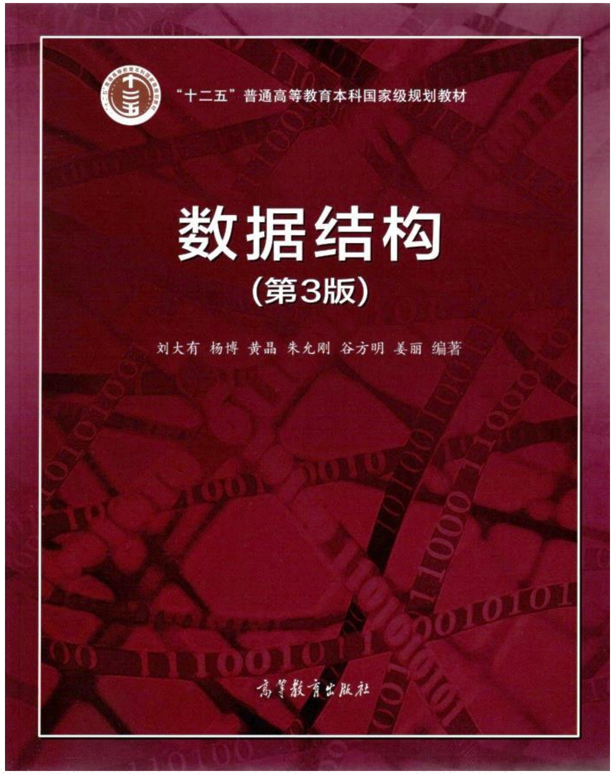
拓扑排序和关键路径



- 拓扑排序
- 关键路径

数据之法
结构之美
算法之道

Last updated on 2023.11

zhuyungang@jlu.edu.cn

毛啸
麻省理工学院17级本科/21级硕士
斯坦福大学22级博士生
2016年NOI全国中学生信息学奥赛决赛第1名
2017年IOI世界中学生信息学奥赛银牌
2022年ICPC国际大学生程序设计竞赛全球总决赛冠军
国际顶级会议FOCS 2021最佳学生论文奖

我深知我写代码非常容易出错，经常调错调半天。

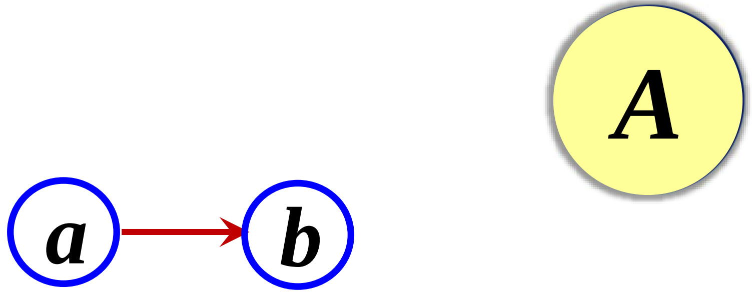
当你试图解决一个问题时，应该先独立思考。如果实在没有思路，那么应该去一点一点的读别人的题解，而不是一次性都读完，尽可能确保能有更多的部分是你自己做出来的。之后仔细思考一下，为什么你会想不出来依赖于题解解决的那些部分。Think twice, code once。

拓扑排序——动机

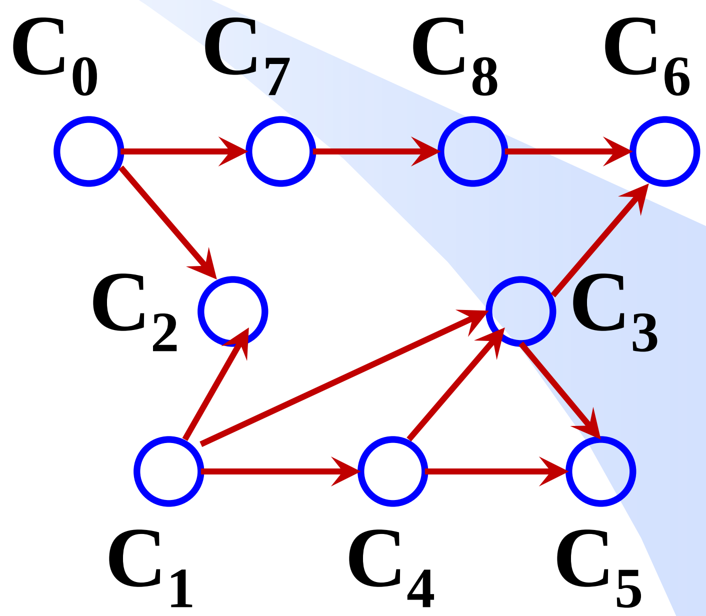
A

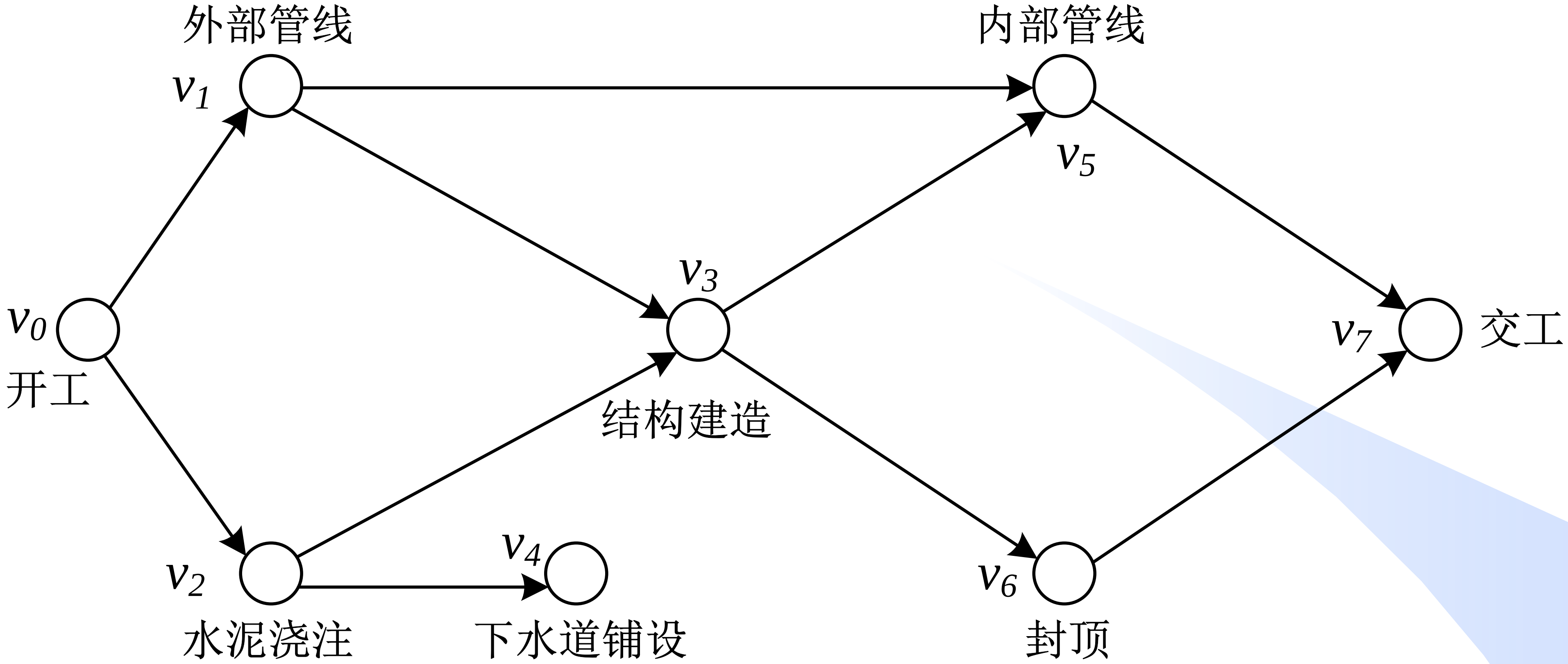
- 一个任务（例如一个工程）通常可以被分解成若干个子任务，要完成整个任务就可以转化为完成所有的子任务。
- 在某些情况下，各子任务之间有序，要求一些子任务必须先于另外一些子任务被完成。
- 各任务之间的先后关系可以用有向图来表示。
- 例：计算机专业学生的学习就是一个任务，每一门课程的学习就是整个任务的一个子任务。其中有些课程要求先修课程，有些则不要求。这样在有的课程之间有先后关系，有的课程可以并行地学习。

计算机专业部分课程的先后关系



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



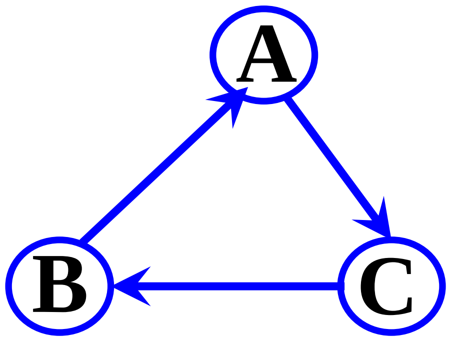


例：建楼工程示意图，图中V_3 必须在V_1和V_2被完成后才能开始；V_4必须在V_2被完成后才能开始；V_5必须在V_1和V_3被完成后才能开始；V_6必须在V_3被完成后才能开始；V_5和V_6最后被完成时才能说整个工程可以交工。

AOV网

A

- **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。即AOV网是一个有向无环图（Directed Acyclic Graph, DAG）



拓扑排序——动机

A

- **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 $V_i \rightarrow V_j$，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有向图的所有顶点都排入一个拓扑序列中，则该有向图中必定不含环；相反，若不能把所有顶点都排入一个拓扑序列，则说明有向图中存在环，此图表示的AOV网所代表的任务是不可行的。

A

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。

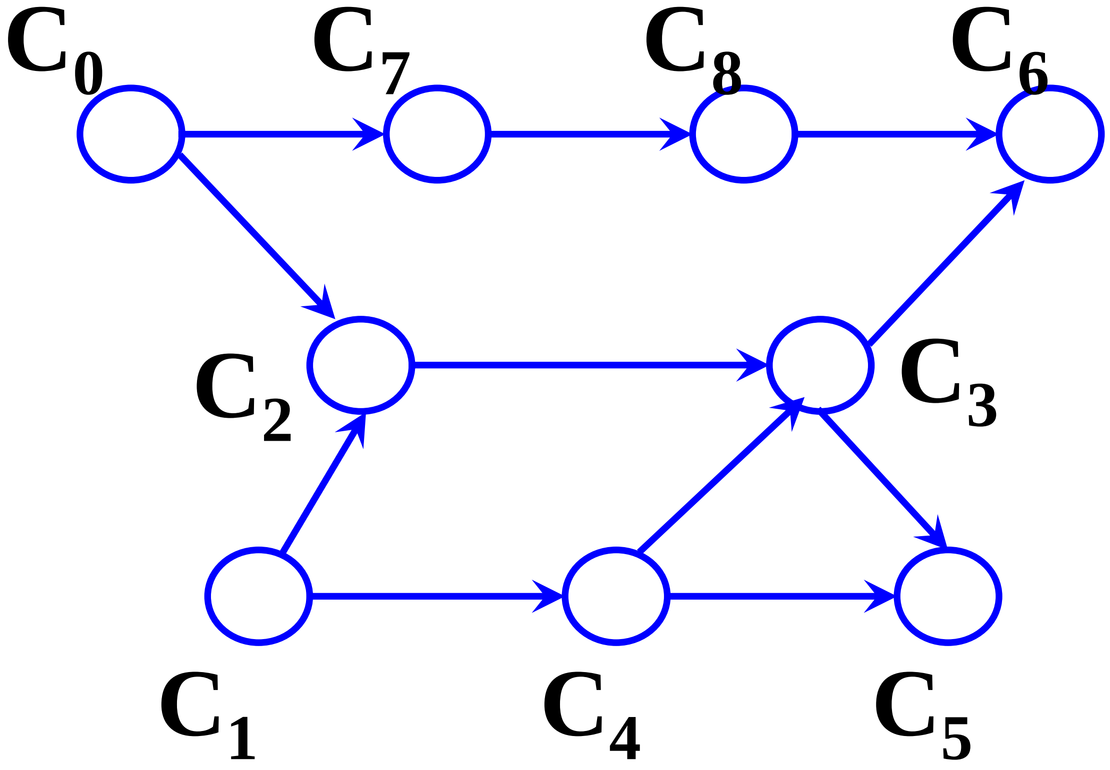
③ 执行①②，直至所有顶点已输出，或图中剩余顶点入度均不为0（说明存在环，无法继续拓扑排序）。

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。

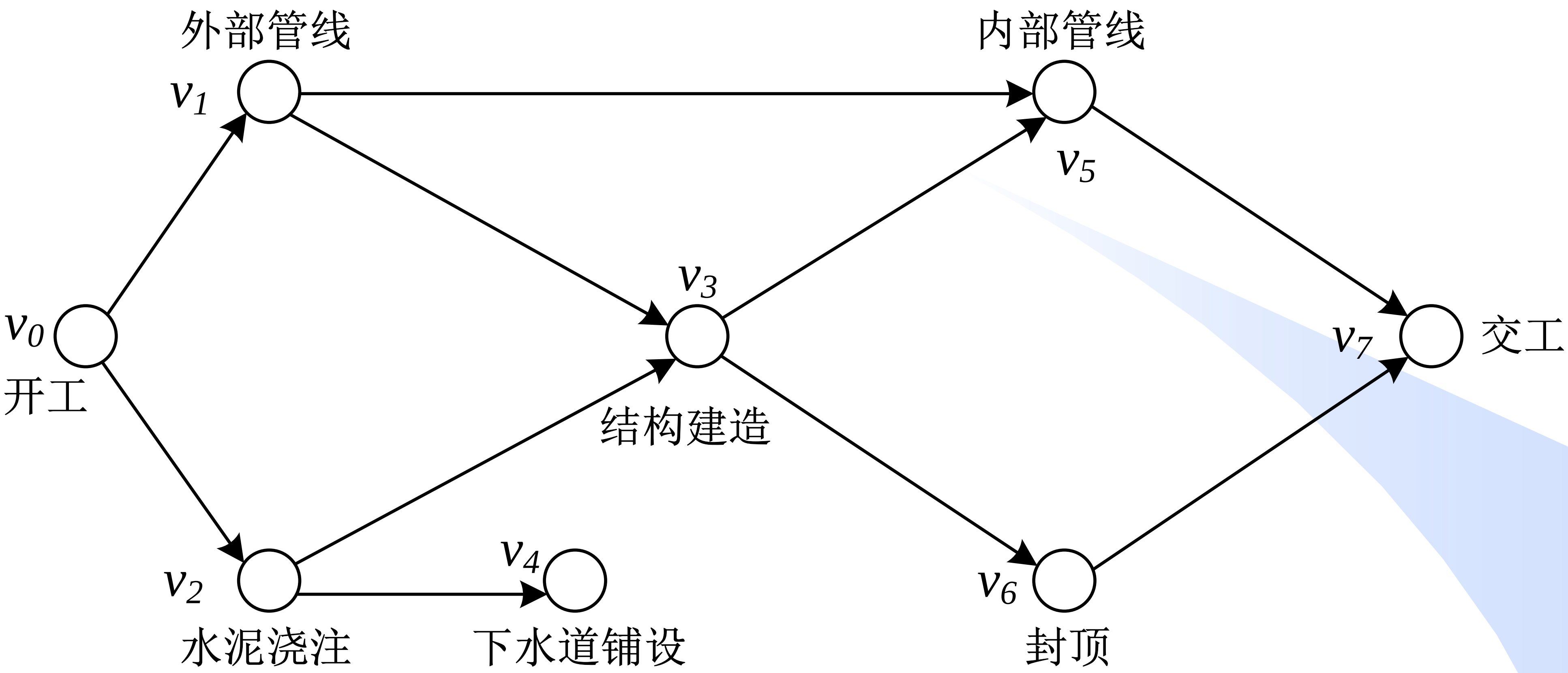
例：对下图进行拓扑排序，得到的拓扑序列为

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

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



A



$V_0,V_1,V_2,V_4,V_3,V_5,V_6,V_7$和

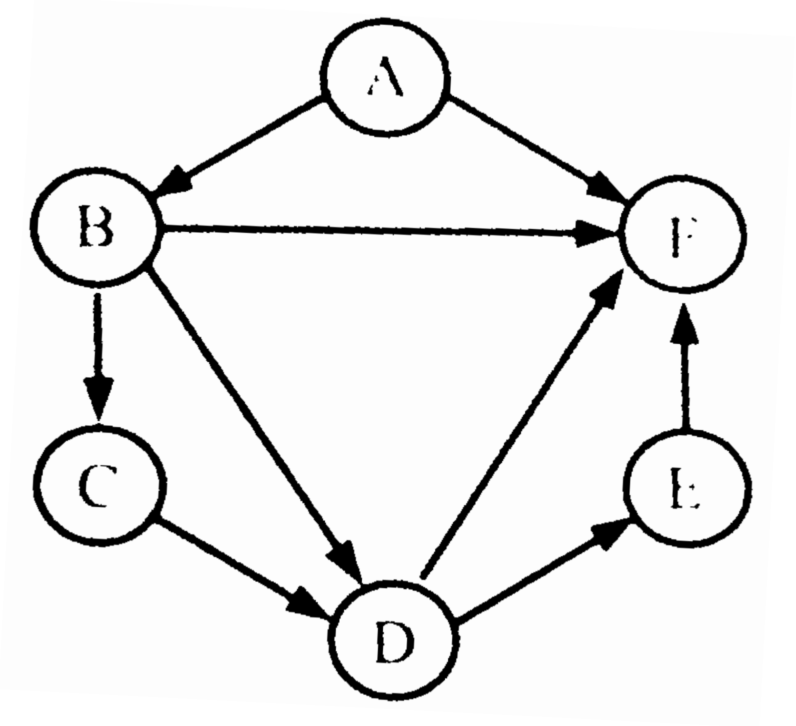
$V_0,V_2,V_4,V_1,V_3,V_6,V_5,V_7$均是上图的拓扑序列。

课下思考

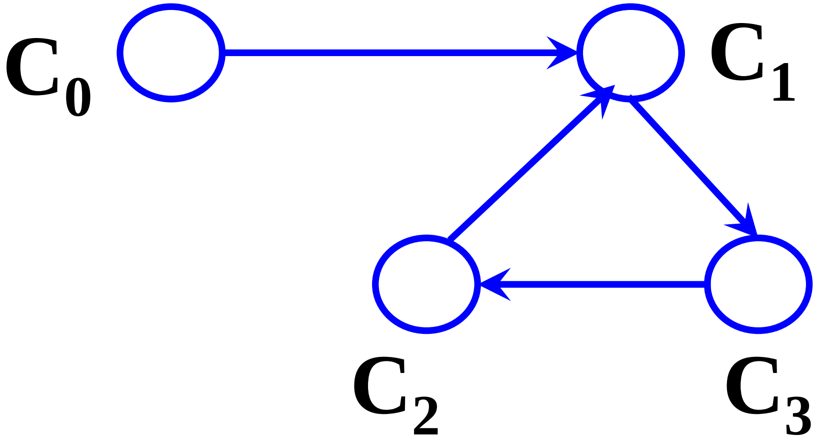
A

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】

A. 1　　B. 2　　C. 3　　D. 4



拓扑排序判断有向图中是否含有环



拓扑排序的实现——准备工作

A

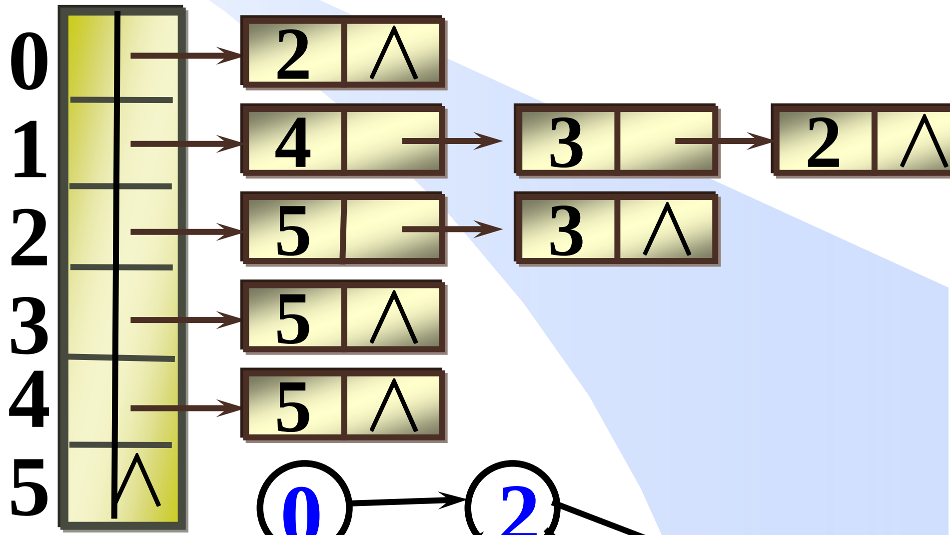
- 假定AOV网以邻接表的形式存储。为实现拓扑排序算法，事先需好两项准备工作：
- 建立一个数组InDegree[ ]：InDegree[*i*]为顶点*i*的入度；
- 建立一个栈存放入度为0的顶点：每当一个顶点的入度为0，就将其压栈；每次找入度为0的顶点时，就弹栈。

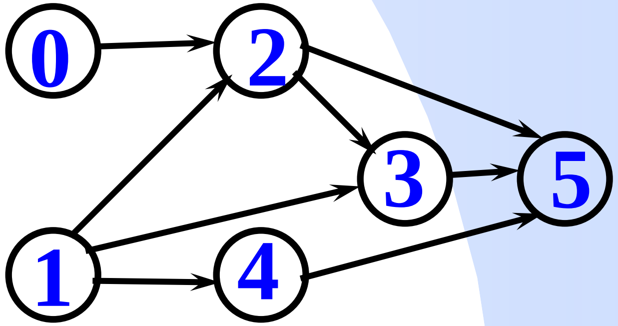
课下思考：用队列可以么？

回顾：求每个顶点的入度

```
void getInDegree(Vertex Head[], int n, int InDegree[]){
   for(int i=0; i<n; i++) InDegree[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;
         InDegree[k]++;
         p = p->link;
      }
   }
}
```

用p扫描每个顶点的邻接顶点（边结点）



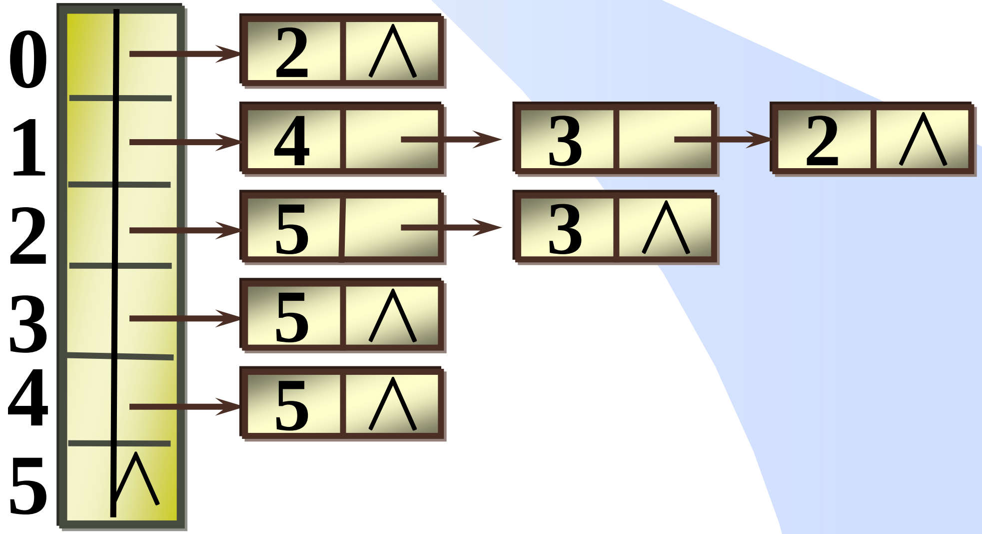


InDegree						
	0	1	2	3	4	5

回顾：求每个顶点的入度

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

用p扫描每个顶点的邻接顶点（边结点）

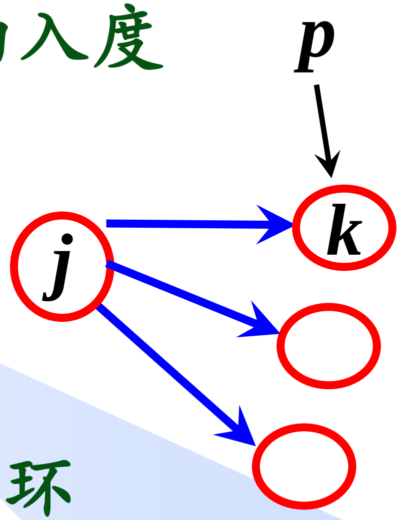


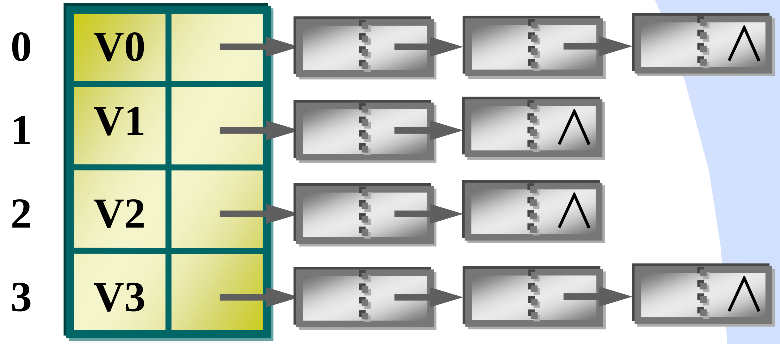
InDegree

0	1	2	3	4	5

B

```
bool TopoOrder(Vertex Head[], int n){
    int InDegree[N]; Stack s;
    getInDegree(Head, n, InDegree);  //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(InDegree[i]==0) s.PUSH(i);
    for(int i=0; i<n; i++){
        if(s.Empty()) 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; InDegree[k]--; //顶点k的入度减1
            if(InDegree[k]==0) s.PUSH(k);
        }
    }
    return true;
}
```





时间复杂度为O(*n*+*e*)

B

```
bool TopoOrder(Vertex Head[], int n){
    int InDegree[N]; int stack[N],top=-1;
    InDegree(Head, n, InDegree);//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(InDegree[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; InDegree[k]--; //顶点k的入度减1
            if(InDegree[k]==0) stack[++top]=k;
        }
    }
    return true;
}
```

最简单写法
用数组实现栈

思考：如何判断拓扑序列是否唯一

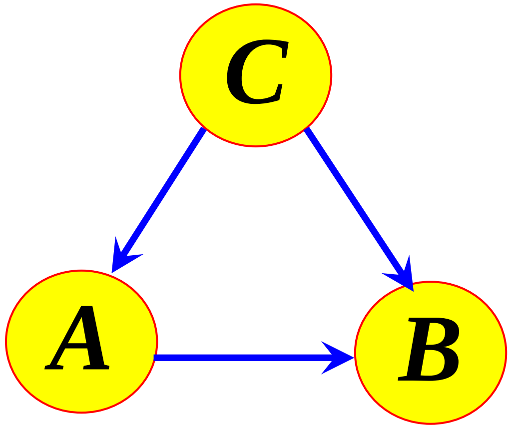
练习

B

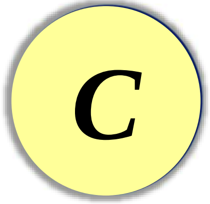
给定一个图和顶点序列，编写算法判断该序列是否是图的拓扑序列。【北京航空航天大学考研题】

- 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，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



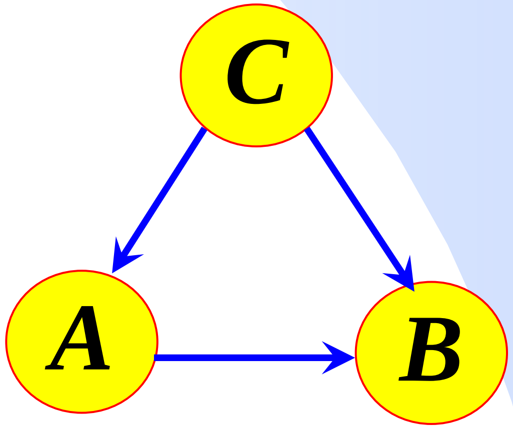
DFS不能保证A一定在B之前输出，但能保证A一定在B之后输出。即DFS可以输出拓扑序的逆序。



深度优先遍历生成逆拓扑序

C

```
void DFS_TopoSort(Vertex*Head, int v, int visited[]){
     visited[v]=1;
     for(Edge*p=Head[v].adjacent; p!=NULL; p=p->link)
        if(visited[p->VerAdj]==0)
           DFS_TopoSort(Head, p->VerAdj, visited);
     visited[v]=2;
     printf("%d ",v);
}
```



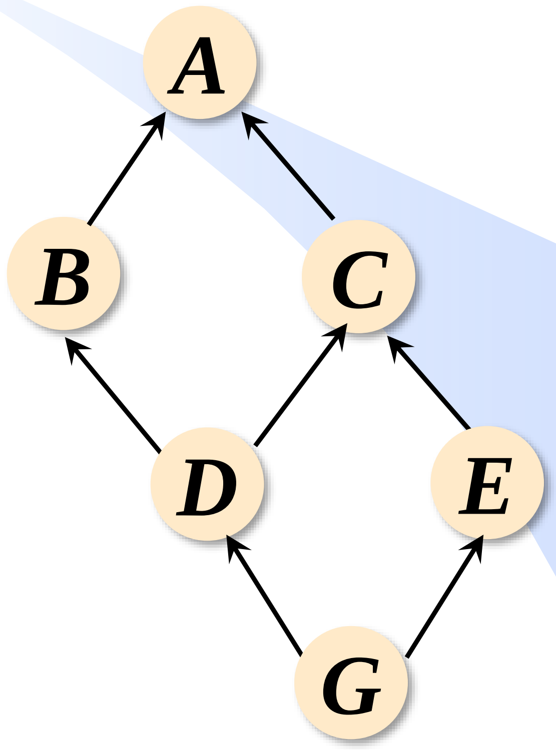
初始调用

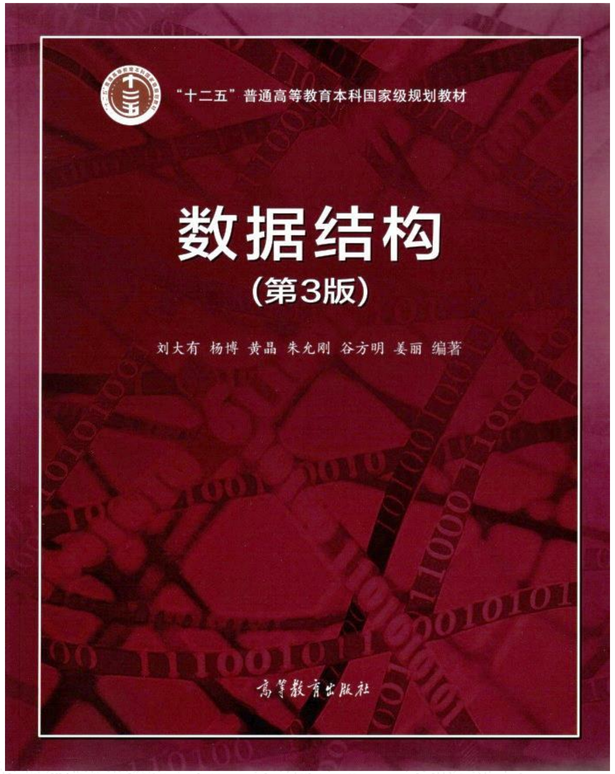
```
for(int i=0; i<n; i++)
   if(visited[i]==0)
      DFS_TopoSort(Head, i, visited);
```

拓扑排序的应用举例——多重继承

D

```
class A{
public:
   void f(){…}
};
class B:public A{…};
class C:public A{
public:
   void f(){…}
};
class D:public B, public C{…};
class E:public C{…};
class G:public D, public E{…};
G g;
g.f();
```





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

Last updated on 2023.11

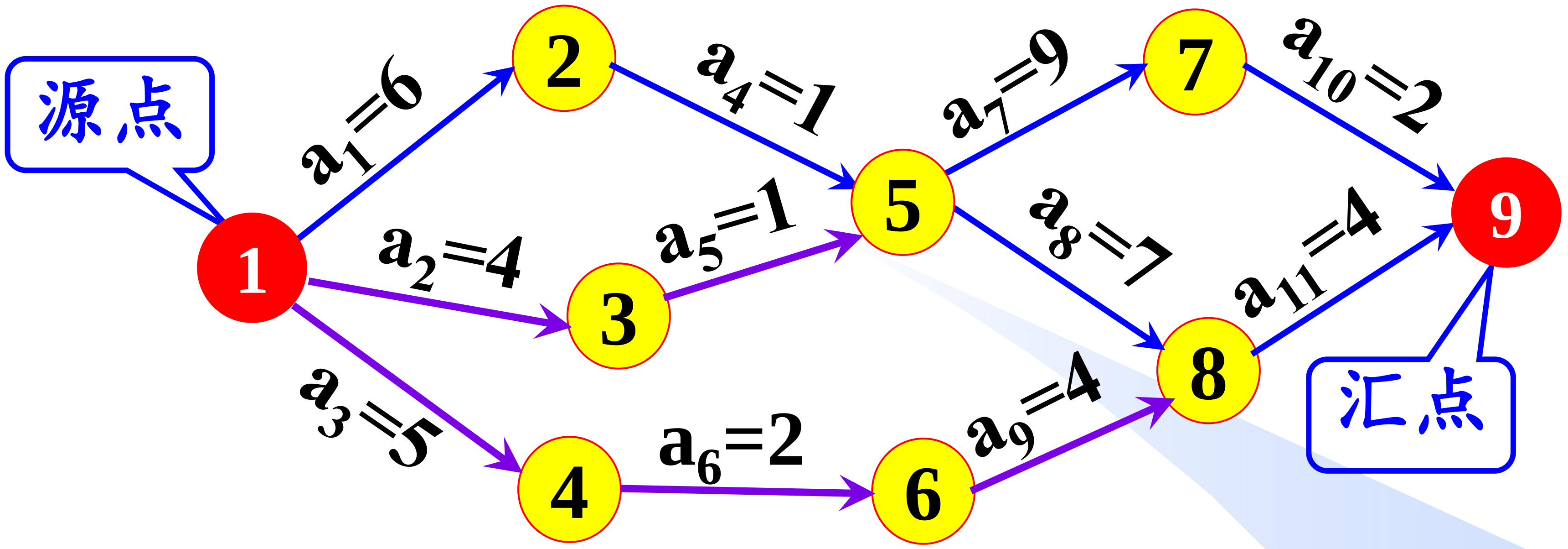
zhuyungang@jlu.edu.cn

关键路径

A

- 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?

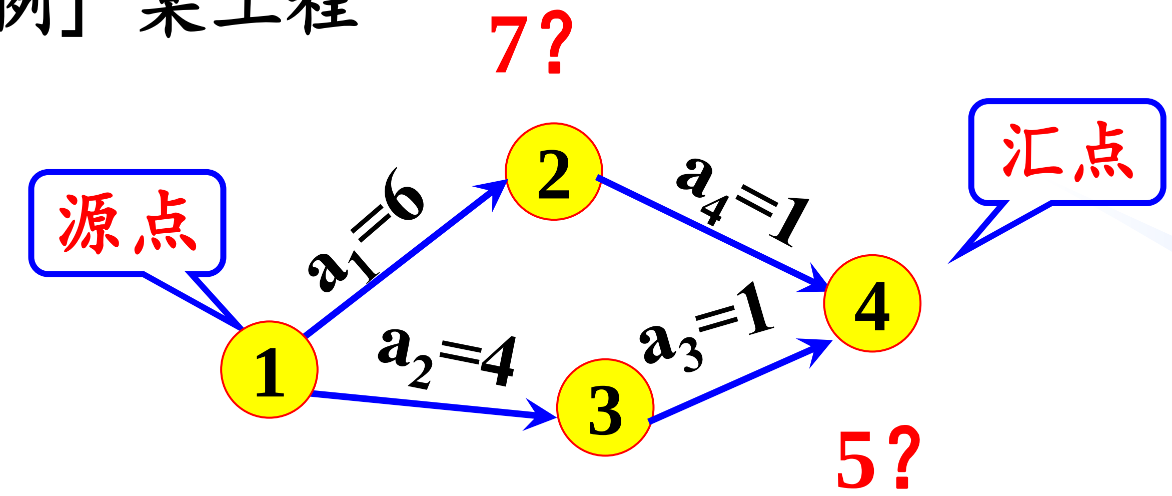
A

- 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

A

[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 $vl(j)$

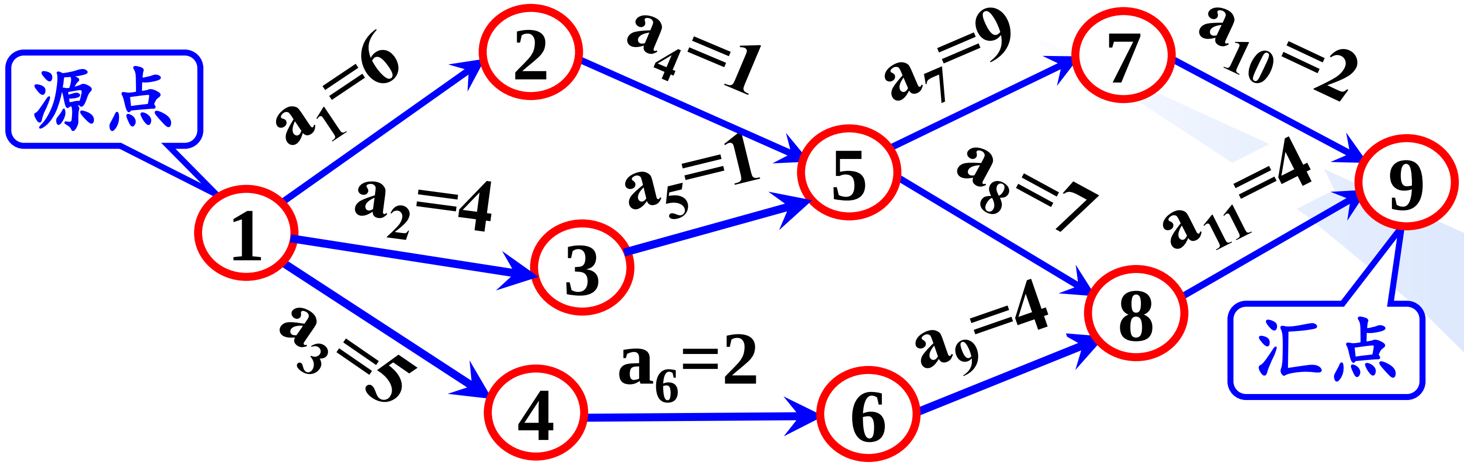
③ 活动 a_i 的最早开始时间 $e(i)$

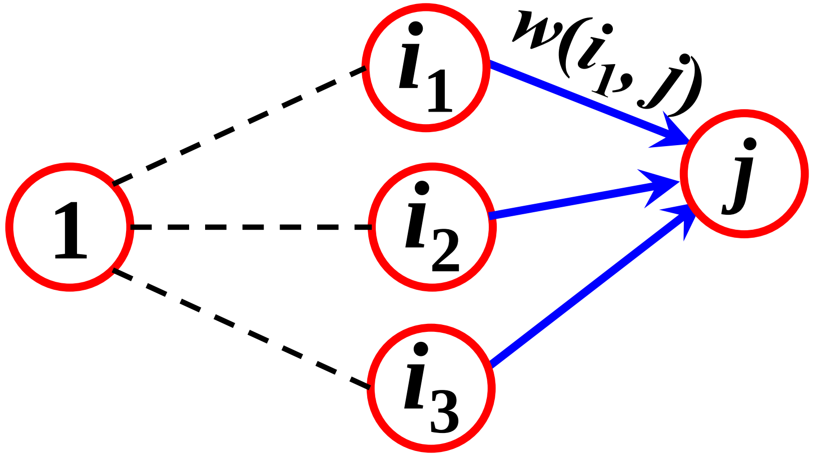
④ 活动 a_i 的最迟开始时间 $l(i)$

① 事件v_j的最早发生时间 $ve(j)$

A

从源点 v_1 到 v_j 的最长路径的长度。显然有$ve(1)=0$



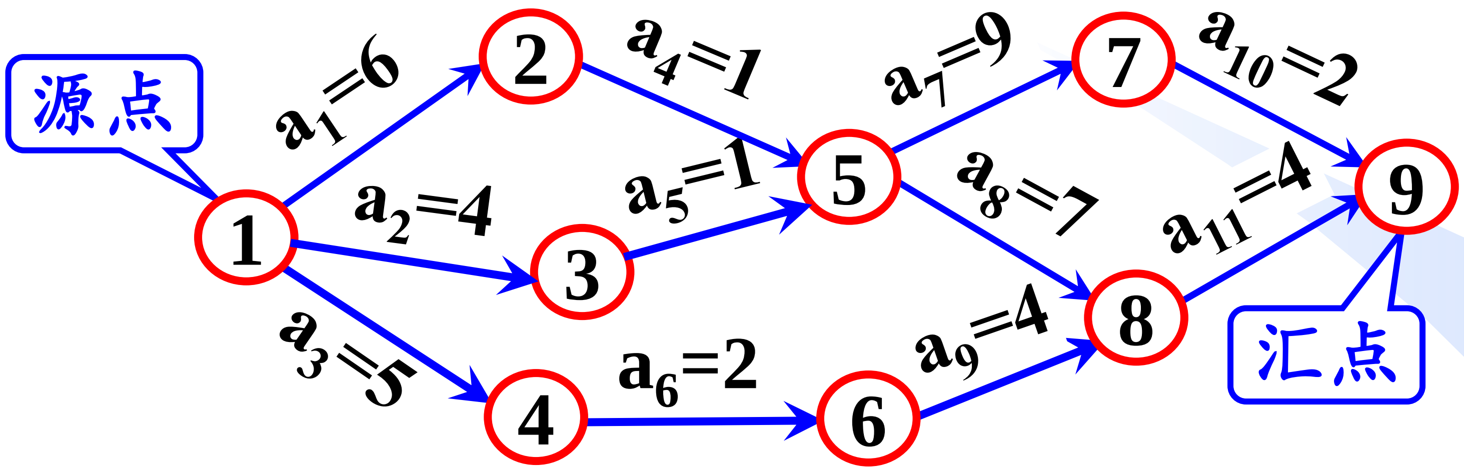


$$\max \begin{Bmatrix} ve(i_1) + w(i_1, j) \\ ve(i_2) + w(i_2, j) \\ ve(i_3) + w(i_3, j) \end{Bmatrix}$$

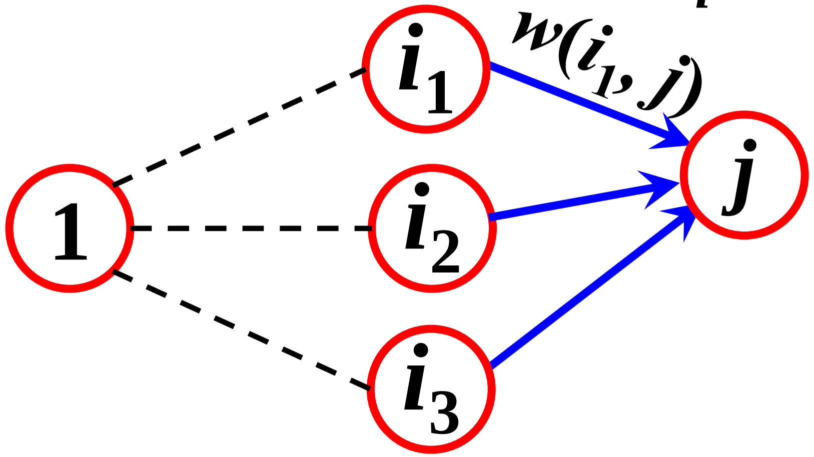
① 事件v_j的最早发生时间 $ve(j)$

A

从源点 v_1 到 v_j 的最长路径的长度。显然有$ve(1)=0$



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



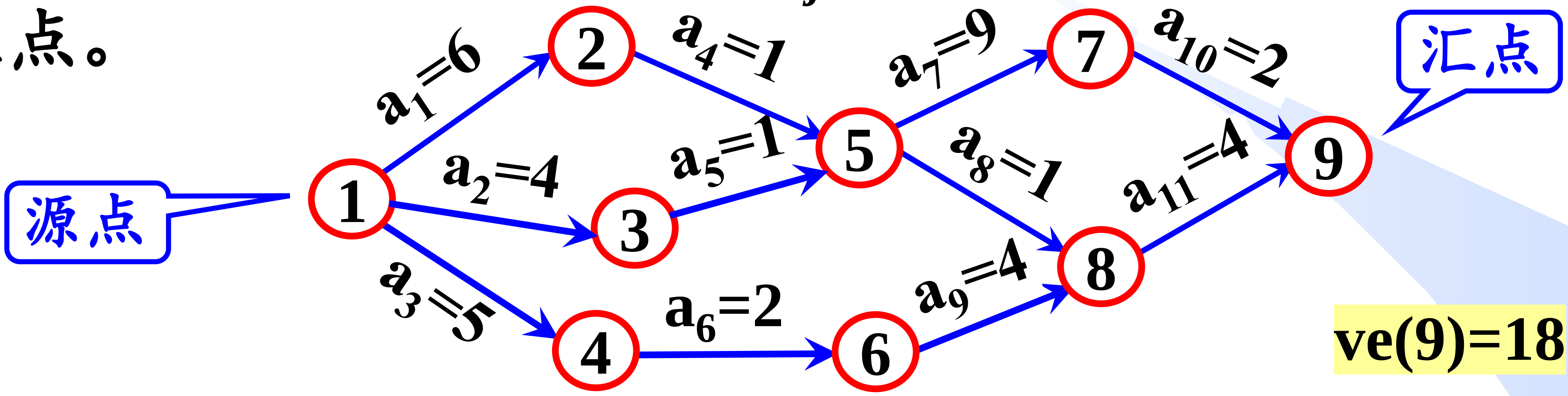
计算$ve(j)$需要已知顶点v_j的所有前驱顶点的最早发生时间。

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 $vl(j)$

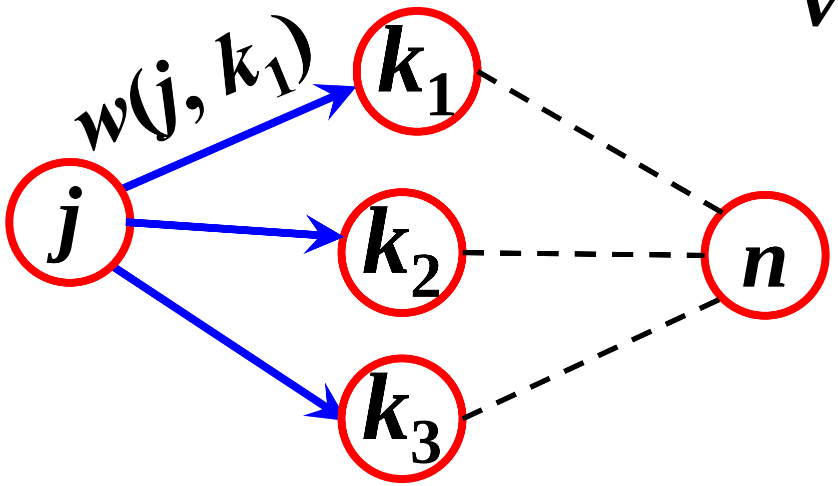
A

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。



ve(9)=18

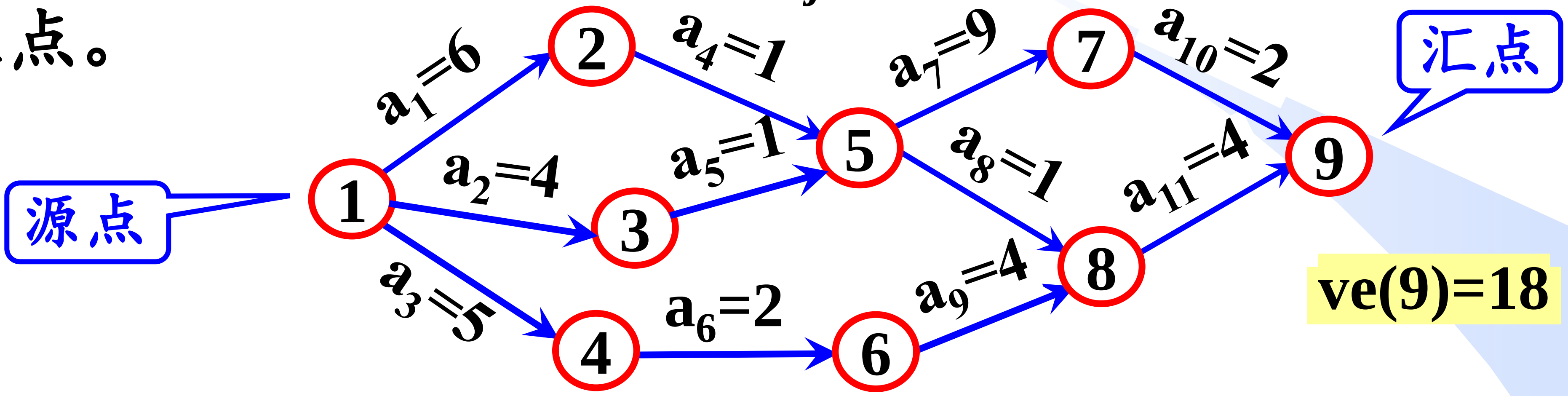
$$vl(j) = ve(n) - max\{j\ldots n\}$$

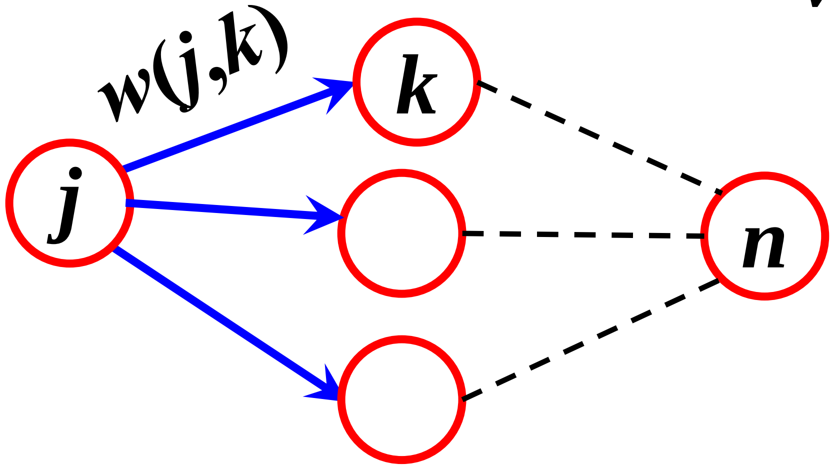


$$\min\begin{cases} ve(n) - max\{k_1\ldots n\} - w(j, k_1) \\ ve(n) - max\{k_2\ldots n\} - w(j, k_2) \\ ve(n) - max\{k_3\ldots n\} - w(j, k_3) \end{cases}$$

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。



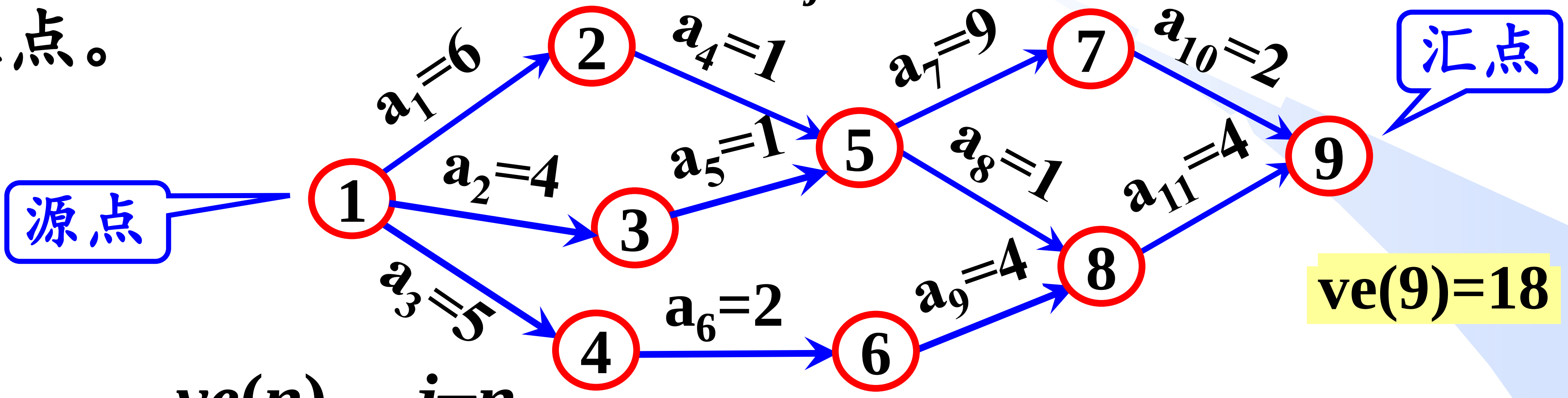


$$vl(j) = \min_{k}\{ \underbrace{ve(n) - max\{k…n\}}_{vl(k)} - w(j, k) \}$$

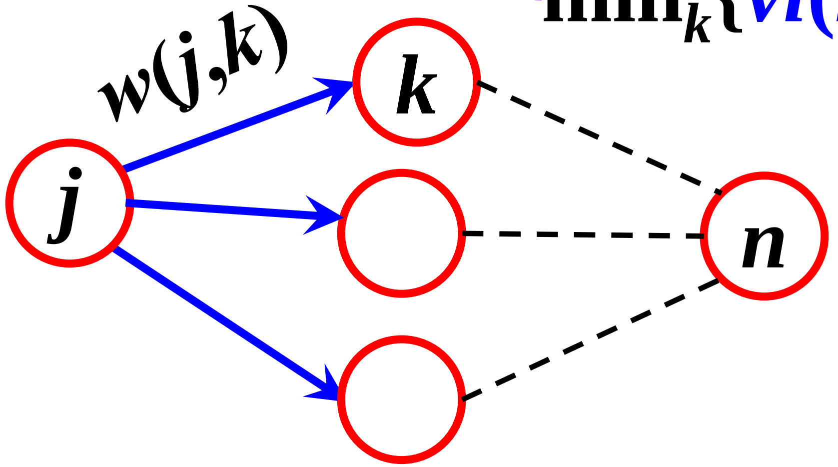
② 事件v_j的最迟发生时间 $vl(j)$

A

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。



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



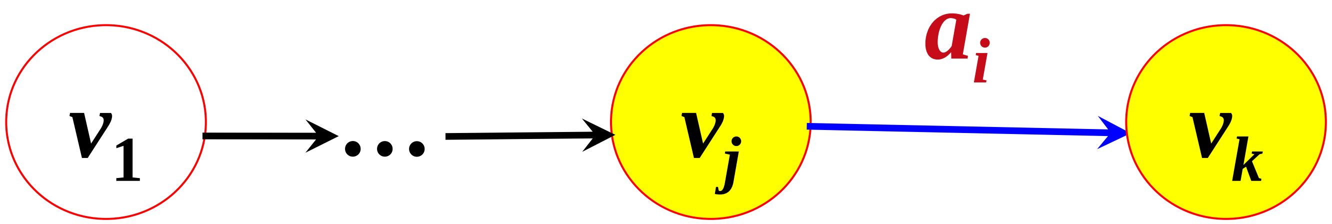
计算$vl(j)$需要已知顶点v_j的所有后继顶点的最晚发生时间。

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 $e(i)$

A

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

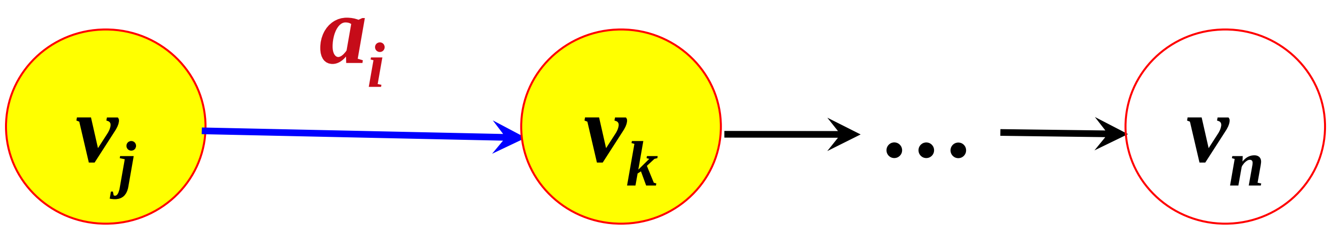


④ 活动a_i的最迟开始时间 $l(i)$

A

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

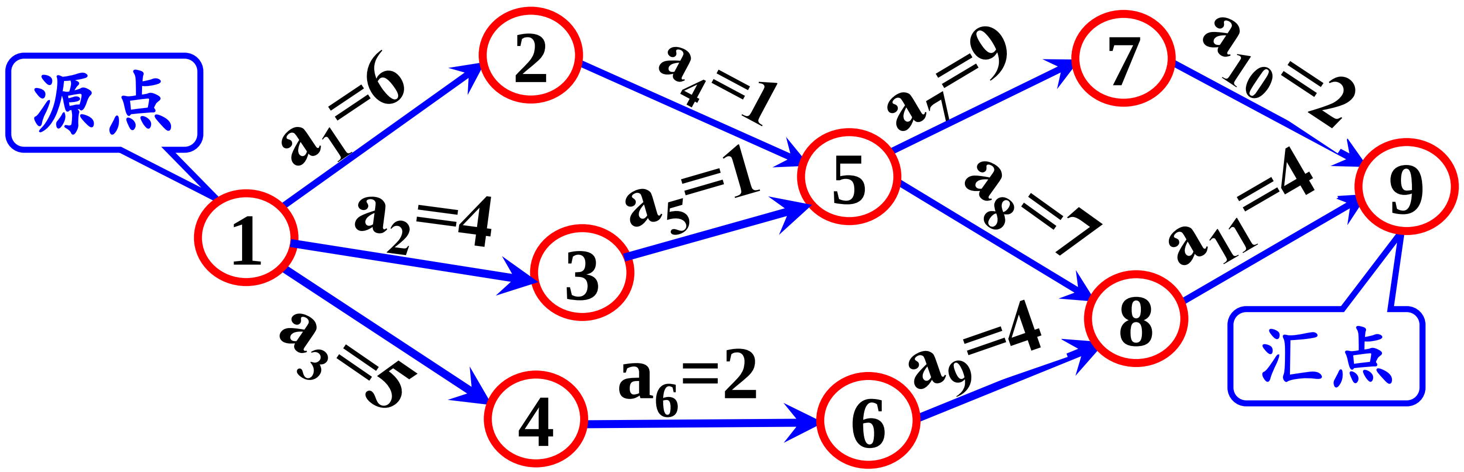
$$l(i) = vl(k) - weight(j, k)$$



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

A

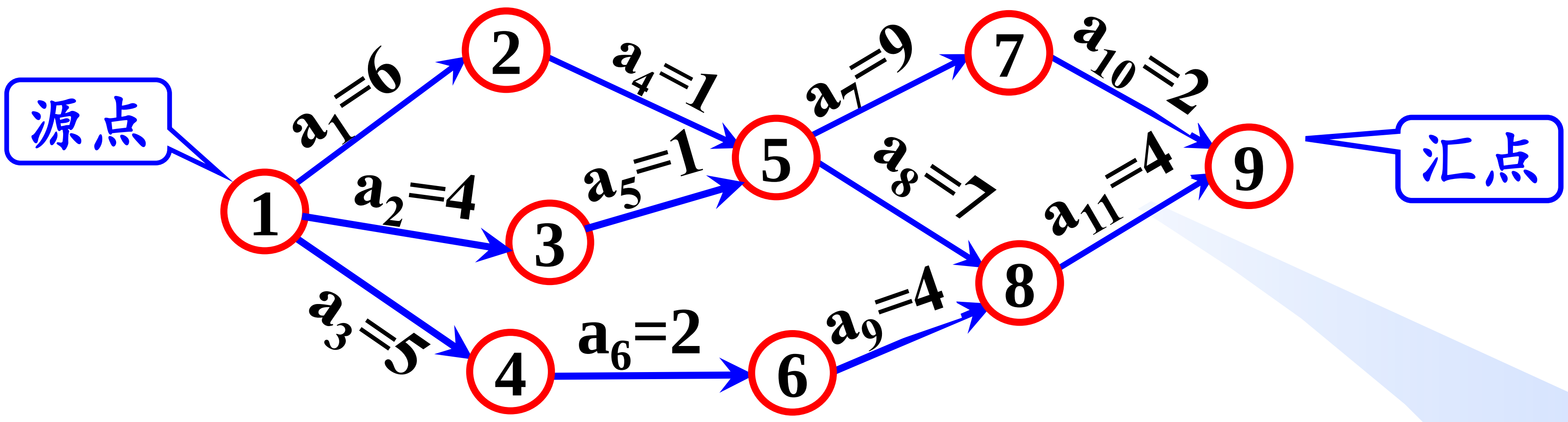
① 对AOE网进行拓扑排序，按顶点拓扑序求各顶点v_j的最早发生时间 $ve(j)$；

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。

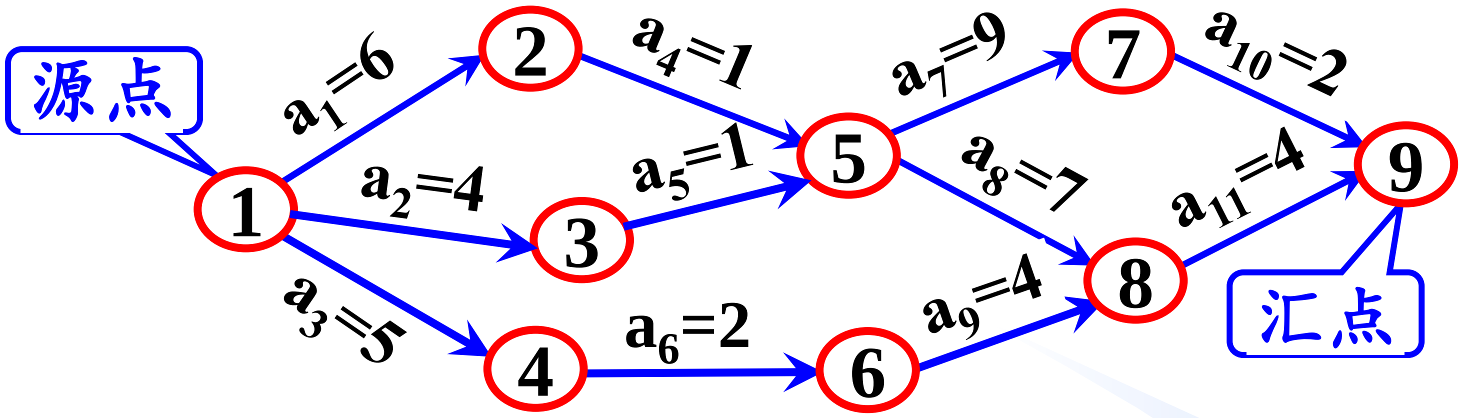
例：求下图的关键活动和关键路径

A



	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

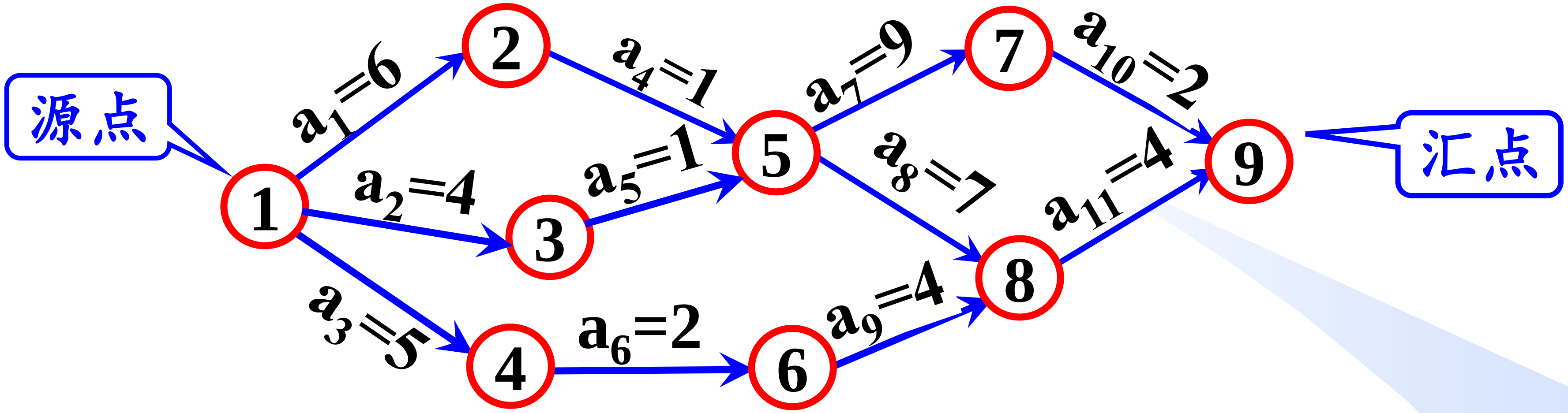
ve(7)= ve(5)+weight(<5, 7>)=7+9=16

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

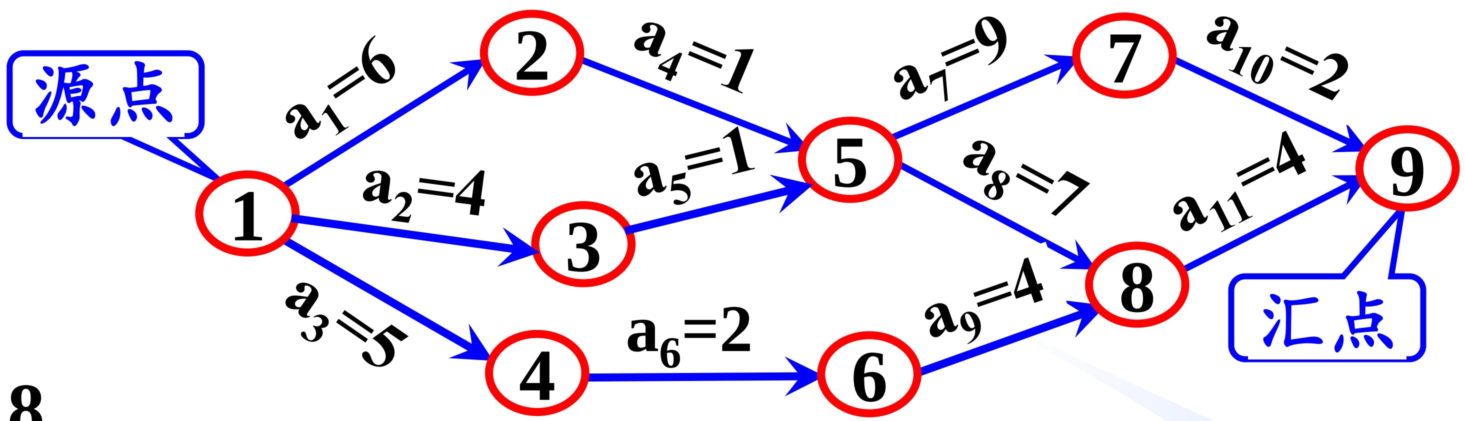
$$ve(j)=\begin{cases} 0, & j=1 \\ \max_i \{ve(i)+w(<i,j>) \mid <i,j>\in E(G),\ j=2,\ldots,n\} & \end{cases}$$

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

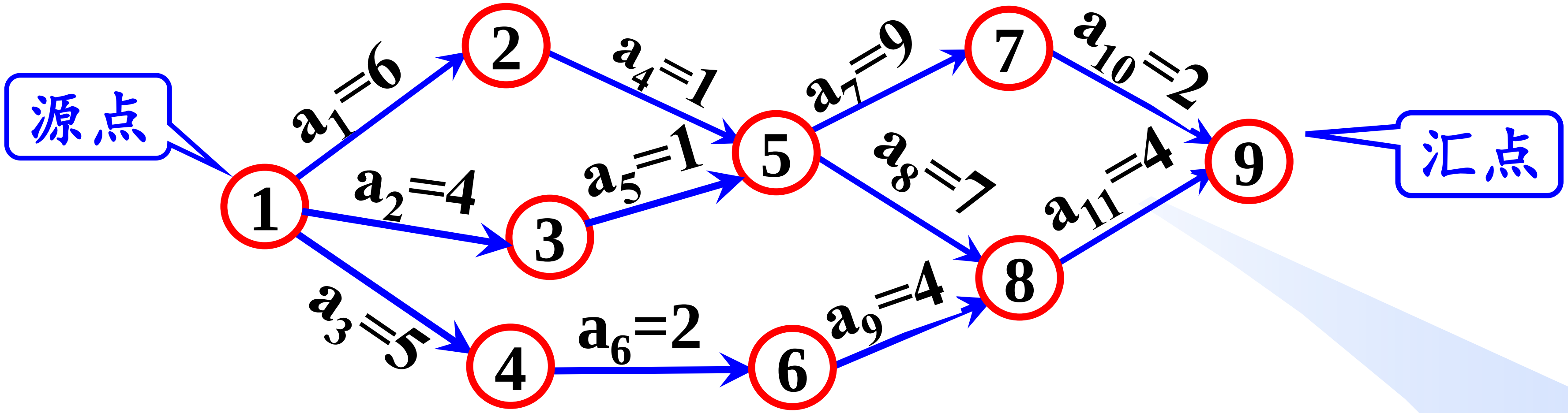
vl(2)= vl(5)-weight(<2,5>)=7-1=6

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

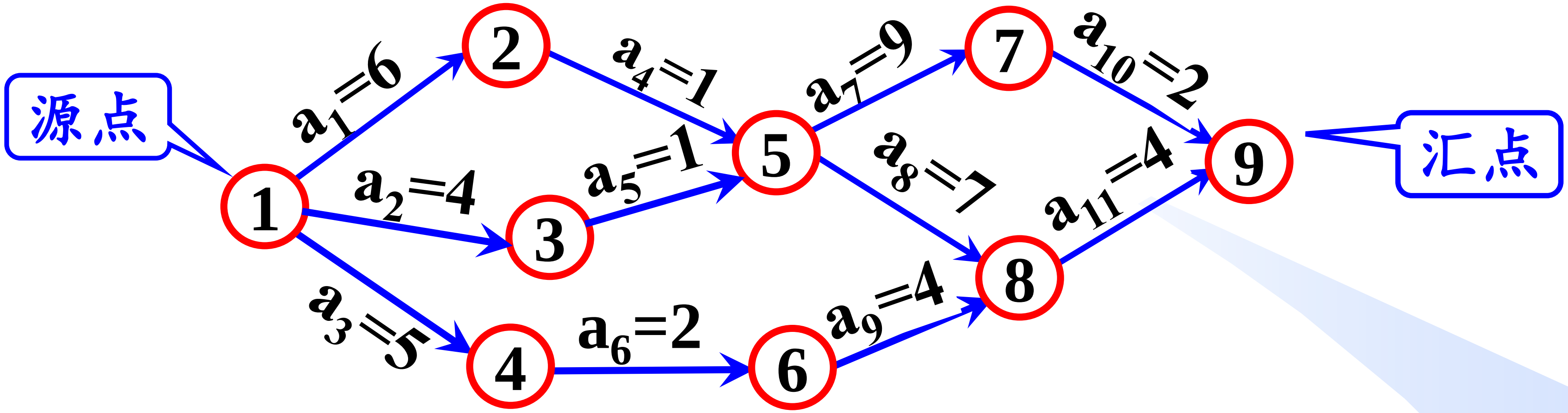
A



	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

A

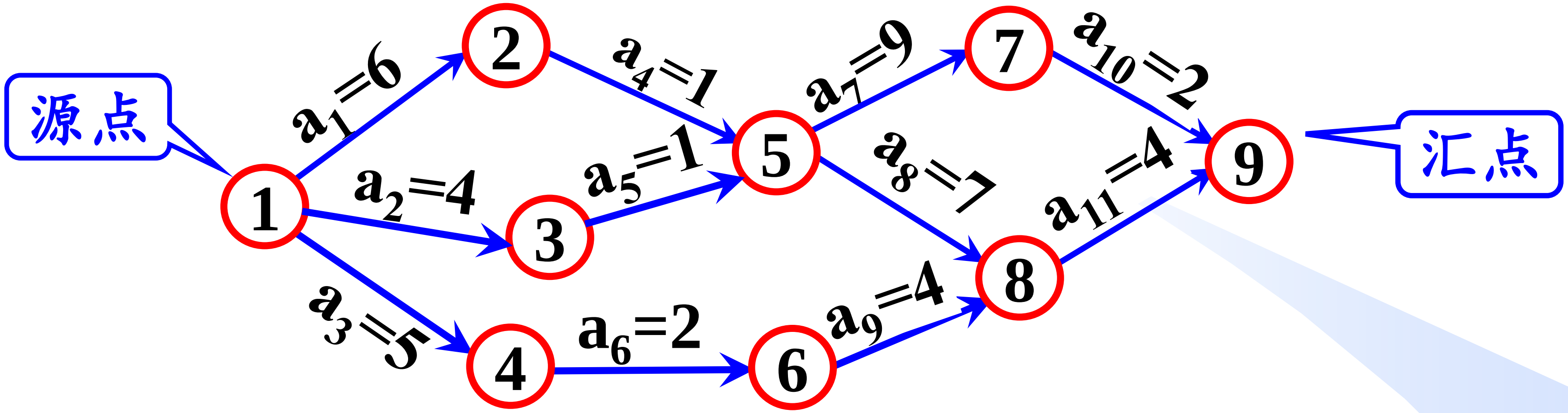


	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	16	14	18
vl	0	6	6	8	7	10	16	14	18

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

A

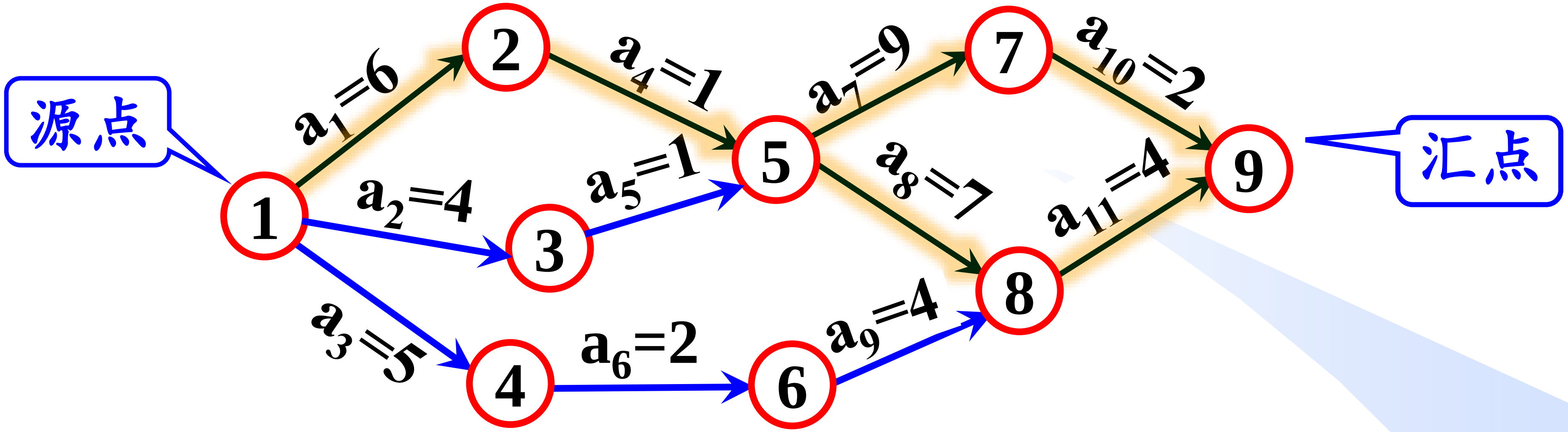


	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	16	14	18
vl	0	6	6	8	7	10	16	14	18

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

A



	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

A

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；

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

②按逆拓扑序求各顶点的最迟发生时间vl；

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

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$\langle j,k\rangle$上：$e(i)=ve(j)$，$l(i)=vl(k)-weight(\langle j,k\rangle)$

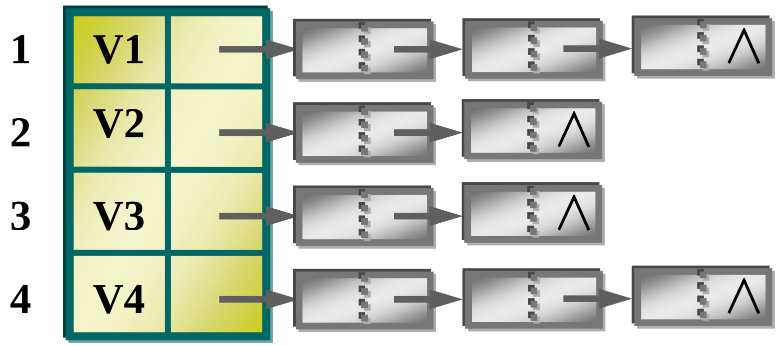
B

计算顶点的最早发生时间，存入ve数组

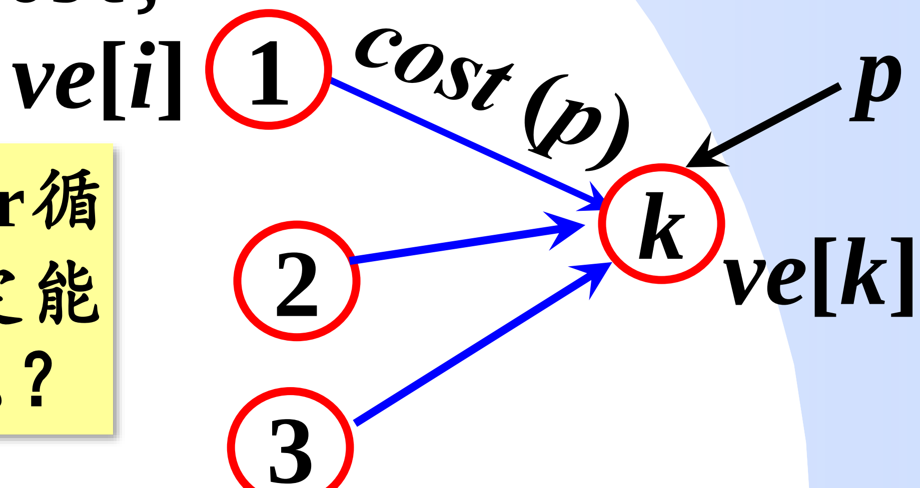
```
void VertexEarliestTime(Vertex Head[],int n, int ve[]){
     for(int i=1;i<=n;i++)
          ve[i]=0;
     for(int i=1;i<=n;i++)  //按拓扑序计算各顶点最早发生时间
          for(Edge* p=Head[i].adjacent; p!=NULL; p=p->link){
               int k = p->VerAdj;
               if(ve[i] + p->cost > ve[k])
          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

假定图中顶点已按拓扑排序编号

时间复杂度 $O(n+e)$

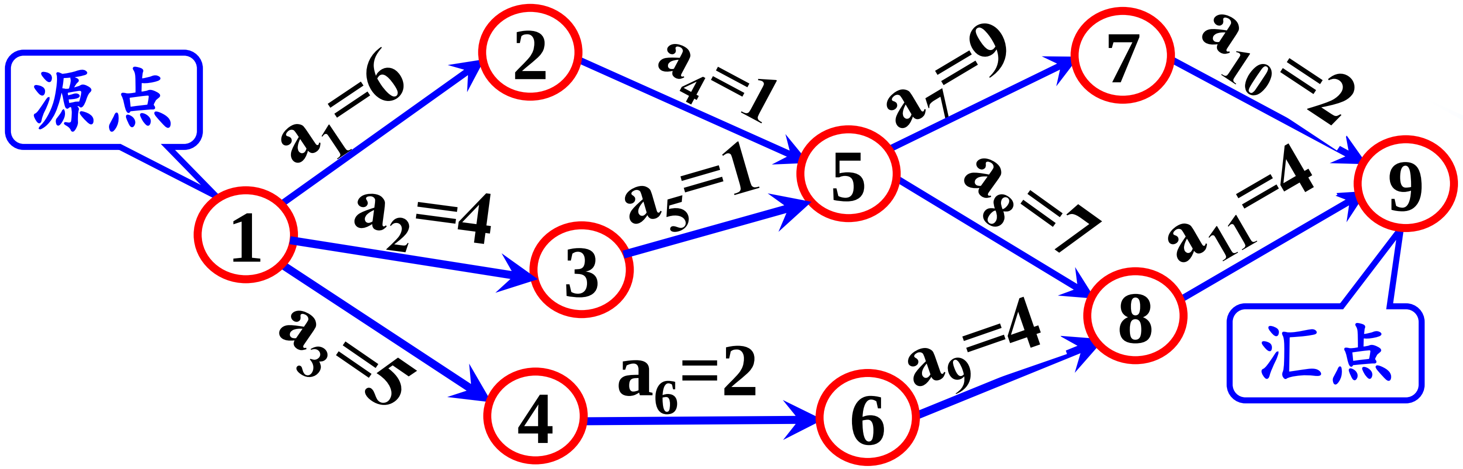


思考1：内层for循环结束后，一定能确定ve[k]的值么？



计算顶点的最早发生时间

B



$i=2$
计算 $ve(2)+w(2, 5)$

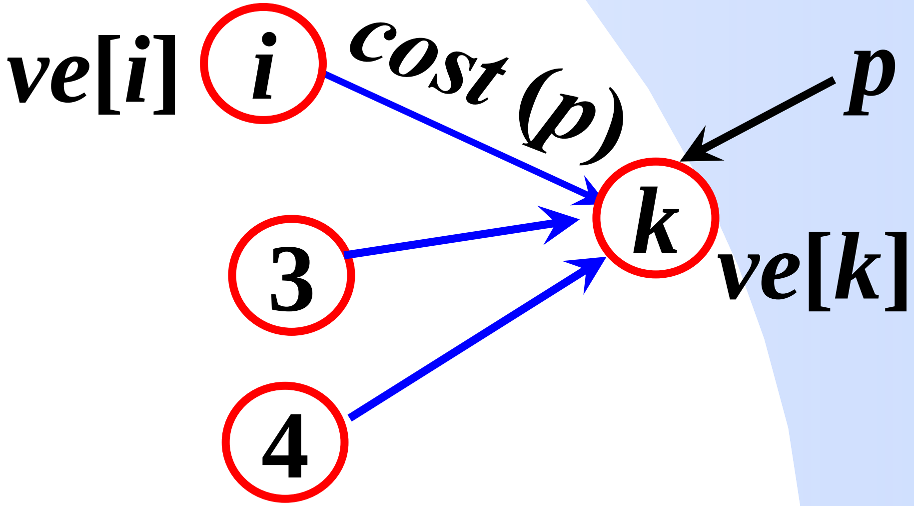
$i=3$
计算 $ve(3)+w(3, 5)$

```
for(int i=1;i<=n;i++)
   for(Edge* p=Head[i].adjacent;p;p=p->link){
        int k = p->VerAdj;
        if(ve[i] + p->cost > ve[k])
          ve[k] = ve[i]+p->cost;
   }
```

计算顶点的最早发生时间

B

```
void VertexEarliestTime(Vertex Head[],int n, int ve[]){
    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？

B

```
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;
        }
}
```

图中顶点未按拓扑序编号

不是处理顶点*i*，而是处理拓扑序列中第*i*个顶点

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

初始调用：

```
TopoSort(Head,Topo);//求拓扑序并存入Topo数组
VertexEarliestTime(Head, Topo, n, ve);
```

B

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

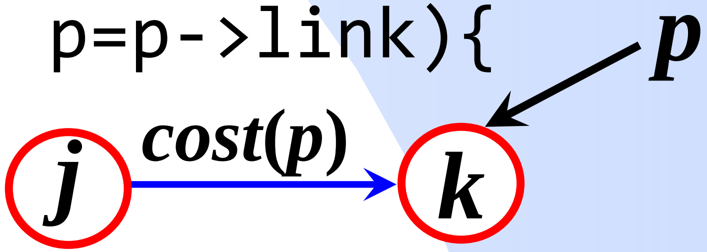
拓扑排序

按拓扑序选出一个点
扫描其邻接顶点*k*
更新顶点*k*的入度

求*ve*值

按拓扑序选出一个点
扫描其邻接顶点*k*
更新顶点*k*的*ve*值

两个过程合并



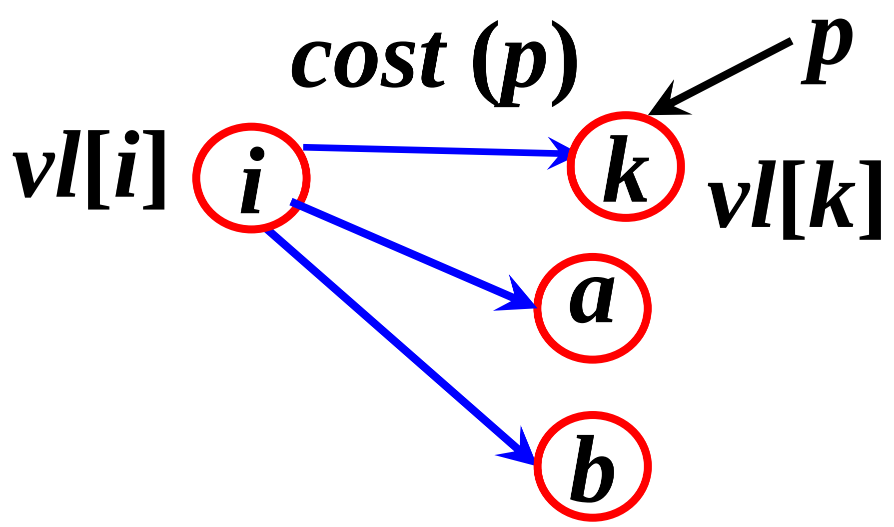
方案2：拓扑排序过程中，弹栈选出入度为0的顶点并更新其邻接顶点的入度时，顺带更新ve值，从而无需调用VertexEarliestTime函数

计算顶点的最迟发生时间，存入Vl数组

B

```
void VertexLatestTime(Vertex* Head,int n,int ve[],int vl[]){
    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;
       }
}
```

时间复杂度O(n+e)



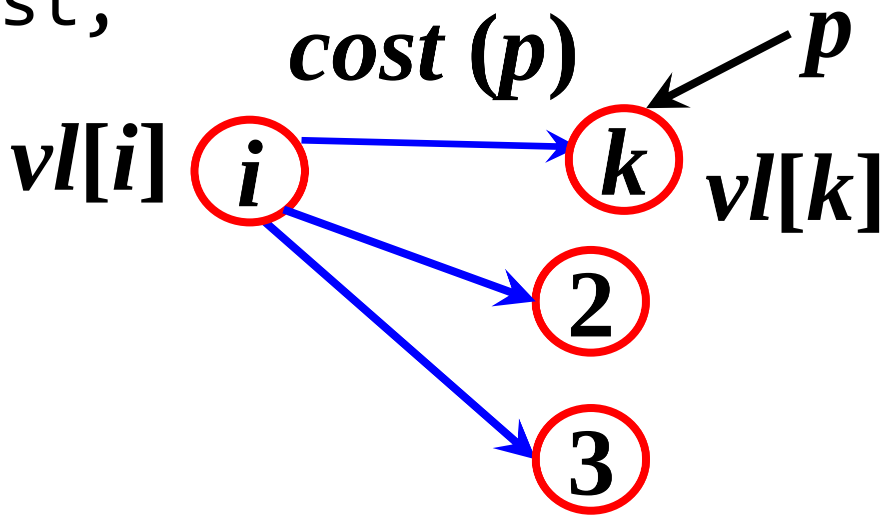
思考1：内层for循环结束后，能确定$vl[i]$的值么？

B

计算顶点的最迟发生时间

```
void VertexLatestTime(Vertex *Head,int n,int Topo[],int ve[],int vl[]){
    for(int i=1;i<=n;i++)
       vl[i]=ve[Topo[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



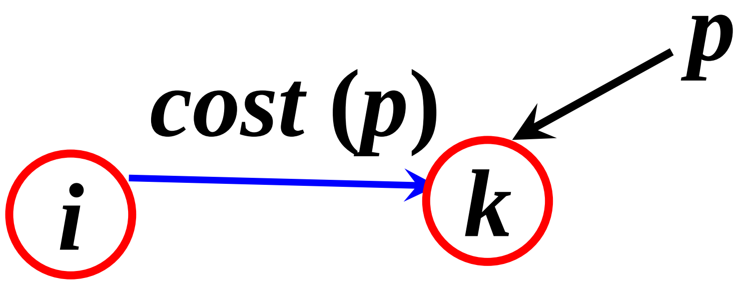
思考2：如果图中顶点未按拓扑序编号，怎么办？

计算活动的最早和最迟开始时间

B

```
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
        }
}
```

时间复杂度 $O(e)$



求关键路径和关键活动

B

```
const int maxn=1010;
void CriticalPath(Vertex* Head, int n){
   //假定图中顶点已按拓扑序编号
   int ve[maxn],vl[maxn];
   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
}
```

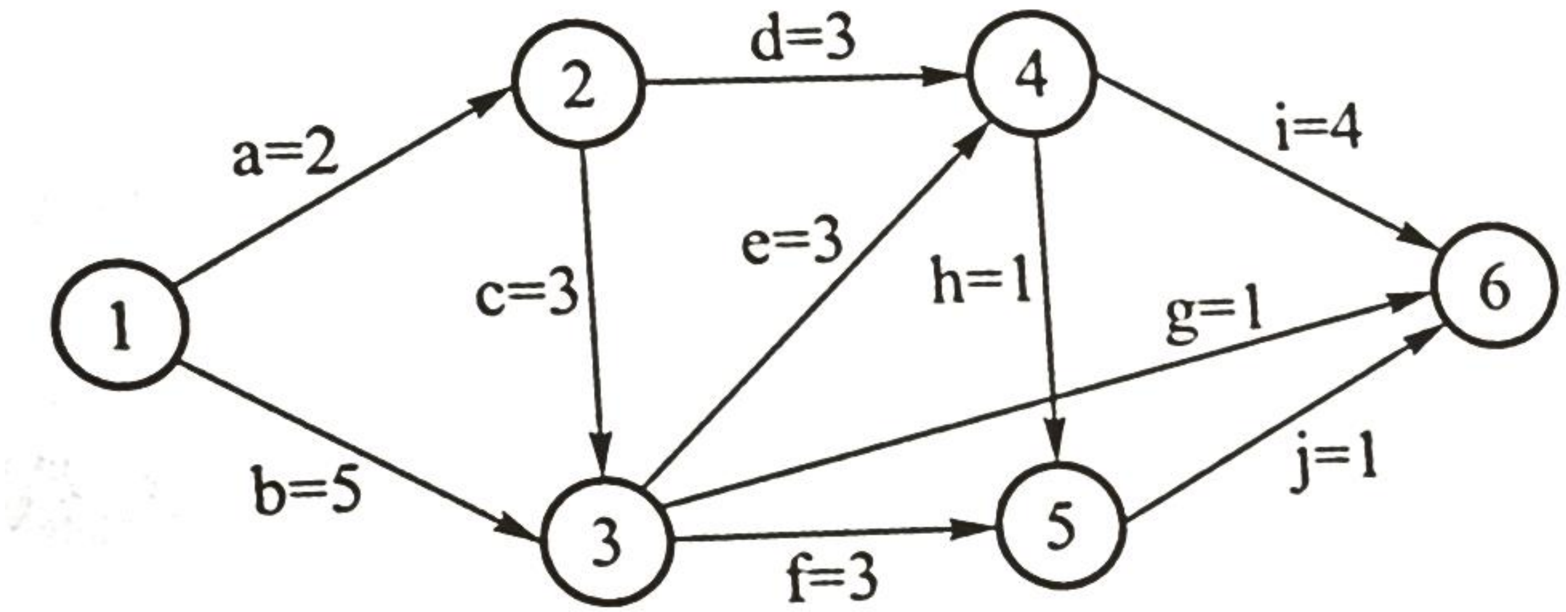
时间复杂度
$O(n+e)$

课下思考

A

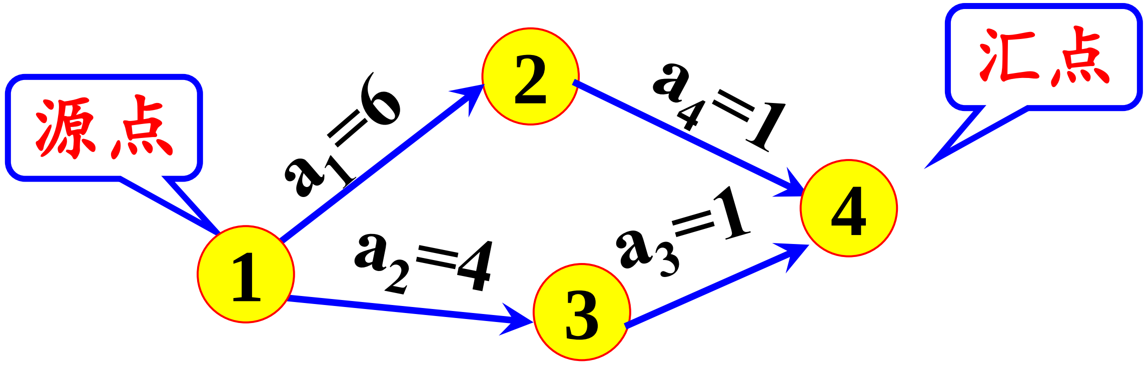
下图是有10个活动的AOE网，其中时间余量最大的活动是________【2022年考研题全国卷】

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？



课下思考

B

求ve值的过程中已经算出了源点到汇点的最长距离（同时也可以求出对应的最长路径），最长路径（关键路径）上包含的边即为关键活动，为什么还需要继续算vl值以及各活动的最早最晚开始时间?