第七章 图

7.5 拓扑排序

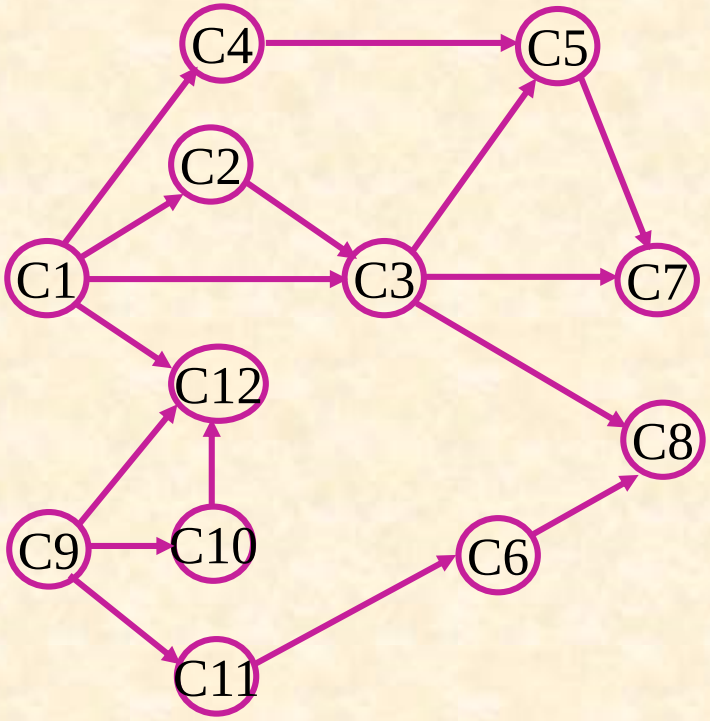
计划、**施工过程**、**生产流程**、**程序流程**等都是“**工程**”。除了很小的工程外，一般都把工程分为若干个叫做“**活动**”的子工程。完成了这些活动，这个工程就可以完成了。

AOV网：在有向图中，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)**。

[例]按拓扑次序安排计算机专业必修课程

计算机专业必修课程

课程代	课程名称	先修课
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1, C2
C4	汇编语言	C1
C5	语言设计和分析	C3, C4
C6	计算机原理	C11
C7	编译原理	C3. C5
C8	操作系统	C3, C6
C9	高等数学	无
C10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C1, C9, C10



例如，计算机专业学生的学习就是一个**工程**，每一门课程的学习就是整个工程的一些**活动**。其中有些课程**要求先修**，有些则不要求。这样在有的课程之间有**领先关系**，有的课程可以**并行**地学习。

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$<V_i, V_j>$。

AOV网络中**不能出现有向回路**，即**有向环**。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**。因此，必须先判断它是否存在**有向环**。

拓扑序列：

把**AOV网**中的所有顶点排成一个**线性序列**，使每个活动的所有**前驱活动**都排在该活动的前边。

拓扑排序：

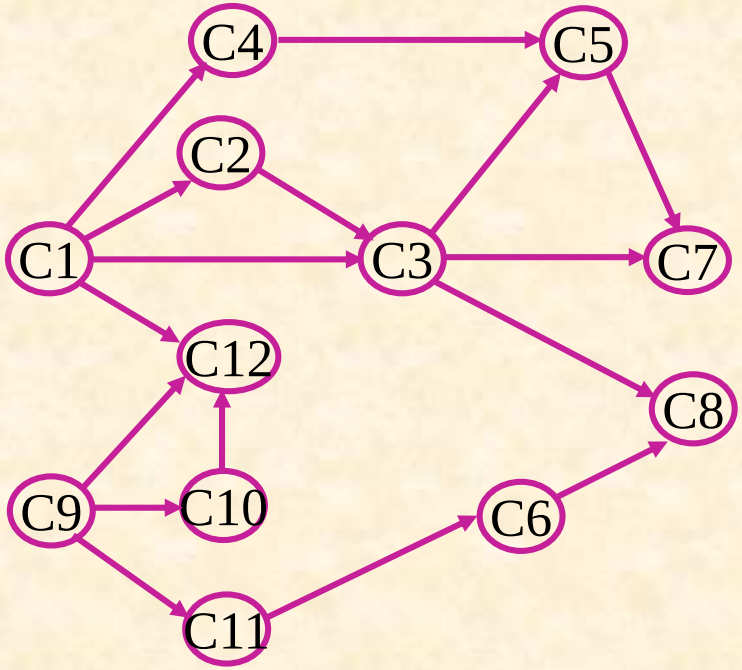
构造**AOV网**的**拓扑序列**的过程被称为**拓扑排序**。

拓扑排序基本步骤：

① 从网中选择一个**入度为0**的顶点且**输出之**。

② 从网中**删除**该**顶点及其所有出边**。

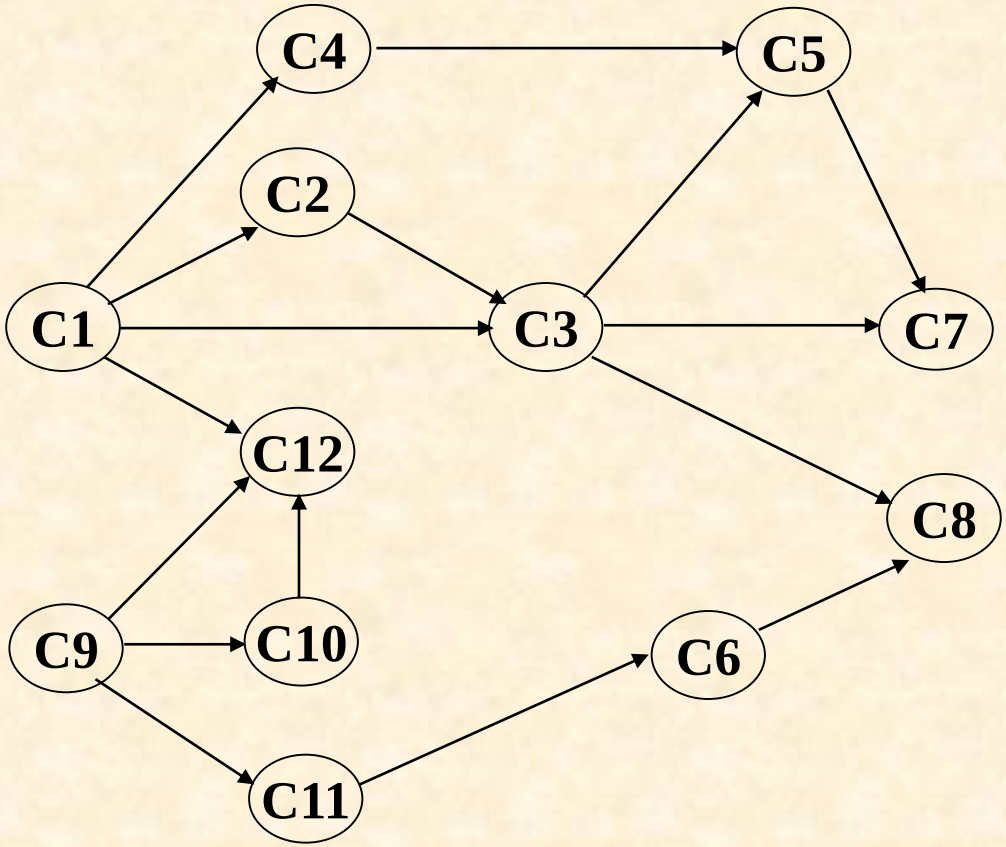
③ 执行① ②，直至所有顶点已输出，或网中**剩余顶点入度均不为0** (说明网中**存在回路**，无法继续拓扑排序)



拓扑序列：C1--C2--C3--C4--C5--C7--C9--C10--C11--C6--C12--C8

或　　：C9--C10--C11--C6--C1--C12--C4--C2--C3--C5--C7--C8

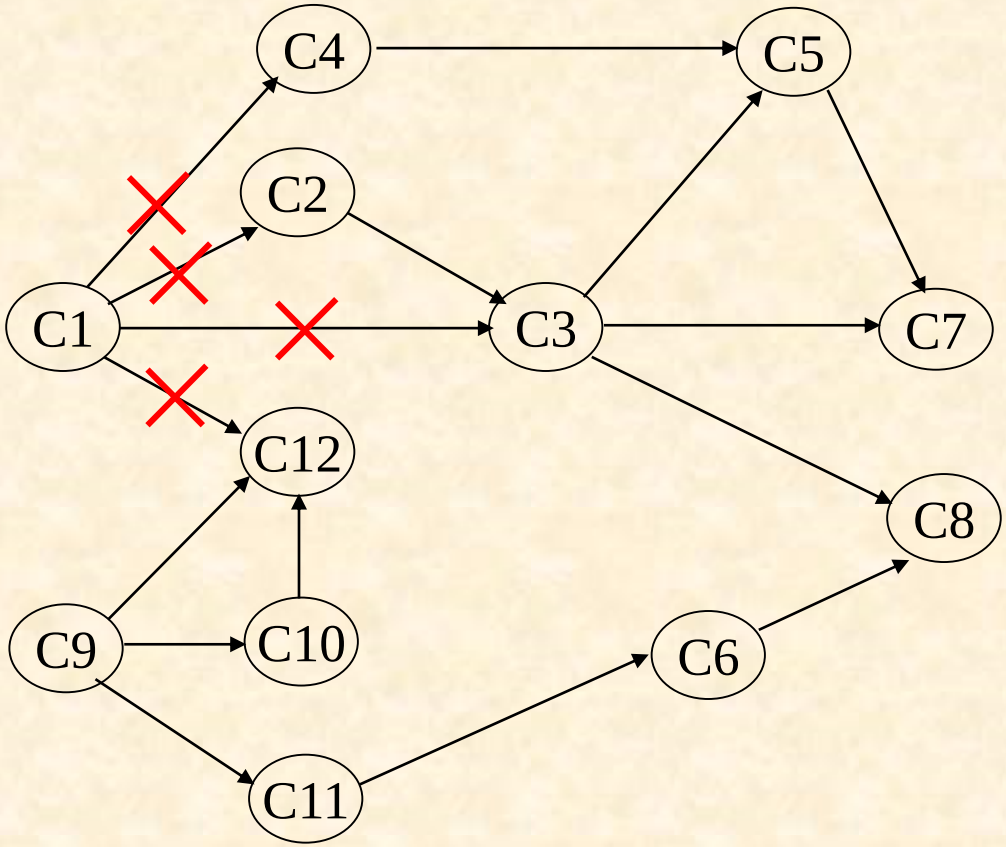
一个AOV网的拓扑序列不是唯一的

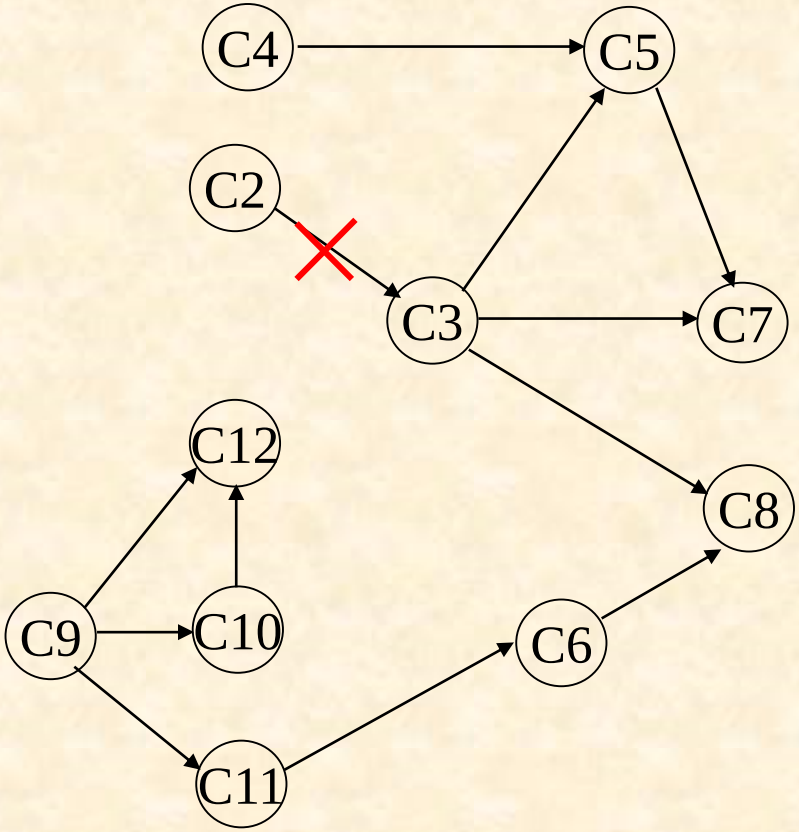


拓扑序列：C1--C2--C3--C4--C5--C7--C9--C10--C11--C6--C12--C8

或　　：C9--C10--C11--C6--C1--C12--C4--C2--C3--C5--C7--C8

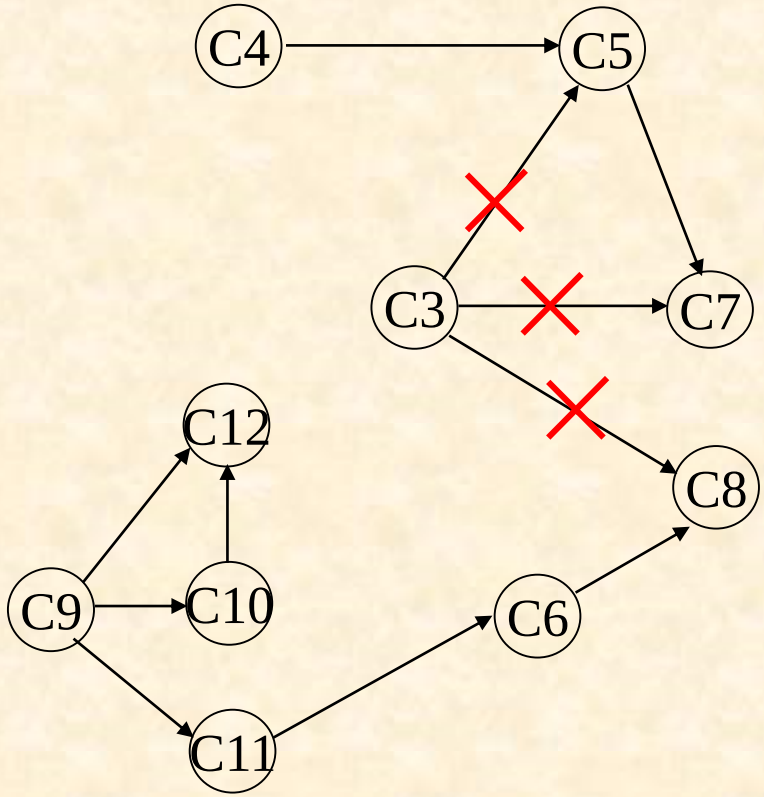
一个AOV网的拓扑序列不是唯一的


C4
C5
C2
C1
C3
C7
C12
C8
C9
C10
C6
C11



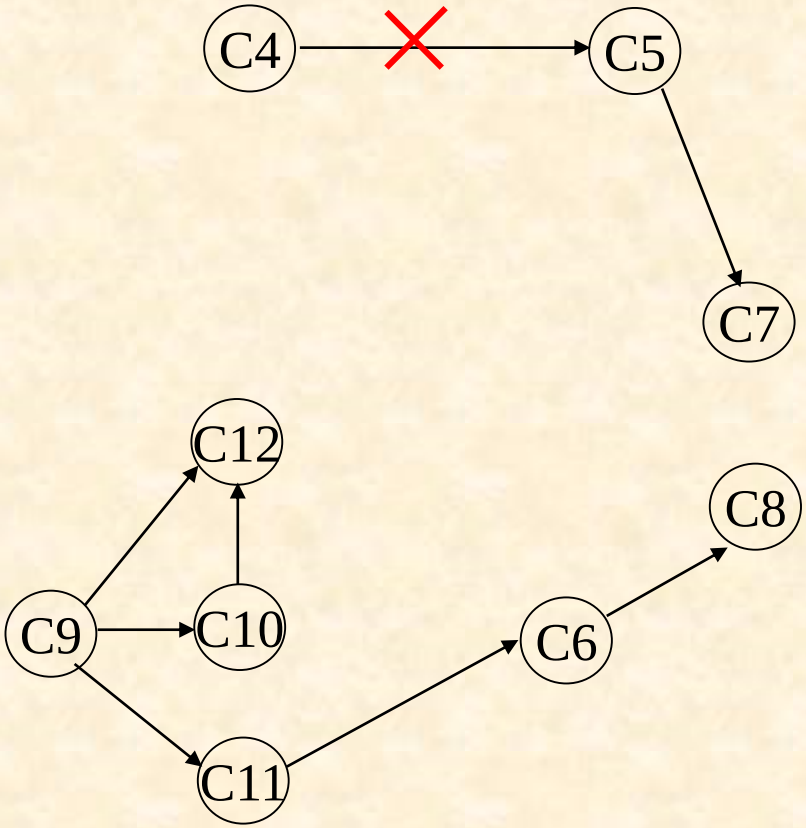
拓扑序列：C1

（1）



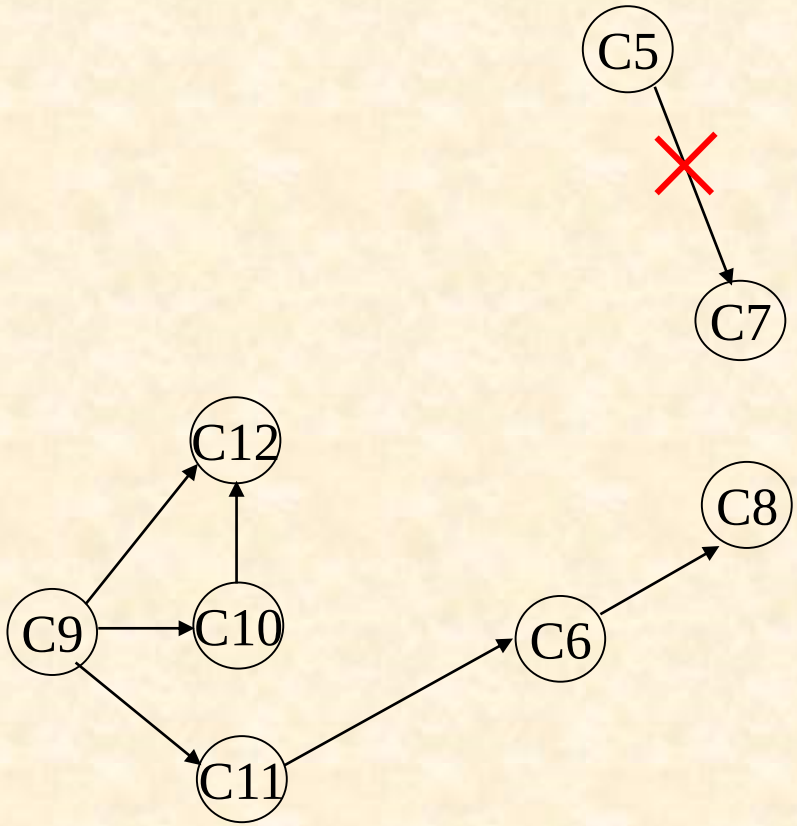
拓扑序列：C1--C2

（2）



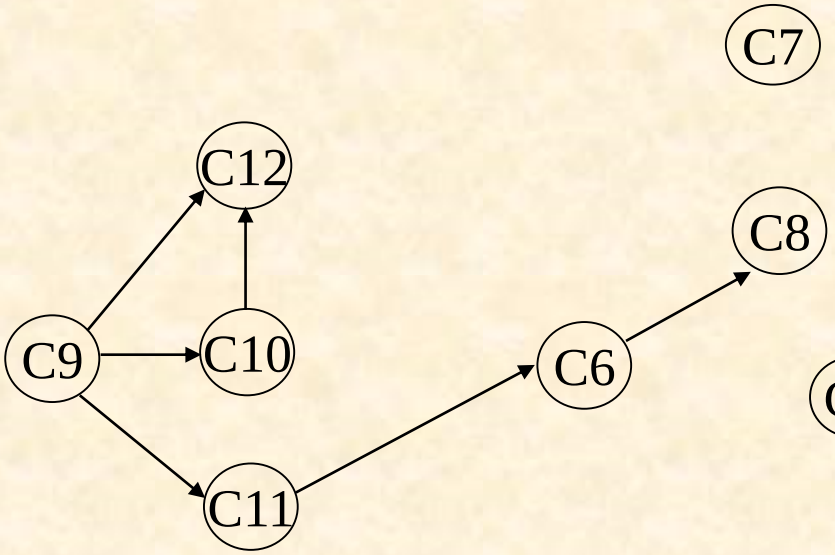
拓扑序列：C1--C2--C3

（3）



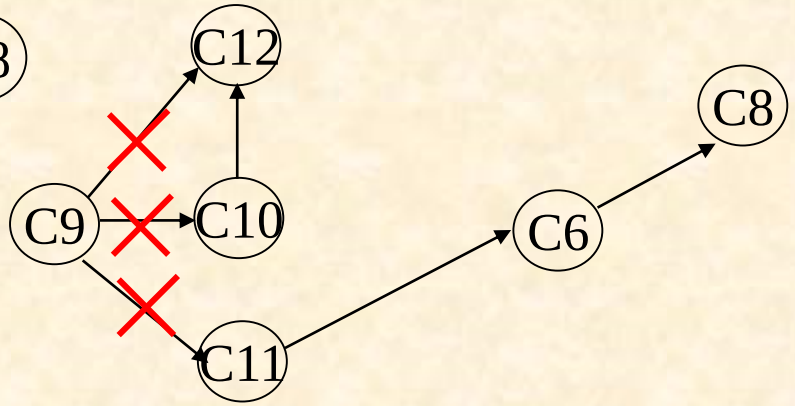
拓扑序列：C1--C2--C3--C4

（4）



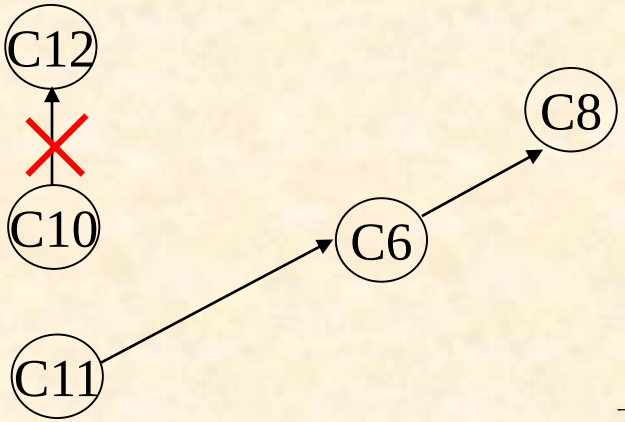
拓扑序列：C1--C2--C3--C4--C5

（5）

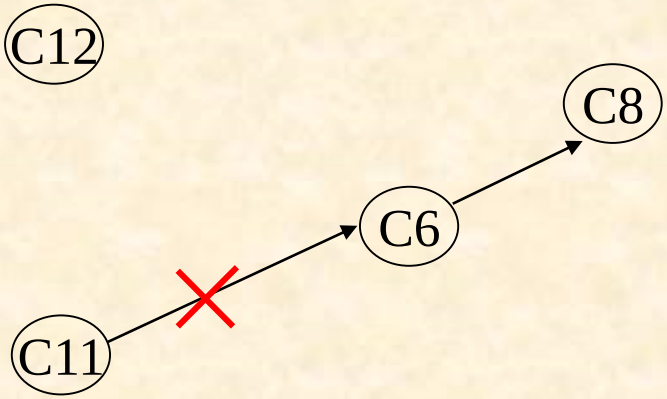


拓扑序列：C1--C2--C3--C4--C5--C7

（6）



拓扑序列：C1--C2--C3--C4--C5--C7--C9



拓扑序列：C1--C2--C3--C4--C5--C7--C9

--C10

（8）

（9）

拓扑序列：C1--C2--C3--C4--C5--C7--C9
--C10--C11

C12

C8

（10）

拓扑序列：C1--C2--C3--C4--C5--C7--C9
--C10--C11--C6

C8

（11）

拓扑序列：C1--C2--C3--C4--C5--C7--C9
--C10--C11--C6--C12

拓扑序列：C1--C2--C3--C4--C5--C7--C9
--C10--C11--C6--C12--C8

（12）

回路与拓扑排序

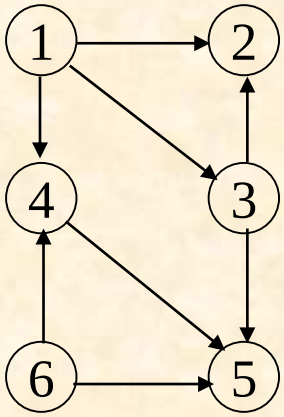
- 任何**无回路**的AOV网，其顶点均**可排成拓扑序列**(其拓扑序列**不一定唯一**)；
- 如果能将AOV网的所有顶点都排入一个**拓扑序列**，则该AOV网中必定**无有向环**；
- 如果**得不到**所有顶点的**拓扑序列**，则说明AOV网中**存在有向环**(AOV网所代表的工程是不可行的)。
- **存在回路**的AOV网，**找不到**所有顶点的**拓扑序列**。
- 因此，可以**用拓扑排序**判断有向图中**是否有回路**。

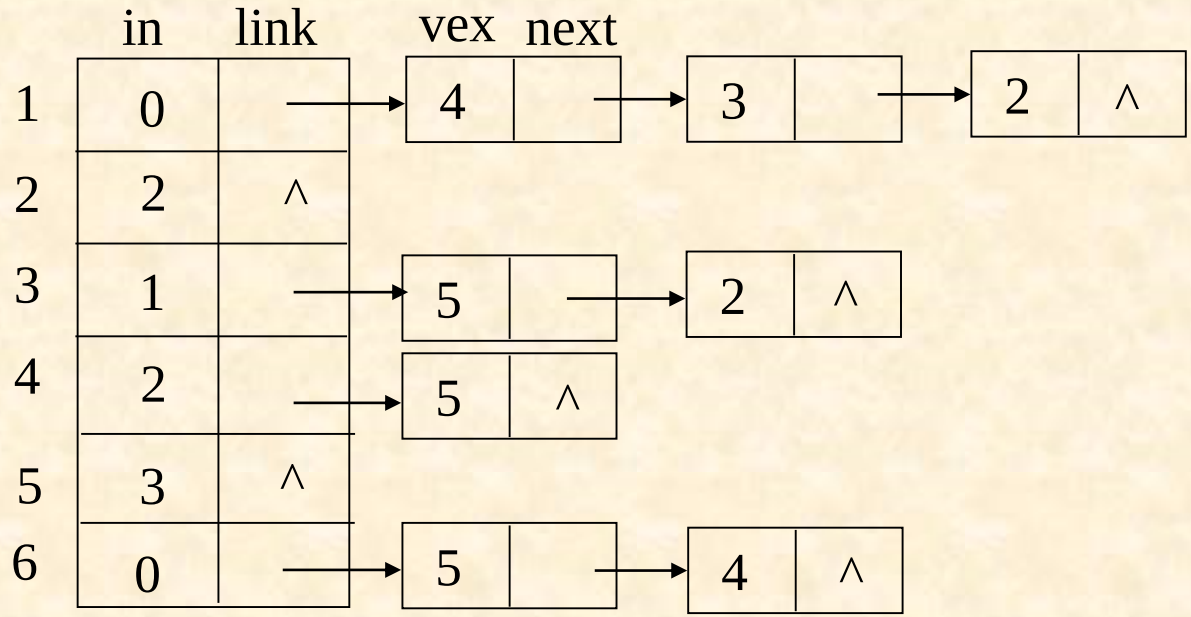
■算法实现

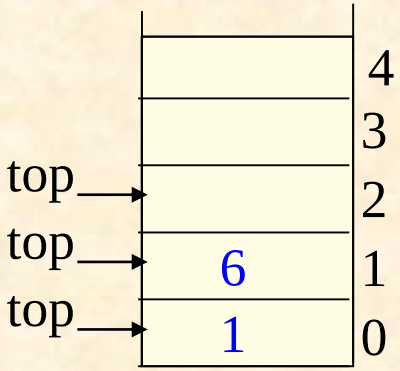
- ◆以**邻接表**作存储结构
- ◆把邻接表中所有**入度为0**的顶点**进栈**
- ◆栈非空时，**输出栈顶元素V_j**并**退栈**；在**邻接表**中查找V_j的**直接后继V_k**，把V_k**的入度减1**；若V_k的**入度为0则进栈**
- ◆**重复**上述操作直至**栈空**为止。若栈空时输出的顶点**个数不是n**，则有向图**有环**；否则，拓扑排序**完毕**

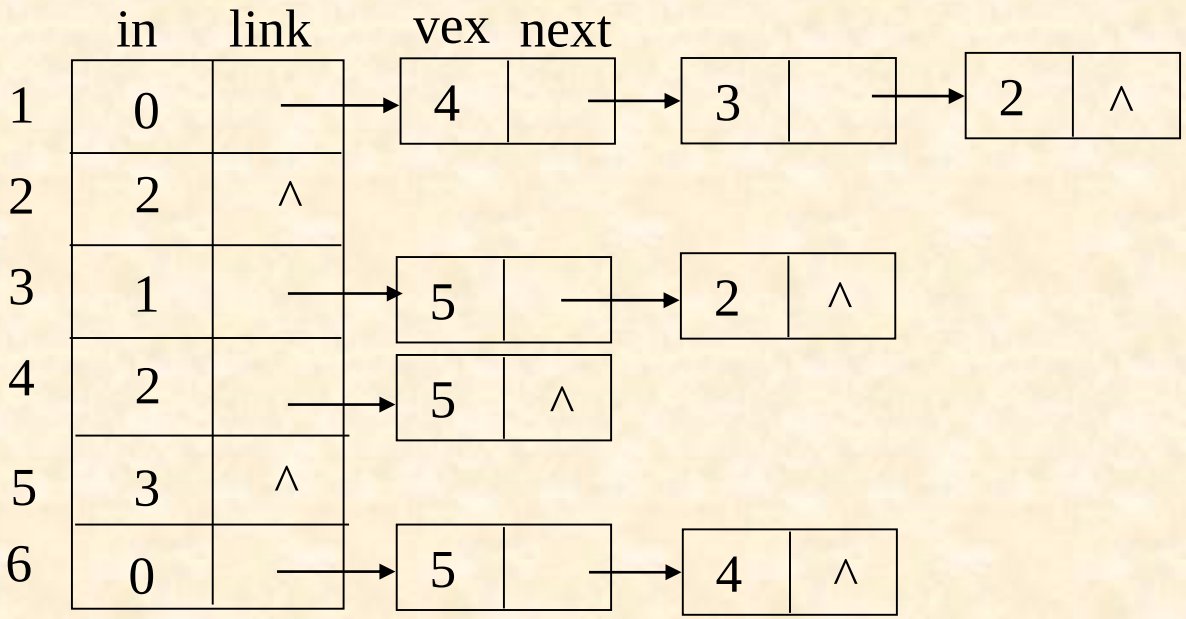
■ 算法描述

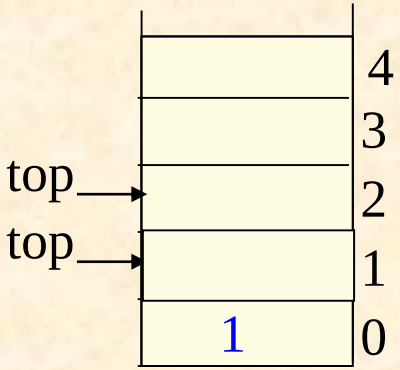
例


1
2
4
3
6
5

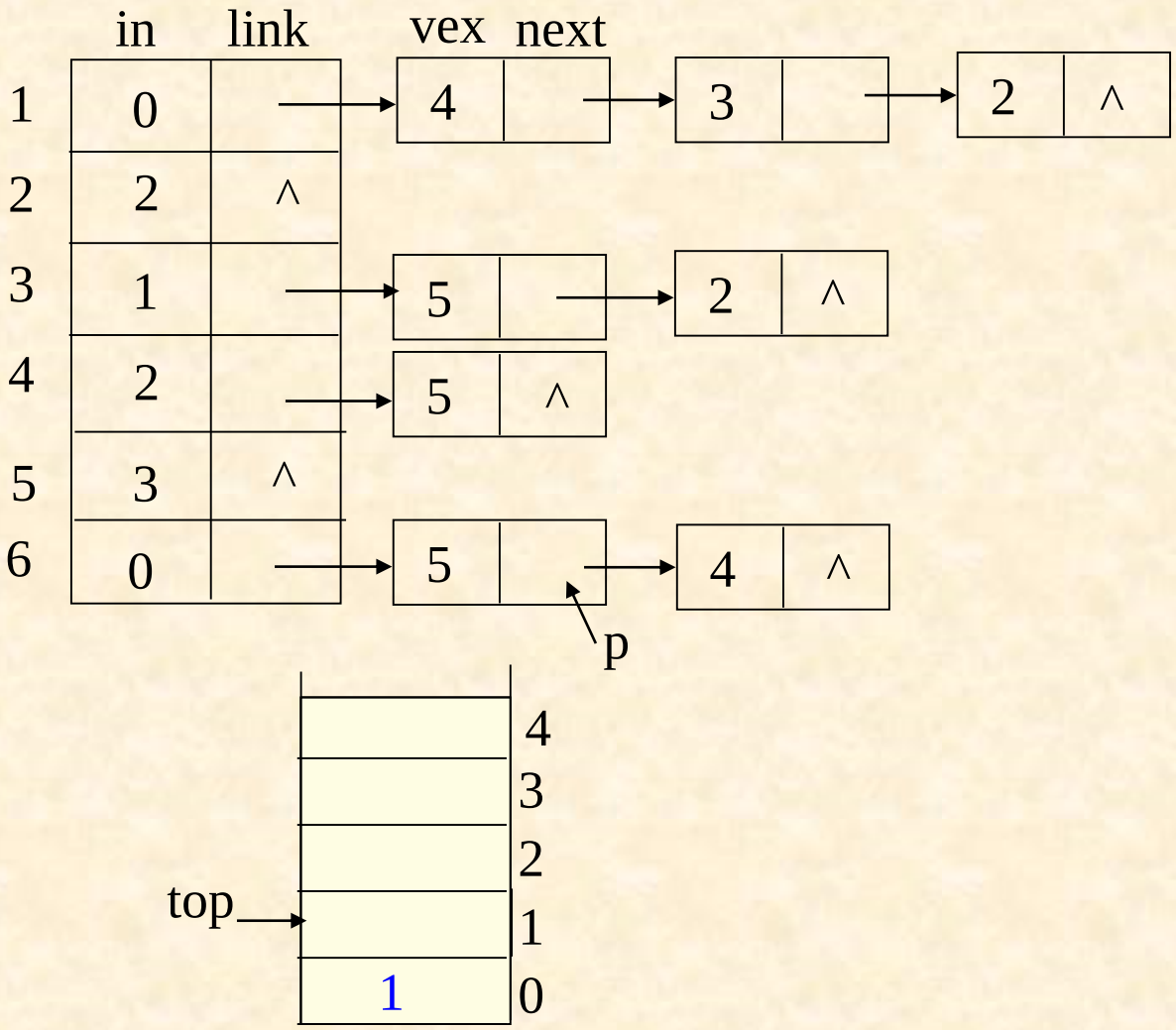

in
link
vex
next
1
0
4
3
2
∧
2
2
∧
3
1
5
2
∧
4
2
5
∧
5
3
∧
6
0
5
4
∧


top
top
top
6
1
4
3
2
1
0

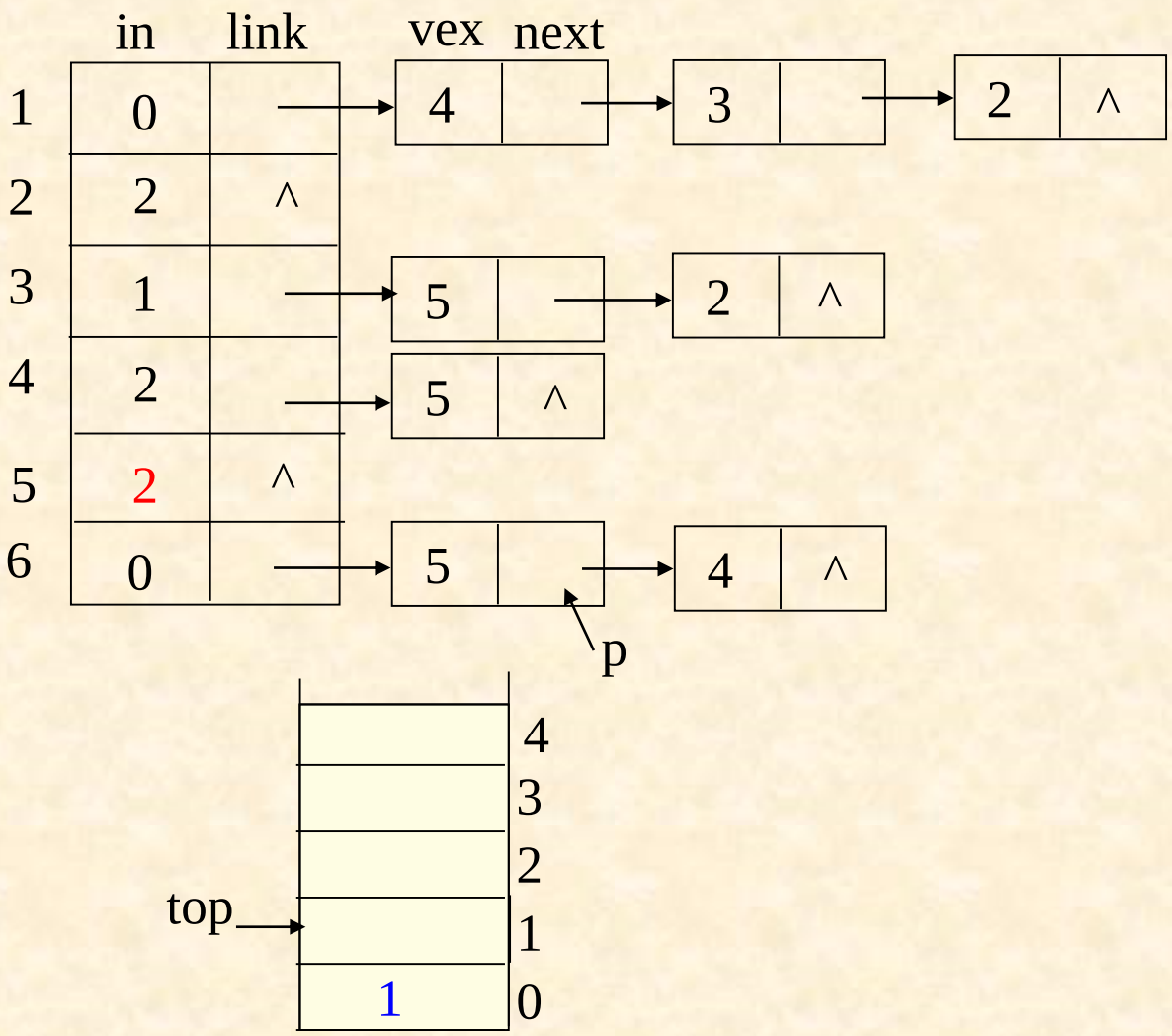




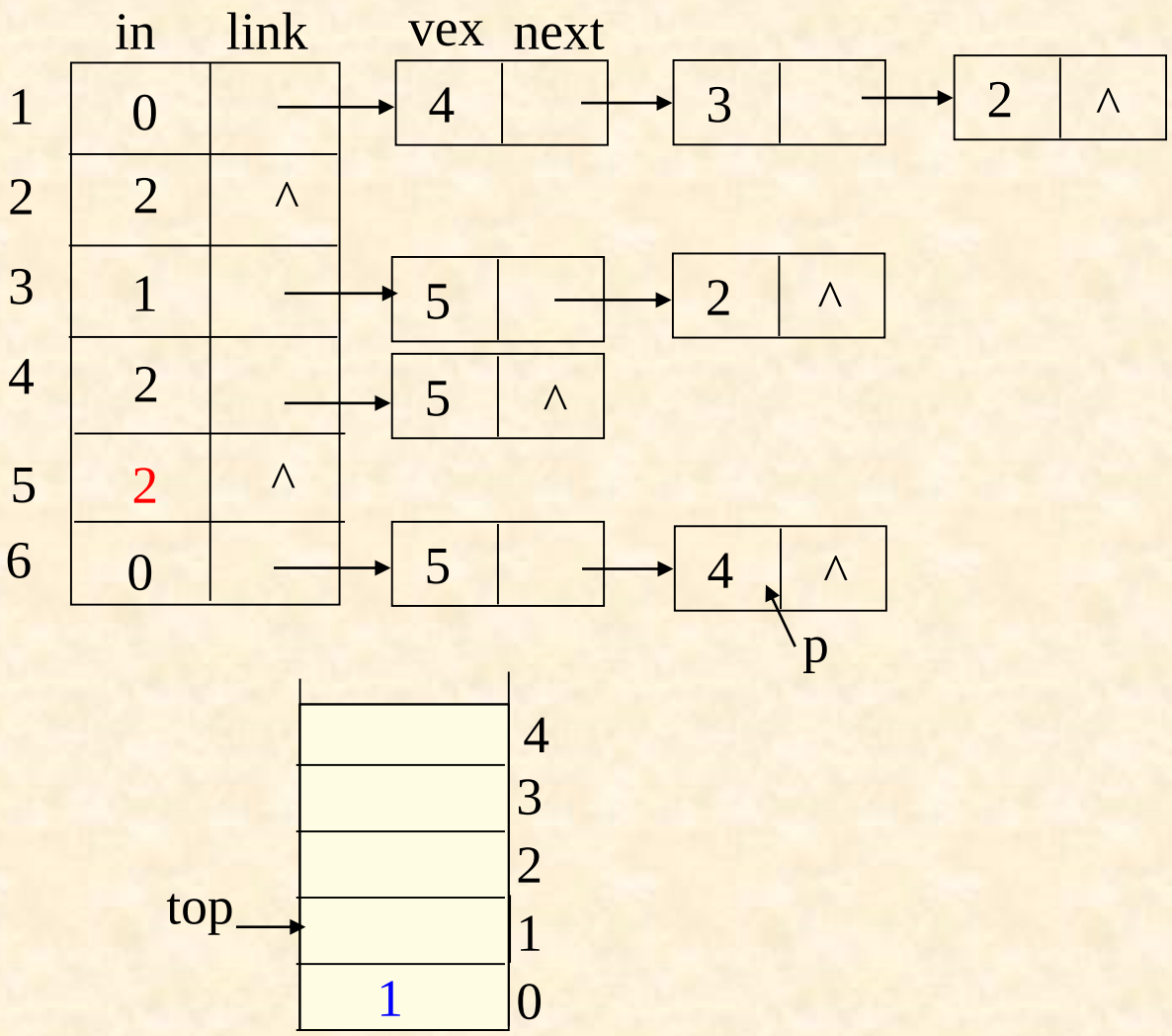
输出序列：6



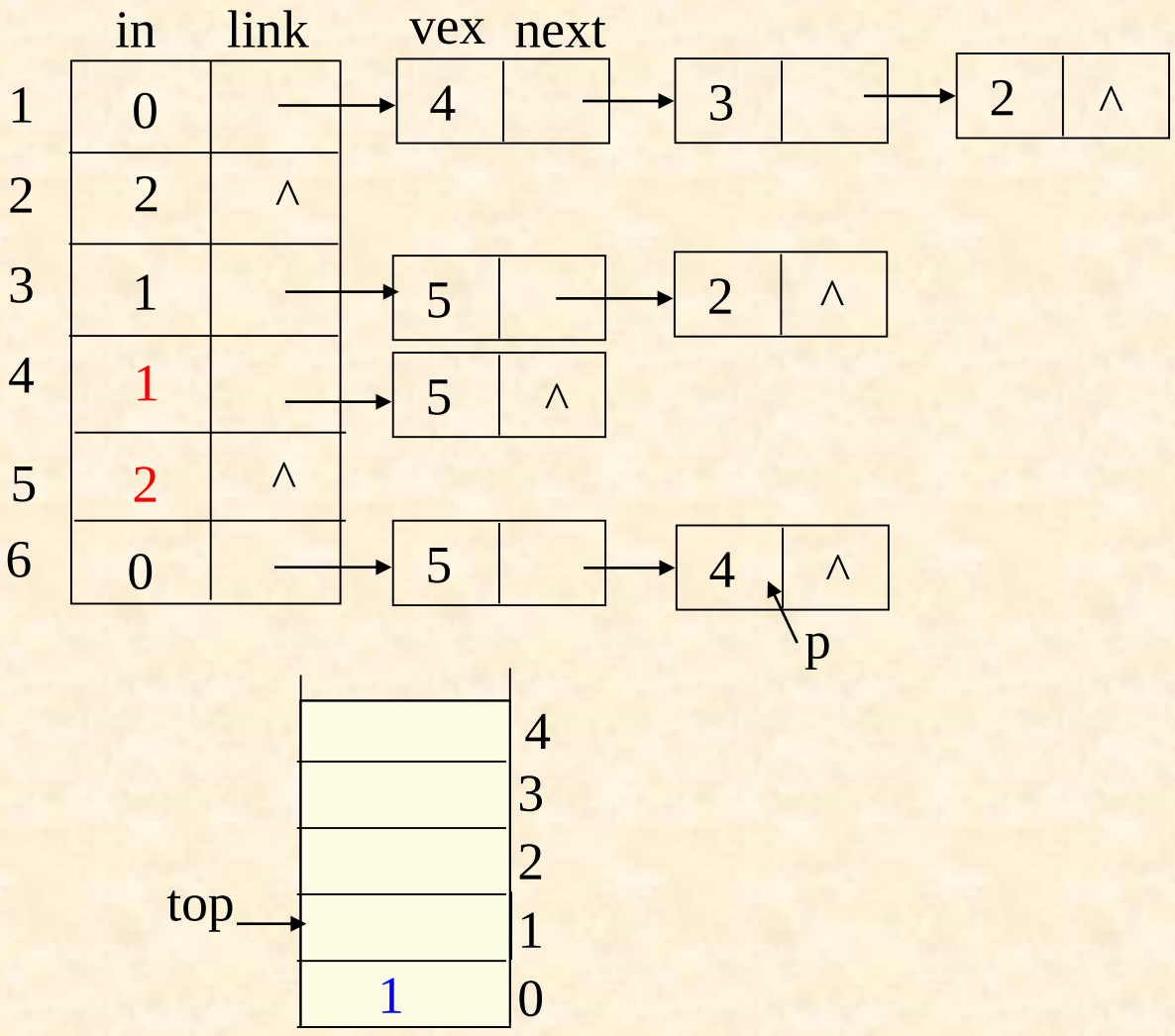
输出序列：6



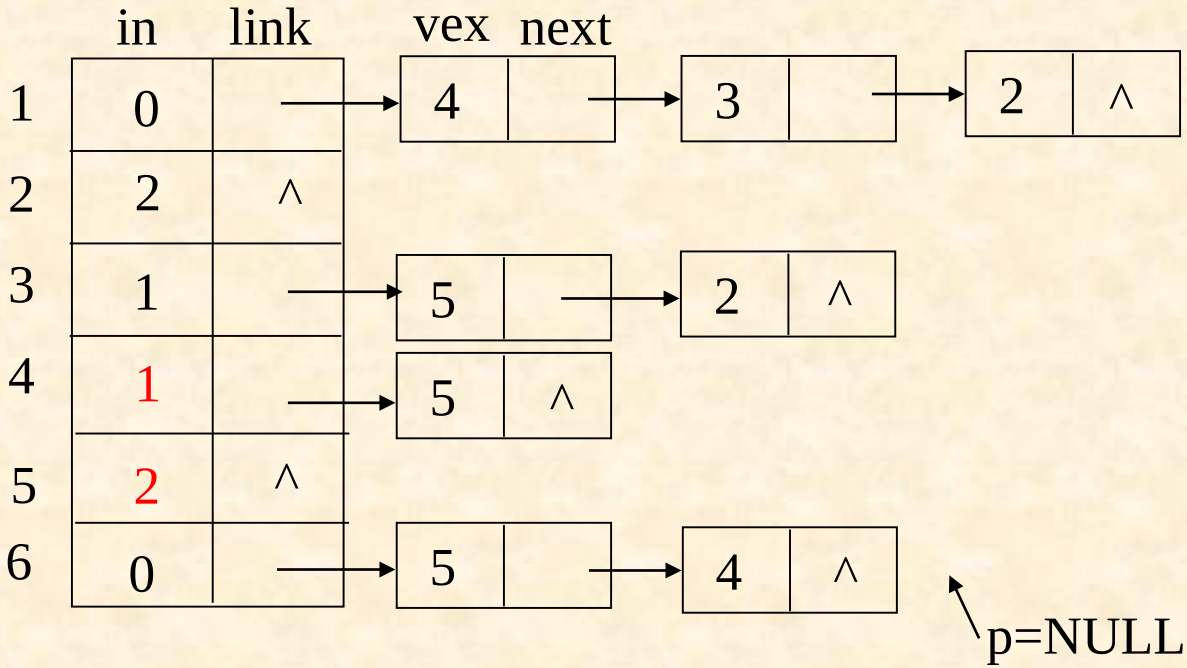
输出序列：6

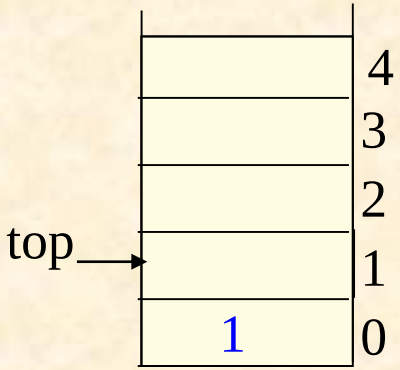


输出序列：6

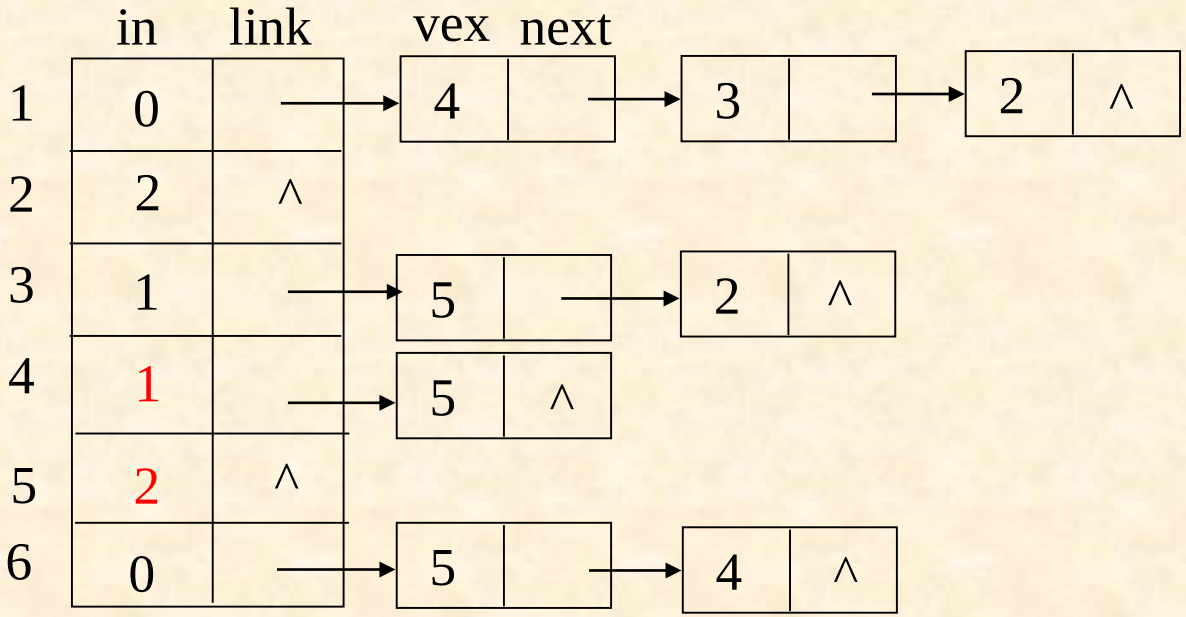


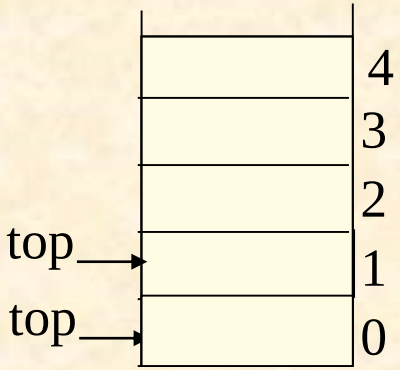
输出序列：6



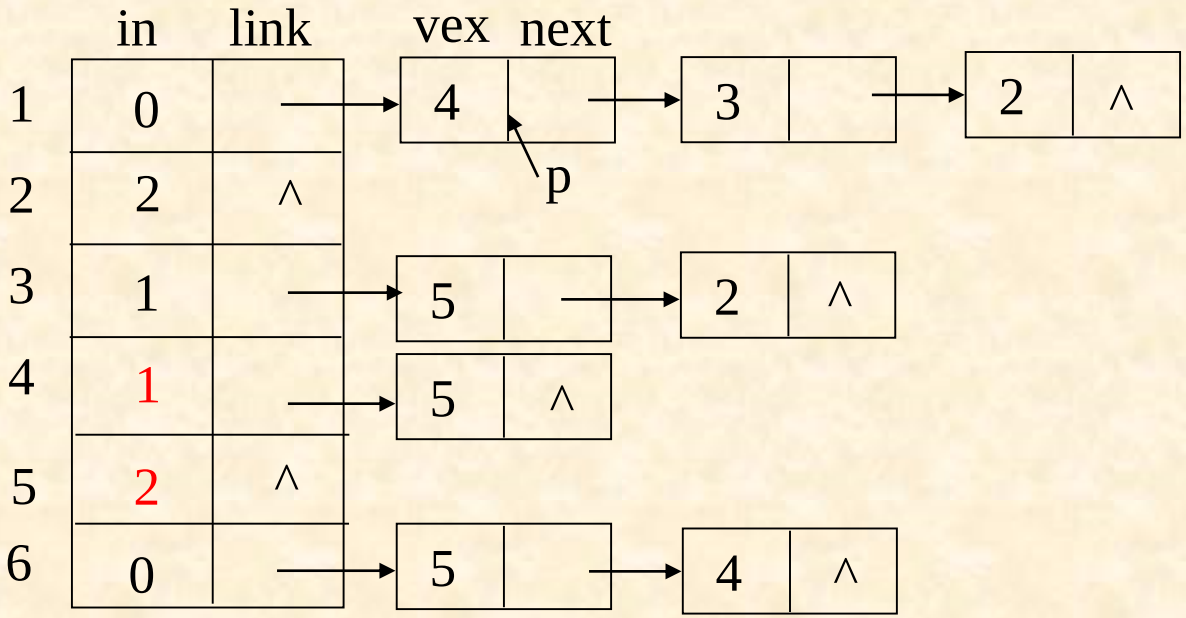


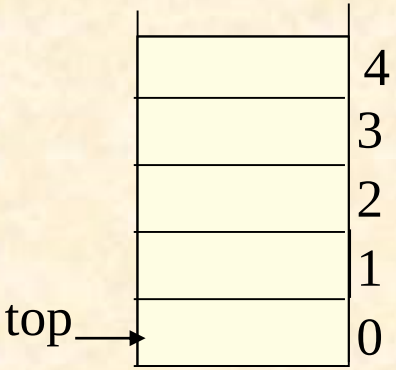
输出序列：6



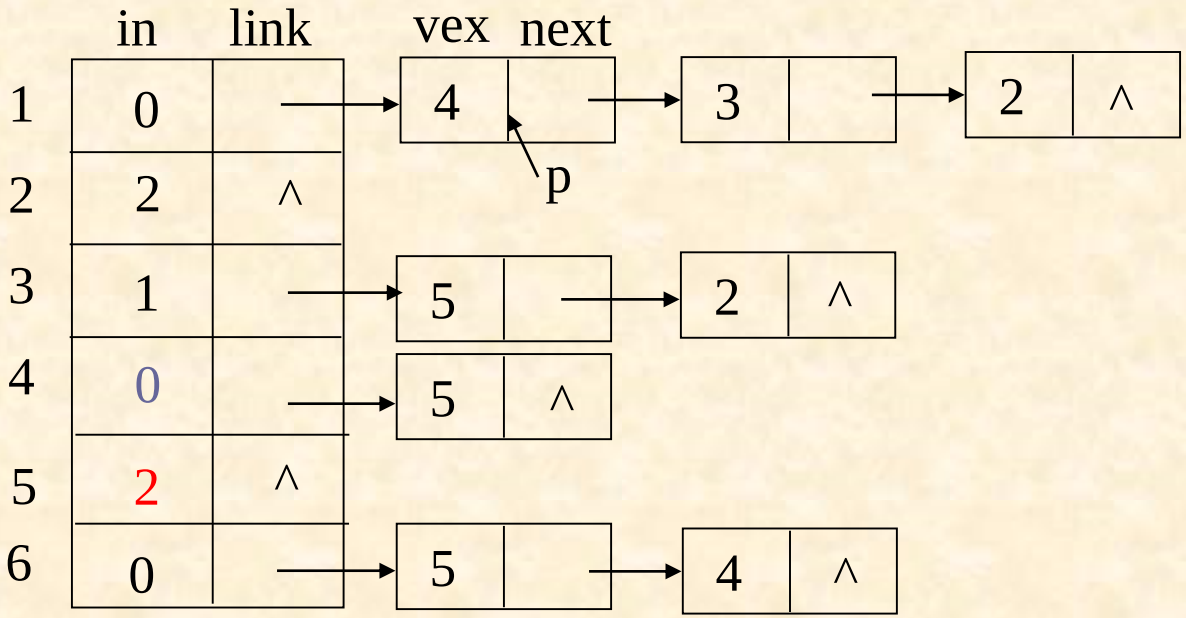


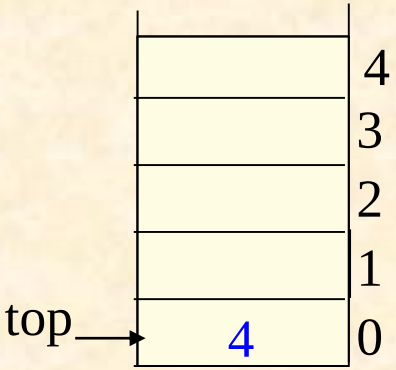
输出序列：6 1



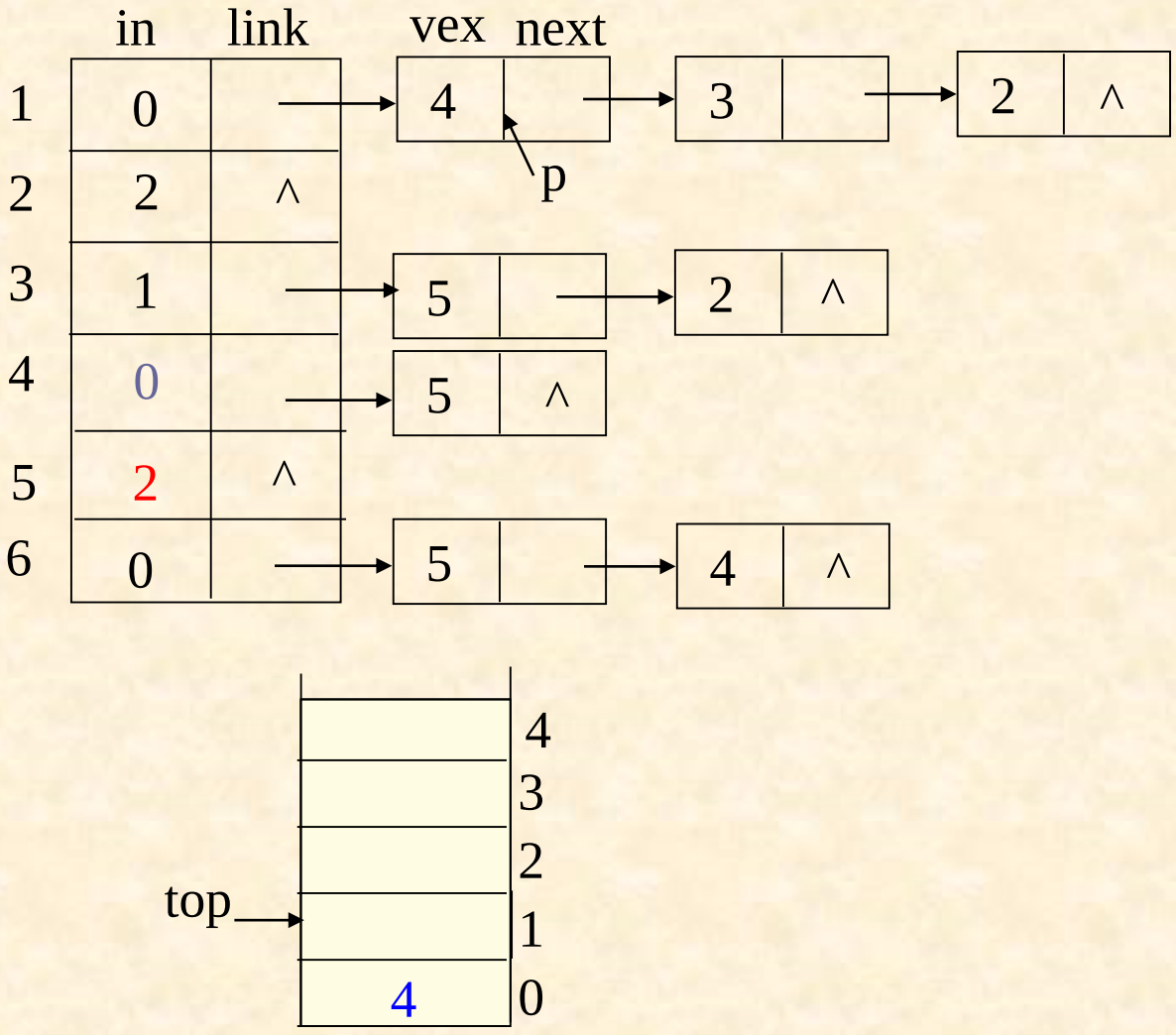


输出序列：6 1

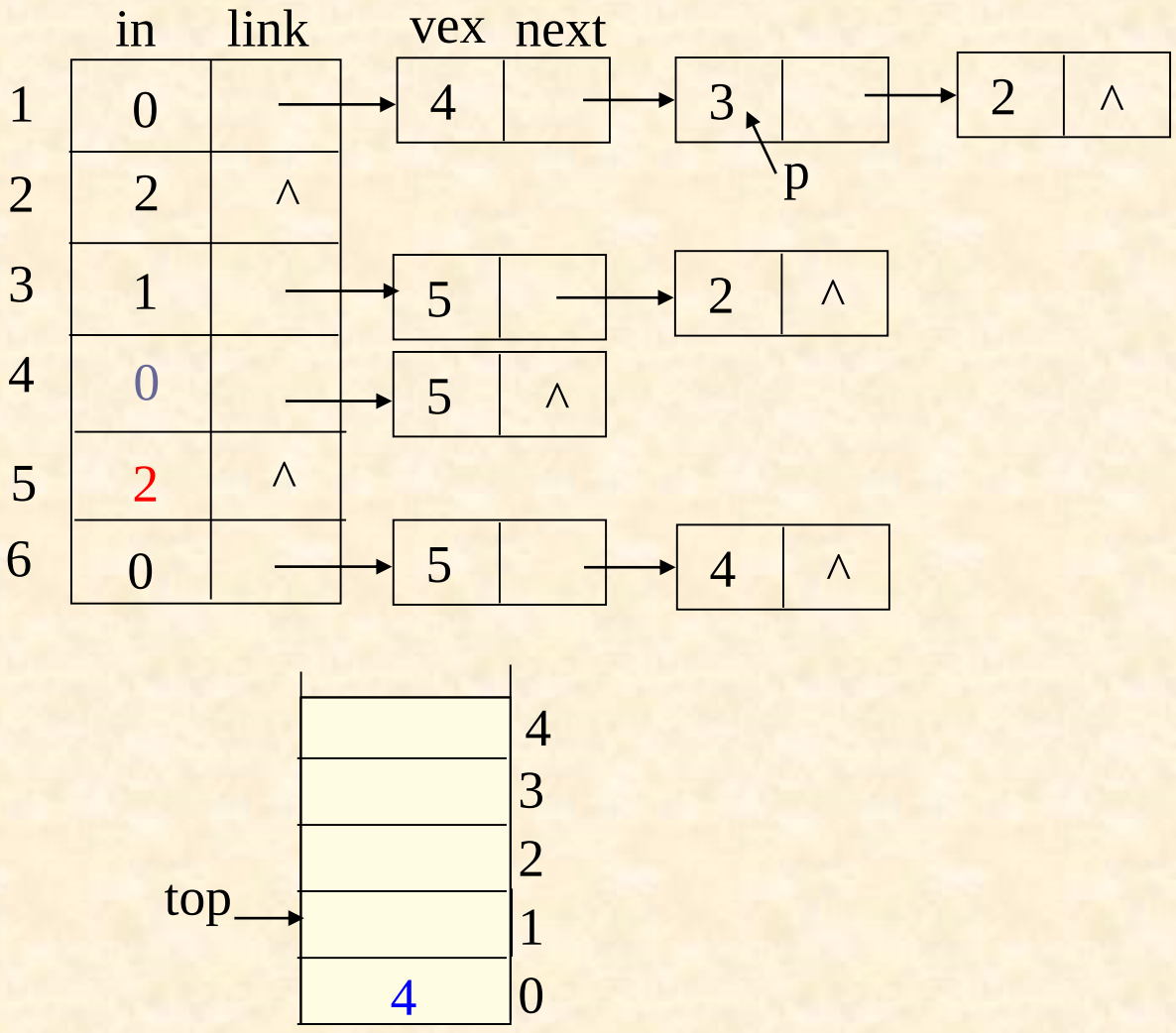




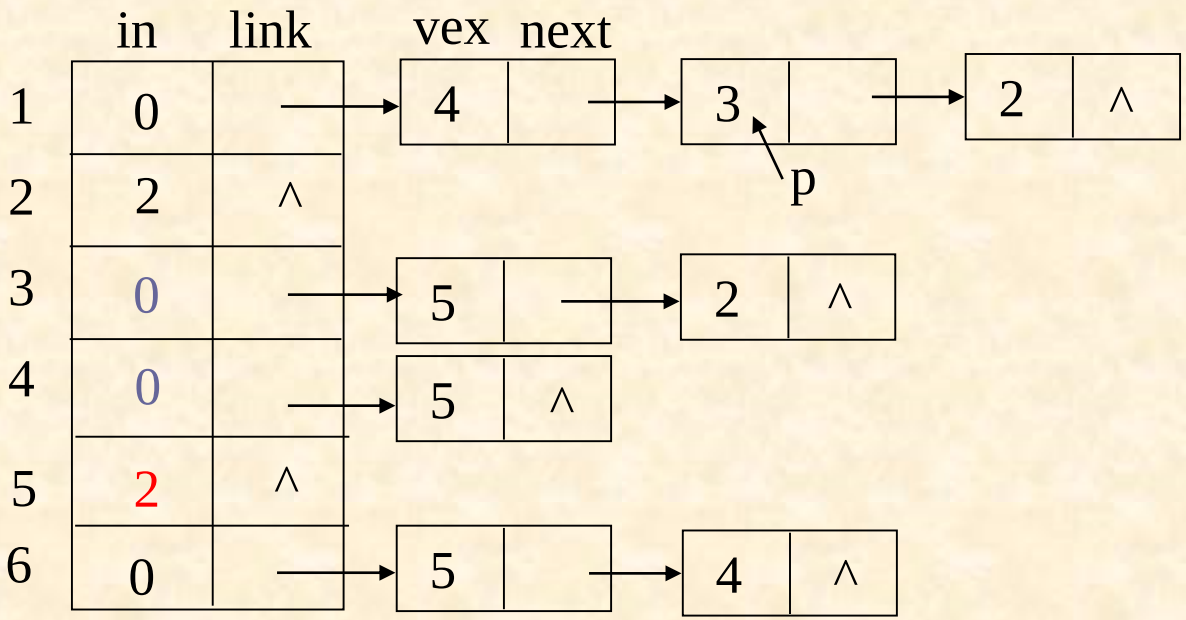
输出序列：6 1

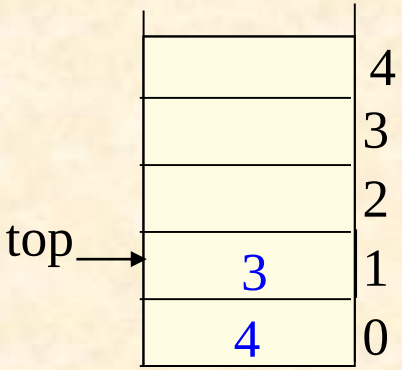


输出序列：6 1

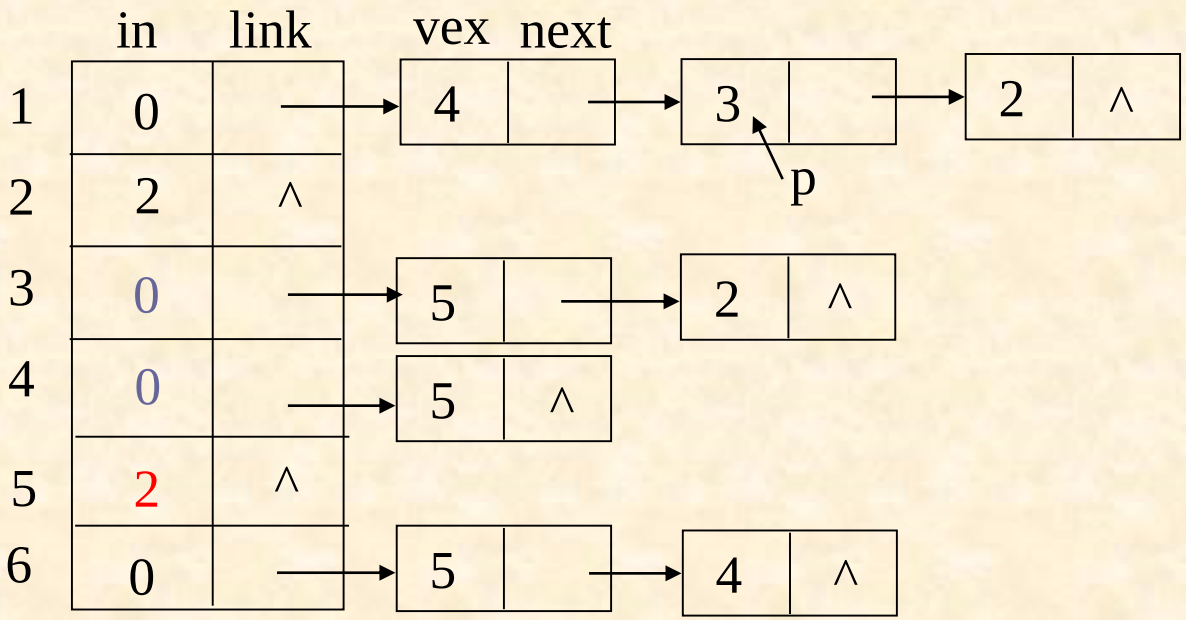


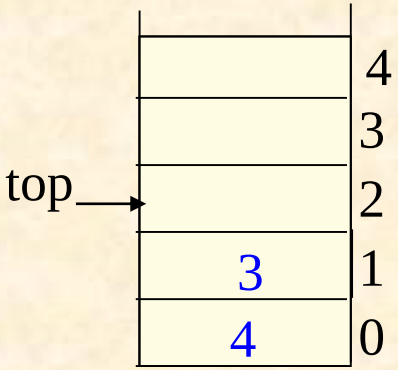
输出序列：6 1



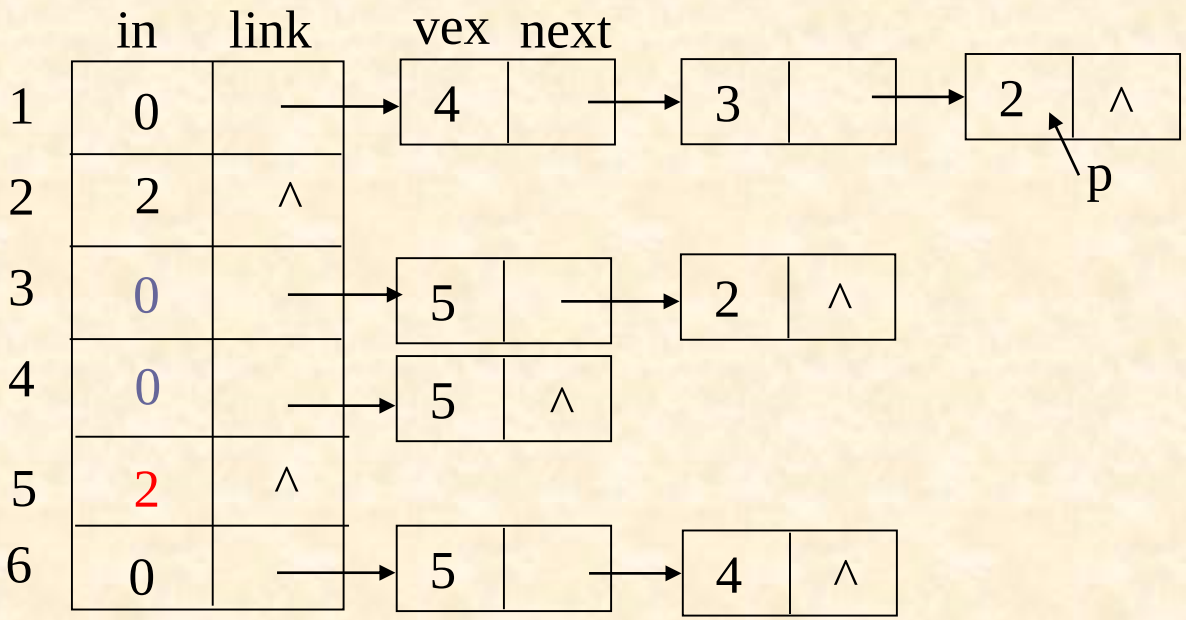


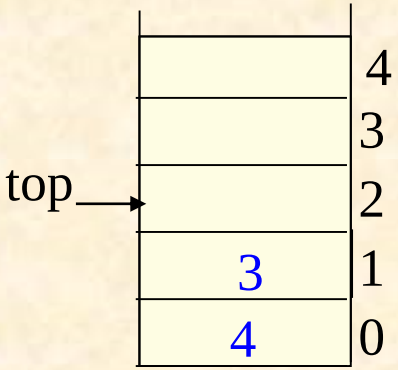
输出序列：6　1



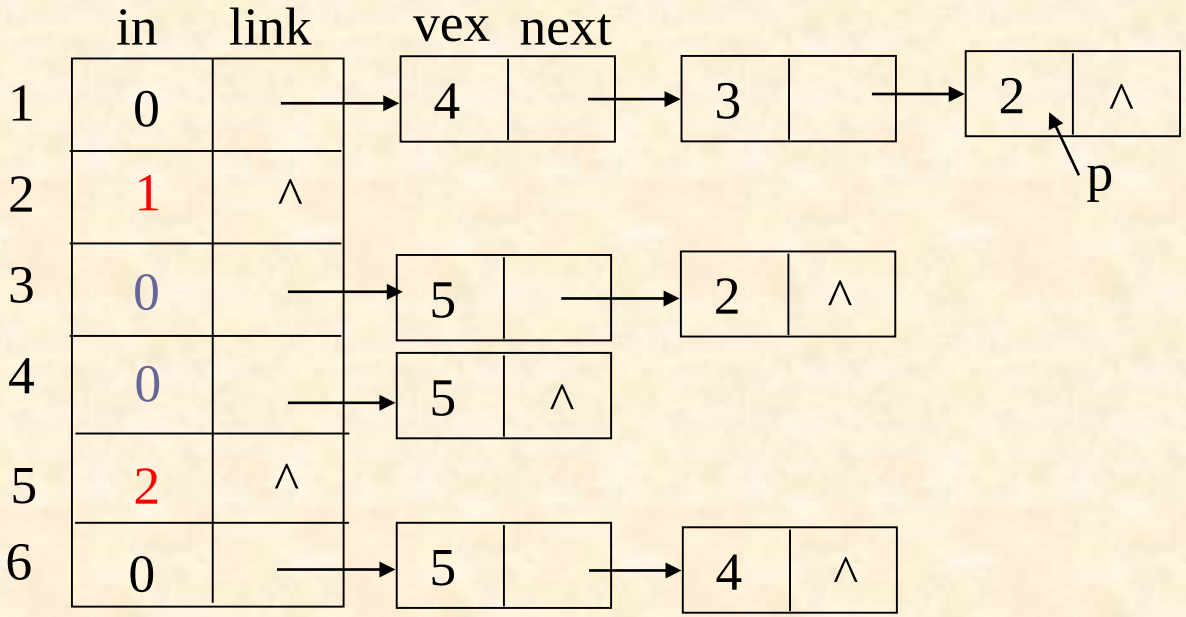


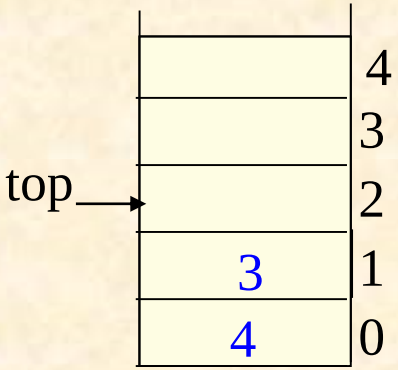
输出序列：6　1



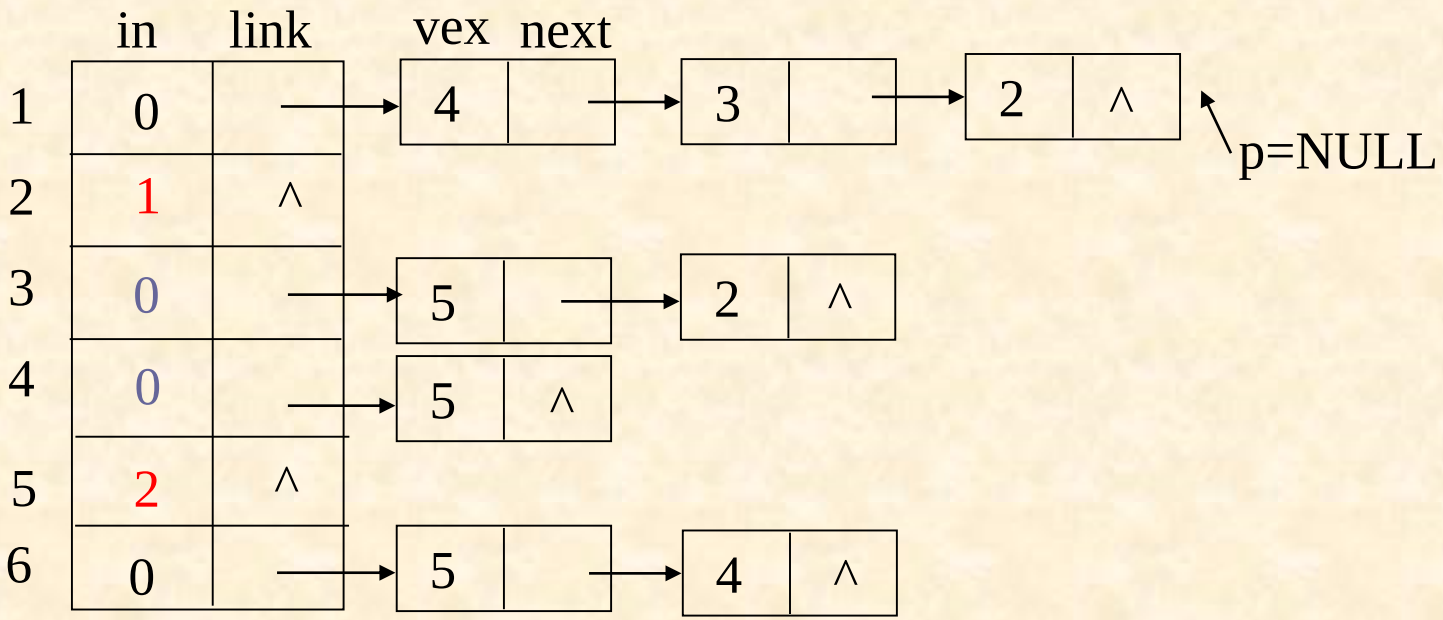


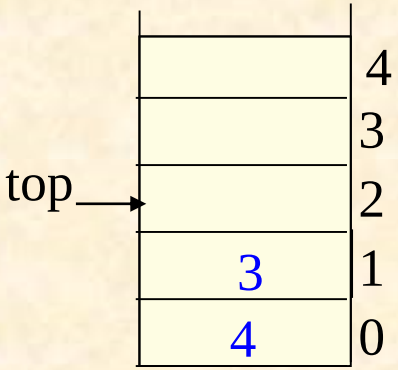
输出序列：6 1



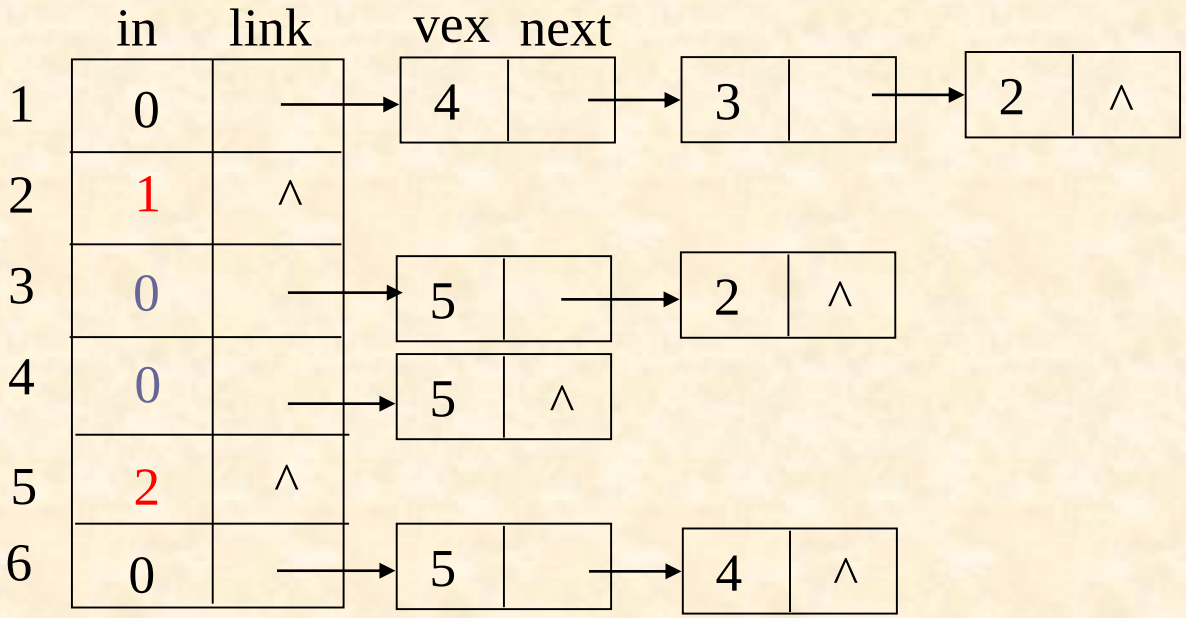


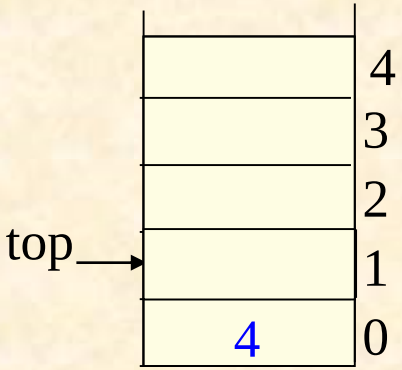
输出序列：6　1



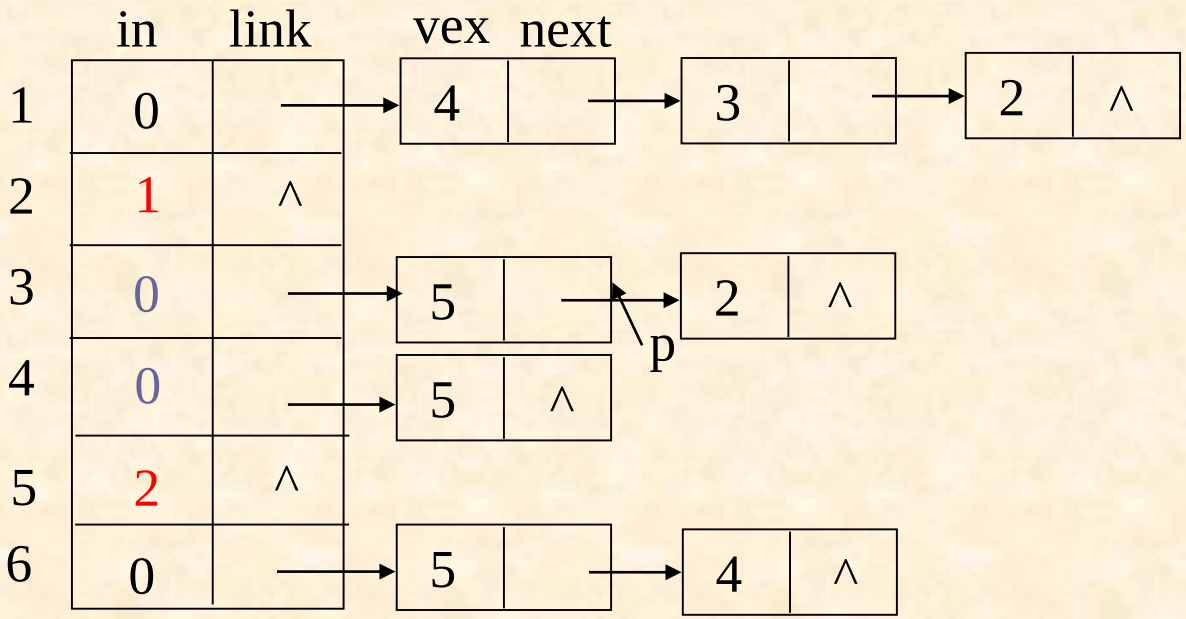


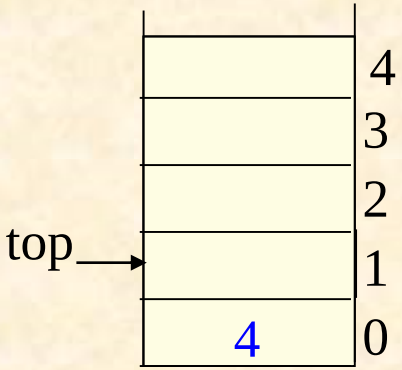
输出序列：6　1



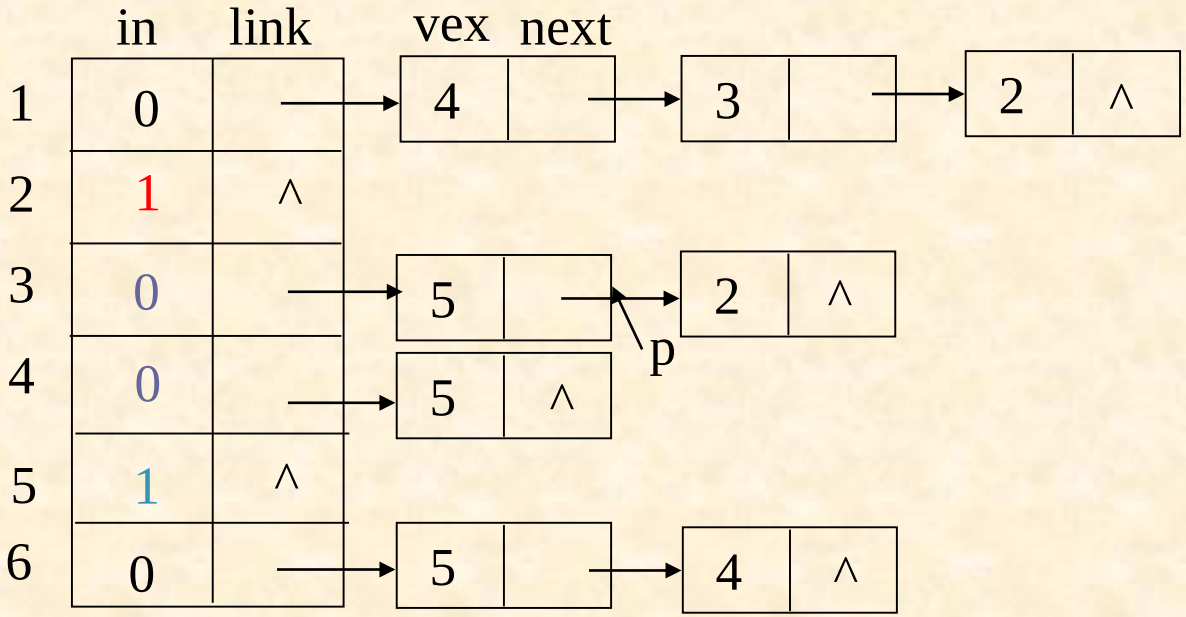


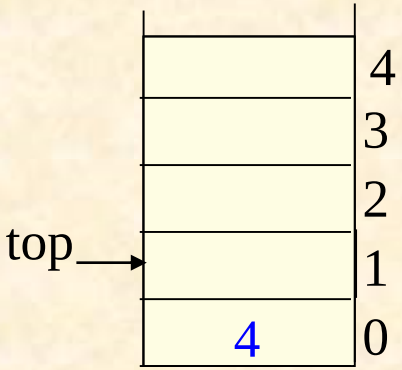
输出序列：6 1 3



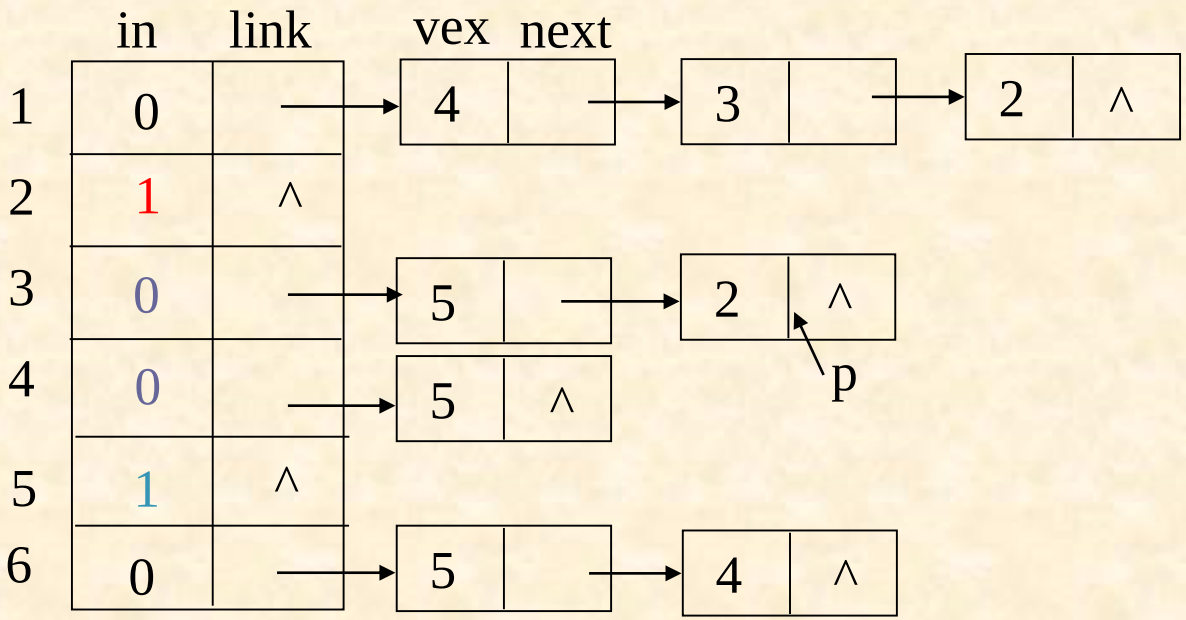


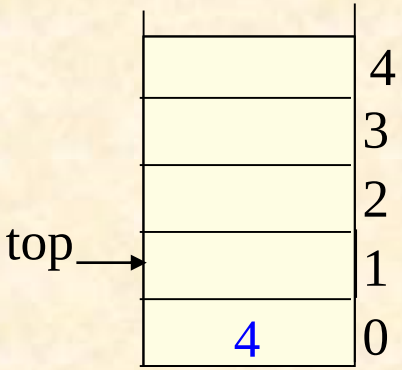
输出序列：6 1 3



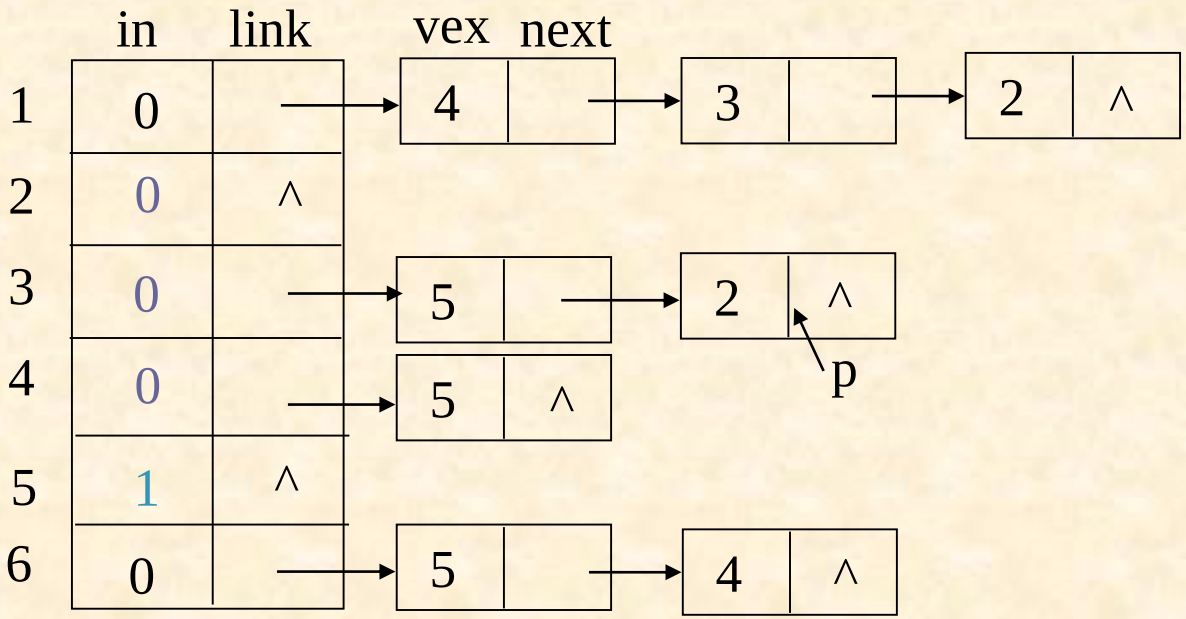


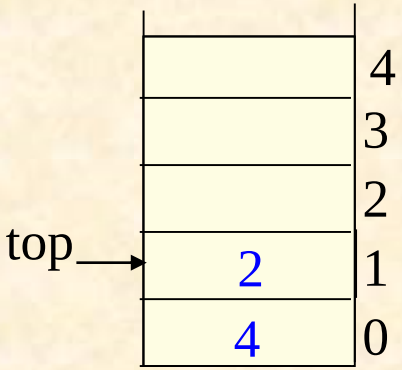
输出序列：6 1 3



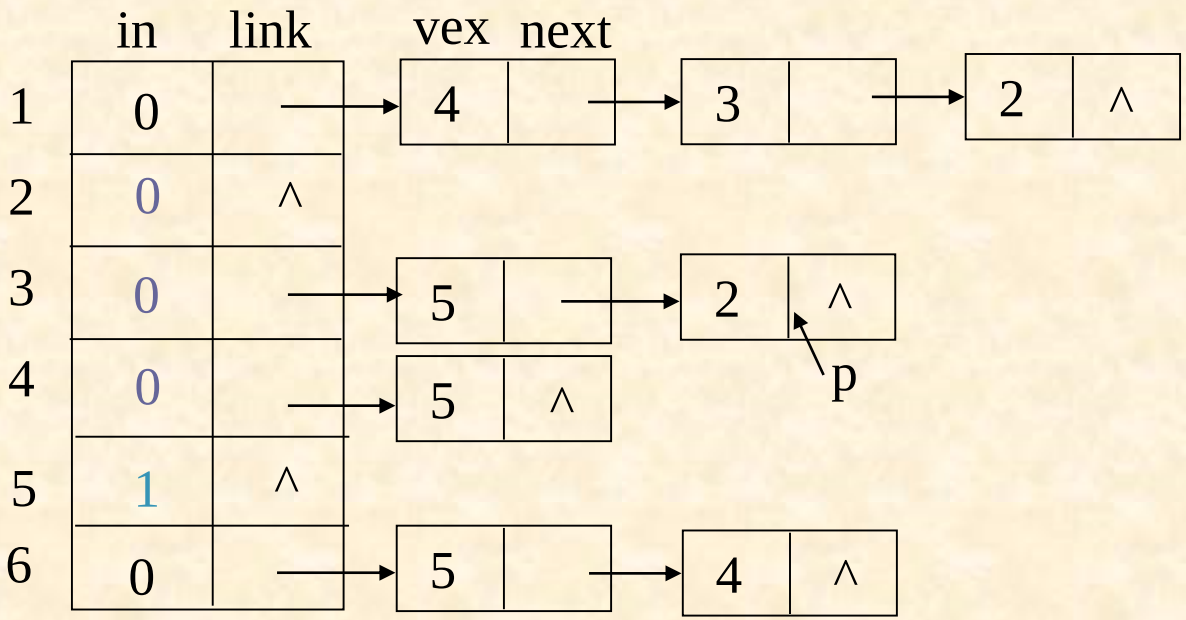


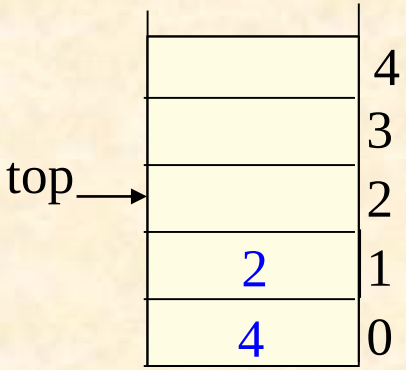
输出序列：6 1 3



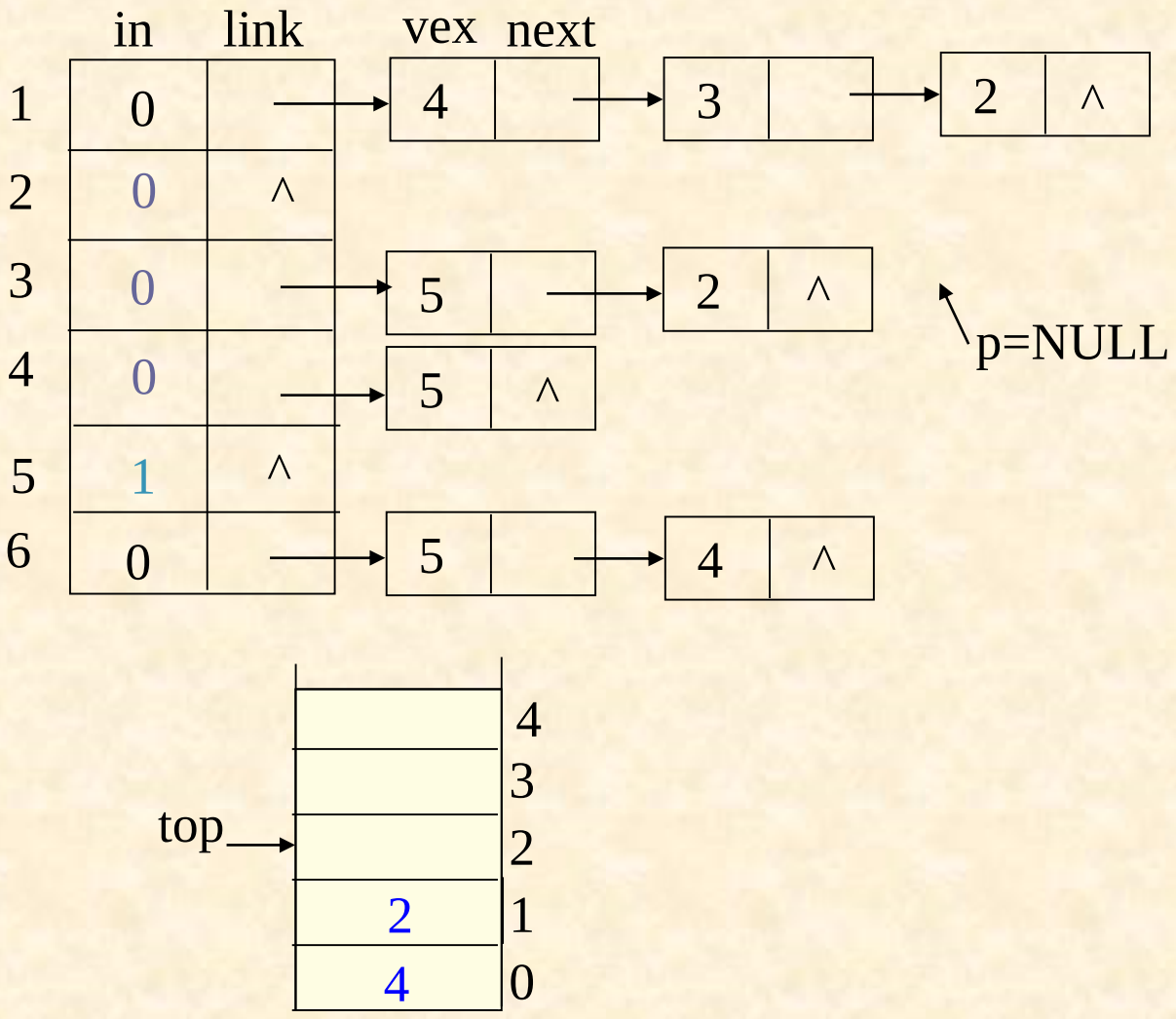


输出序列：6 1 3

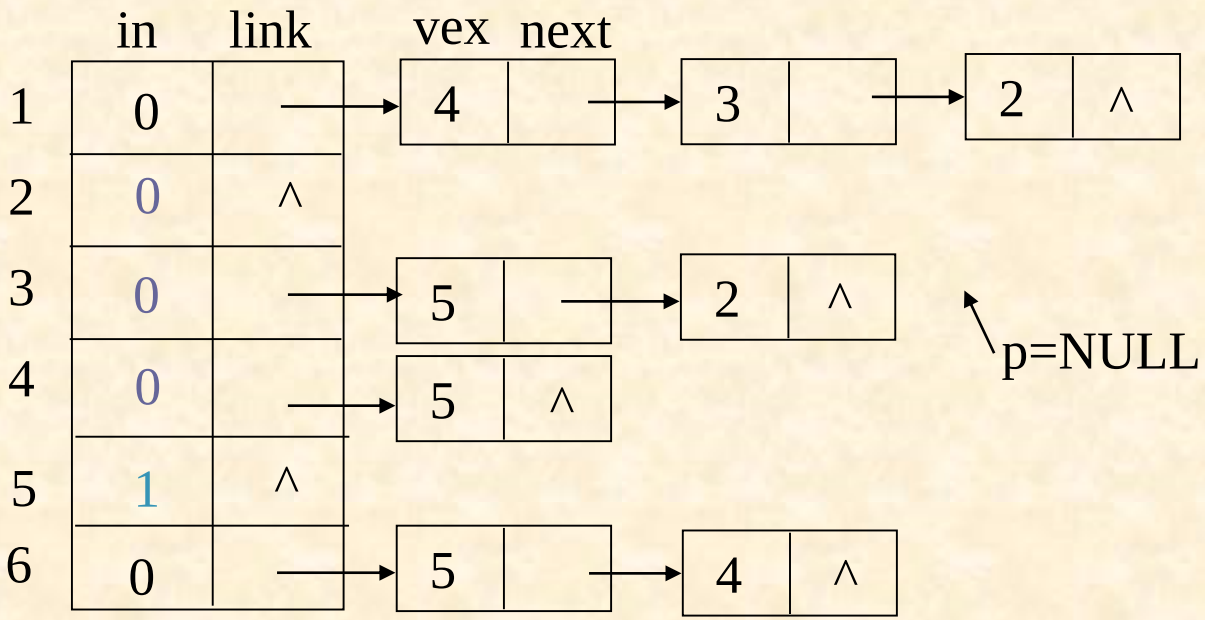


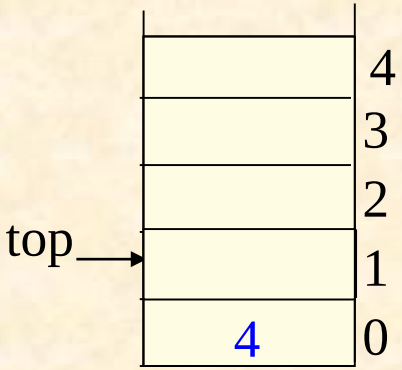


输出序列：6 1 3

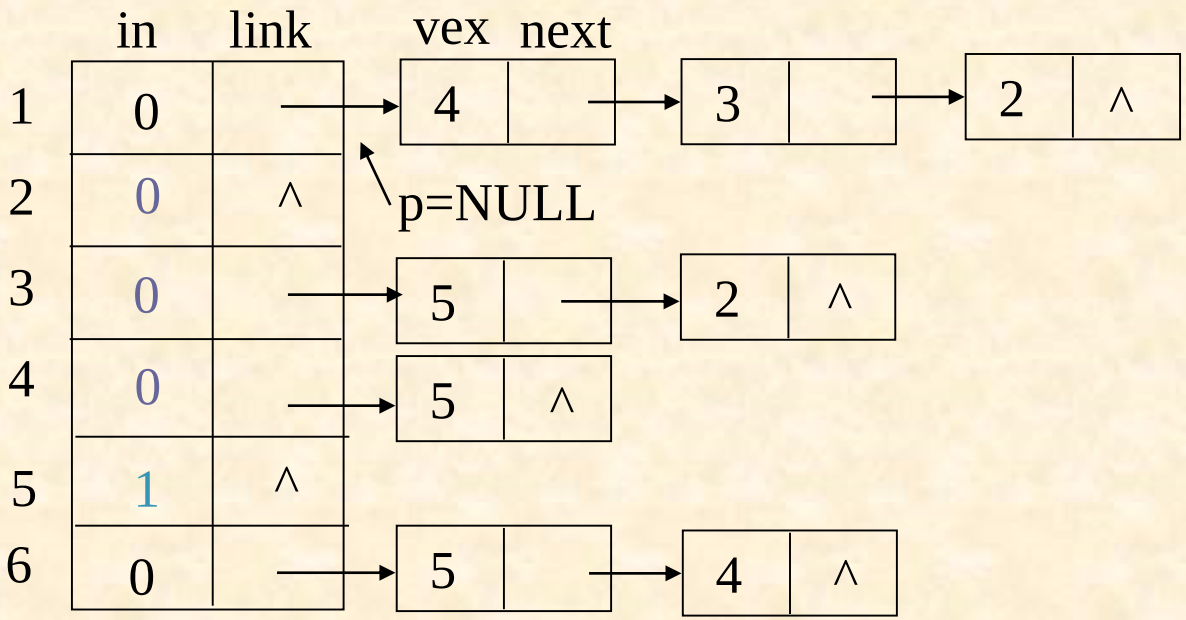


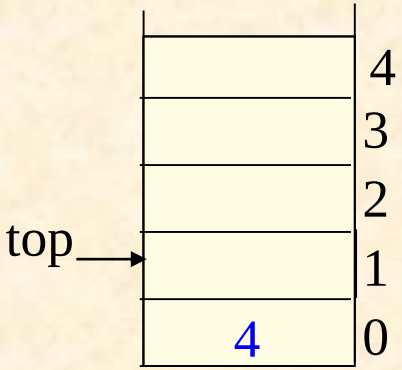
输出序列：6 1 3



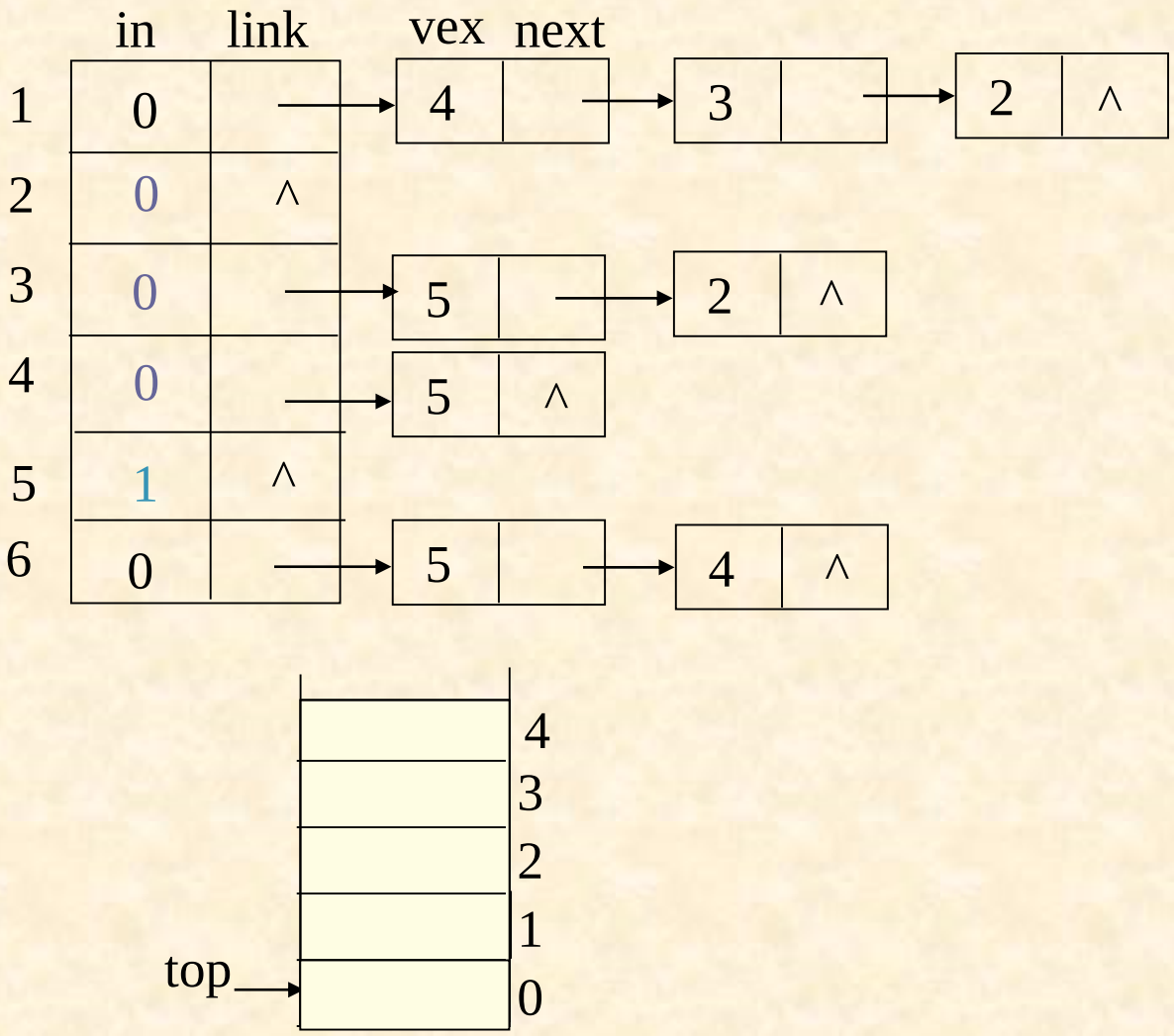


输出序列：6 1 3 2

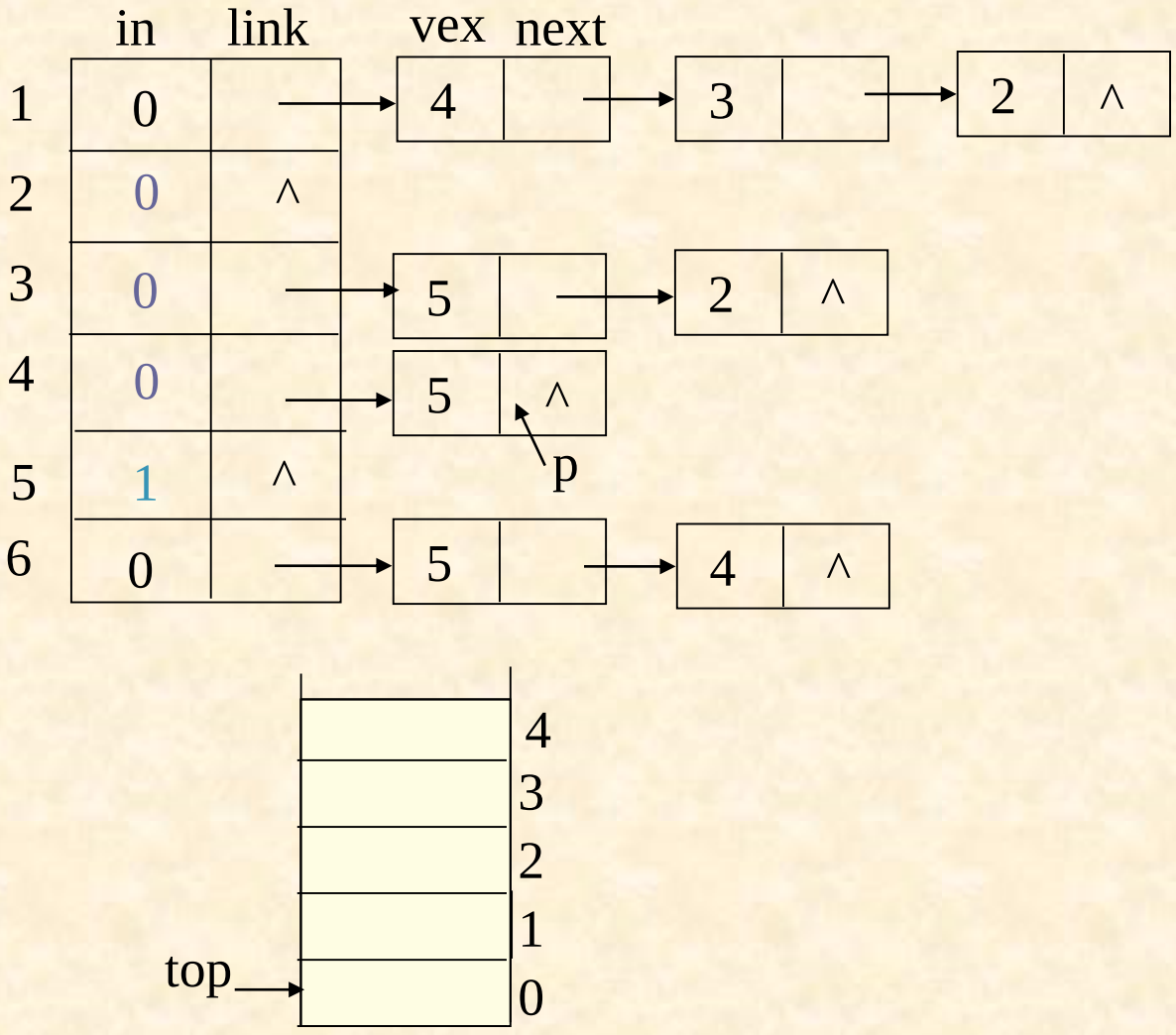




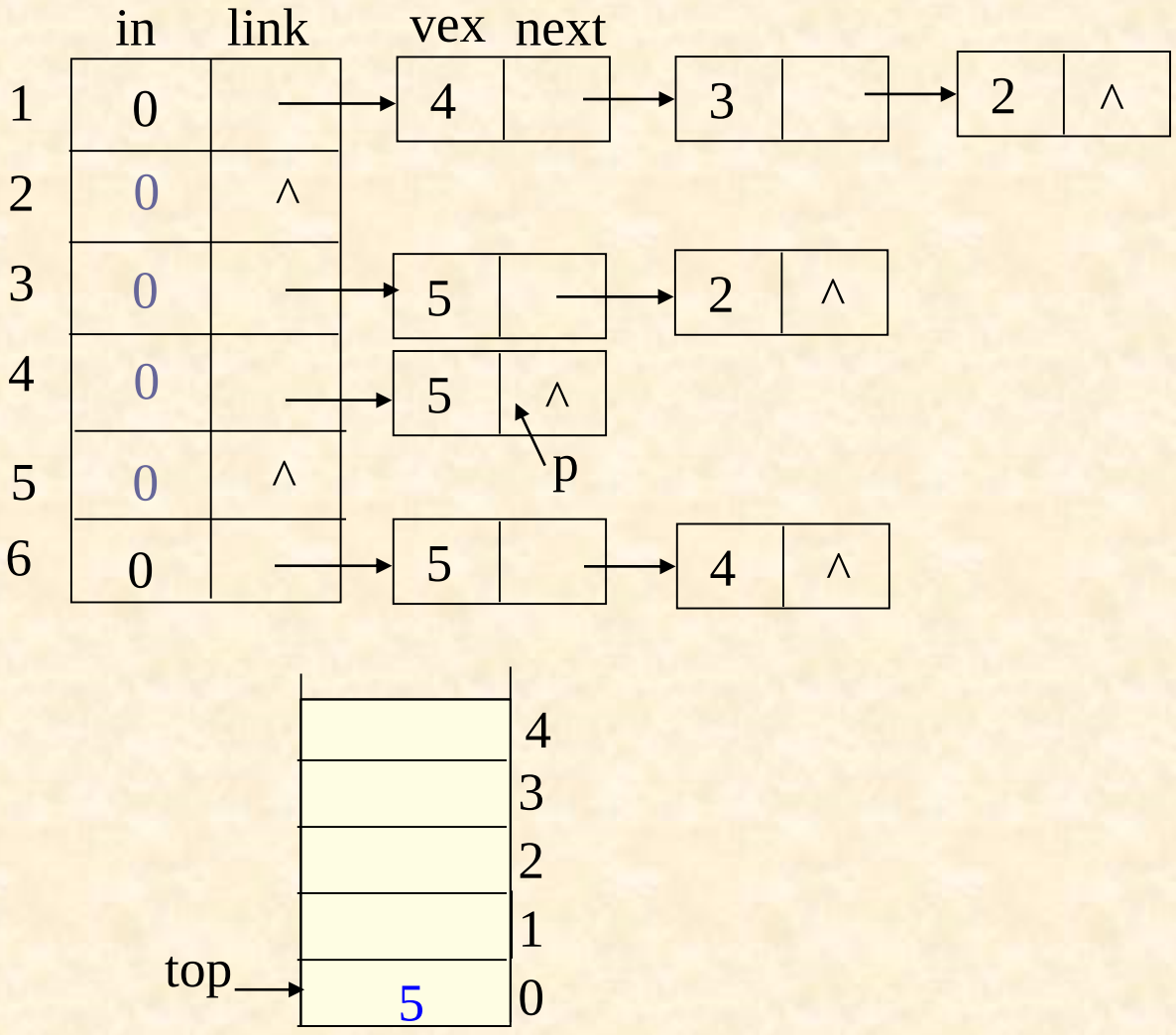
输出序列：6 1 3 2



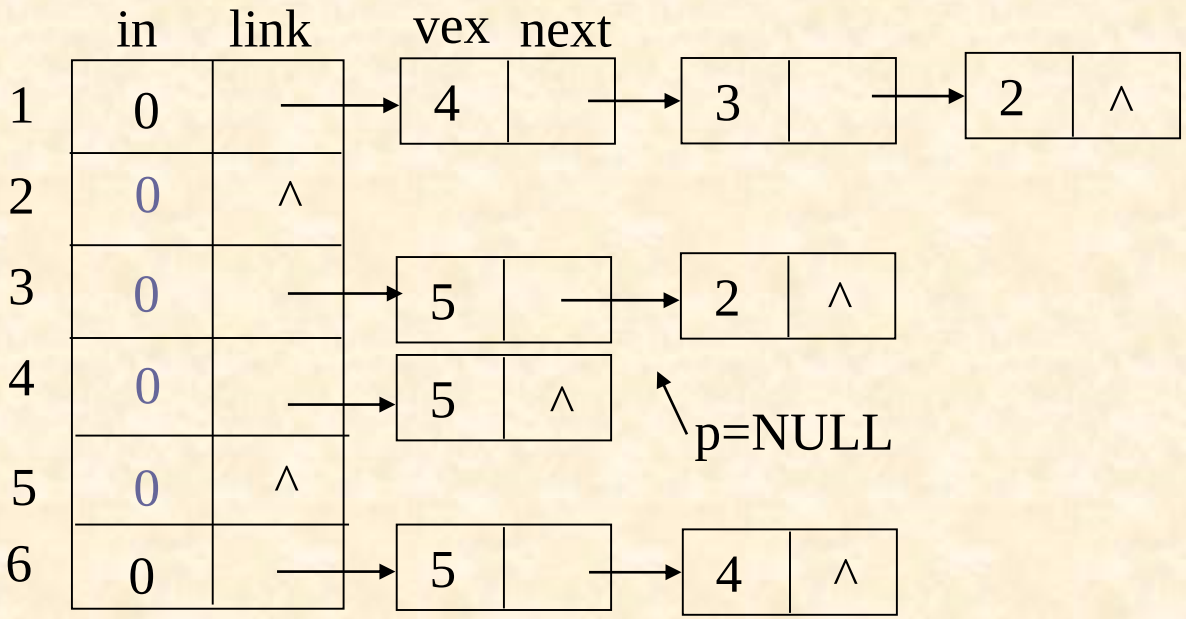
输出序列：6 1 3 2 4

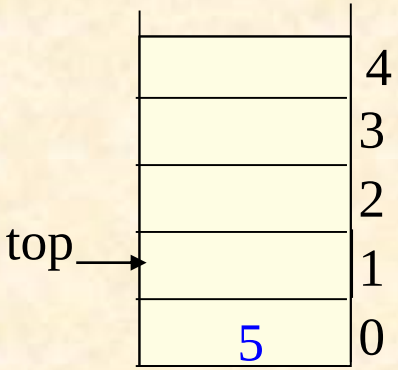


输出序列：6 1 3 2 4

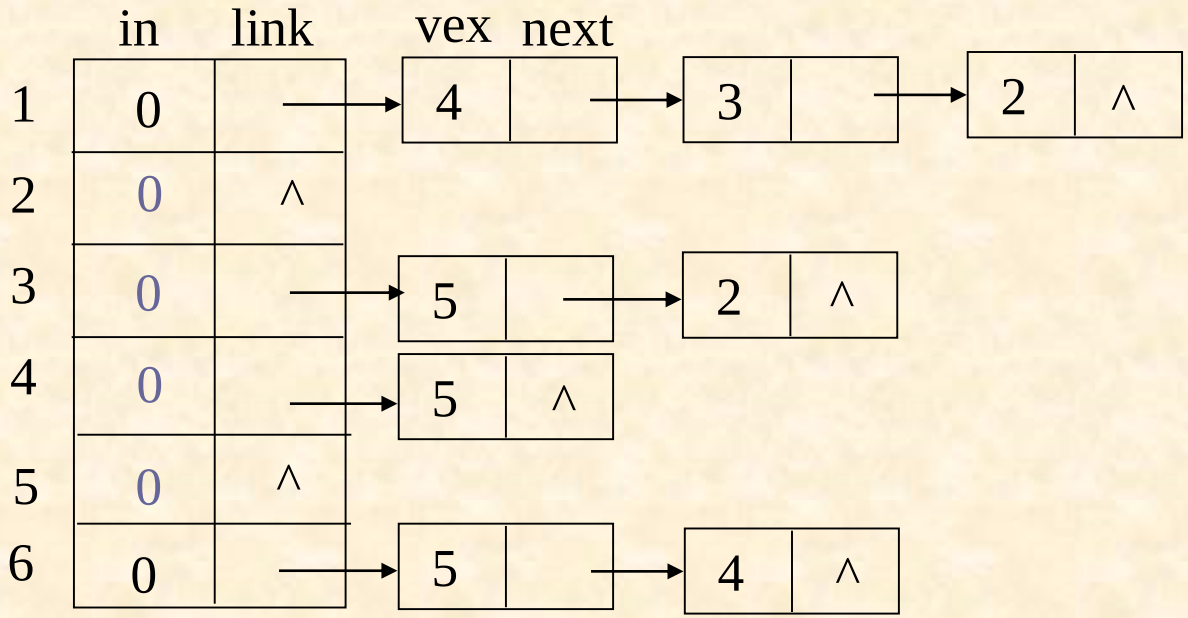


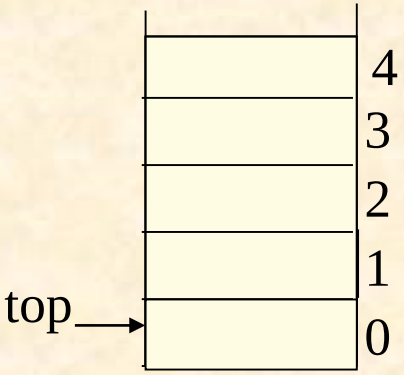
输出序列：6 1 3 2 4



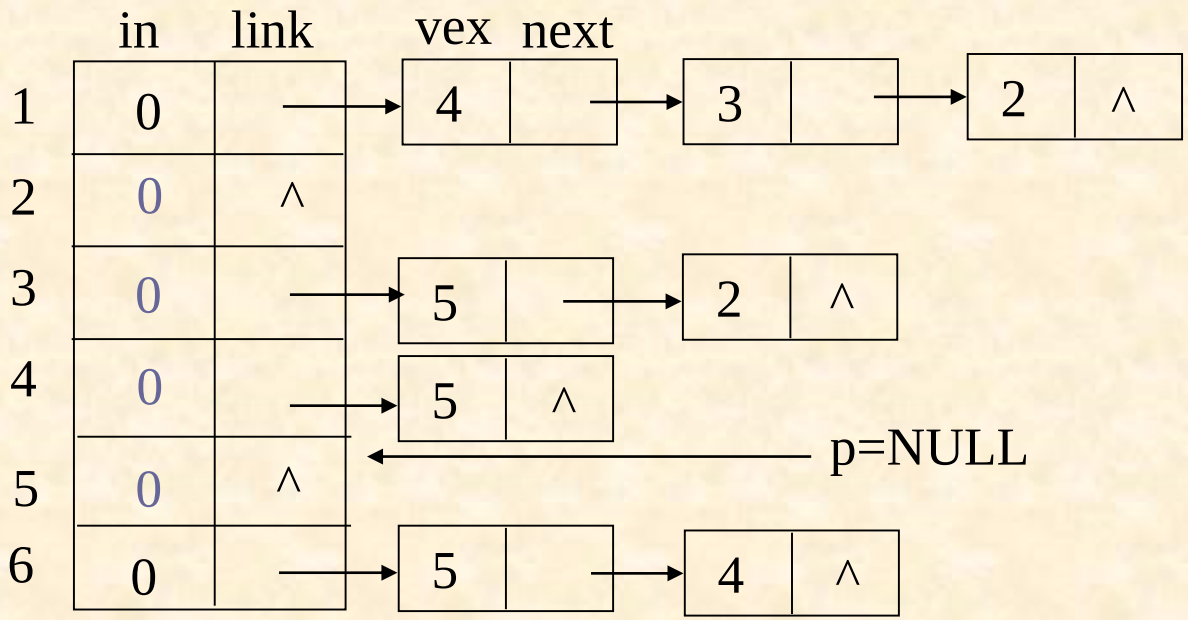


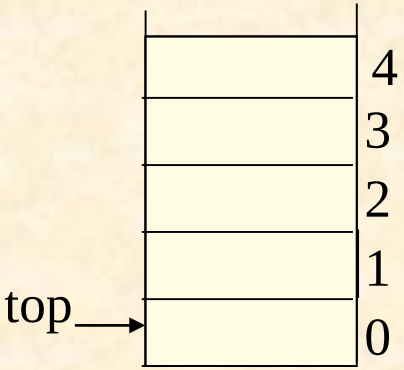
输出序列：6 1 3 2 4





输出序列：6 1 3 2 4 5





输出序列：6 1 3 2 4 5

■ 算法分析

建邻接表：$T(n)=O(e)$

搜索入度为0的顶点的时间：$T(n)=O(n)$

拓扑排序：$T(n)=O(n+e)$

拓扑排序算法

- 假定AOV网用邻接表的形式存储。为实现拓扑排序算法，事先需做好两项准备工作：
- 建立一个数组count[ ]，count[i]的元素值取对应顶点i的入度；
- 建立一个堆栈，栈中存放入度为0的顶点，每当一个顶点的入度为0，就将其压入栈。

```
Status TopologicalSort(ALGraph G) {  // 算法7.12
  // 有向图G采用邻接表存储结构。
  // 若G无回路，则输出G的顶点的一个拓扑序列并返回OK，  否则ERROR
  SqStack S;
  int count,k,i;
  ArcNode *p;
  char indegree[MAX_VERTEX_NUM];
  FindInDegree(G, indegree);  // 对各顶点求入度indegree[0..vernum-1]
  InitStack(S);
  for (i=0; i<G.vexnum; ++i)       // 建零入度顶点栈S
    if (!indegree[i]) Push(S, i);  // 入度为0者进栈
  count = 0;                       // 对输出顶点计数
```

```
while (!StackEmpty(S)) {
    Pop(S, i);
    printf(i, G.vertices[i].data);  ++count;  // 输出i号顶点并计数
    for (p=G.vertices[i].firstarc;  p;  p=p->nextarc) {
        k = p->adjvex;              // 对i号顶点的每个邻接点的入度减1
        if (!(--indegree[k])) Push(S, k);  // 若入度减为0，则入栈
    }
}
if (count<G.vexnum) return ERROR;      // 该有向图有回路
else return OK;
} // TopologicalSort
```

引理5.1 设图G = (V, E)是非循环图, V(G)≠Φ, 则G中一定存在入度为零的顶点

定理5.2 设G=(V, E)是非循环图，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).

第七章 图

7.6 关键路径

基本概念

◆如果在有向无环的带权图中

- 用有向边表示一个工程中的各项活动(Activity)
- 用边上的权值表示活动的持续时间(Duration)
- 用顶点表示事件(Event)

◆则这样的有向图叫做用边表示活动的网络，简称AOE (Activity On Edges)网络。

- 源点：表示整个工程的开始（入度为零）。
- 汇点：表示整个工程的结束（出度为零）。

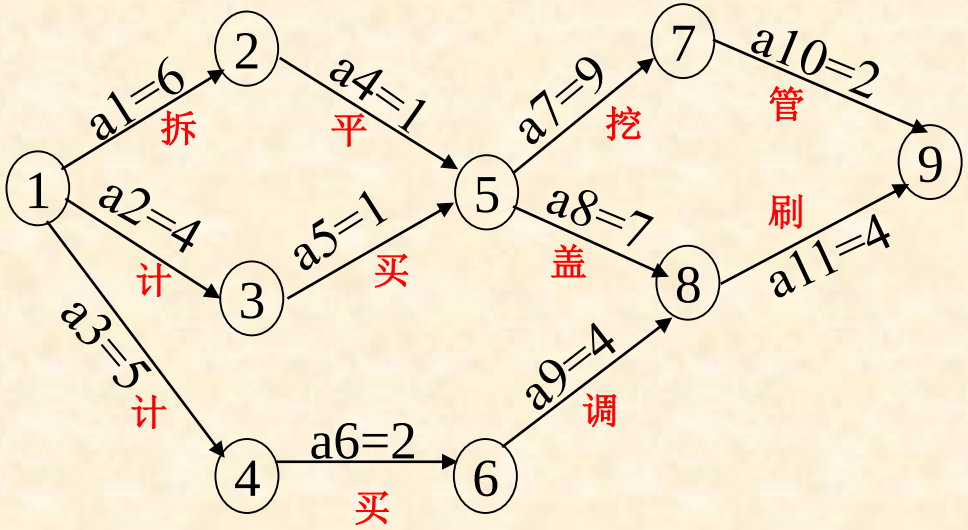
❑问题提出

把工程计划表示为有向图，用**顶点**表示**事件**，**弧**表示**活动**；
每个事件表示在它之前的活动已完成，在它之后的活动可以开始。

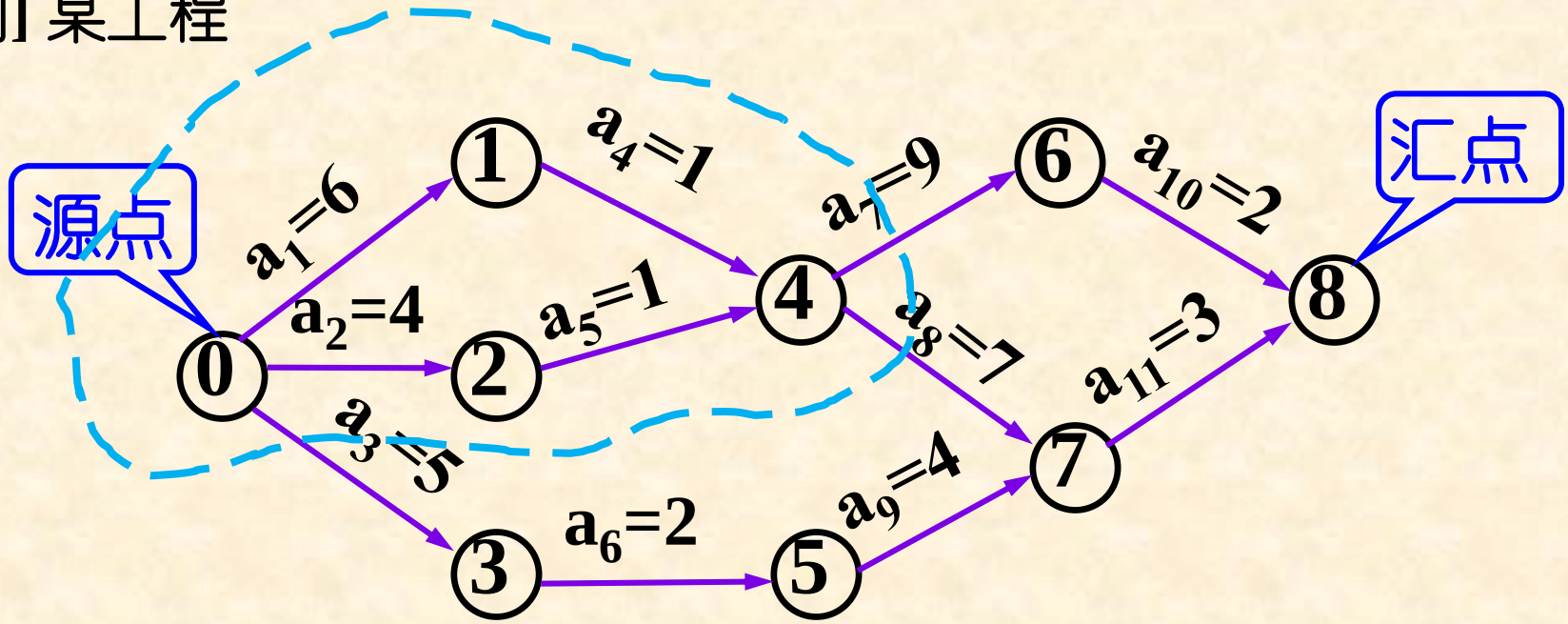
例　设一个工程有11项活动，9个事件

事件 V1——表示整个工程开始

事件 V9——表示整个工程结束



[例] 某工程



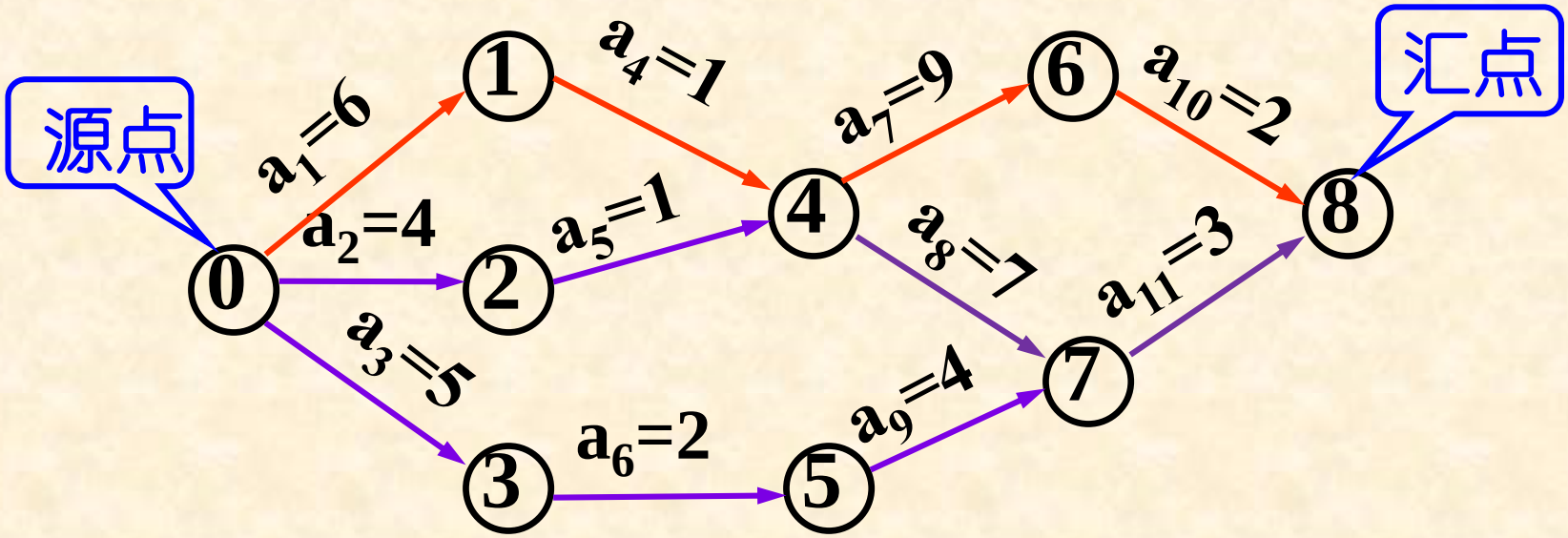
- 完成整个工程至少需要多少时间？
- 哪些活动不能延期，否则将会影响整个工程进度？
- 在不推迟整个工程进度的情况下，哪些活动可以适当延期？

- 在AOE网络中，有些活动顺序进行，有些活动并行进行。
- 从源点到各个顶点，以至从源点到汇点的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路径的活动所需的时间虽然不同，但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)。

相关定义与术语

- **AOE网(Activity On Edge)**——也叫边表示活动的网。AOE网是一个带权的**有向无环图**，其中**顶点**表示**事件**，**弧**表示**活动**，**权**表示活动**持续时间**
- **路径长度**——路径上各活动持续时间之和
- **关键路径**——路径长度**最长**的路径叫~
- **Ve(j)**——表示**事件Vj**的**最早**发生时间
- **Vl(j)**——表示**事件Vj**的**最迟**发生时间
- **e(i)**——表示**活动a_i**的**最早**开始时间
- **l(i)**——表示**活动ai**的**最迟**开始时间
- **l(i)-e(i)**——表示完成**活动ai**的时间**余量**
- **关键活动**——关键路径上的活动叫~，即**l(i)=e(i)**的活动

[例] 某工程



- 关键路径：从源点到汇点具有最大长度的路径称为关键路径。
- 路径长度：指路径上的各边权值之和。
- 关键活动：关键路径上的活动。

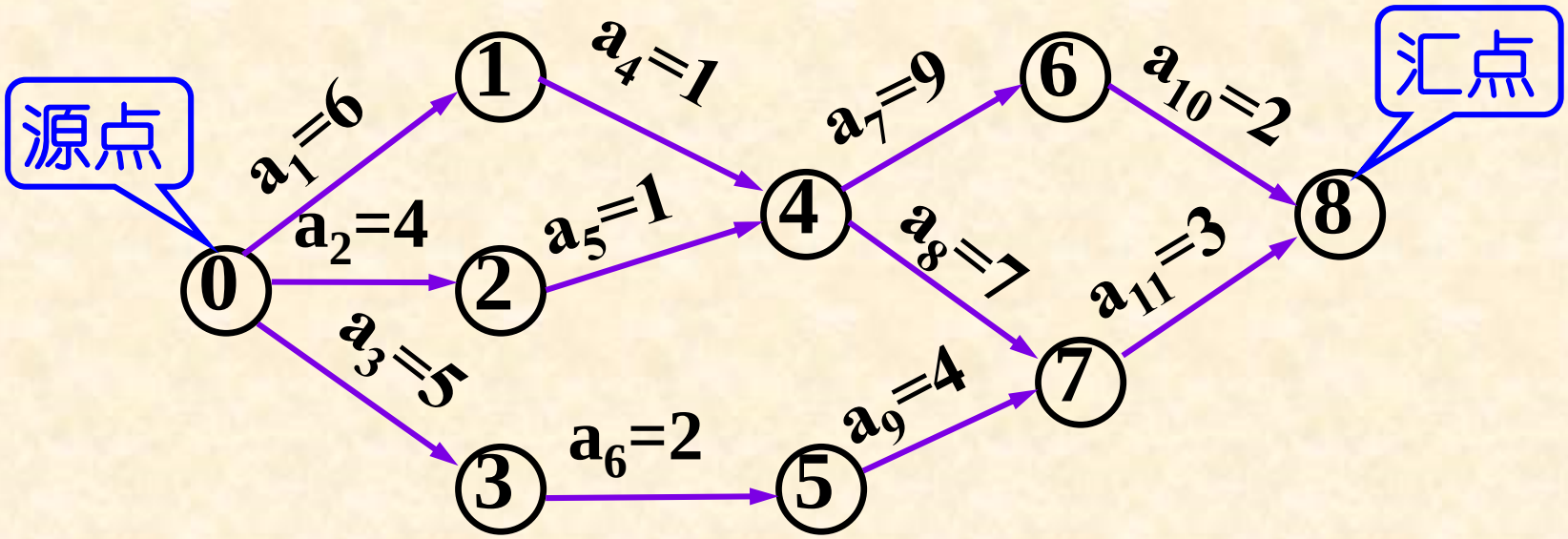
- 关键活动有关的量：

① 事件v_j的最早发生时间ve(j):

从源点v_0到v_j的最长路径长度。

② 事件v_j的最迟发生时间vl(j):

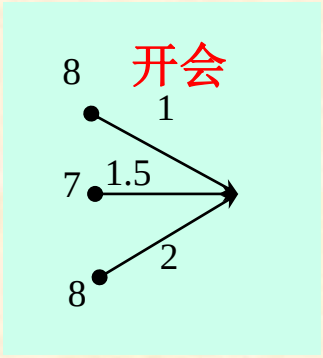
保证汇点的最早发生时间不推迟的前提下，事件v_j允许的最迟开始时间，等于ve(n-1)减去从v_j到v_{n-1}最长路径长度。



- 求所有事件的最早发生时间：

递推公式：// 拓扑排序正序

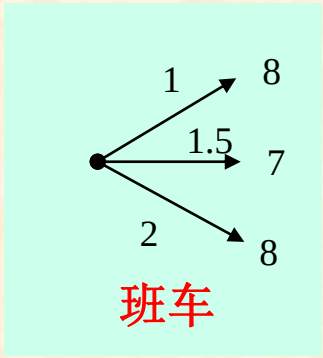
$$ve(k)=\begin{cases}0 & k=1\\ \max\{ve(j)+weight(<j,k>)\} & <v_j,v_k>\in E(G),\ k=2,3,\ldots,n\end{cases}$$



- 求所有事件的最迟发生时间：

递推公式：// 拓扑排序逆序

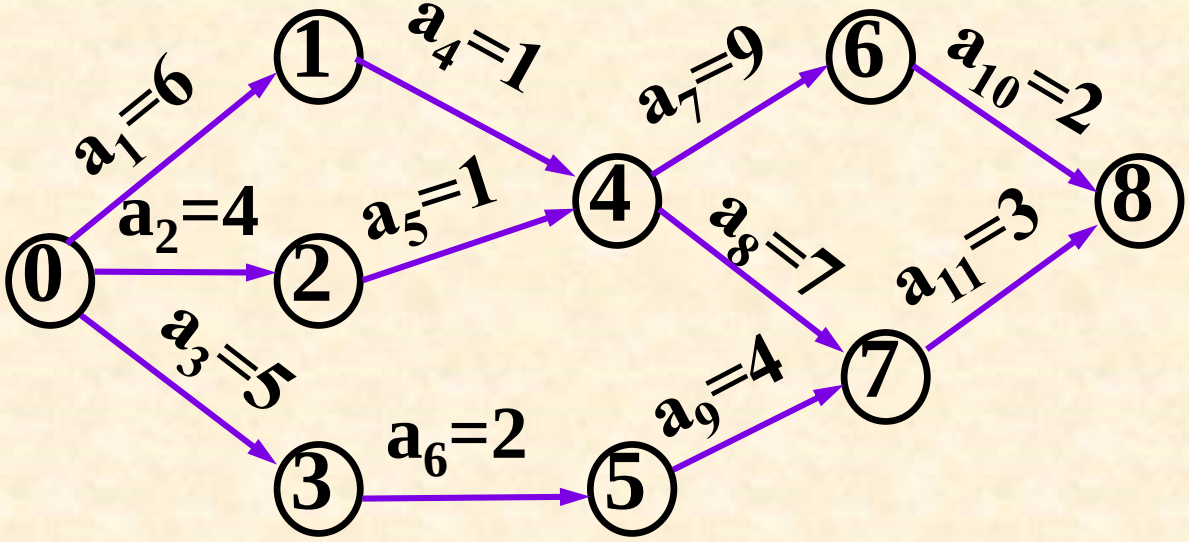
$$vl(j)=\begin{cases}ve(n) & j=n\\ \min\{vl(k)-weight(<j,k>)\} & <v_j,v_k>\in E(G),\ j=n-1,n-2,\ldots,1\end{cases}$$



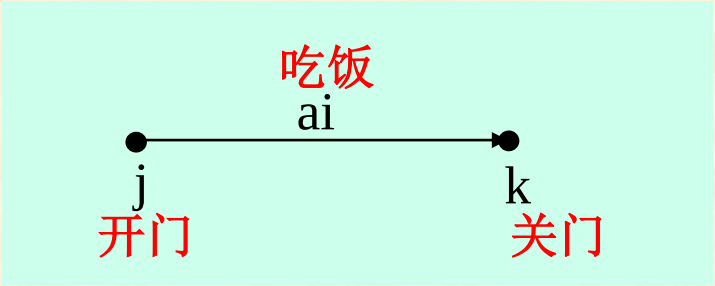
ve(0)=0	vl(8)= 18
ve(1)= 6	vl(7)= 15
ve(2)= 4	vl(6)= 16
ve(3)= 5	vl(5)= 11
ve(4)= 7	vl(4)= 7
ve(5)= 7	vl(3)= 9
ve(6)= 16	vl(2)= 6
ve(7)= 14	vl(1)= 6
ve(8)= 18	vl(0)= 0

$ve(k)= \max\{ve(j)+ weight(<j , k>)\}$

$vl(j)= \min\{vl(k)- weight(<j , k>)\}$

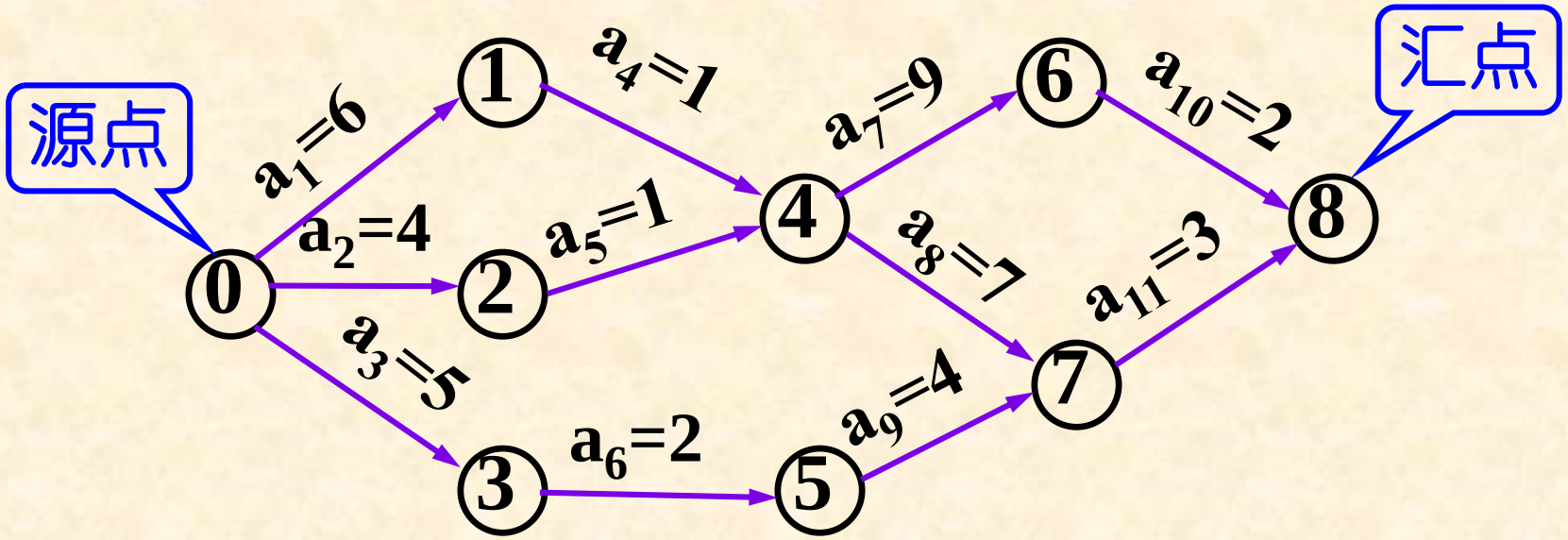


- 关键活动有关的量：



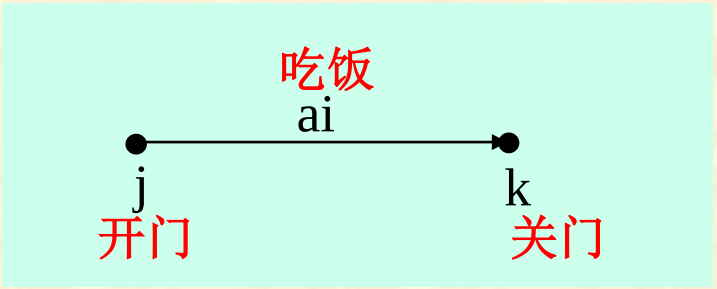
③ 活动a_i的最早开始时间e(i)：

设活动a_i在有向边$< v_j, v_k >$上，e(i)是从源点v_0到v_j的最长路径长度。因此e(i)=ve(j)。



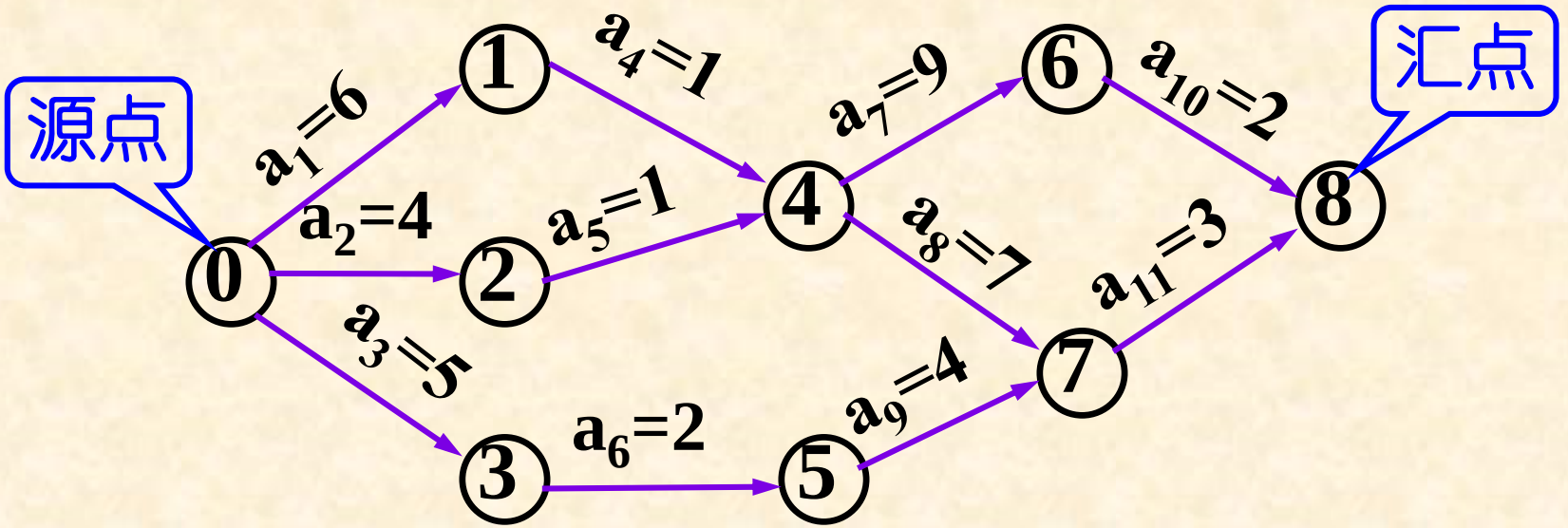
- 关键活动有关的量：

④ 活动a_i的最迟开始时间**l(i)**：



l(i) 是在不会引起时间延误的前提下，该活动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>)

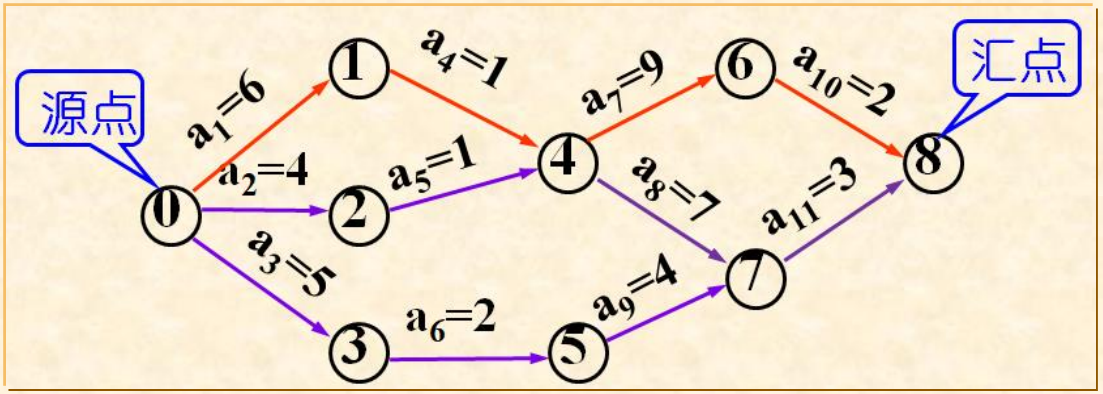


ve(0)=0	vl(8)= 18
ve(1)= 6	vl(7)= 15
ve(2)= 4	vl(6)= 16
ve(3)= 5	vl(5)= 11
ve(4)= 7	vl(4)= 7
ve(5)= 7	vl(3)= 9
ve(6)= 16	vl(2)= 6
ve(7)= 14	vl(1)= 6
ve(8)= 18	vl(0)= 0

设活动a_i在有向边$< v_j, v_k >$上

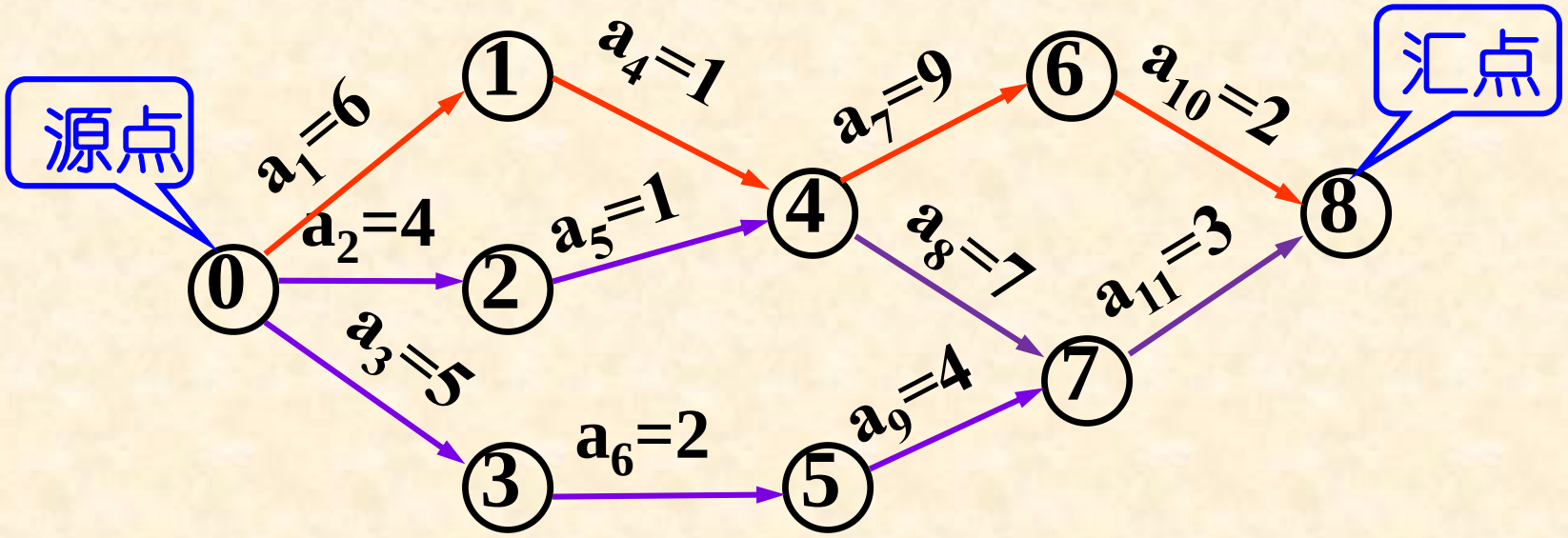
e(i)=ve(j)

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



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	4	6	6	9	7	8	11	16	15

- 关键活动： l(i)＝ e(i) 表示活动a_k是没有时间余量的关键活动



为找出关键活动，需要求各个活动的 e(i) 与 l(i)，以判别是否 l(i)＝ e(i)

为求得e(i) 与 l(i)，需要先求得从源点V_0到各个顶点V_j的 ve(j) 和 vl(j)。

所有的关键活动组成的路径就是关键路径

求关键活动算法

求关键活动的基本步骤：

①对AOE网进行**拓扑排序**，按**拓扑次序**求出各顶点**事件**的**最早发生时间ve**，若网中有回路，则终止算法；

② 按**拓扑序列的逆序**求各顶点事件的**最迟发生时间vl**；

③根据ve和vl的值，求各**活动**的**最早开始时间e(*i*)**与**最迟开始时间l(*i*)**，若e(*i*)=l(*i*)，则*i*是**关键活动**。

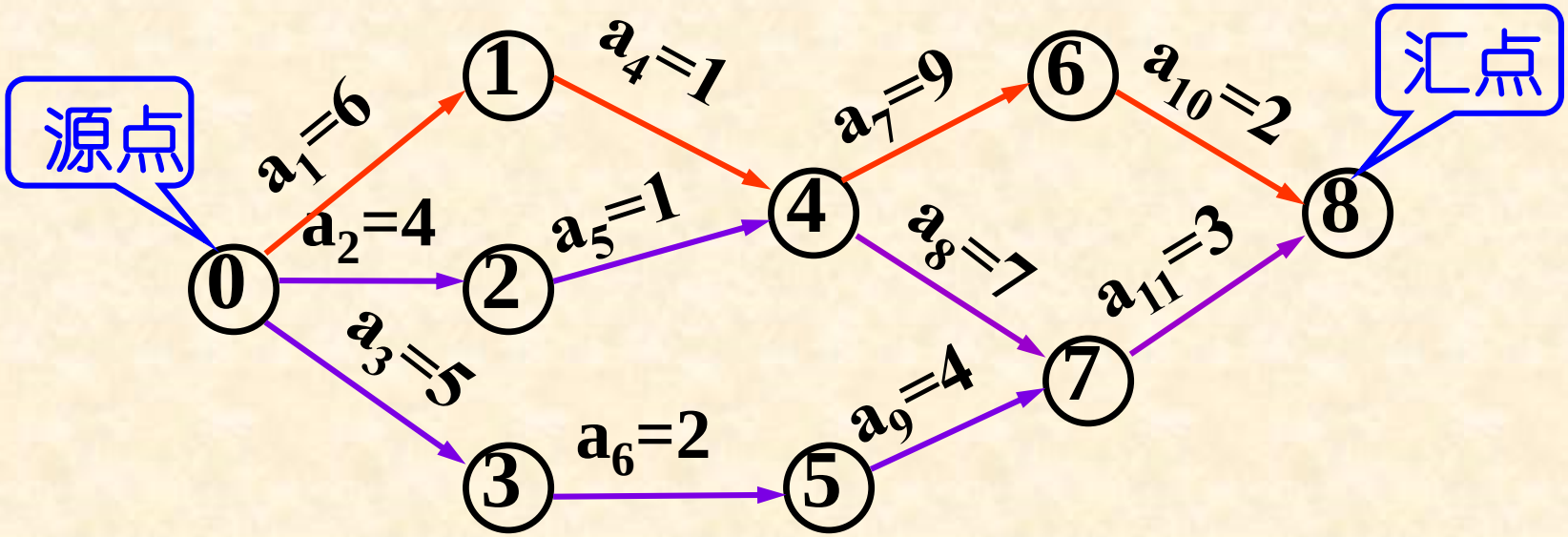
■ 算法实现

- ✧以邻接表作存储结构
- ✧从源点V1出发，令Ve[1]=0, 按**拓扑序列**求各顶点的Ve[i]
- ✧从汇点Vn出发，令Vl[n]=Ve[n]，按**逆拓扑序列**求其余各顶点的Vl[i]
- ✧根据各顶点的Ve和Vl值，计算每条弧的e[i]和l[i]，找出e[i]=l[i]的**关键活动**

■ 算法描述

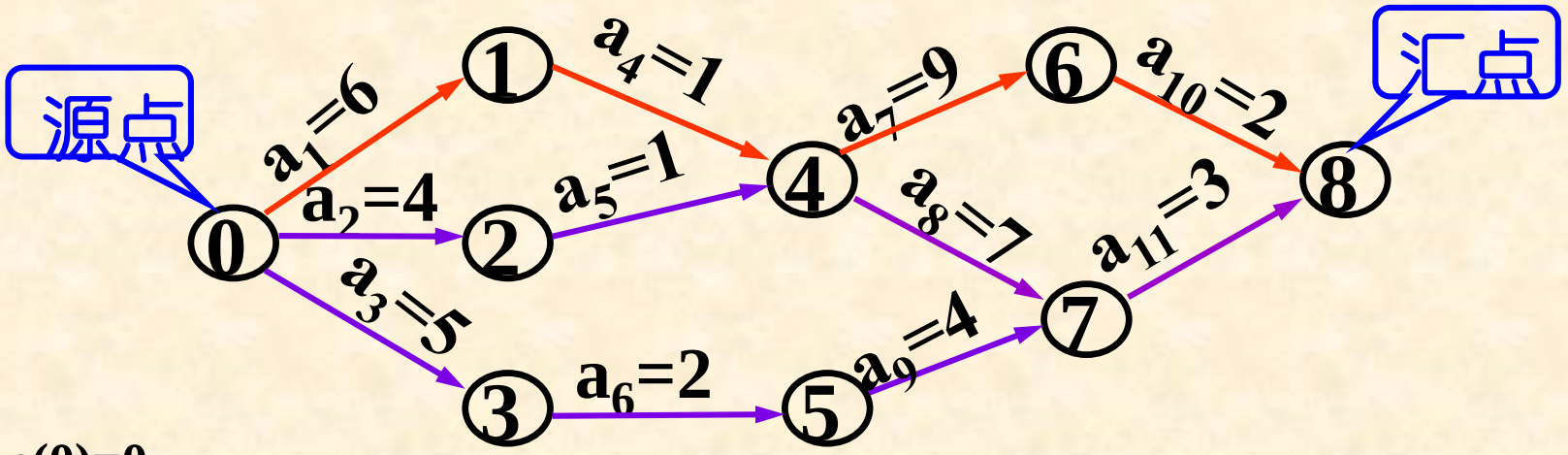
- ✧输入顶点和弧信息，建立其邻接表
- ✧计算每个顶点的入度
- ✧对其进行**拓扑排序**
  - ⌑排序过程中求顶点的Ve[i]
  - ⌑将得到的拓扑序列**进栈**
  - ⌑按**逆拓扑序列**求顶点的Vl[i]
- ✧计算每条弧的e[i]和l[i],找出e[i]=l[i]的关键活动

[例] 求关键活动 — 第1步



按拓扑正序递推：

$$ve(k)\begin{cases} ve(0)=0 & k=0 \\ \max\{ve(j)+weight(<j,k>)\} & <v_j,v_k>\in E(G),\ k=1,2,\ldots,n-1 \end{cases}$$



ve(0)=0

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

ve(2)= ve(0)+weight(<0,2>)=0+4=4

ve(3)= ve(0)+weight(<0,3>)=0+5=5

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

ve(5)= ve(3)+weight(<3,5>)=5+2=7

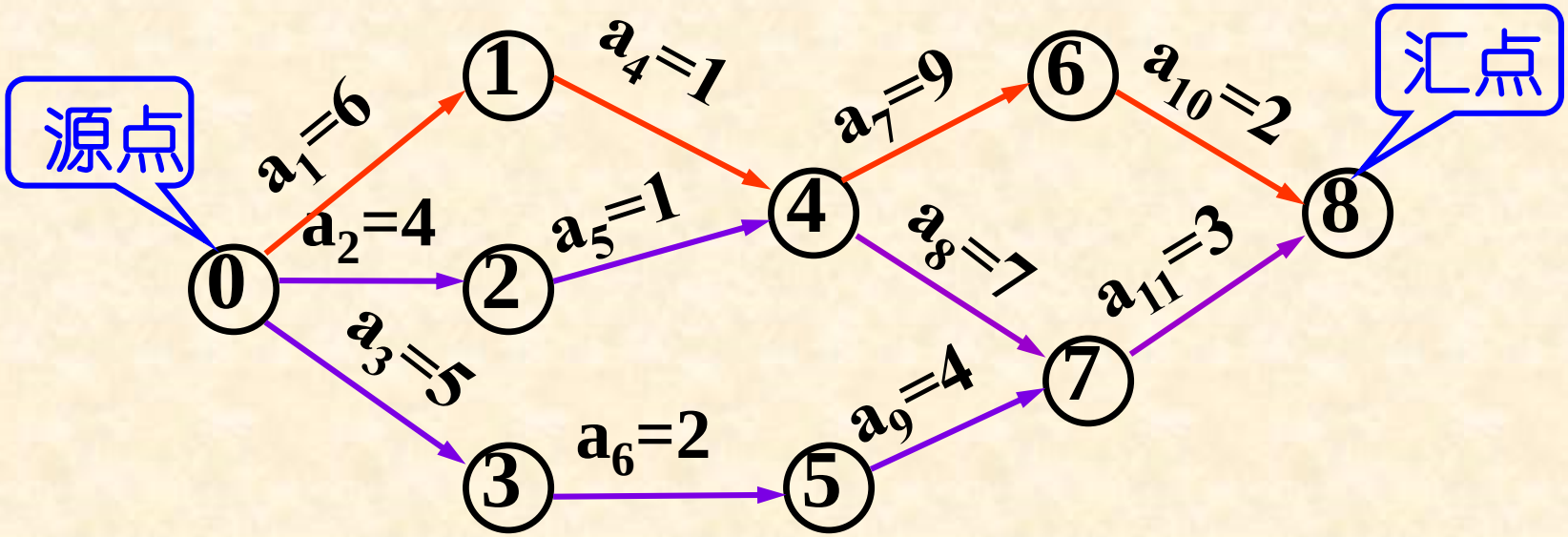
ve(6)= ve(4)+weight(<4,6>)=7+9=16

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

ve(8)= max{ve(6)+ weight(<6,8>),
ve(7)+ weight(<7,8>)}=max{16+2,14+4}=18

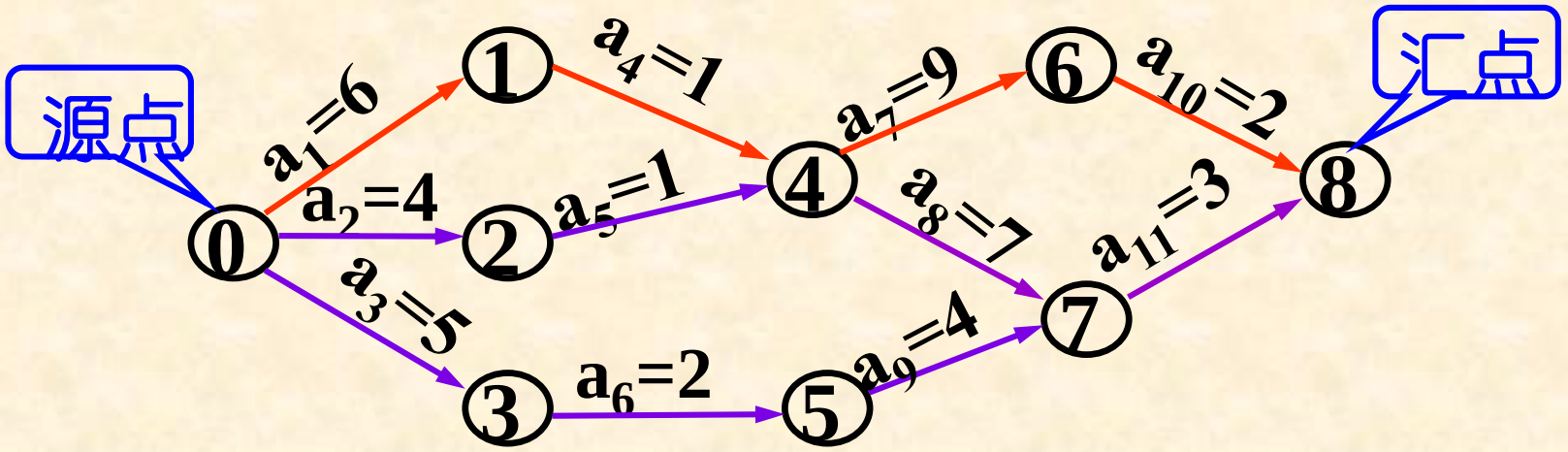
ve(k)=
max{ve(j)+ weight(<j , k>)}

[例] 求关键活动— 第2步



按拓扑逆序递推：

$$vl(j)\begin{cases} ve(n-1) & j=n-1 \\ \min\{vl(k)-weight(<j,k>)\} & <v_j, v_k> \in E(G),\ j=n-2, n-3,\ldots,0 \end{cases}$$



vl(8)= ve(8)=18

vl(7)= vl(8)-weight(<7,8>)=18-3=15

vl(6)= vl(8)-weight(<6,8>)=18-2=16

vl(5)= vl(7)-weight(<5,7>)=15-4=11

vl(4)= min{vl(7)- weight(<4 ,7>),
vl(6)- weight(<4 ,6>)} =min{15-7,16-9}=7

vl(3)= vl(5)-weight(<3,5>)=11-2=9

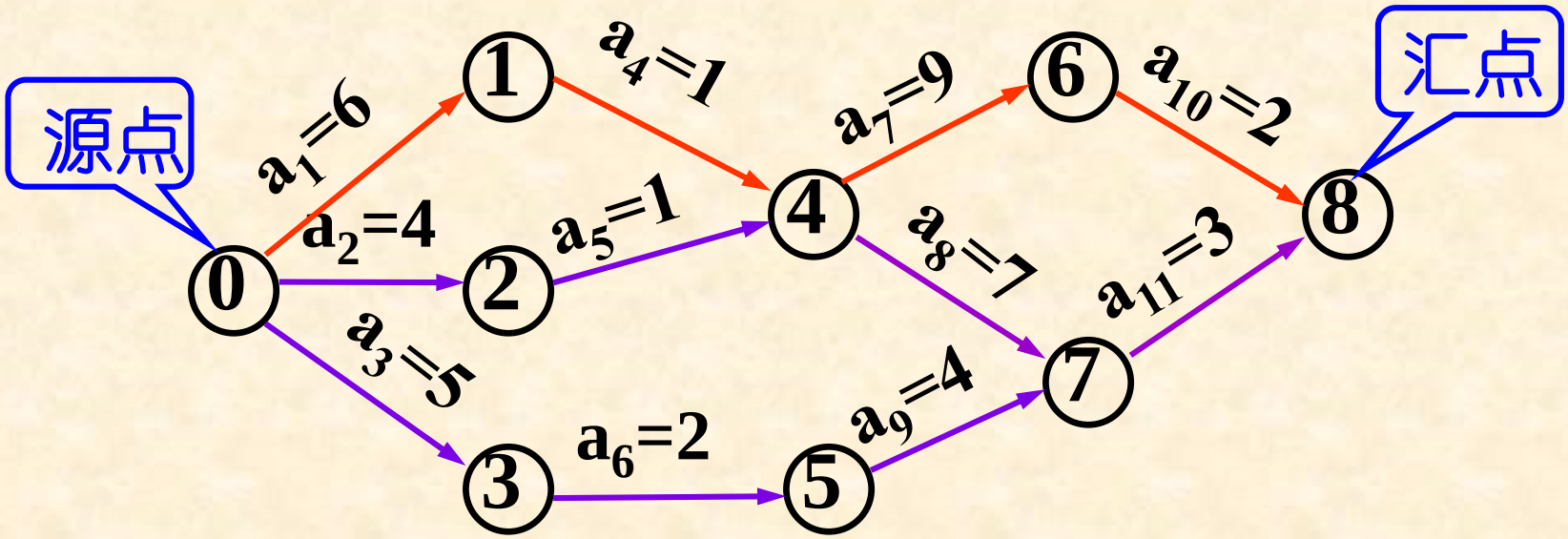
vl(2)= vl(4)-weight(<2,4>)=7-1=6

vl(1)= vl(4)-weight(<1,4>)=7-1=6

vl(0)= min{vl(1)- weight(<0 ,1>),
vl(2)- weight(<0 ,2>),
vl(3)- weight(<0 ,3>)} = min{6-6,6-4,9-5}=0

vl(j)=
min{vl(k)- weight(<j , k>)}

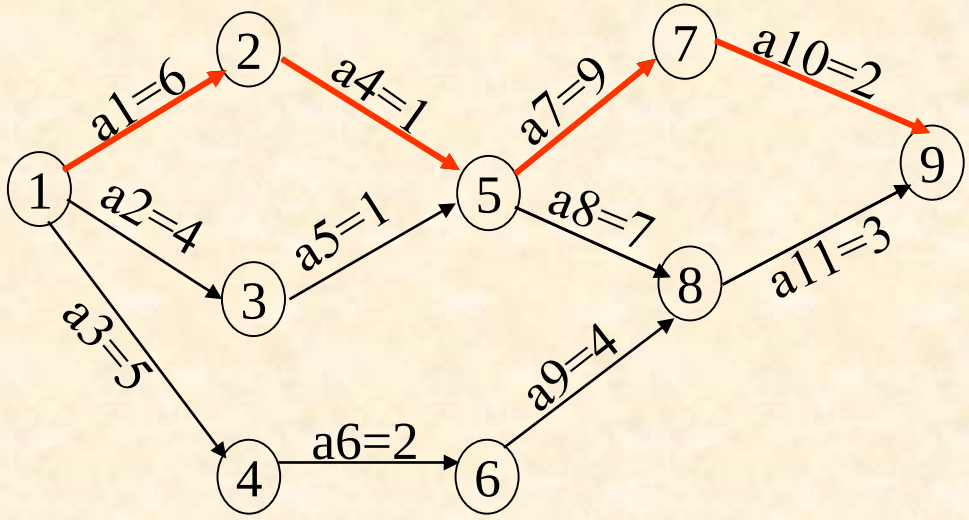
[例] 求关键活动— 第3步：



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

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	4	6	6	9	7	8	11	16	15
l_i-e_i	0	2	4	0	2	4	0	1	4	0	1

- 求关键路径步骤
  - ✧求Ve(i)
  - ✧求Vl(j)
  - ✧求e(i)
  - ✧求l(i)
  - ✧计算l(i)-e(i)



顶点	Ve	Vl
V1	0	0
V2	6	6
V3	4	6
V4	5	9
V5	7	7
V6	7	11
V7	16	16
V8	14	15
V9	18	18

活动	e	l	l-e
a1	0	0	0 ✓
a2	0	2	2
a3	0	4	4
a4	6	6	0 ✓
a5	4	6	2
a6	5	9	4
a7	7	7	0 ✓
a8	7	8	1
a9	7	11	4
a10	16	16	0 ✓
a11	14	15	1

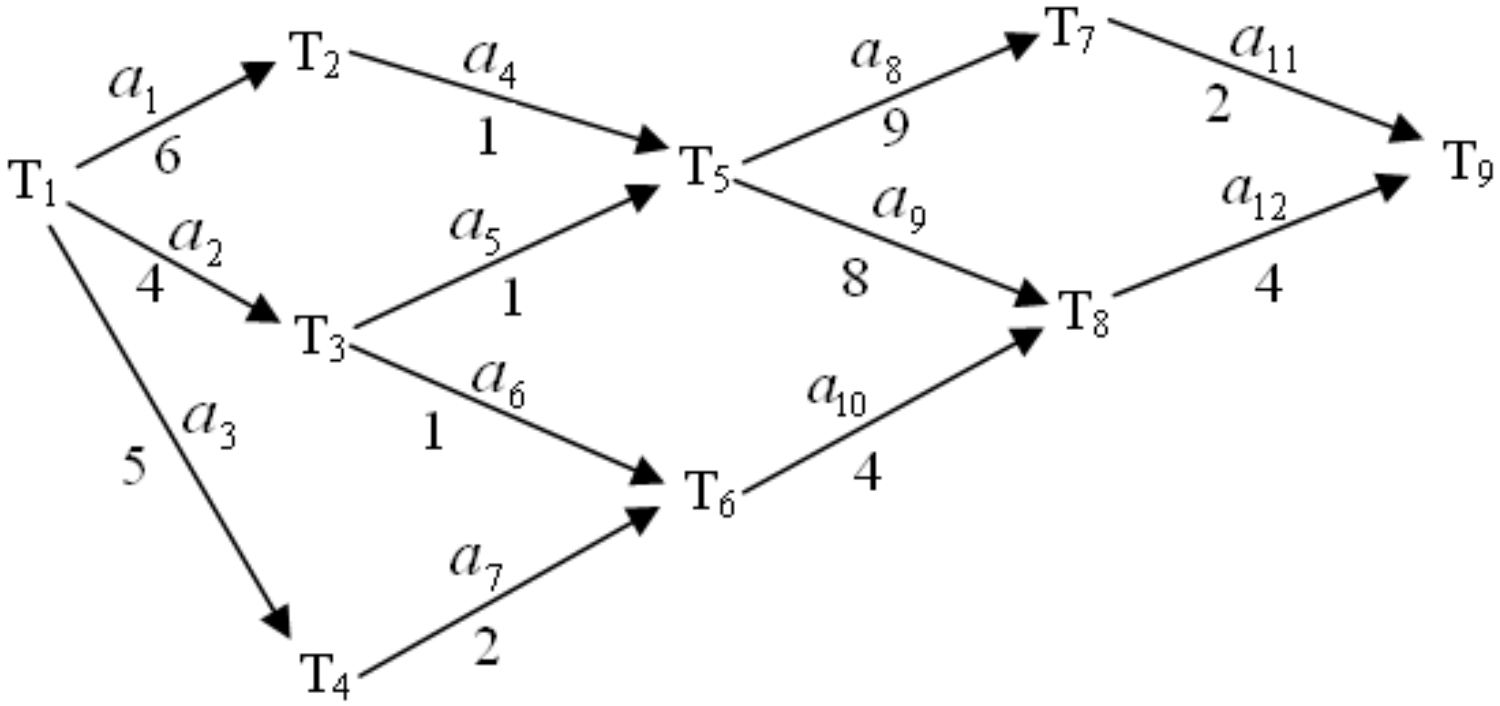


图 5.14　AOE 网

表 5.1　图 5.14 中各个活动的最早开始时间和最晚开始时间

a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
e(i)	0	0	0	6	4	4	5	7	7	7	16	19
l(i)	0	2	4	6	6	10	9	8	7	11	17	19
l(i)-e(i)	0	2	4	0	2	6	4	1	0	4	1	0

```
Status TopologicalOrder(ALGraph G, Stack &T) {  // 算法7.13
// 有向网G采用邻接表存储结构，求各顶点事件的最早发生时间ve(全局变量)
// T为拓扑序列定点栈，S为零入度顶点栈。
// 若G无回路，则用栈T返回G的一个拓扑序列，且函数值为OK，否则为ERROR。
    SqStack S;
    int count,k,i;
    ArcNode *p;
    char indegree[MAX_VERTEX_NUM];
    FindInDegree(G, indegree);   // 对各顶点求入度indegree[0..vernum-1]
    InitStack(S);
    for (i=0; i<G.vexnum; ++i)       // 建零入度顶点栈S
       if (!indegree[i]) Push(S, i);  // 入度为0者进栈
    count = 0;                        // 对输出顶点计数
```

```
for(int i=0; i<G.vexnum; i++) ve[i] = 0;  // 初始化
while (!StackEmpty(S)) {
    Pop(S,i);  Push(T, i);  ++count;       // i号顶点入T栈并计数
    for (p=G.vertices[i].firstarc;  p;  p=p->nextarc) {
        k = p->adjvex;                     // 对i号顶点的每个邻接点的入度减1
        if (!(--indegree[k])) Push(S, k);  // 若入度减为0，则入栈
        if (ve[i]+p->info > ve[k])  ve[k] = ve[i]+p->info;
    } //for  *(p->info)=dut(<j,k>)
 }//while
 if (count<G.vexnum) return ERROR;      // 该有向图有回路
 else return OK;
} // TopologicalOrder
```

```
Status CriticalPath(ALGraph G) {  // 算法7.14
  // G为有向网，输出G的各项关键活动。
  Stack T;
  int a,j,k,el,ee,dut;
  char tag;
  ArcNode *p;
  if (!TopologicalOrder(G, T)) return ERROR;
  for(a=0; a<G.vexnum; a++)
    vl[a] = ve[G.vexnum-1];    // 初始化顶点事件的最迟发生时间
```

```
    while (!StackEmpty(T))      // 按拓扑逆序求各顶点的vl值
        for (Pop(T, j), p=G.vertices[j].firstarc;  p;  p=p->nextarc) {
            k=p->adjvex;  dut=p->info;     //dut<j,k>
            if (vl[k]-dut < vl[j]) vl[j] = vl[k]-dut;
        }
    for (j=0; j<G.vexnum; ++j)          // 求ee,el和关键活动
        for (p=G.vertices[j].firstarc;  p;  p=p->nextarc) {
            k=p->adjvex;dut=p->info;
            ee = ve[j];  el = vl[k]-dut;
            tag = (ee==el) ? '*' : ' ';
            printf(j, k, dut, ee, el, tag);   // 输出关键活动
        }
    return OK;
} // CriticalPath
```

时间复杂性：

对定点进行拓扑排序的时间复杂性为O(n+e), 以拓扑排序求$ve[i]$和以拓扑逆序求$vl[i]$时，所需时间为均为O(e), 求各个活动的$e[k]$和$l[k]$的时间复杂度为O(e), 整个算法的时间复杂性是O(n+e)。

第七章 图

5.6 最短路径问题

- 两顶点间可能存在**多条路径**，每条路径经过的**边数不同**，每条路径的各边**权值之和**也不同。
- 从一个**指定的顶点到达另一指定顶点**的路径上**各边权值之和最小的路径**被称为**最短路径**，这类问题亦称为**最短路径问题**。

❑ 问题提出

用带权的有向图表示一个**交通运输网**，图中：

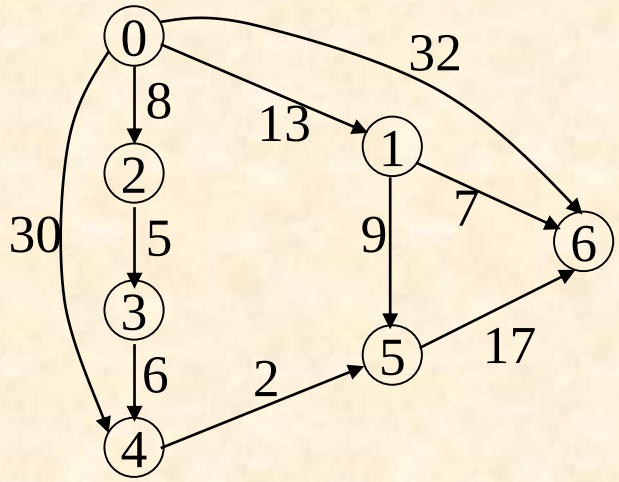
顶点：表示城市

边：　表示城市间的交通联系

权：　表示此线路的长度或沿此线路运输所花的时间或费用等

问题：从某顶点出发，沿图的边到达另一顶点所经过的路径中，各边上权值之和最小的一条路径——**最短路径**

❑ 从某个源点到其余各顶点的最短路径



最短路径	长度
<V0,V1>	13
<V0,V2>	8
<V0,V2,V3>	13
<V0,V2,V3,V4>	19
<V0,V2,V3,V4,V5>	21
<V0,V1,V6>	20

Dijkstra算法

基本思想：

- 将图中所有顶点集合V分成两个集合S和T，即V=S+T；
- 第一个集合S包括已确定最短路径的顶点；
- 第二个集合T包括尚未确定最短路径的顶点；
- 按照**最短路径长度递增的顺序**逐个把集合T的顶点加到集合S中去；
- 直至从源点出发可以到达的所有顶点都包括到集合S中。

保证：

- 从源点V0到S中各顶点的最短路径长度都不大于从V0到T中任何顶点的最短路径长度
- 每个顶点对应一个距离值

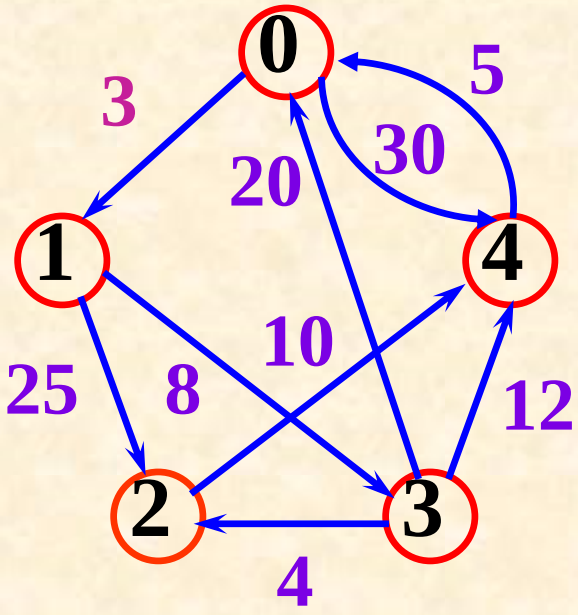
  S中顶点：从V0到此顶点的最短路径长度

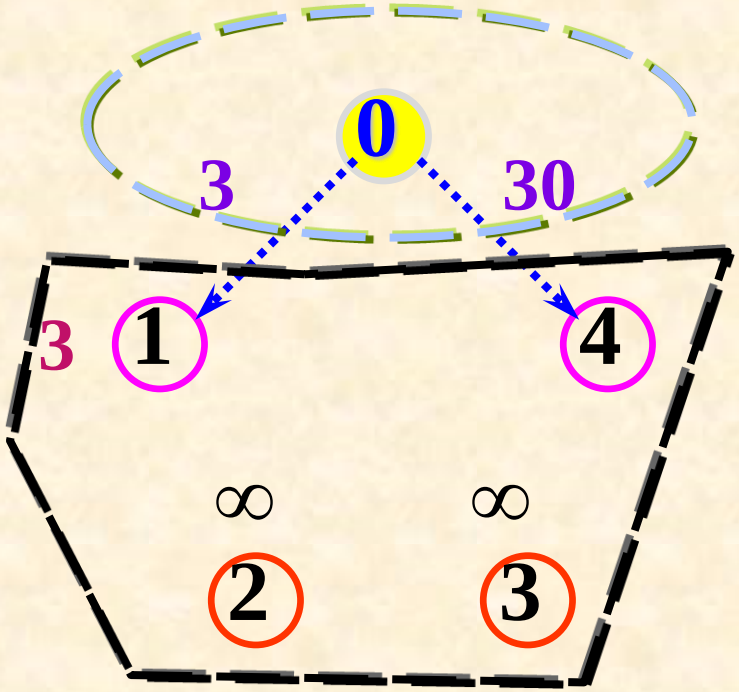
  T中顶点：从V0到此顶点的只包括S中顶点作中间顶点的最短路径长度

依据： 可以证明V0到T中顶点Vk的最短路径（反证法可证）

- 或是从V0到Vk的直接路径的权值；
- 或是从V0经S中顶点到Vk的路径权值之和。

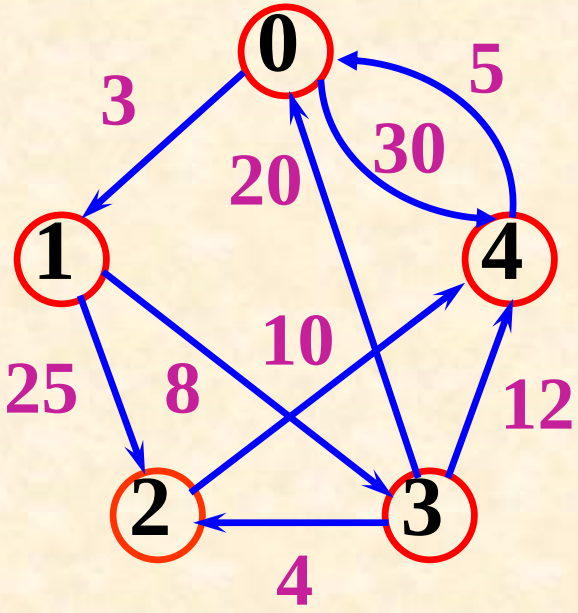
Dijkstra算法可以按照非递减次序依次得到各顶点的最小路径长度。

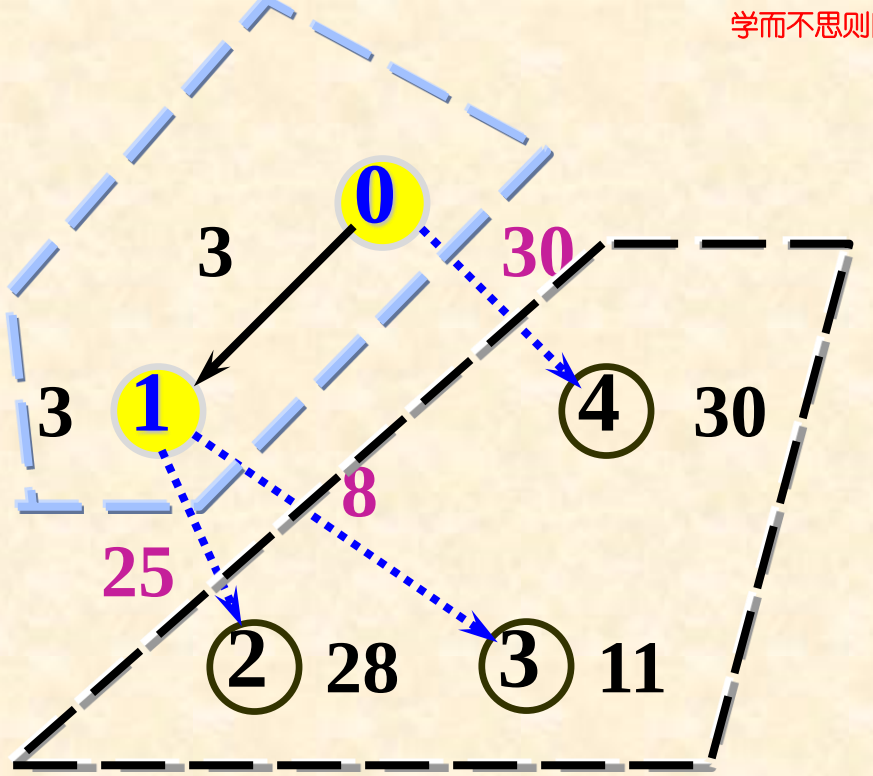




最小最短路径

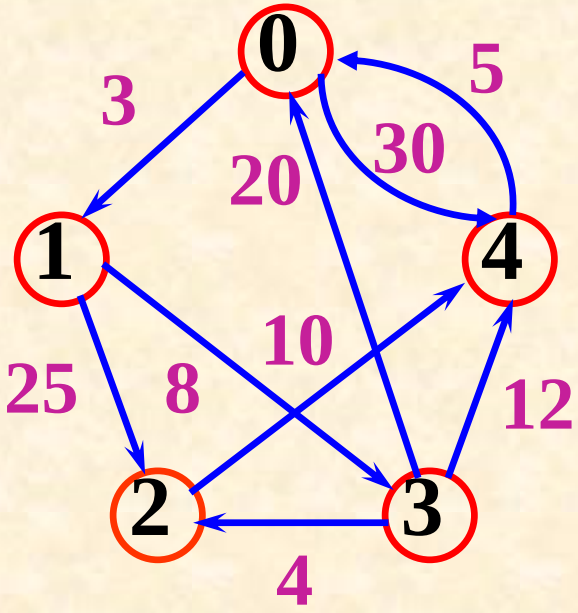
Dijkstra算法按递增次序依次得到各顶点的最小路径长度

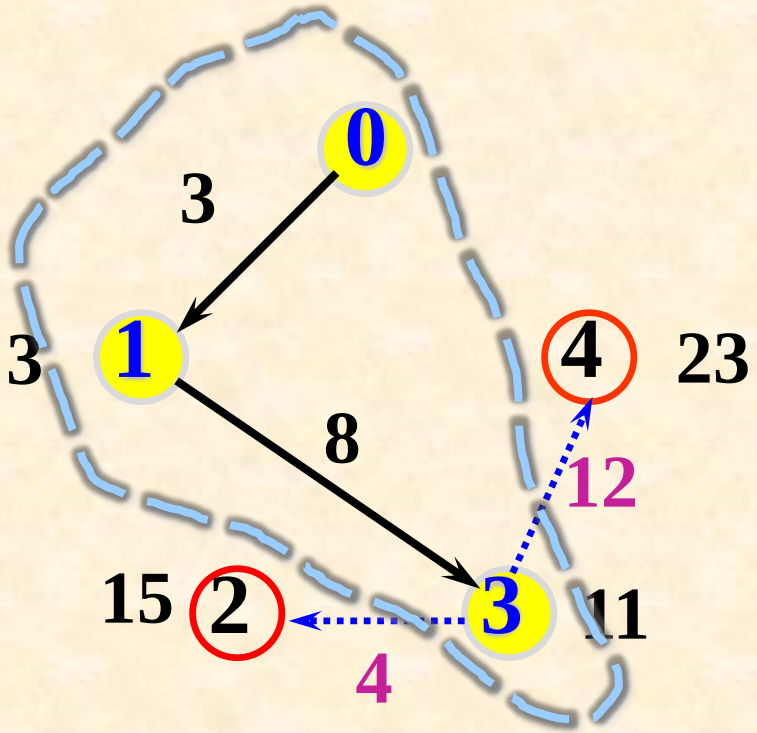




次小最短路径

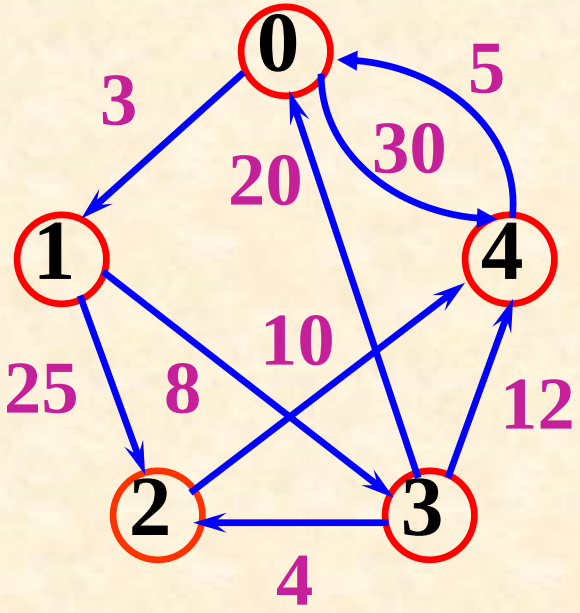
Dijkstra算法按递增次序依次得到各顶点的最小路径长度

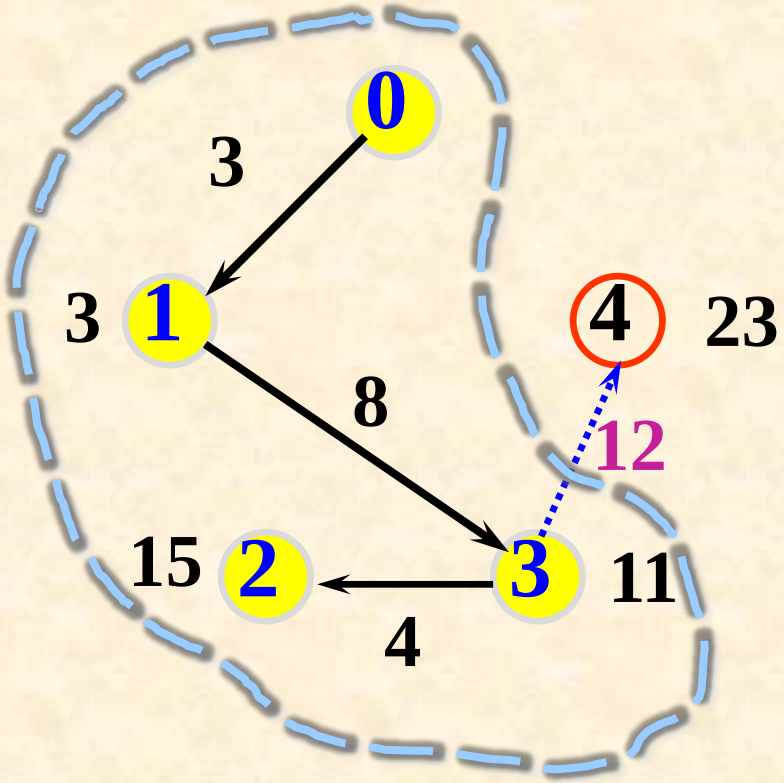




第三小最短路径

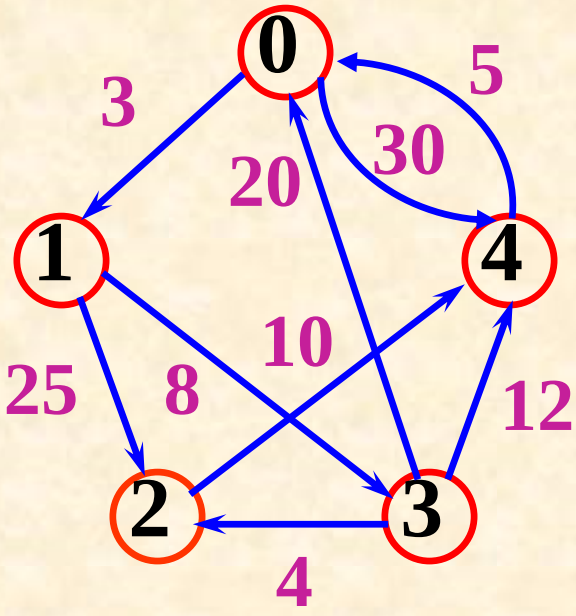
Dijkstra算法按递增次序依次得到各顶点的最小路径长度

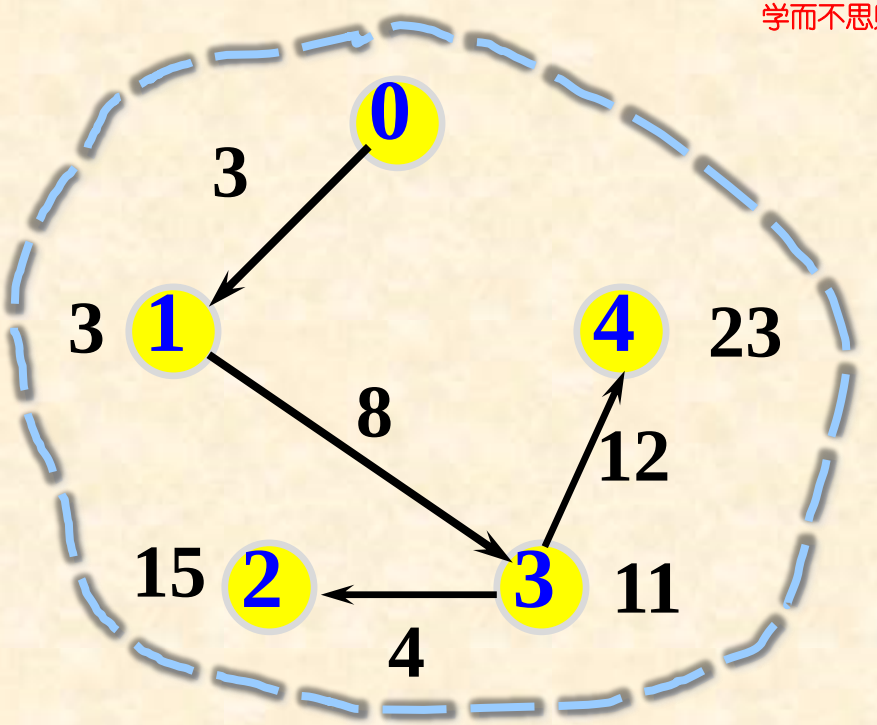




第四小最短路径

Dijkstra算法按递增次序依次得到各顶点的最小路径长度





第五小最短路径

Dijkstra算法按递增次序依次得到各顶点的最小路径长度

Dijkstra算法

基本思想：

- 将图中所有顶点集合V分成两个集合S和T，即V=S+T；
- 第一个集合S包括已确定最短路径的顶点；
- 第二个集合T包括尚未确定最短路径的顶点；
- 按照**最短路径长度递增的顺序**逐个把集合T的顶点加到集合S中去；
- 直至从源点出发可以到达的所有顶点都包括到集合S中。

保证：

- 从源点V0到S中各顶点的最短路径长度都不大于从V0到T中任何顶点的最短路径长度
- 每个顶点对应一个距离值

  S中顶点：从V0到此顶点的最短路径长度

  T中顶点：从V0到此顶点的只包括S中顶点作中间顶点的最短路径长度

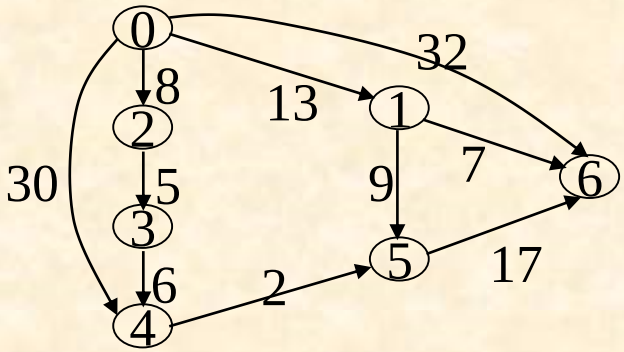
依据： 可以证明V0到T中顶点Vk的最短路径（反证法可证）

- 或是从V0到Vk的直接路径的权值；
- 或是从V0经S中顶点到Vk的路径权值之和。

Dijkstra算法可以按照非递减次序依次得到各顶点的最小路径长度。

求最短路径步骤

- 初使时令 S={V0}, T={其余顶点}，T中顶点对应的距离值：
  - 若存在<V0,Vi>，为<V0,Vi>弧上的权值
  - 若不存在<V0,Vi>，为∞
- 从T中选取一个其距离值为最小的顶点W，加入S
- 对T中顶点的距离值进行修改：若加进W作中间顶点，从V0到Vi的距离值比不加W的路径要短，则修改此距离值
- 重复上述步骤，直到S中包含所有顶点，即S=V为止



终点	从V0到各终点的最短路径及其长度				
V1	13 <V0,V1>	13 <V0,V1>	-------	-------	-------
V2	8 <V0,V2>	-------	-------	-------	-------
V3	∝	13 <V0,V2,V3>	13 <V0,V2,V3>	-------	-------
V4	30 <V0,V4>	30 <V0,V4>	30 <V0,V4>	19 <V0,V2,V3,V4>	-------
V5	∝	∝	22 <V0,V1,V5>	22 <V0,V1,V5>	21 <V0,V2,V3,V4,V5>
V6	32 <V0,V6>	32 <V0,V6>	20 <V0,V1,V6>	20 <V0,V1,V6>	20 <V0,V1,V6>
Vj	V2:8 <V0,V2>	V1:13 <V0,V1>	V3:13 <V0,V2,V3>	V4:19 <V0,V2,V3,V4>	V6:20 <V0, V1,V6>

Dijkstra算法描述

初始时（S为初始顶点），$D_s=0$且 $\forall i \neq S$，有$D_i = +\infty$。

①在未访问顶点中选择D_v最小的顶点v，访问v，令 S[v]=1。

②依次考察v的邻接顶点w，若

$D_v+weight(<v,w>)< D_w$ ，

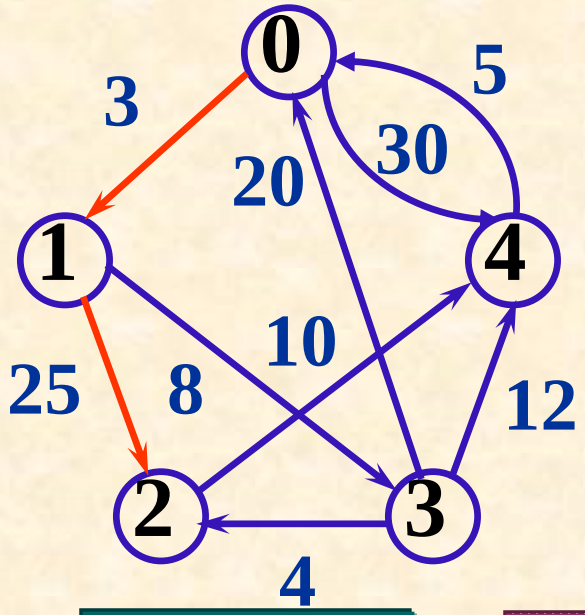
则改变D_w的值，使$D_w = D_v + weight(<v,w>)$ 。

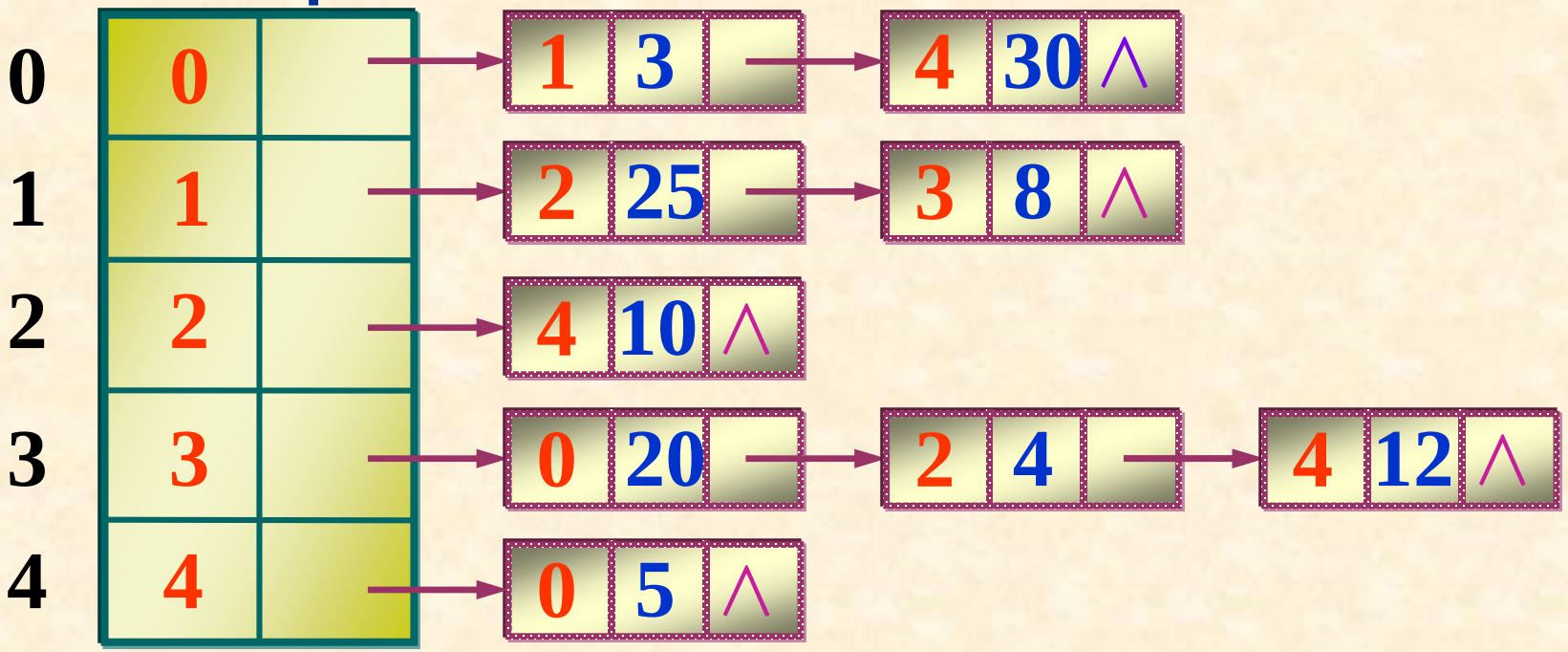
③重复① ②，直至所有顶点被访问。

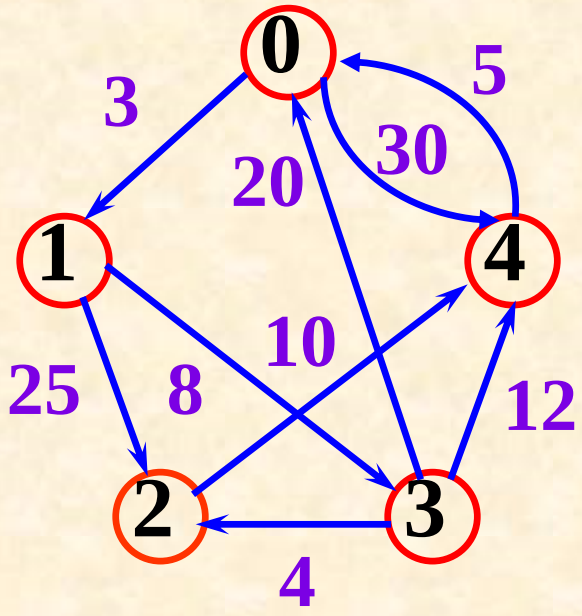
说明

- 引入一个**辅助数组***dist*。它的每一个分量*dist*[*i*]表示当前找到的从源点*s*到顶点*i* 的最短路径的长度。初始状态：*dist*[*s*]=0，对其它节点*i*有*dist*[*i*]=+∞ 。
- 引入path[*i*]记录*s*到*i*最短路径中*i*的前驱节点编号

[例]


0
1
2
3
4
3
5
20
30
25
8
10
12
4


0
1
2
3
4
0
1
2
3
4
1 3
4 30 ∧
2 25
3 8 ∧
4 10 ∧
0 20
2 4
4 12 ∧
0 5 ∧



①在未访问顶点中选择D_v最小的顶点v，访问v，令 S[v]=1。

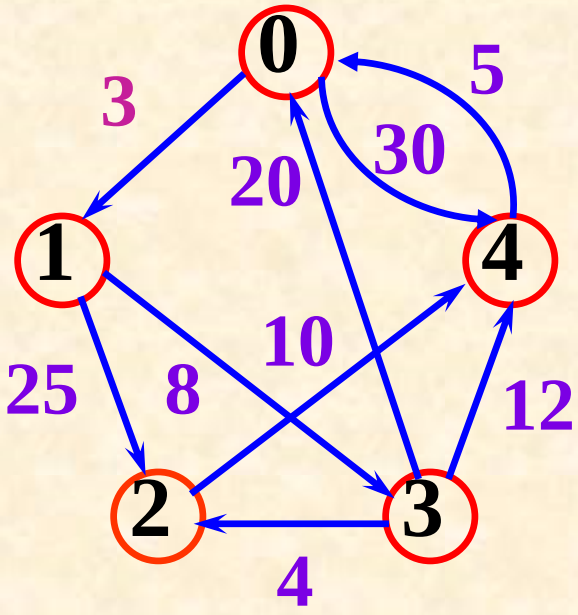
②依次考察v的邻接顶点w，若

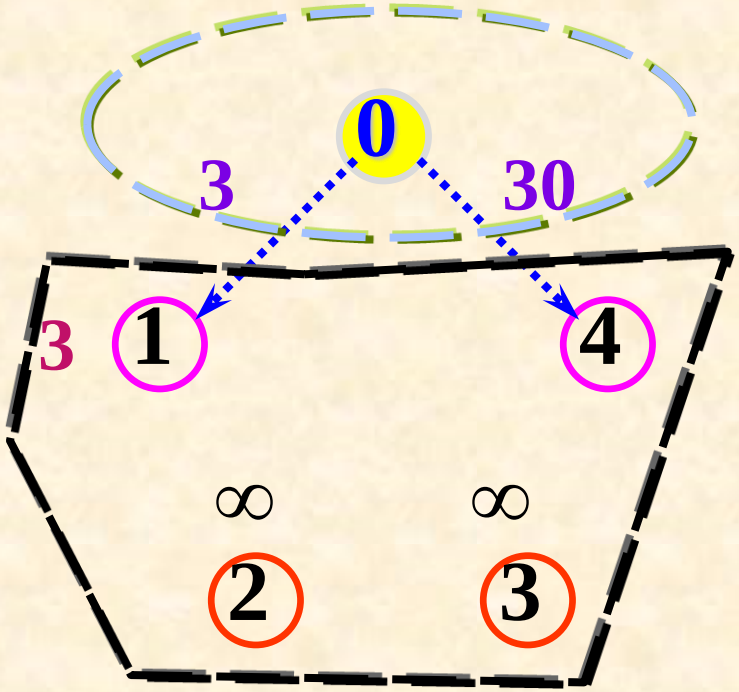
D_v+weight(<v, w>)< D_w ，

则改变D_w的值，使$D_w = D_v$ + weight(<v, w>) 。

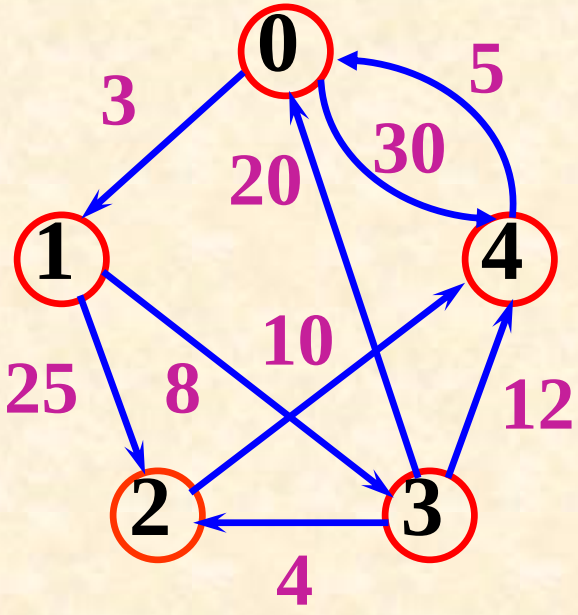
③重复① ②，直至所有顶点被访问。

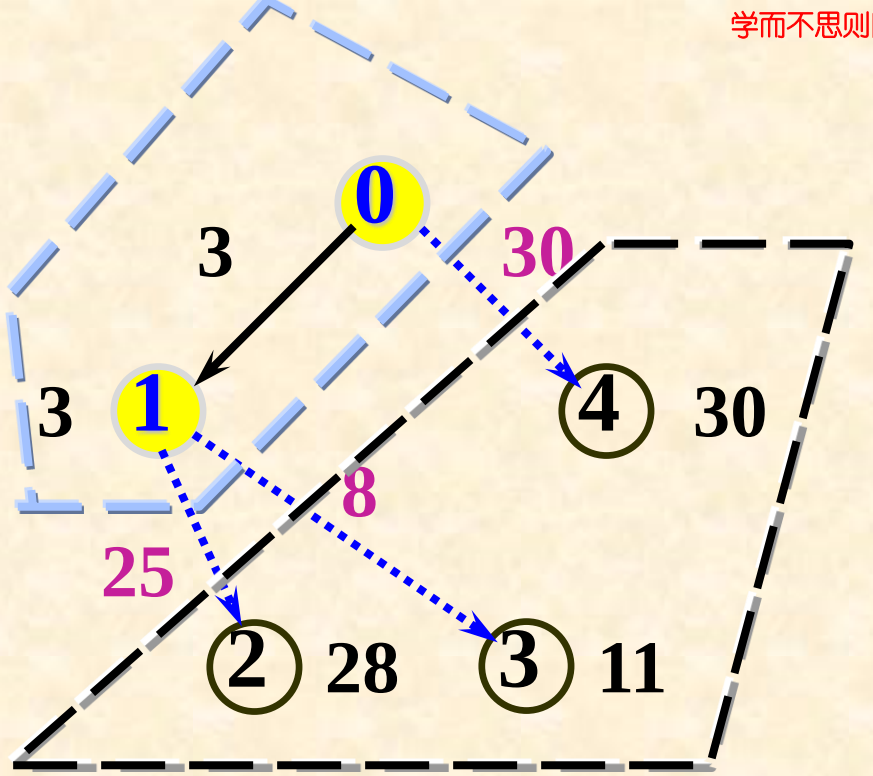
	0	1	2	3	4
s	0	0	0	0	0
dist	0	∞	∞	∞	∞
path					



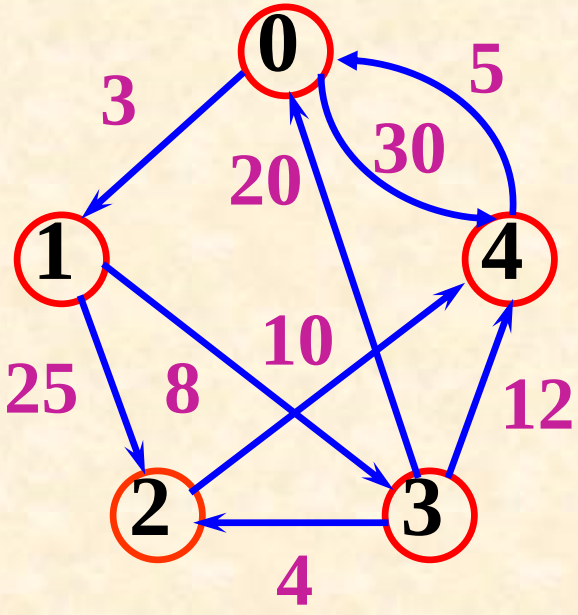


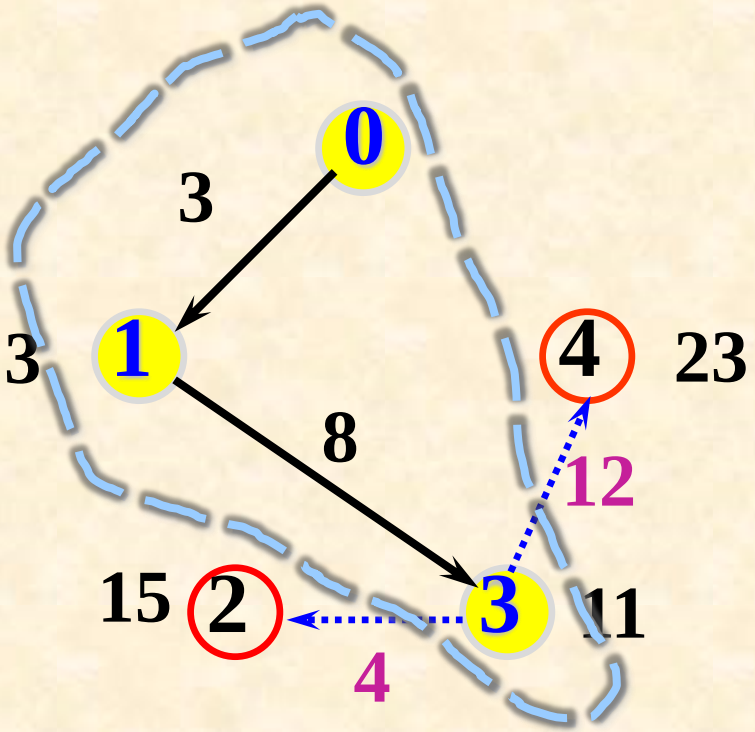
	0	1	2	3	4
s	1	0	0	0	0
dist	0	3	∞	∞	30
path		v_0			v_0



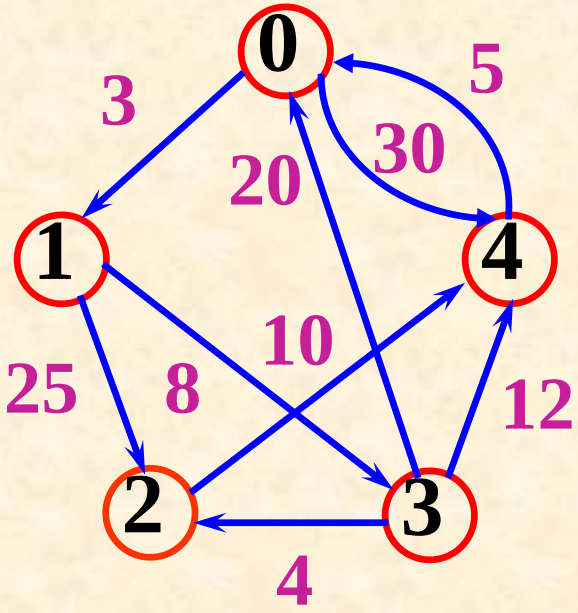


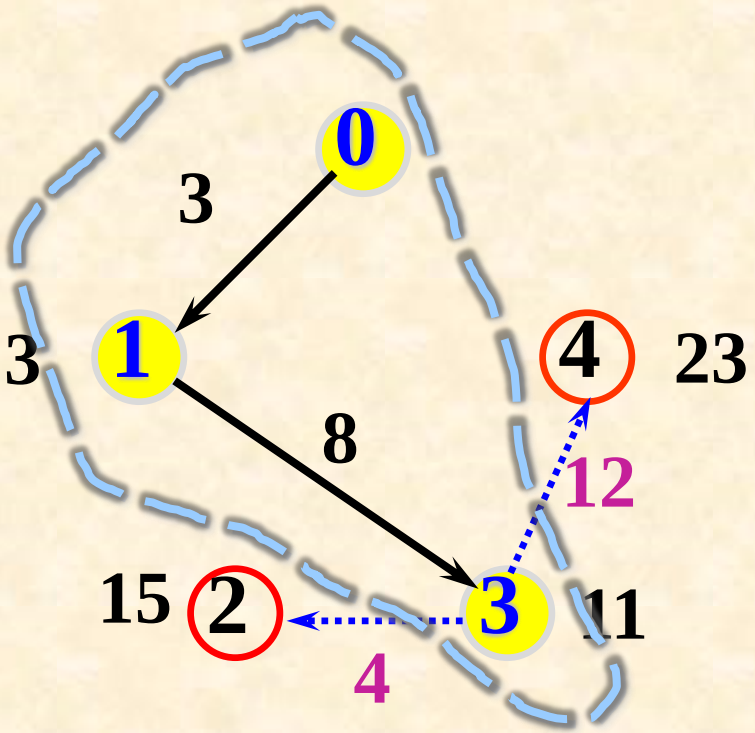
	0	1	2	3	4
s	1	1	0	0	0
dist	0	3	28	11	30
path		v_0	v_1	v_1	v_0



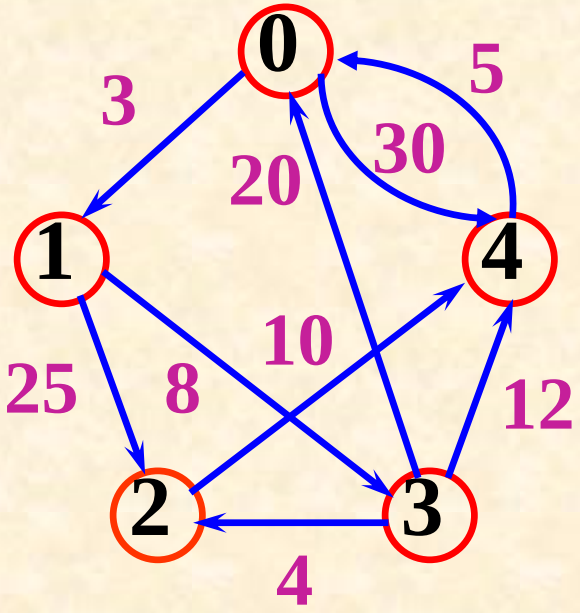


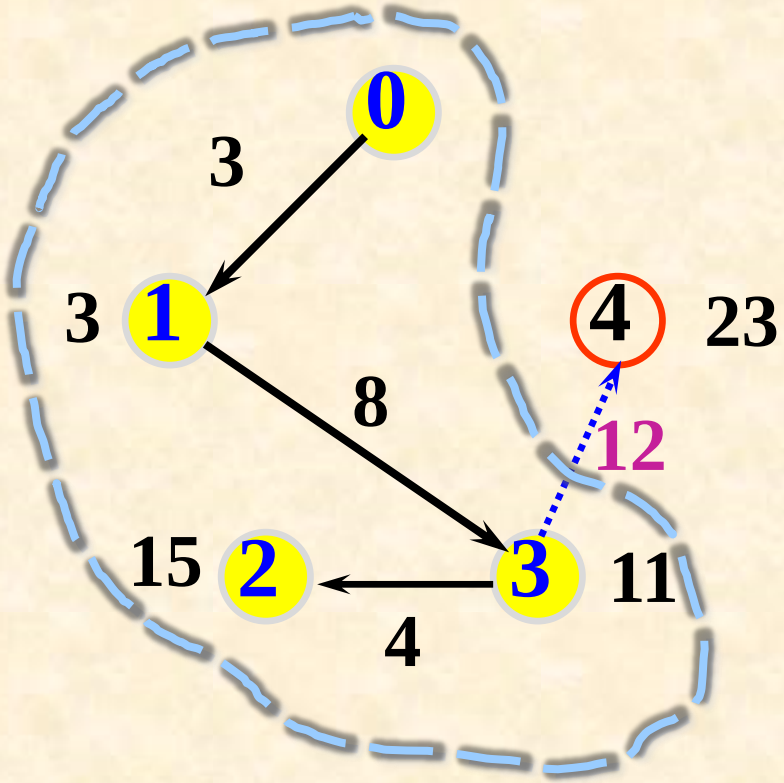
	0	1	2	3	4
s	1	1	0	1	0
dist	0	3	28	11	30
path		v_0	v_1	v_1	v_0



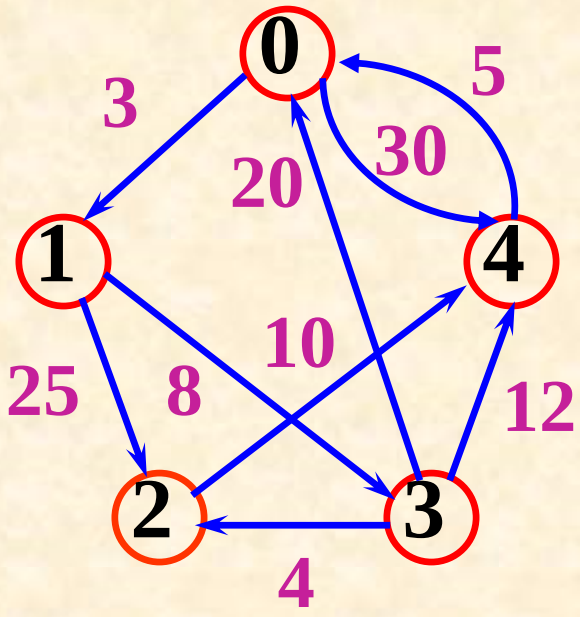


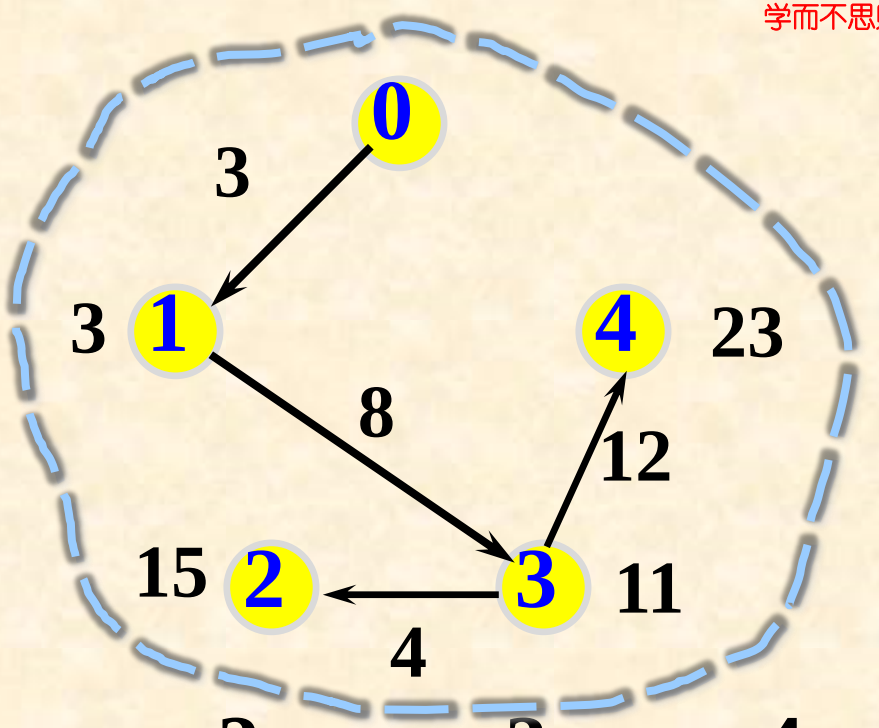
	0	1	2	3	4
s	1	1	0	1	0
dist	0	3	15	11	23
path		v_0	v_3	v_1	v_3





s	1	1	1	1	0
dist	0	3	15	11	23
path		v_0	v_3	v_1	v_3





	0	1	2	3	4
s	1	1	1	1	1
dist	0	3	15	11	23
path		v_0	v_3	v_1	v_3

Dijkstra算法按照非递减次序依次得到各顶点的最小路径长度。

时间复杂性分析

$$O(\sum_{i=1}^{n}(n+d_i)) = O(n^2 + \sum_{i=1}^{n} d_i) = O(n^2 + e)$$

该算法的时间复杂性为$O(n^2+e)$，也可认为是$O(n^2)$

7.6.3 每对顶点间的最短路径

- 问题：已知一个各边权值均大于0的带权有向图，对每一对顶点 $v_i \neq v_j$，求v_i与v_j间的最短路径和最短路径长度。
- 方法一：每次以一个顶点为源点，重复执行Dijkstra算法n
- 方法二：弗洛伊德(Floyd)算法
  - ➢算法思想：逐个顶点试探法
  - ➢求最短路径步骤
    - ① 初始时设置一个n阶方阵，令其对角线元素为0，若存在弧<Vi,Vj>，则对应元素为权值；否则为∞
    - ② 逐步试着在原直接路径中增加中间顶点，若加入中间点后路径变短，则修改之；否则，维持原值
    - ③ 所有顶点试探完毕，算法结束