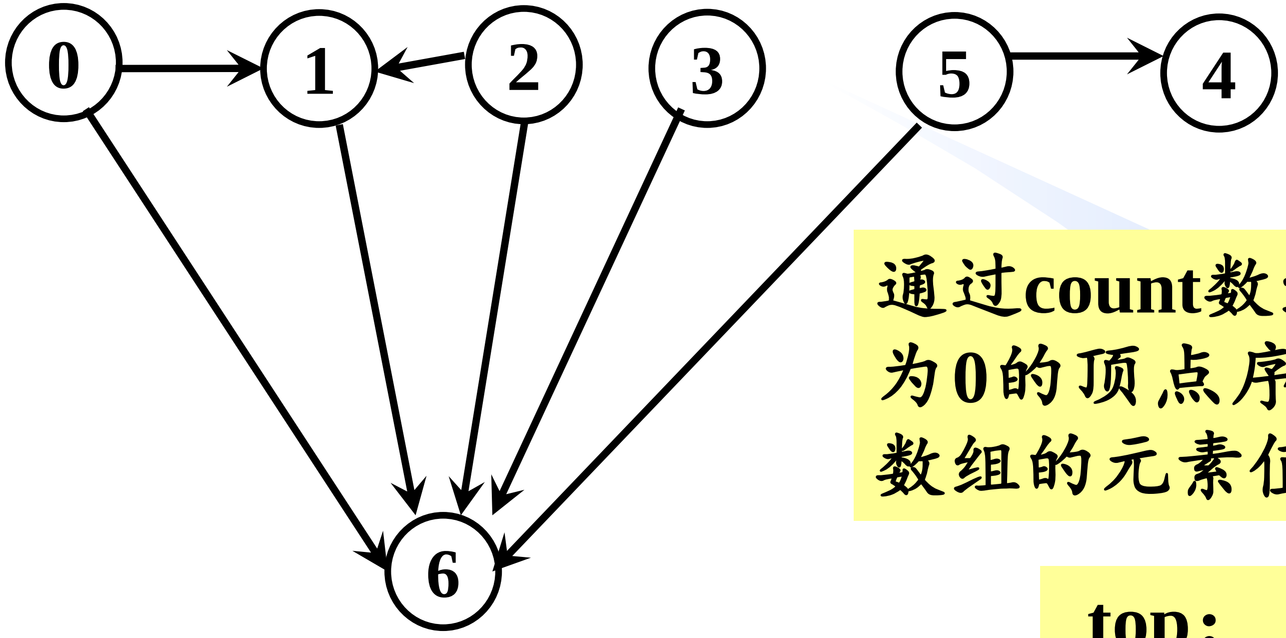
用静态链表实现的链式栈



top =5

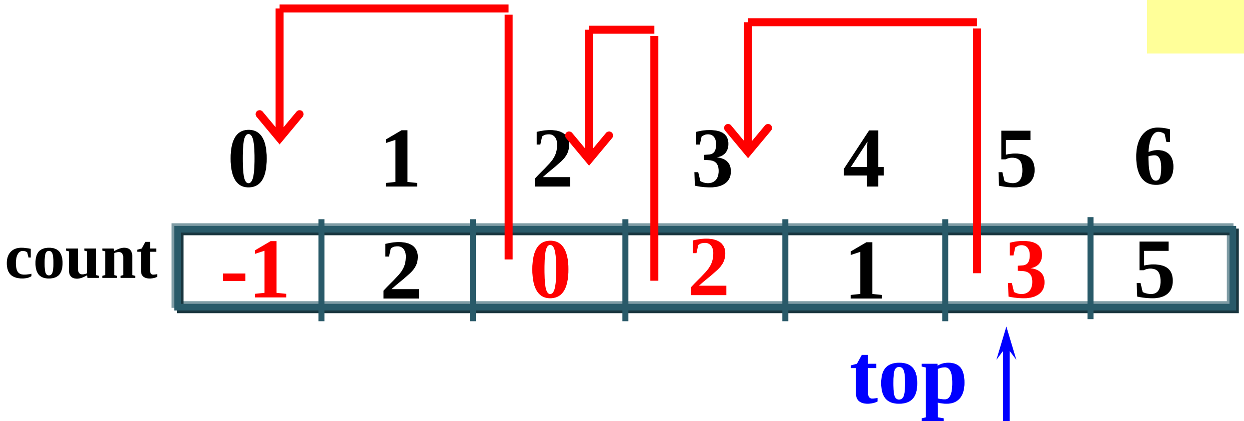
可以用count数组空闲的空间来组织栈



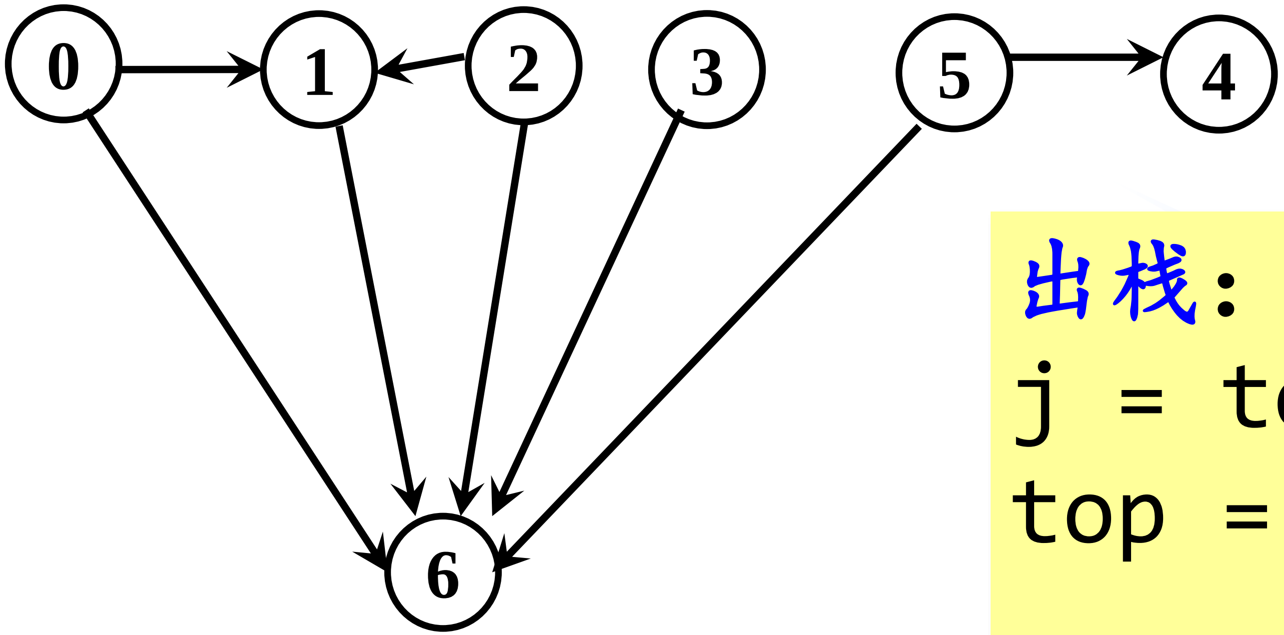


通过count数组的下标找入度为0的顶点序号。通过count数组的元素值来链接栈。

top：“栈顶”位置初始为-1



top =5

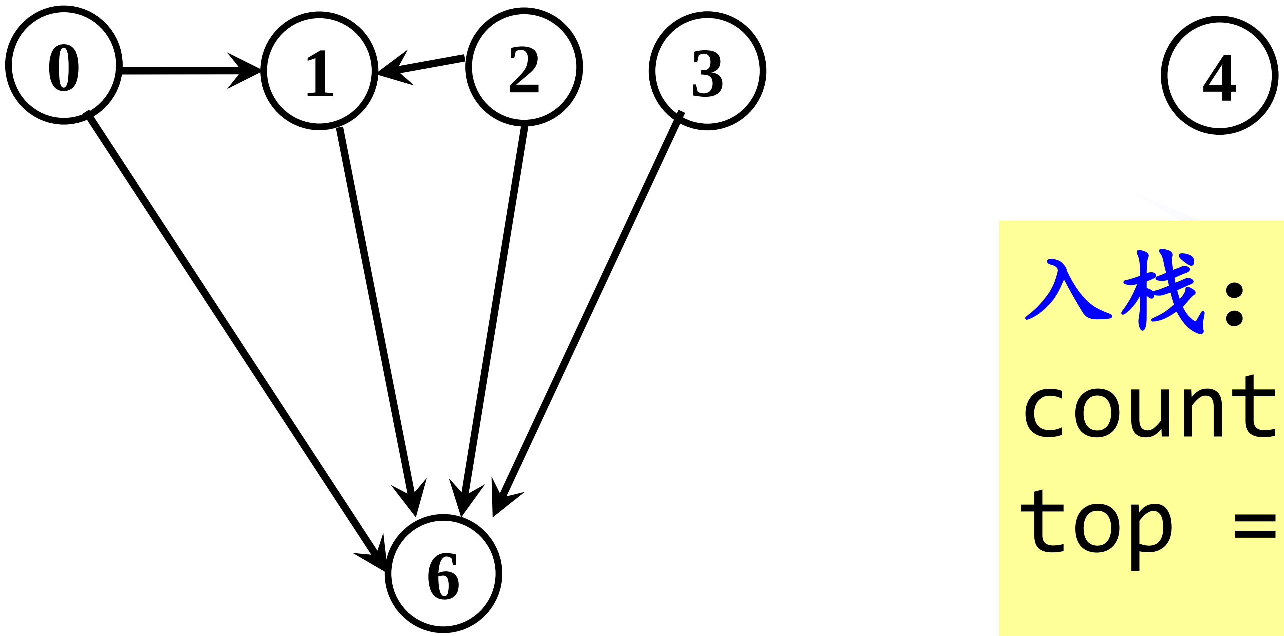


出栈:

```
j = top;
top = count[top];
```

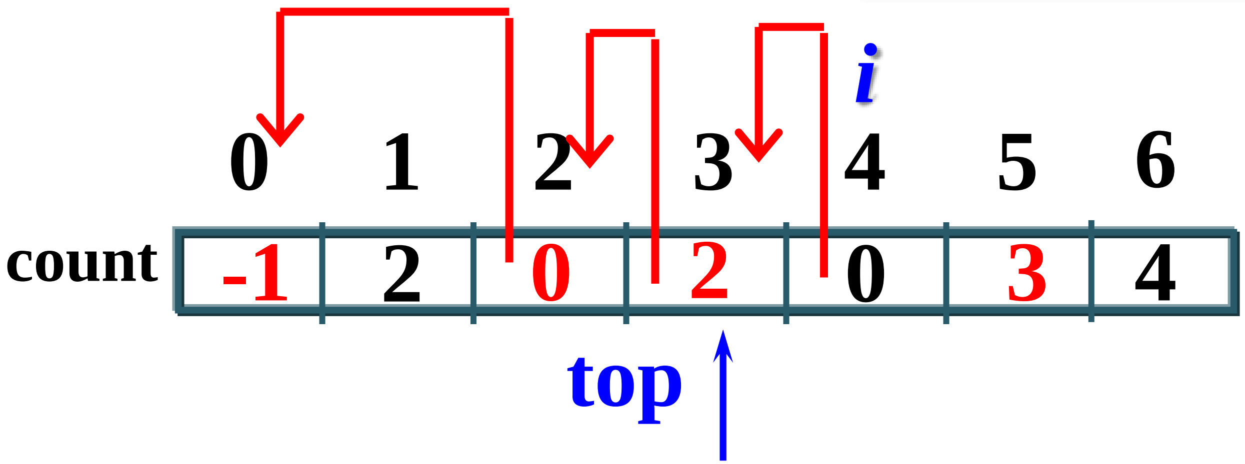
	0	1	2	3	4	5	6
count	-1	2	0	2	1	3	5

top



入栈:

```
count[i] = top;
top = i;
```



```
bool TopoOrder(Vertex Head[], int n){
   int count[N]; int top=-1;
   InDegree(Head, n, count);   //求每个顶点的入度
   for(int i = 0; i < n; i++)  //入度为0的顶点进栈
      if(count[i]==0) {count[i]=top;top=i;}
   for(int i = 0; i < n; i++){
      if(  top==-1  ) return false; //有环
      int j=top; top=count[top];
      printf("%d ",j);//选出1个入度为0的顶点输出
      for(Edge *p=Head[j].adjacent; p!=NULL; p=p->link){
         int k=p->VerAdj;  count[k]--; //顶点k的入度减1
         if(count[k]==0) {count[k]=top;top=k;}
      }
   } return true;
}
```

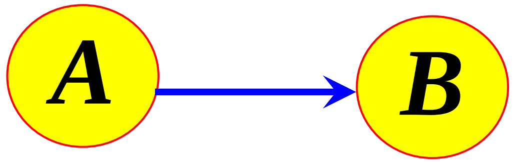
压栈：
count[i]=top;
top=i;

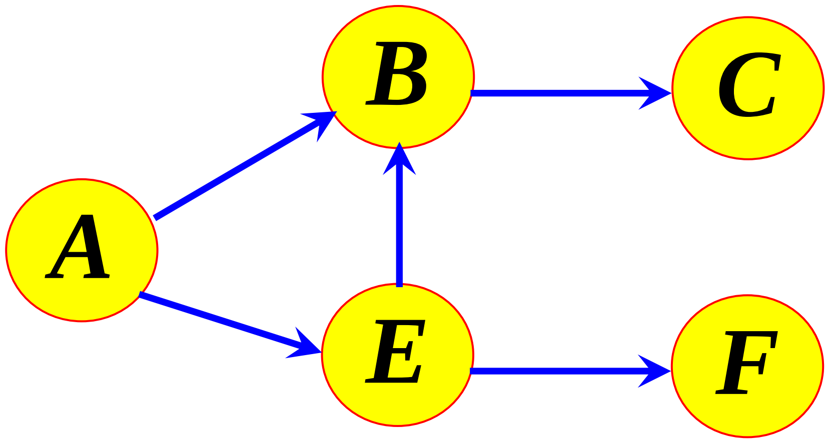
弹栈：
j=top;
top=count[top];

给定一个图和顶点序列，编写算法判断该序列是否是图的拓扑序列。【北京航空航天大学考研题】

- 扫描序列中的每个顶点，在图中看其入度是否为0：
  - ✓若入度不为0，则非拓扑序列，算法退出。
  - ✓若入度为0，则在图中删去该顶点及其引出的边，继续扫描。

深度优先遍历实现拓扑排序

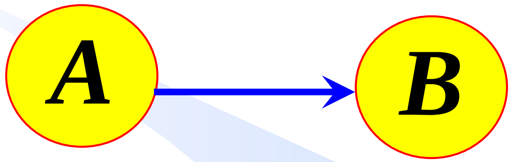




DFS不能保证A一定在B之前输出，但能保证B一定在A之前输出。即DFS很难输出拓扑序的正序,但可以输出拓扑序的逆序。

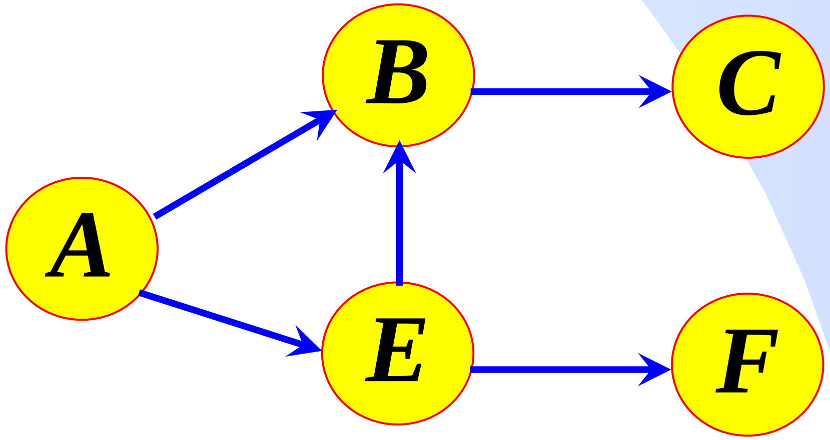
深度优先遍历生成逆拓扑序

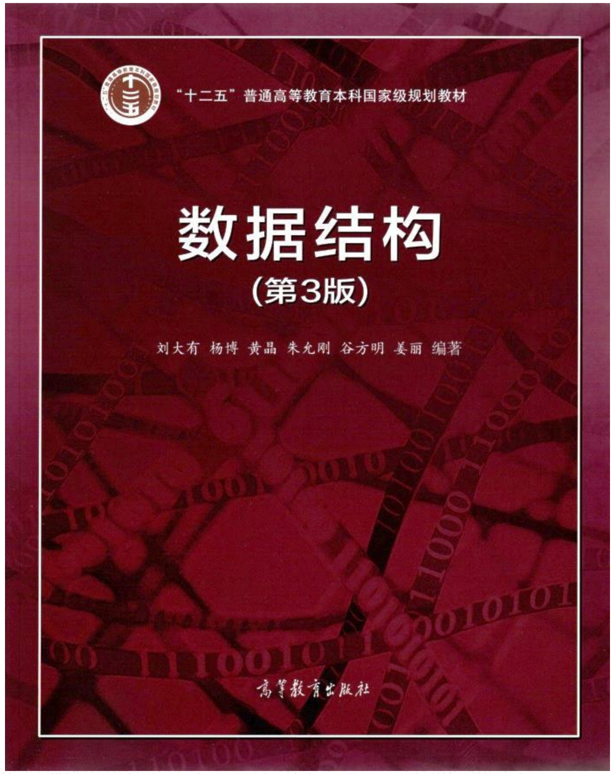
```
void DFS_TopoSort(Vertex*Head, int v, int vis[]){
     vis[v]=1; //访问顶点v
     Edge* p= Head[v].adjacent;
     while(p!=NULL){
        if(vis[p->VerAdj]==0)
           DFS_TopoSort(Head,p->VerAdj,vis);
        p=p->link;
     }
     printf("%d ",v);
}
```



初始调用

```
for(int i=0;i<n;i++)
   if(vis[i]==0)
      DFS_TopoSort(Head, i, vis);
```





图的拓扑排序与关键路径

- 拓扑排序
- **关键路径**

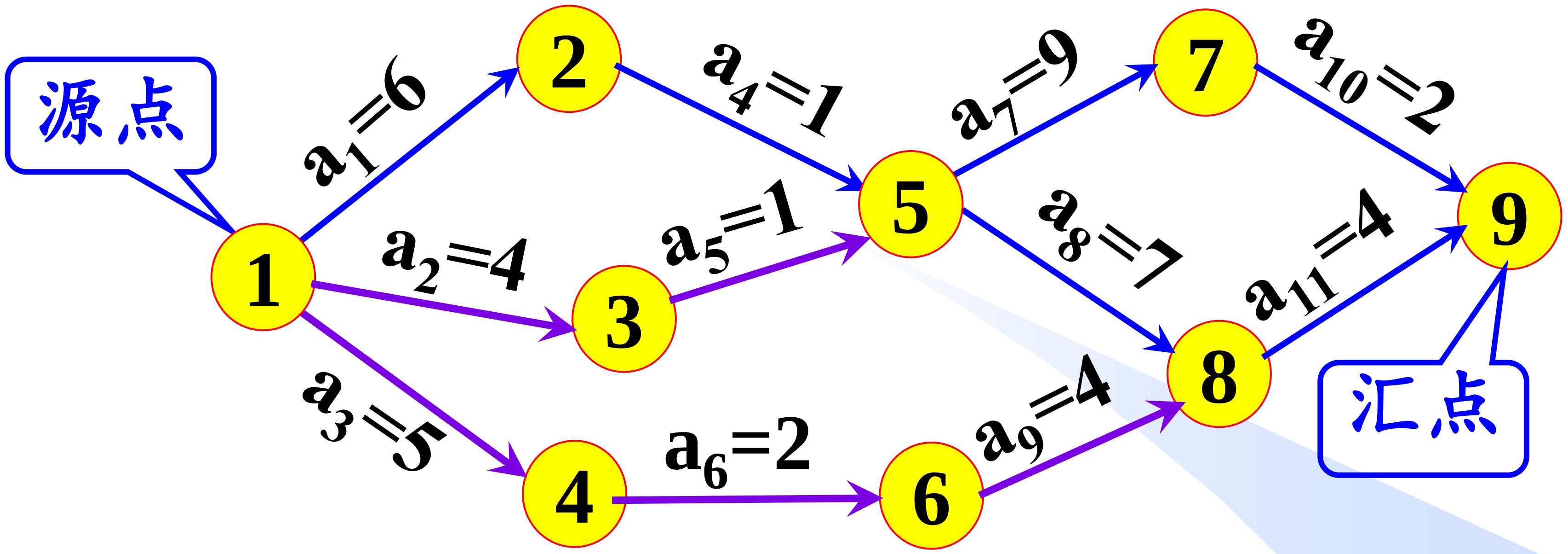
数据结构之法
结构之美
算法之道

吉林大学计算机科学与技术学院 朱允刚
zhuyungang@jlu.edu.cn

关键路径

- AOV网(Activity On Vertex):顶点表示活动或任务(Activity)，有向边表示活动（或任务）间的先后关系。
- AOE网(Activity On Edges):有向边表示活动或任务(Activity)，用边上的权值表示活动的持续时间，顶点称为事件(Event):表示其入边的任务已完成，出边的任务可开始的状态。

[例] 某工程



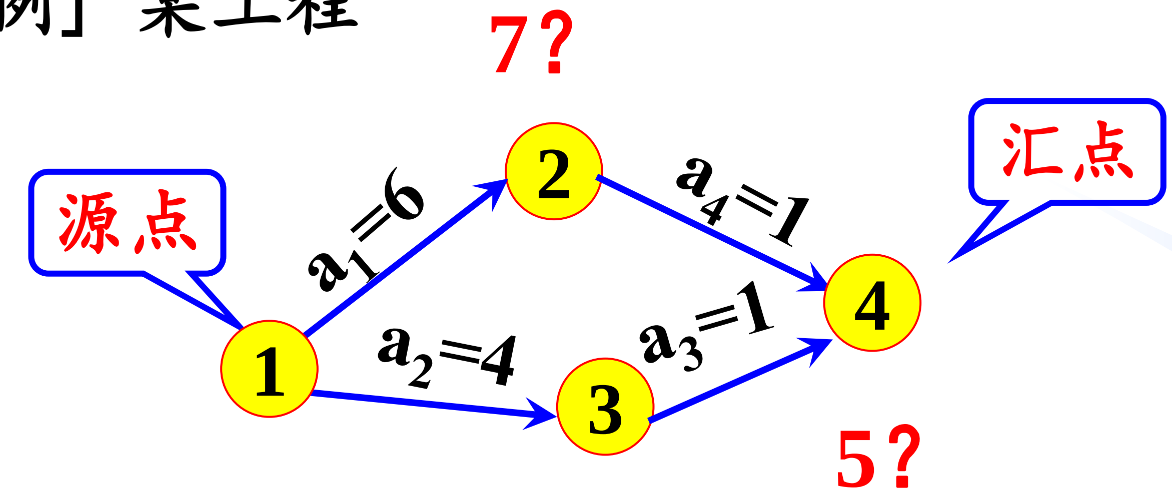
- 源点： 表示整个工程的开始(入度为0).
- 汇点： 表示整个工程的结束(出度为0).

✓ 完成整个工程至少需要多少时间？
✓ 哪些活动不能延期，否则将会影响整个工程进度？
✓ 在不整个工程进度的情况下，哪些活动可以适当延期？

- 在AOE网络中，有些活动可以并行进行，但有些活动必须顺序进行。
- 从源点到各个顶点，以至从源点到汇点的路径可能不止一条。这些路径的长度也可能不同。
- 只有各条路径上所有活动都完成了，整个工程才算完成。
- 因此，完成整个工程所需的最短时间取决于从源点到汇点的最长路径长度，即在这条路径上所有活动的持续时间之和。这条路径长度最长的路径就叫做关键路径(Critical Path)。

路径长度：路径上的各边权值之和

[例] 某工程



完成工程所需的最短时间?

- 关键路径：从源点到汇点的最长路径。
- 关键活动：关键路径上的活动。

与关键活动有关的量

① 事件 v_j 的最早发生时间 $ve(j)$

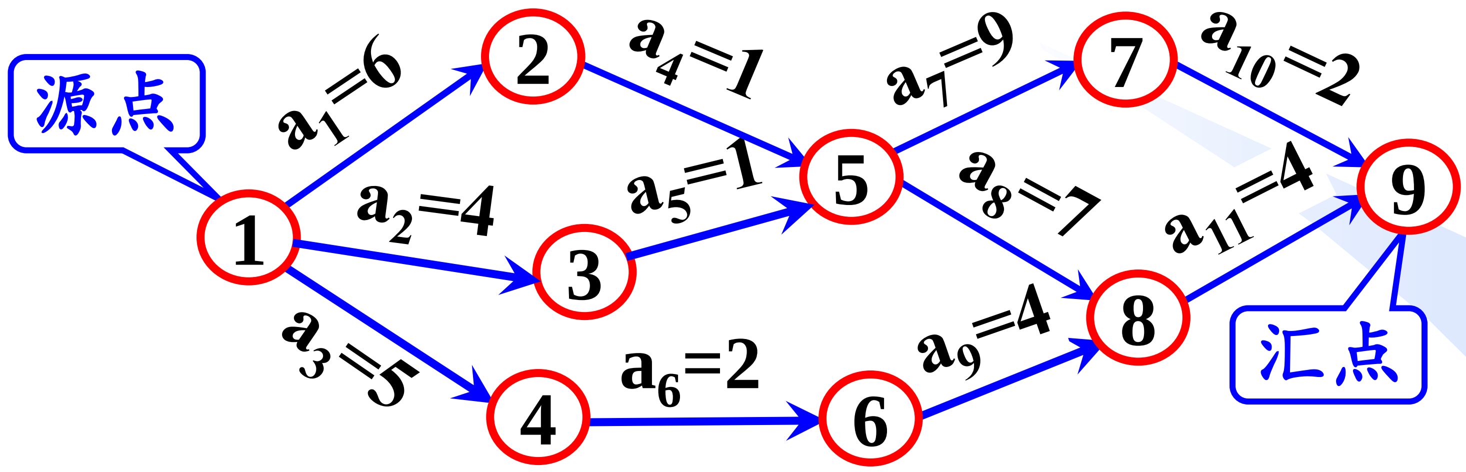
② 事件 v_j 的最迟发生时间 $vl(j)$

③ 活动 a_i 的最早开始时间 $e(i)$

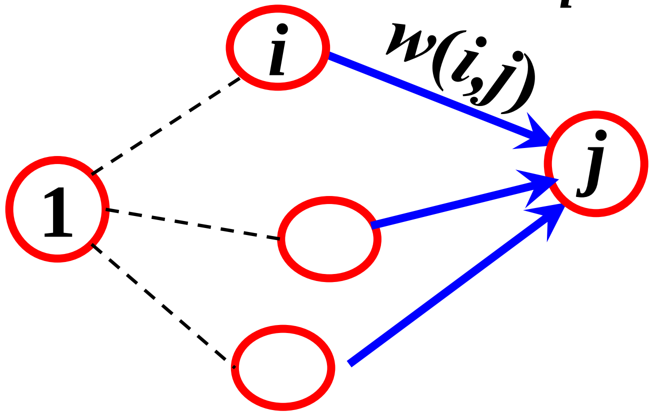
④ 活动 a_i 的最迟开始时间 $l(i)$

① 事件v_j的最早发生时间 $ve(j)$

从源点 v_1 到 v_j 的最长路径的长度。显然有$ve(1)=0$



$$ve(j)=\begin{cases} 0, & j=1 \\ \max_i \{ve(i)+w(<i,j>) \mid <i,j> \in E(G), j=2,\ldots,n\} \end{cases}$$

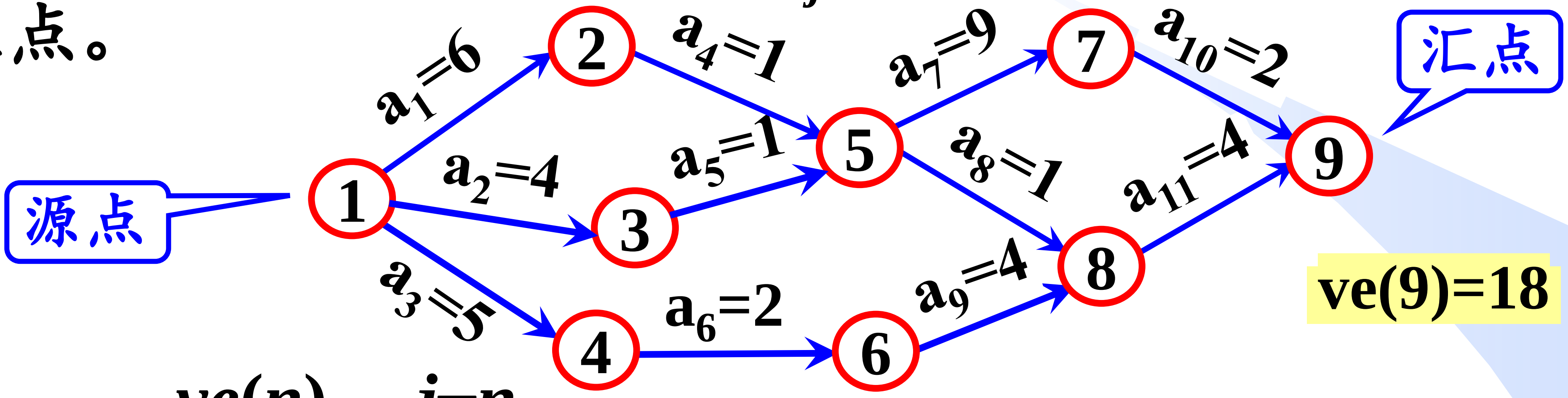


计算$ve(j)$需要已知顶点v_j的所有前驱顶点的最早发生时间。

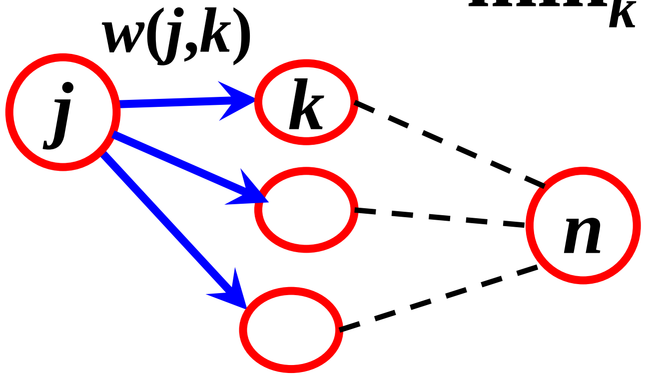
如何保证？先对AOE网进行拓扑排序，然后按拓扑序递推。

② 事件v_j的最迟发生时间 $vl(j)$

在保证汇点的最早发生时间不推迟的前提下，事件 v_j 允许的最迟开始时间，等于$ve(n)$减去 v_j 到 v_n 的最长路径长度，其中v_n为汇点。



$$vl(j)=\begin{cases} ve(n), & j=n \\ \min_k\{vl(k)-w(<j,k>) \mid <j,k>\in E(G), j=n-1,\dots,1\} \end{cases}$$

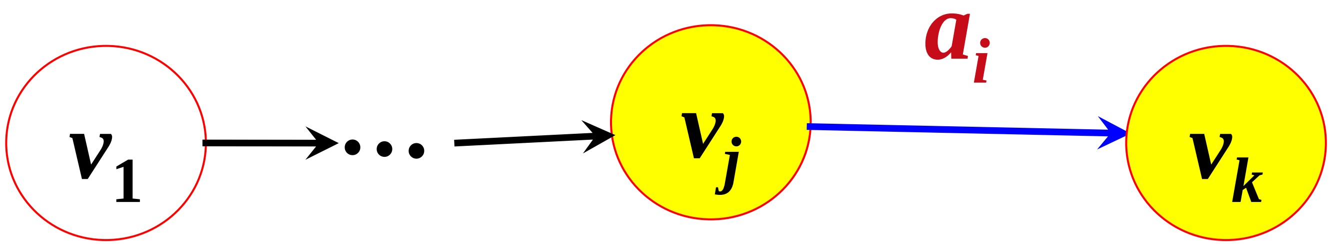


计算$vl(j)$需要已知顶点v_j的所有后继顶点的最晚发生时间。

如何保证？先对AOE网进行拓扑排序，然后按拓扑逆序递推

③ 活动 a_i 的最早开始时间 $e(i)$

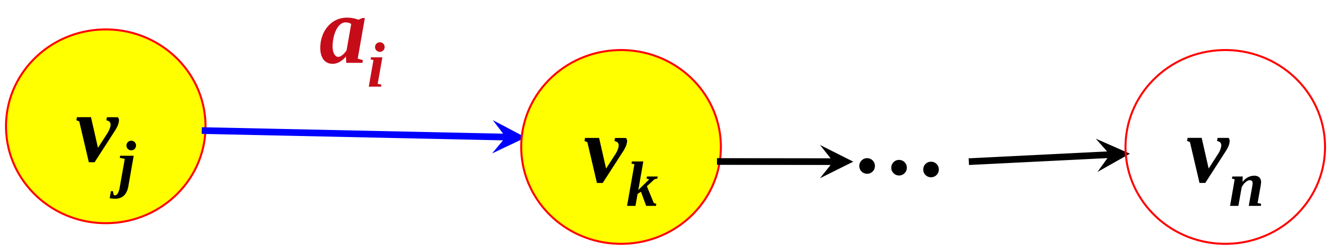
设活动 a_i 在有向边 $v_j \rightarrow v_k$ 上，则 $e(i) = ve(j)$.



④ 活动a_i的最迟开始时间$l(i)$

不会引起时间延误的前提下，活动a_i允许的最迟开始时间。设活动a_i在有向边$<v_j, v_k>$上，则

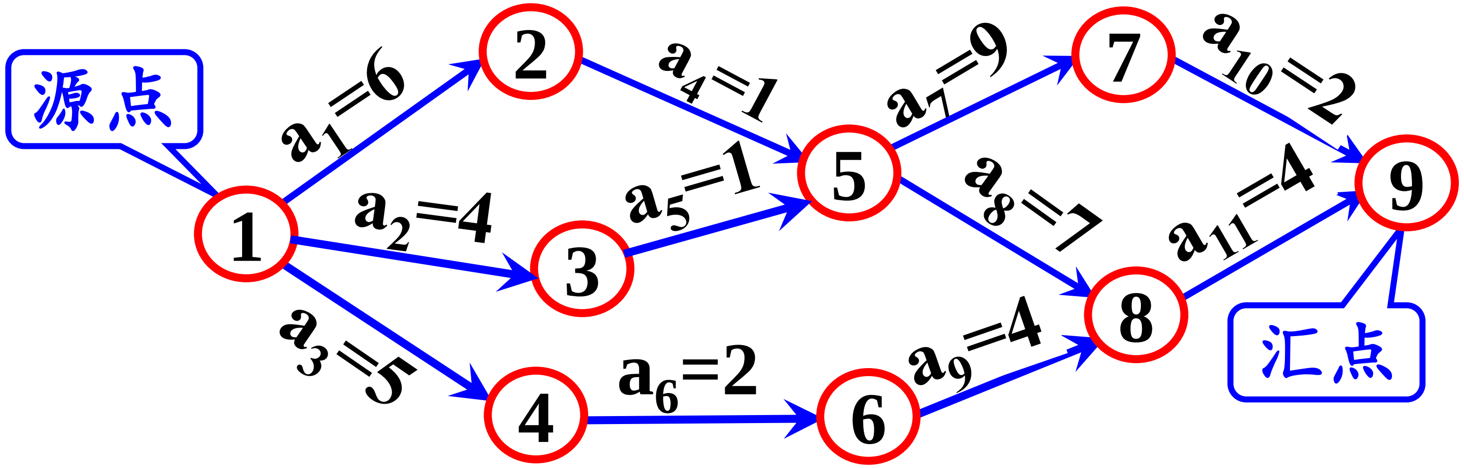
$$l(i) = vl(k) - weight(<j, k>)$$



关键路径与关键活动

关键路径： 从源点到汇点的最长路径。

关键活动： 关键路径上的活动，活动的最早开始时间等于活动的最迟开始时间，即$l(i)=e(i)$.



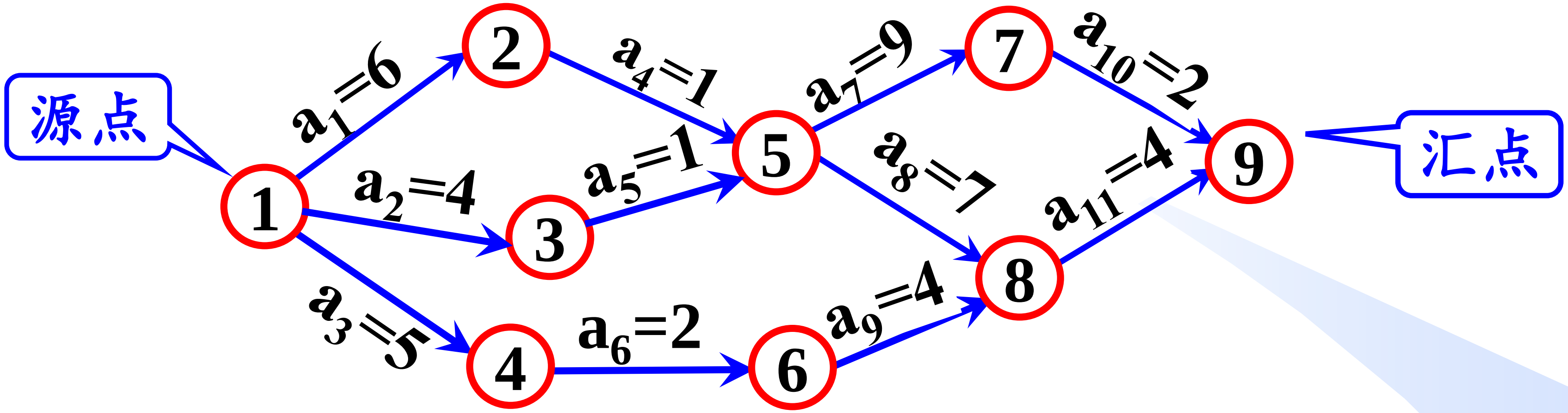
求关键活动的步骤

① 对AOE网进行拓扑排序，按顶点拓扑序求各顶点v_j的最早发生时间 $ve(j)$；

② 按顶点的逆拓扑序求各顶点 v_j 的最迟发生时间$vl(j)$；

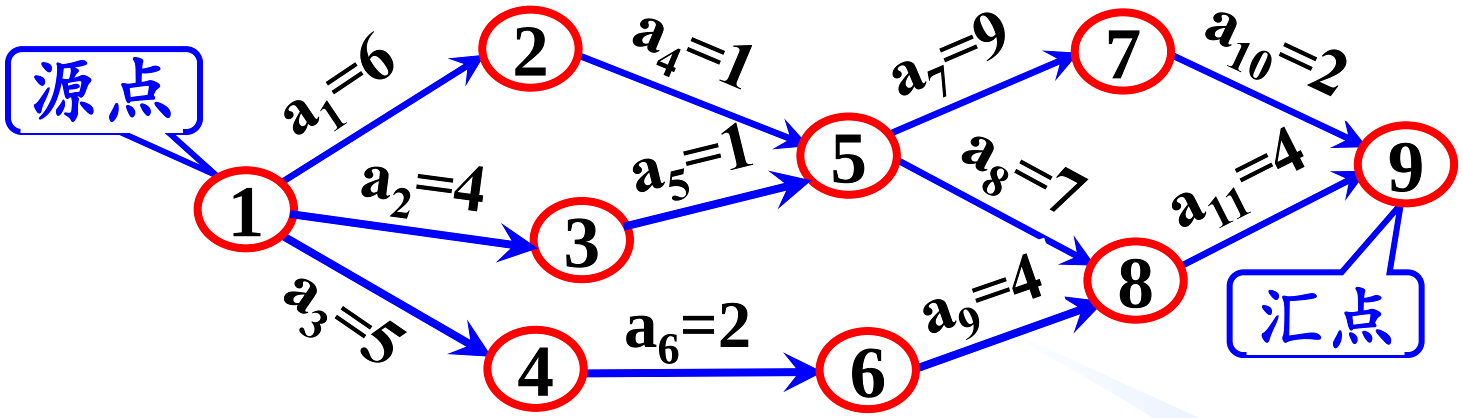
③ 根据各顶点ve和vl值，求出各活动 a_i 的最早开始时间 $e(i)$和最迟开始时间$l(i)$，若 $e(i)= l(i)$，则a_i是关键活动。

例：求下图的关键活动和关键路径



	1	2	3	4	5	6	7	8	9
ve									
vl									

例



$ve(1)=0$

$ve(2)= ve(1)+weight(<1, 2>)=0+6=6$

$ve(3)= ve(1)+weight(<1, 3>)=0+4=4$

$ve(4)=ve(1)+weight(<1, 4>)=0+5=5$

$ve(5)= \max\{ve(2)+weight(<2,5>),\ ve(3)+weight<3,5>\}=\max\{6+1, 4+1\}=7$

$ve(6)= ve(4)+weight(<4, 6>)=5+2=7$

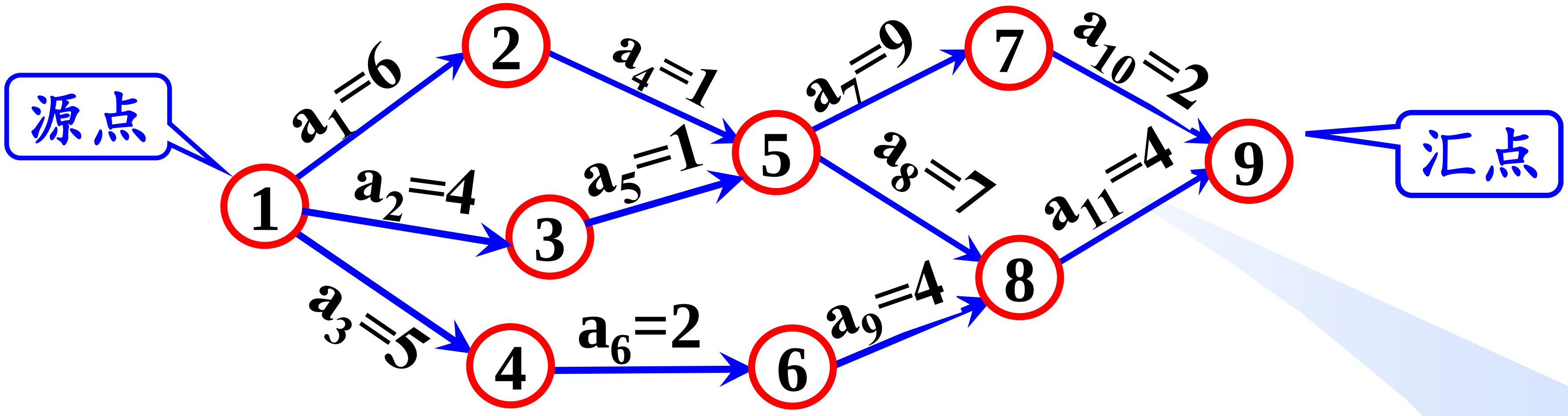
$ve(7)= ve(5)+weight(<5, 7>)=7+9=16$

$ve(8)=\max\{ve(5)+weight(<5,8>), ve(6)+weight(<6,8>)\}=\max\{7+7,7+4\}=14$

$ve(9)=\max\{ve(7)+weight(<7,9>),ve(8)+weight(<8,9>)\}=\max\{16+2,14+4\}=18$

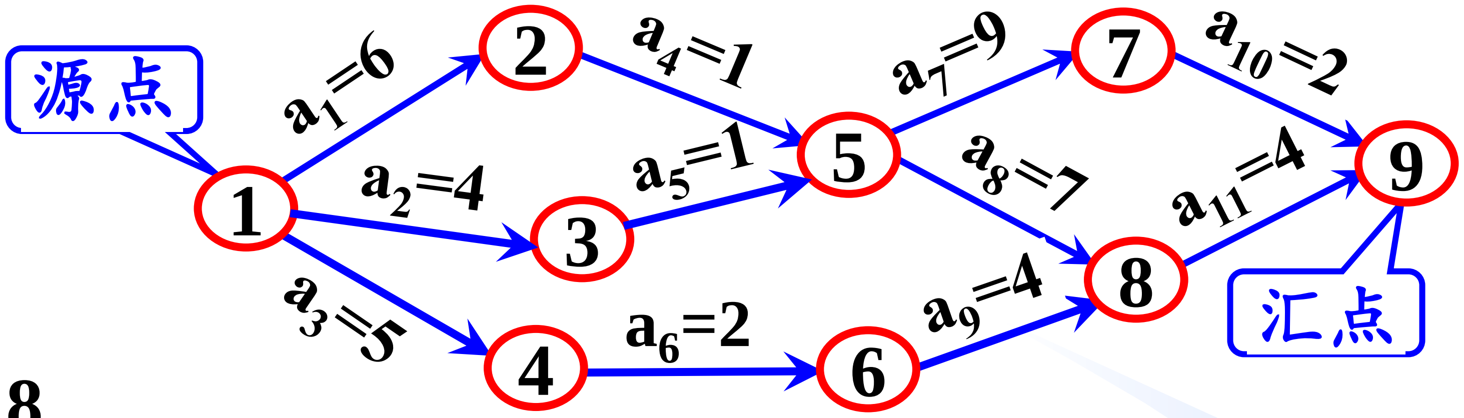
$$ve(j)= \begin{cases} 0, & j=1 \\ \max_i \{ve(i)+ w(<i, j>) \mid <i, j> \in E(G), j=2,\ldots,n\} \end{cases}$$

例：求下图的关键活动和关键路径



	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	16	14	18
vl									

例



vl(9)= ve(9)=18

vl(8)= vl(9)-weight(<8, 9>)=18-4=14

vl(7)= vl(9)-weight(<7, 9>)=18-2=16

vl(6)= vl(8)-weight(<6, 8>)=14-4=10

vl(5)= min{vl(8)-weight(<5, 8>), vl(7)-weight(<5 ,7>)}=min{14-7,16-9}=7

vl(4)= vl(6)-weight(<4,6>)=10-2=8

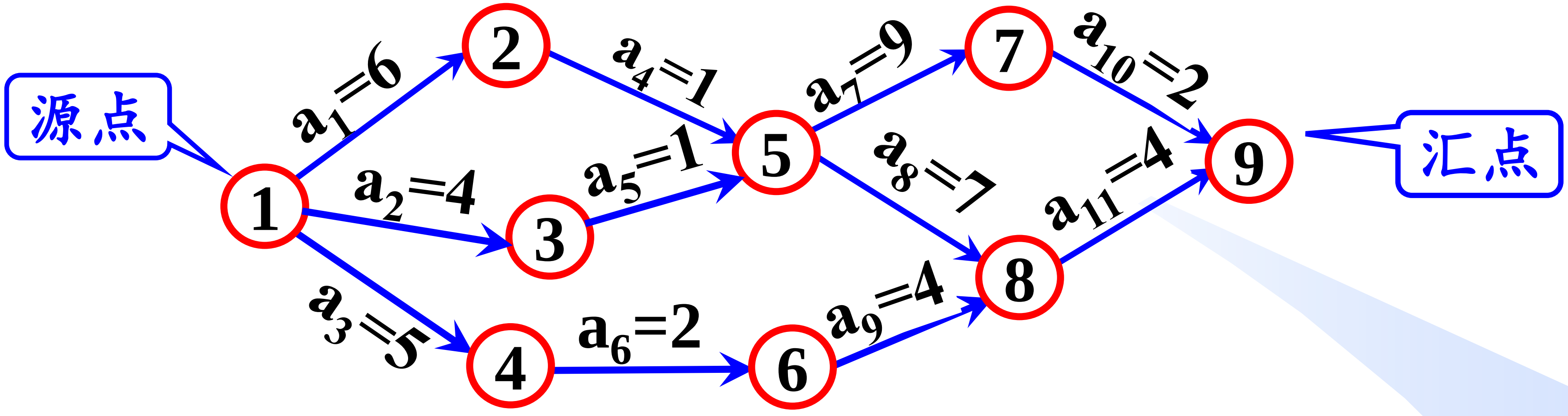
vl(3)= vl(5)-weight(<3,5>)=7-1=6

vl(2)= vl(5)-weight(<2,5>)=7-1=6

vl(1)= min{vl(2)-weight(<1,2>), vl(3)-weight(<1,3>), vl(4)-weight(<1,4>)}=0

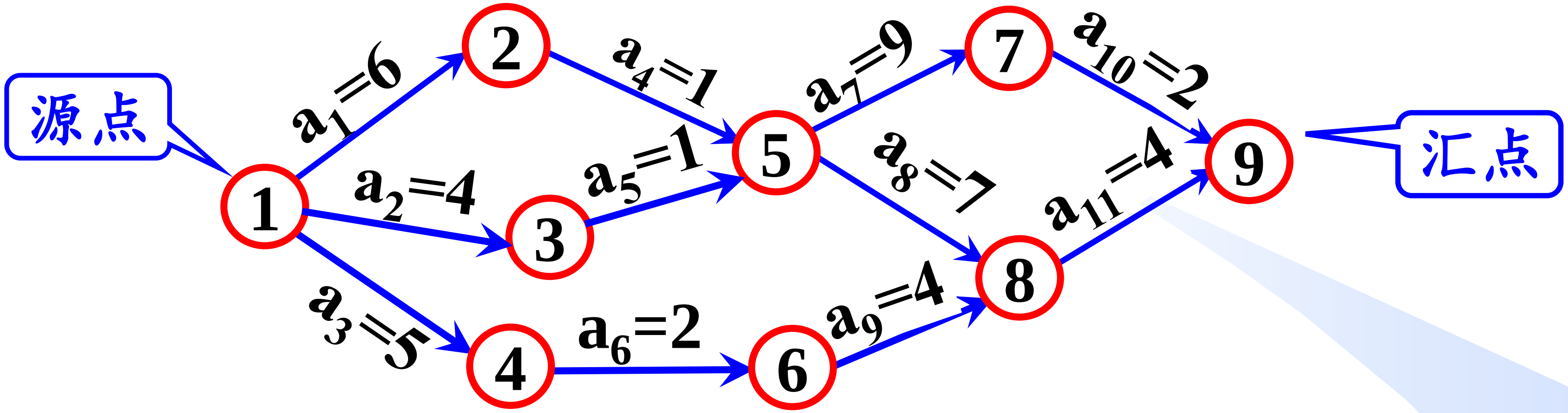
$$vl(j)=\begin{cases} ve(n), & j=n \\ \min_k\{vl(k)-w(<j,k>) \mid <j,k>\in E(G), j=n-1,\ldots,1\} \end{cases}$$

例：求下图的关键活动和关键路径



	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	16	14	18
vl	0	6	6	8	7	10	16	14	18

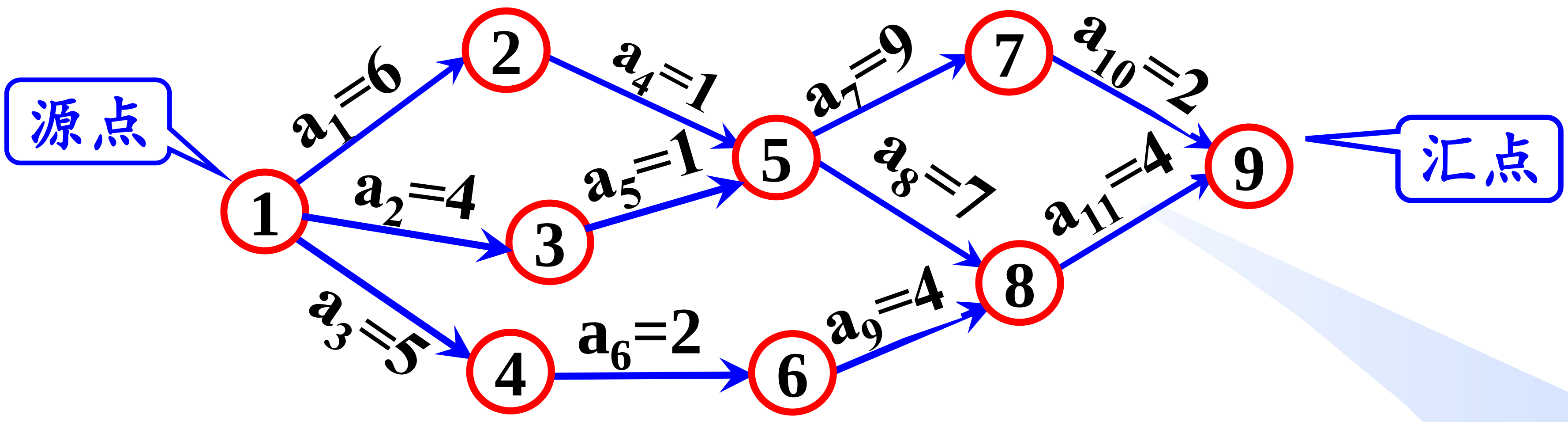
例：求下图的关键活动和关键路径



	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	16	14	18
vl	0	6	6	8	7	10	16	14	18

a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
$e(i)$											
$l(i)$											

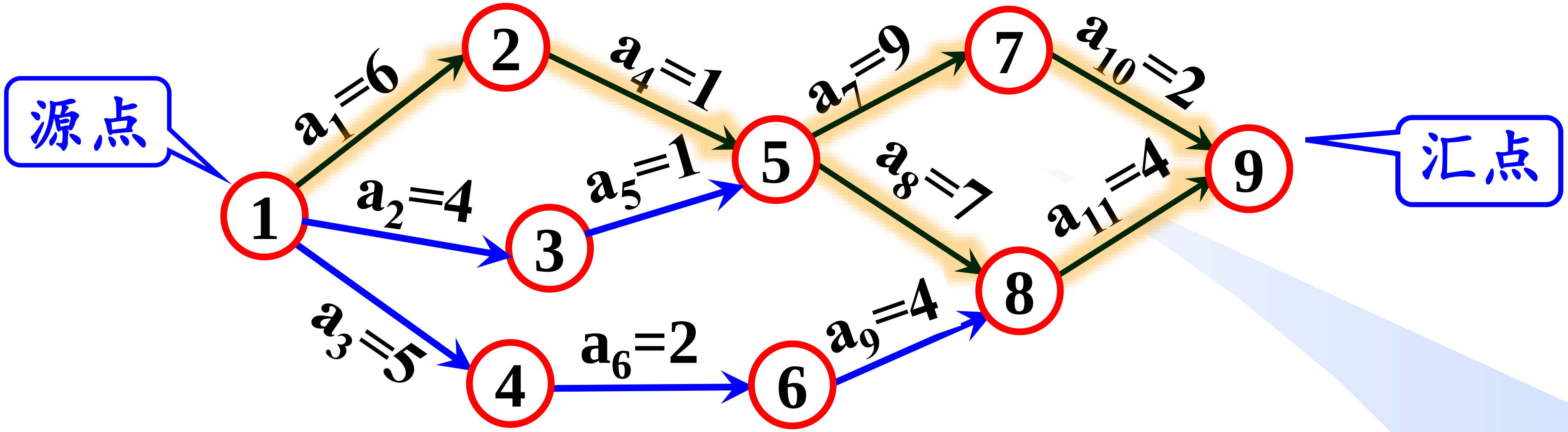
例：求下图的关键活动和关键路径



	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	16	14	18
vl	0	6	6	8	7	10	16	14	18

a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
$e(i)$	0	0	0	6	4	5	7	7	7	16	14
$l(i)$	0	2	3	6	6	8	7	7	10	16	14

例：求下图的关键活动和关键路径



	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	16	14	18
vl	0	6	6	8	7	10	16	14	18

a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
$e(i)$	0	0	0	6	4	5	7	7	7	16	14
$l(i)$	0	2	3	6	6	8	7	7	10	16	14

关键活动算法

①对AOE网进行拓扑排序，若网中有环则终止算法，按拓扑序求出各顶点的最早发生时间*ve*；

$$ve(j)=\begin{cases}0, & j=1\\ \max_i\{ve(i)+w(<i,j>)\,|\,<i,j>\in E(G), j=2,\ldots,n\}\end{cases}$$

②按逆拓扑序求各顶点的最迟发生时间*vl*；

$$vl(j)=\begin{cases}ve(n), & j=n\\ \min_k\{vl(k)-w(<j,k>)\,|\,<j,k>\in E(G), j=n-1,\ldots,1\}\end{cases}$$

③根据*ve*和*vl*的值，求各活动的最早开始时间*e*与最迟开始时间*l*，若*e*=*l*，则对应活动是关键活动。

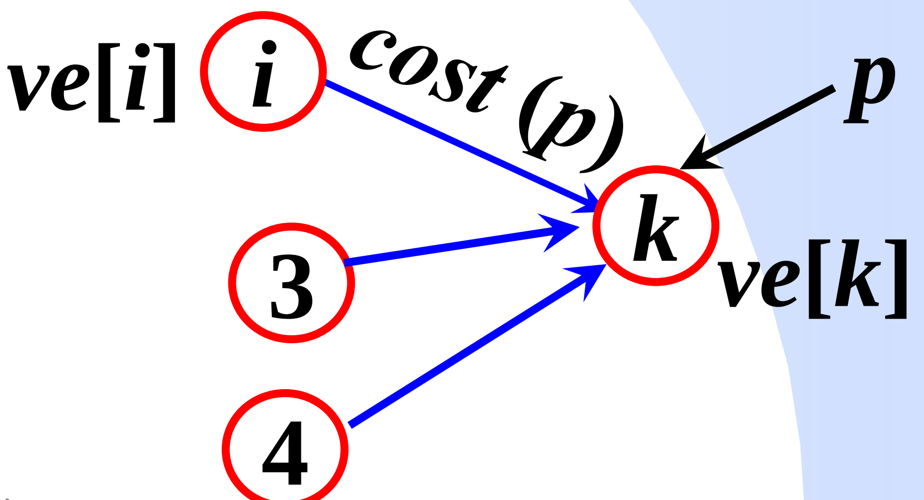
a_i 在边 $<j,k>$ 上：$e(i)=ve(j)$，$l(i)=vl(k)-weight(<j,k>)$

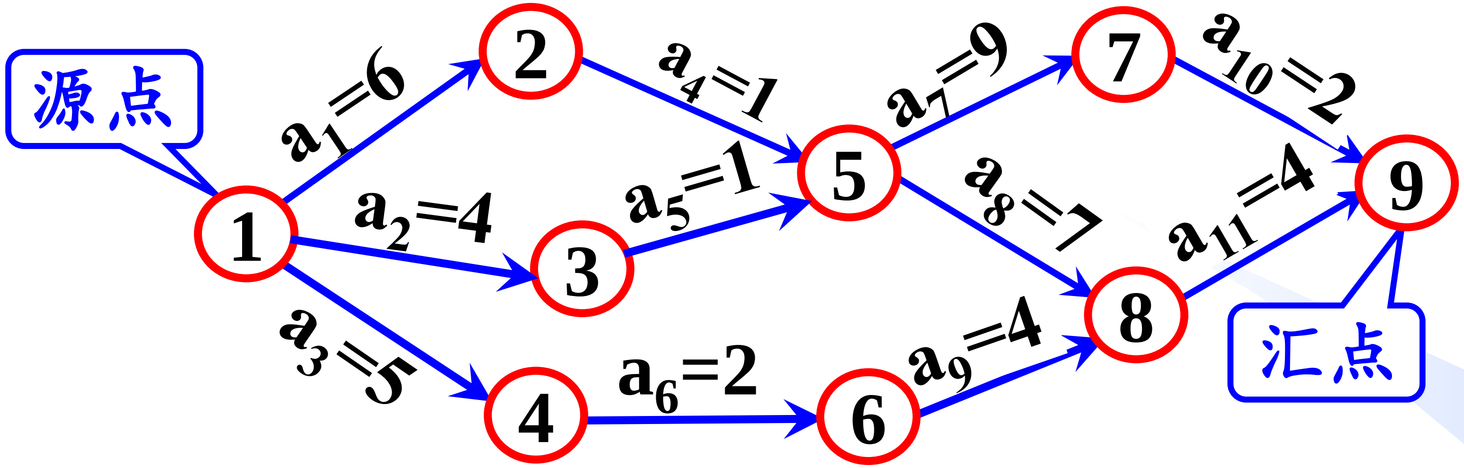
```
void VertexEarliestTime(Vertex Head[],int n, int ve[]){
    //计算顶点的最早发生时间
    for(int i=1;i<=n;i++)
        ve[i]=0;
    for(int i=1;i<=n;i++)  //按拓扑序计算各顶点最早发生时间
        for(Edge* p=Head[i].adjacent; p!=NULL; p=p->link){
            int k=p->VerAdj;
            if(ve[i]+p->cost>ve[k])
                ve[k]=ve[i]+p->cost;
        }
}
```

假定图中顶点已按拓扑排序编号

$O(n+e)$

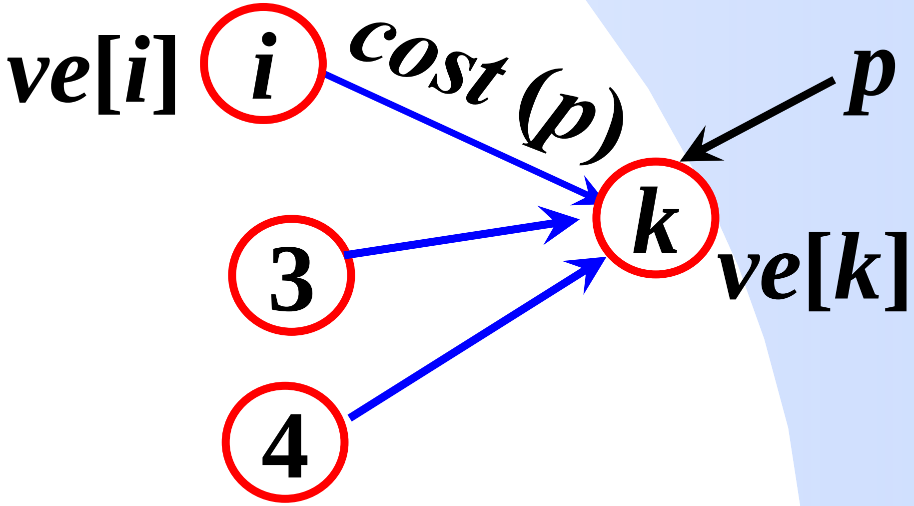
思考1：内层for循环结束后，能确定$ve[k]$的值么？





i=1 计算： <1 , 2><1 , 3><1 , 4>	i=2 计算<2 , 5>	i=3 计算<3 , 5>	i=4 计算<4 , 6>
i=5 计算<5 , 7><5 , 8>	i=6 计算<6 , 8>	i=7 计算<7 , 9>	i=8 计算<8, 9>

```
void VertexEarliestTime(Vertex Head[],int n, int ve[]){
   //计算顶点的最早发生时间
   for(int i=1;i<=n;i++)
      ve[i]=0;
   for(int i=1;i<=n;i++)  //按拓扑序计算各顶点最早发生时间
      for(Edge* p=Head[i].adjacent; p!=NULL; p=p->link){
         int k=p->VerAdj;
         if(ve[i]+p->cost > ve[k])
            ve[k]=ve[i]+p->cost;
      }
}
```



思考2：如果图中顶点未按拓扑序编号，怎么办？

```
void VertexEarliestTime(Vertex Head[],int Topo[],int n,int ve[]){
    //拓扑序存储在Topo数组中
    for(int i=1;i<=n;i++)
        ve[i]=0;
    for(int i=1;i<=n;i++) //按拓扑序计算各顶点最早发生时间
        for(Edge* p=Head[Topo[i]].adjacent; p; p=p->link){
            int k=p->VerAdj;
            if(ve[Topo[i]]+p->cost>ve[k])
                ve[k]=ve[Topo[i]]+p->cost;
        }
}
```

图中顶点未按拓扑排序编号

不是处理顶点*i*，而是处理拓扑序列中第*i*个顶点

初始调用：

```
TopoSort(Head,Topo);//求拓扑序并存储在Topo中
VertexEarliestTime(Head, Topo, n, ve);
```

方案1：先执行拓扑排序，将拓扑序存入数组 Topo ，即 Topo[*i*] 为拓扑序中第*i*个顶点的编号。

```
void TopoOrder(Vertex Head[], int n){
    int count[N]; Stack s;
    InDegree(Head, n, count);
    for(int i = 1; i <= n; i++)
        if(count[i]==0) s.PUSH(i);
    for(int i = 1; i <= n; i++){
        if(s.IsEmpty()) return;
        int j=s.POP(); //选出入度为0的顶点
        for(Edge *p=Head[j].adjacent; p!=NULL; p=p->link){
            int k=p->VerAdj;  count[k]--;
            if(count[k]==0) s.PUSH(k);
            if(ve[j]+p->cost>ve[k]) ve[k]=ve[j]+p->cost;
        }
    }
}
```

拓扑排序

选出一个点
扫描其邻接顶点并更新入度

求ve值

选出一个点
扫描其邻接顶点并更新ve值

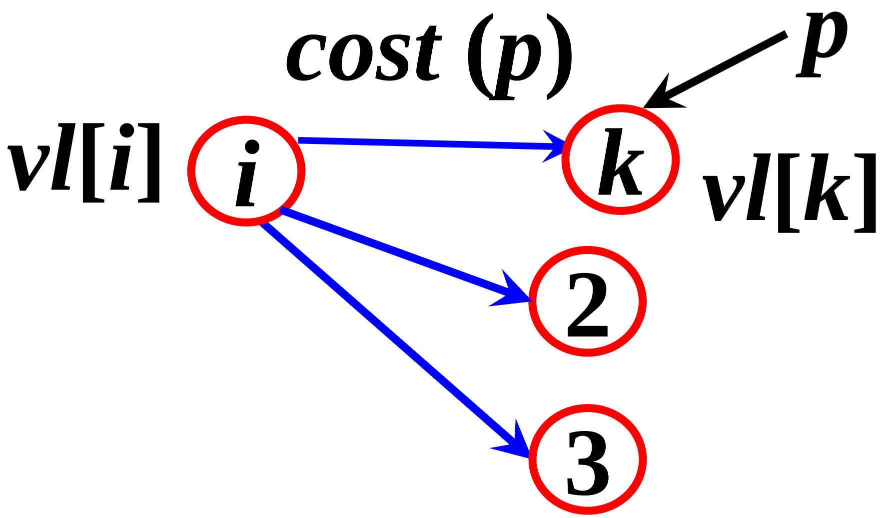
两个过程合并

方案2：拓扑排序过程中，弹栈选出入度为0的顶点并更新其邻接顶点的入度时，同时更新ve值。从而无需调用VertexEarliestTime函数

```
void VertexLatestTime(Vertex* Head,int n,int ve[],int vl[]){
    //计算顶点的最迟发生时间
    for(int i=1;i<=n;i++)
        vl[i]=ve[n];
    for(int i=n;i>=1;i--) //按拓扑逆序计算各顶点最迟发生时间
        for(Edge* p=Head[i].adjacent; p!=NULL; p=p->link){
            int k=p->VerAdj;
            if(vl[k]-p->cost < vl[i])
                vl[i] = vl[k]-p->cost;
        }
}
```

O(n+e)

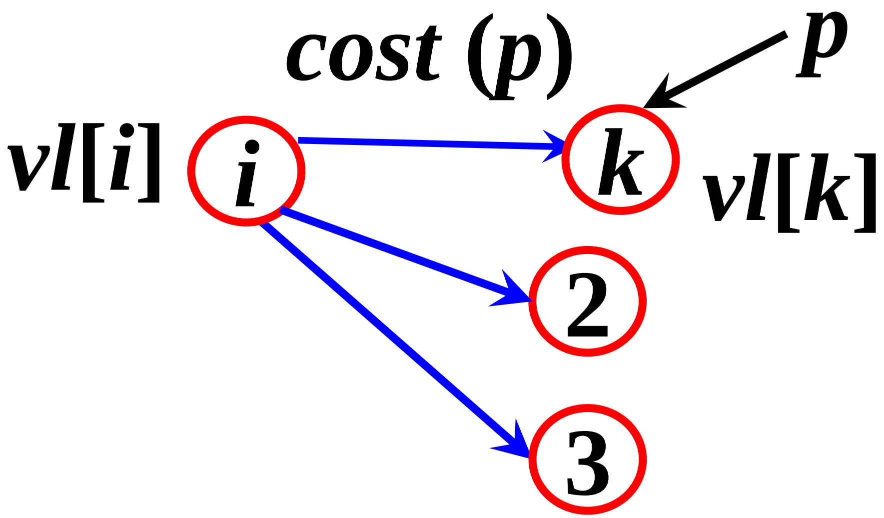
思考1：内层for循环结束后，能确定$vl[i]$的值么？



```
void VertexLatestTime(Vertex *Head,int n, int Topo[],int ve[],int vl[]){
    //计算顶点的最迟发生时间
    for(int i=1;i<=n;i++)
        vl[i]=ve[n];
    for(int i=n;i>=1;i--) //按拓扑逆序计算各顶点最迟发生时间
        for(Edge* p=Head[Topo[i]].adjacent; p; p=p->link){
            int k=p->VerAdj;
            if(vl[k]-p->cost < vl[Topo[i]])
                vl[Topo[i]] = vl[k]-p->cost;
        }
}
```

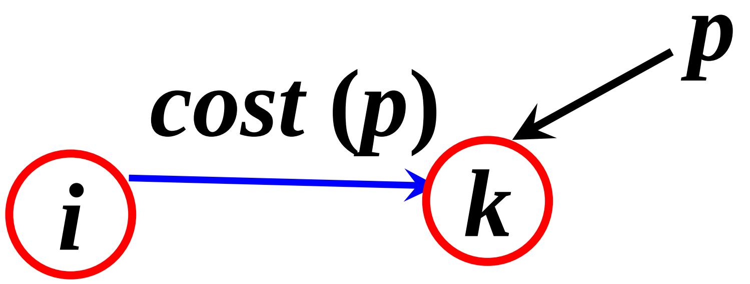
不是处理顶点i，而是处理拓扑序列中第i个顶点

思考2：如果图中顶点未按拓扑序编号，怎么办？



```
void ActivityStartTime(Vertex* Head,int n,int ve[],int vl[]){
    //求诸活动的最早开始时间和最迟开始时间,并求关键活动
    for(int i = 1;i<=n;i++)
        for(Edge* p=Head[i].adjacent; p!=NULL; p=p->link){
            int k = p->VerAdj;
            int e = ve[i];              //最早开始时间
            int l = vl[k] – p->cost;    //最晚开始时间
            if(e == l) printf(“%d->%d\n”,i,k); //输出关键活动
        }
}
```

O(e)



关键路径算法

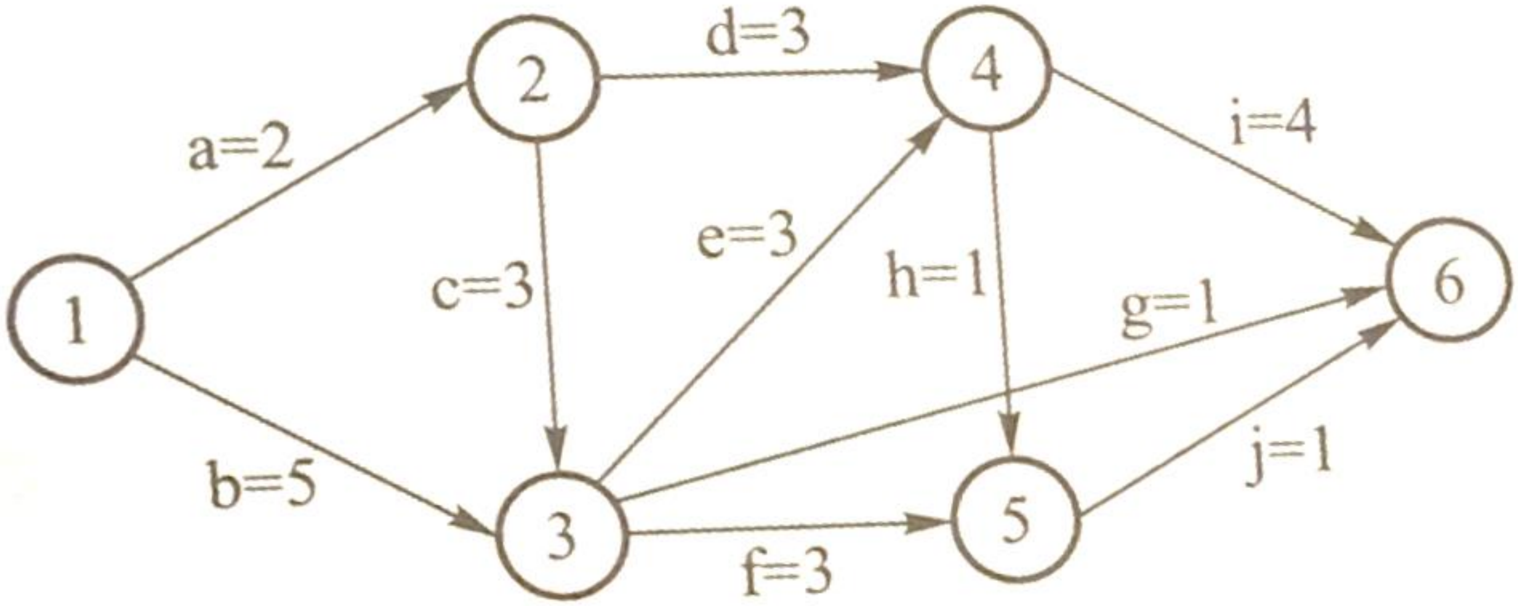
```
const int N=1010;
void CriticalPath(Vertex* Head, int n){
   //假定图中顶点已按拓扑序编号
   int ve[N],vl[N];
   VertexEarliestTime(Head,n,ve);  //顶点最早发生时间
   VertexLatestTime(Head,n,ve,vl); //顶点最迟发生时间
   ActivityStartTime(Head,n,ve,vl);//活动最早最晚开始时间
}
```

时间复杂度O(n+e)

课下思考：

下图是有10个活动的AOE网，其中时间余量最大的活动是______【2022年考研题全国卷，2分】

A. c　　B. g　　C. h　　D. j



课下思考

加速某一关键活动（减少完成该关键活动所需的时间），一定能缩短整个工程的工期么？