数据结构与算法

第六章　树和图

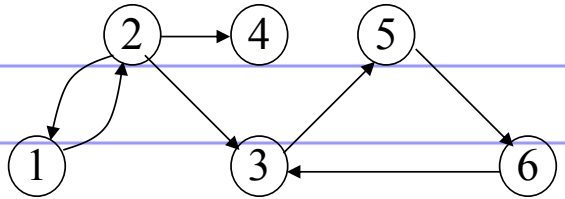
- 图的定义和术语
- 图的存储结构
- 图的遍历
- 生成树
- 拓扑排序
- 关键路径
- 最短路径

6.1 图的定义和术语

- 图(Graph)——图G是由两个集合V(G)和E(G)组成的,记为G=(V,E)。其中:
  V(G)是顶点的非空有限集
  E(G)是边的有限集合，边是顶点的无序对或有序对
- 有向图——有向图G是由两个集合V(G)和E(G)组成的
  其中：V(G)是顶点的非空有限集
  E(G)是有向边（也称弧）的有限集合，弧是顶点的有序对，记为<v,w>，v,w是顶点，v为弧尾，w为弧头
- 无向图——无向图G是由两个集合V(G)和E(G)组成的
  其中：V(G)是顶点的非空有限集
  E(G)是边的有限集合，边是顶点的无序对，记为（v,w）或（w,v），并且（v,w)=(w,v)

图的例子

例1

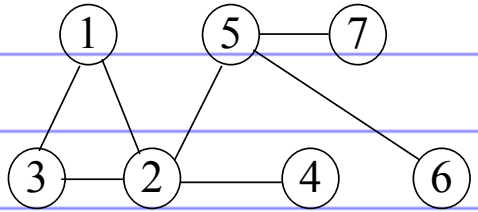


G1

图G1中：V(G1)={1,2,3,4,5,6}

E(G1)={<1,2>, <2,1>, <2,3>, <2,4>, <3,5>, <5,6>, <6,3>}

例2



G2

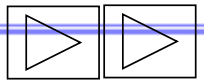
图G2中：V(G2)={1,2,3,4,5,6,7}

E(G1)={(1,2), (1,3), (2,3), (2,4),(2,5), (5,6), (5,7)}

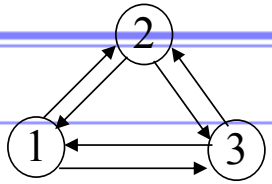
图的定义和术语

- 有向完备图——n个顶点的有向图最大边数是n(n-1) A_n^2
- 无向完备图——n个顶点的无向图最大边数是n(n-1)/2 C_n^2
- 权——与图的边或弧相关的数叫权
- 网——带权的图叫网
- 子图——如果图G(V,E)和图G`(V′,E`),满足：
  - V′⊆V
  - E′⊆E　　则称G`为G的子图
- 顶点的度
  - 无向图中，顶点的度为与每个顶点相连的边数
  - 有向图中，顶点的度分成入度与出度
    - ✧入度：以该顶点为头的弧的数目
    - ✧出度：以该顶点为尾的弧的数目

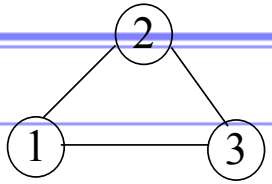
- 路径：路径是顶点的序列V={V_{i0},V_{i1},……V_{in}}，满足(V_{ij-1},V_{ij})∈E 或 <V_{ij-1},V_{ij}>∈E,(1<j≤n)
- 路径长度：沿路径边的数目或沿路径各边权值之和
- 回路：第一个顶点和最后一个顶点相同的路径叫~
- 简单路径：序列中顶点不重复出现的路径叫~
- 简单回路：除了第一个顶点和最后一个顶点外，其余顶点不重复出现的回路叫~
- 连通：从顶点V到顶点W有一条路径，则说V和W是连通的
- 连通图：图中任意两个顶点都是连通的叫~
- 连通分量：非连通图的每一个连通部分叫~
- 强连通图：有向图中，如果对每一对V_i，V_j∈V，V_i≠V_j，从V_i到V_j和从V_j到 V_i都存在路径，则称G是~



例

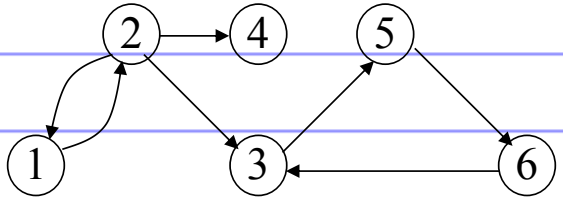


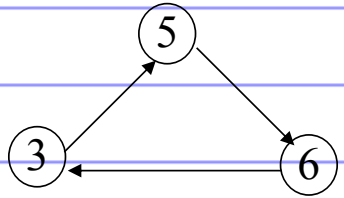
有向完备图



无向完备图

例

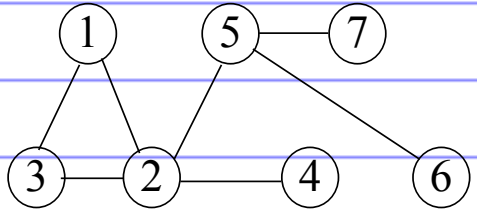




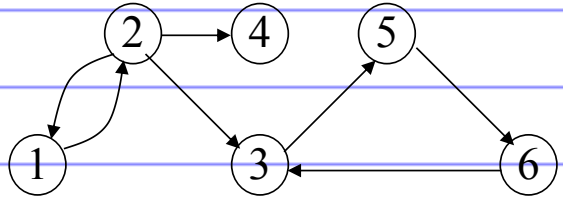
例

图与子图

例



G2



G1

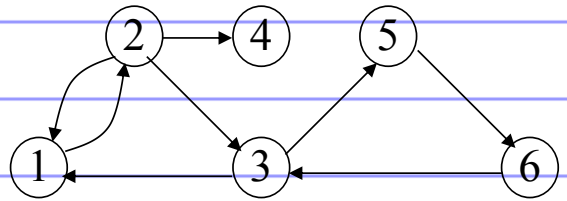
顶点5的度：3

顶点2的度：4

顶点2入度：1　出度：3

顶点4入度：1　出度：0

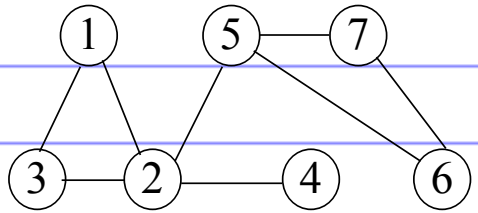
例



G1

路径：1，2，3，5，6，3
路径长度：5
简单路径：1，2，3，5
回路：1，2，3，5，6，3，1
简单回路：3，5，6，3

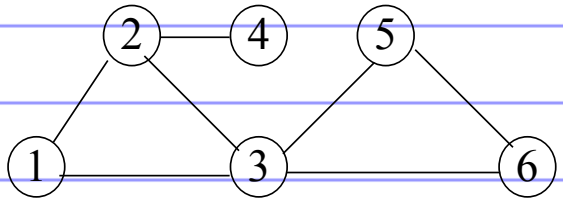
例



G2

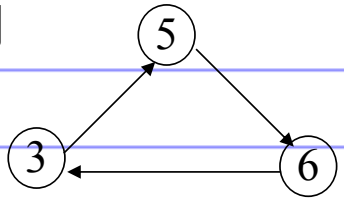
路径：1，2，5，7，6，5，2，3
路径长度：7
简单路径：1，2，5，7，6
回路：1，2，5，7，6，5，2，1
简单回路：1，2，3，1

例



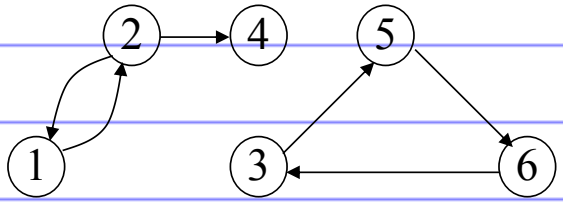
连通图

例



强连通图

例



非连通图
连通分量

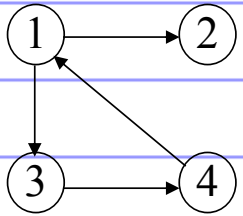
6.2 图的存储结构

- 多重链表
- 邻接矩阵
- 关联矩阵
- 邻接表

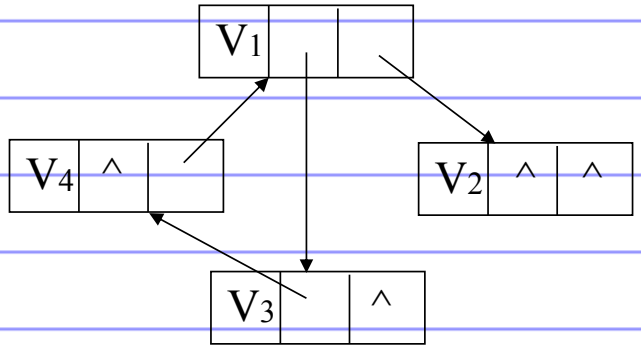
图的存储结构-多重链表

多重链表

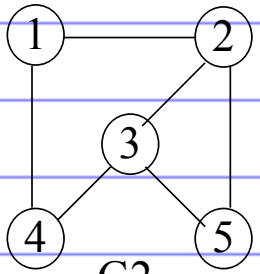
例



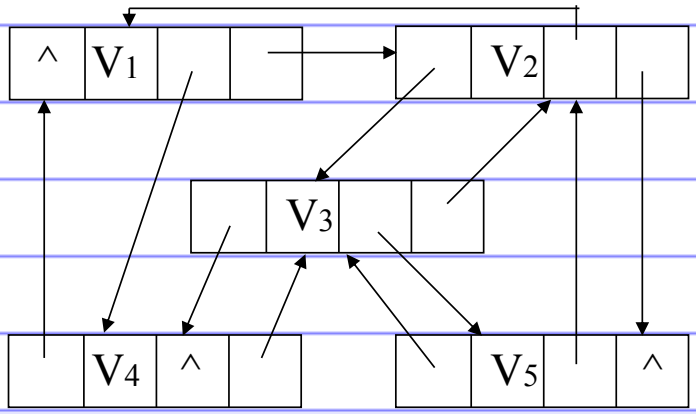
G1



例



G2

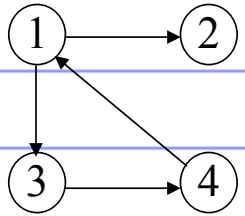


图的存储结构-邻接矩阵

- 邻接矩阵——表示顶点间相联关系的矩阵
- 定义：设G=(V,E)是有n≥1个顶点的图，G的邻接矩阵A是具有以下性质的n阶方阵

$$A[i,j]=\begin{cases}1, 若(v_i,v_j)或<v_i,v_j>\in E(G)\\ 0, \quad 其它\end{cases}$$

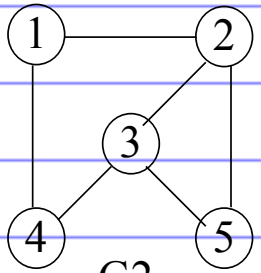
例



G1

	①	②	③	④
①	0	1	1	0
②	0	0	0	0
③	0	0	0	1
④	1	0	0	0

例



G2

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

■ 特点：

✧ 无向图的邻接矩阵对称，可压缩存储；有n个顶点的无向图需存储空间为n(n+1)/2

✧ 有向图邻接矩阵不一定对称；有n个顶点的有向图需存储空间为n^2

✧ 无向图中顶点V_i的度$TD(V_i)$是邻接矩阵A中第i行元素之和

✧ 有向图中，

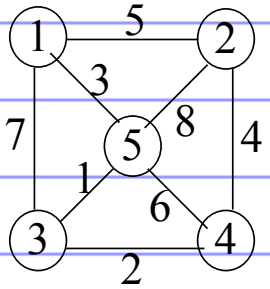
? 顶点V_i的出度是A中第i行元素之和

? 顶点V_i的入度是A中第i列元素之和

✧ 网络的邻接矩阵可定义为：

$$A[i,j]=\begin{cases}\omega_{ij}, 若(v_i,v_j)或<v_i,v_j>\in E(G)\\ 0, \quad 其它\end{cases}$$

例



	①	②	③	④	⑤
①	0	5	7	0	3
②	5	0	0	4	8
③	7	0	0	2	1
④	0	4	2	0	6
⑤	3	8	1	6	0

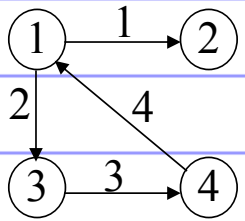
图的存储结构-关联矩阵

- **关联矩阵——表示顶点与边的关联关系的矩阵**
  - **定义：设G=(V,E)是有n≥1个顶点，e≥0条边的图，G的关联矩阵A是具有以下性质的n×e阶矩阵**

有向图：$$A[i,j]=\begin{cases}1, & i\text{顶点与}j\text{边相连，且}i\text{为尾}\\ 0, & i\text{顶点与}j\text{边不相连}\\ -1, & i\text{顶点与}j\text{边相连，且}i\text{为头}\end{cases}$$

无向图：$$A[i,j]=\begin{cases}1, & i\text{顶点与}j\text{边相连}\\ 0, & i\text{顶点与}j\text{边不相连}\end{cases}$$

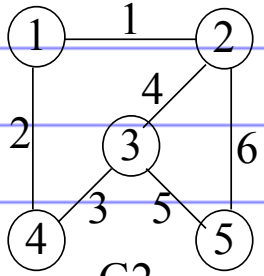
例



G1

$$\begin{array}{c} \\ ① \\ ② \\ ③ \\ ④ \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

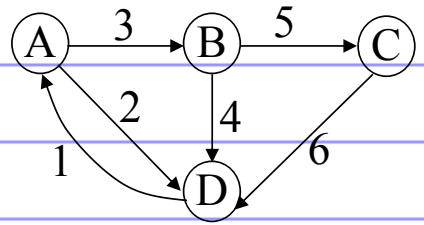
例



G2

$$\begin{array}{c} \\ ① \\ ② \\ ③ \\ ④ \\ ⑤ \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

例



$$
\begin{array}{c}
\\
A\\
B\\
C\\
D
\end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array}\\
\begin{bmatrix}
-1 & 1 & 1 & 0 & 0 & 0\\
0 & 0 & -1 & 1 & 1 & 0\\
0 & 0 & 0 & 0 & -1 & 1\\
1 & -1 & 0 & -1 & 0 & -1
\end{bmatrix}
\end{array}
$$

- 特点
  - 关联矩阵每列只有两个非零元素，是稀疏矩阵；n越大，零元素比率越大
  - 无向图中顶点V_i的度$TD(V_i)$是关联矩阵A中第i行元素之和
  - 有向图中，
    - 顶点V_i的出度是A中第i行中“1”的个数
    - 顶点V_i的入度是A中第i行中“-1”的个数

邻接表

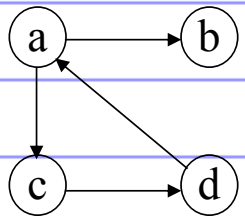
- 实现：为图中每个顶点建立一个单链表，第i个单链表中的结点表示依附于顶点V_i的边（有向图中指以V_i为尾的弧）

```
#define MAX_VERTEX_NUM 20
typedef   struct   ArcNode {
    int  adjvex;      //邻接点域，存放与Vi邻接的点在表头数组中的位置
    struct  ArcNode  *nextarc;   //链域，指示下一条边或弧
    InfoType *info;
}ArcNode;
typedef   struct  VNode {   //表头接点
    VertexType  data;       //存放顶点信息
    ArcNode  *firstarc;     //指示第一个邻接点
}VNode, AdjList[MAX_VERTEX_NUM];
typedef struct {
    AdjList vertices;
    int vexnum, arcnum;
    int kind;     //图的种类标识
} ALGraph;
```

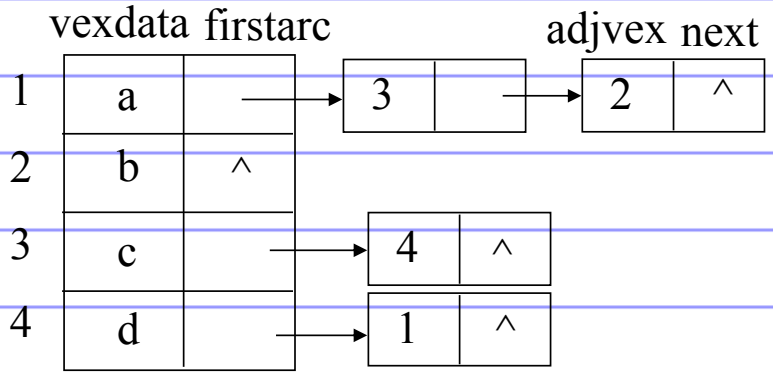
adjvex	nextarc	info

vexdata	firstarc

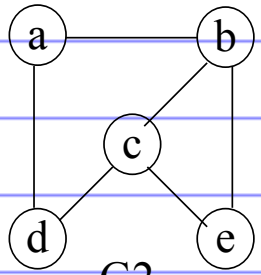
例


a
b
c
d

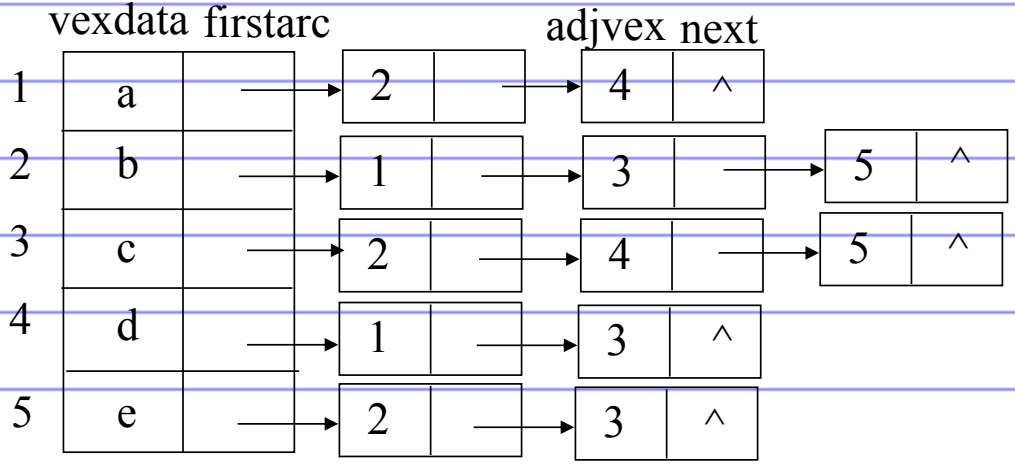
G1


vexdata firstarc
adjvex next
1
a
3
2
^
2
b
^
3
c
4
^
4
d
1
^

例


a
b
c
d
e

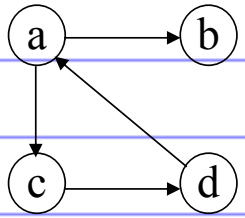
G2


vexdata firstarc
adjvex next
1
a
2
4
^
2
b
1
3
5
^
3
c
2
4
5
^
4
d
1
3
^
5
e
2
3
^

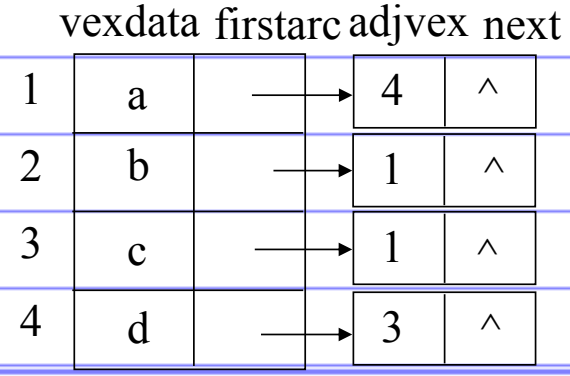
特点

- 无向图中顶点V_i的度为第i个单链表中的结点数
- 有向图中
  - 顶点V_i的出度为第i个单链表中的结点个数
  - 顶点V_i的入度为整个单链表中邻接点域值是i的结点个数
  - 逆邻接表：有向图中对每个结点建立以V_i为头的弧的单链表

例



G1



有向图的十字链表表示法

```
#define MAX_VERTEX_NUM 20

 typedef  struct  ArcBox {     //弧结点
    int  tailvex, headvex;      //弧尾、弧头在表头数组中位置
    struct ArcBox  *hlink, *tlink；//分别指向弧头、狐尾相同的下一条弧
    InfoType *info;
 }ArcBox;
```

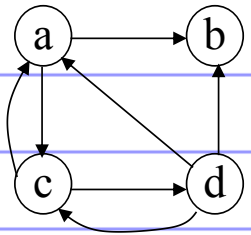
tailvex	headvex	hlink	tlink	info

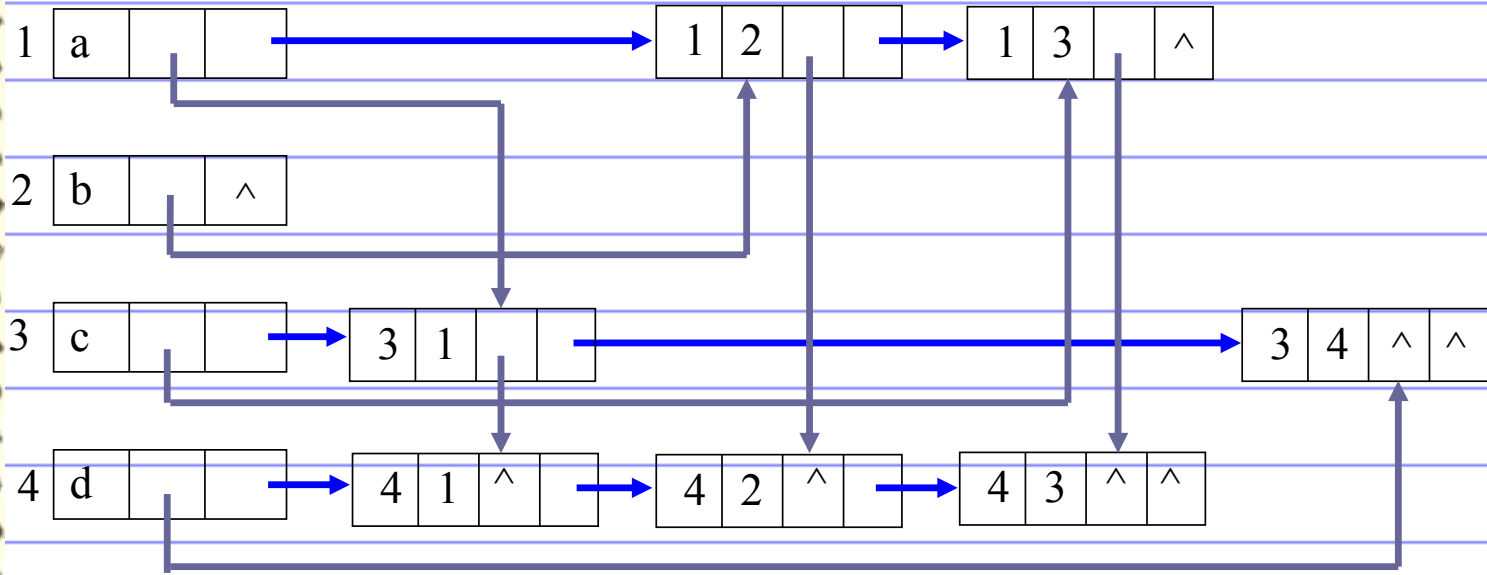
```
typedef  struct  VexNode {     //顶点结点
   VertexType data;  //存与顶点有关信息
   ArcBox *firstin, *firstout；//分别指向该顶点第一条入弧和出弧
}VexNode;
```

data	firstin	firstout

```
typedef struct {
   VexNode xlist[MAX_VERTEX_NUM];
   int vexnum, arcnum;
} OLGraph;
```

例





无向图的邻接多重表表示法

```
#define MAX_VERTEX_NUM 20
#define enum {unvisited, visited} VisitIf;
typedef  struct  EBox {       //边结点
   VisitIf  mark;     //标志域
   int  ivex, jvex;   //该边依附的两个顶点在表头数组中位置
   struct  EBox   *ilink,  *jlink;  //分别指向依附于ivex和jvex的下一条边
   InfoType *info;
}EBox;
```

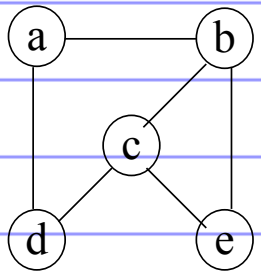
mark	ivex	ilink	jvex	jlink

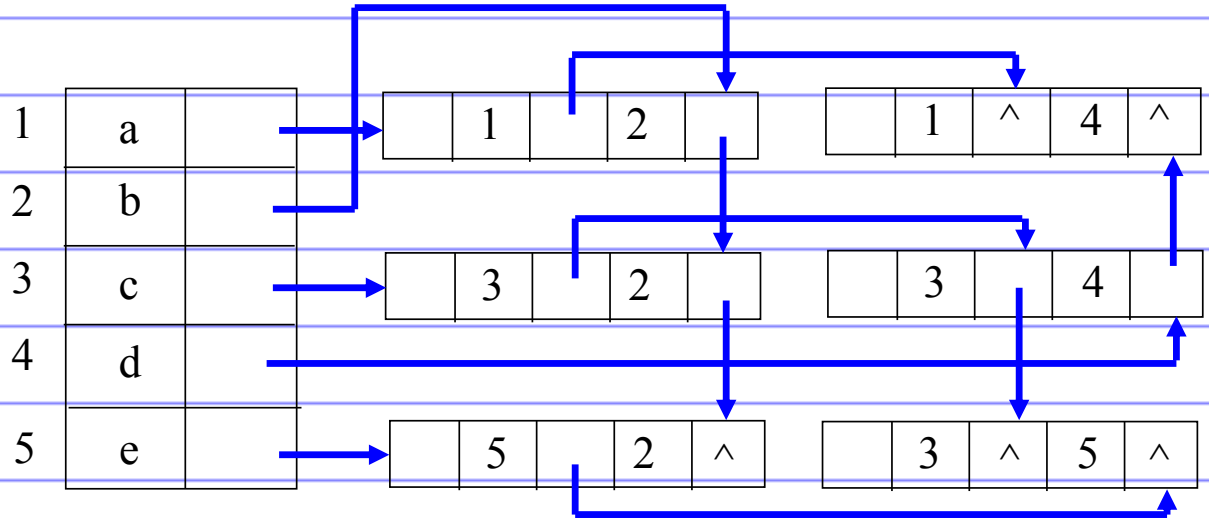
```
typedef  struct  VexNode {    //顶点结点
   VertexType data;   //存与顶点有关的信息
   EBox  *firstedge;  //指向第一条依附于该顶点的边
}VexBox;
```

data	firstedge

```
typedef struct {
   VexBox adjmulist[MAX_VERTEX_NUM];
   int vexnum, edgenum;
} AMLGraph;
```

例

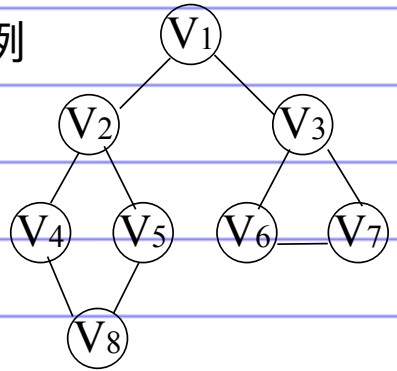




6.3 图的遍历

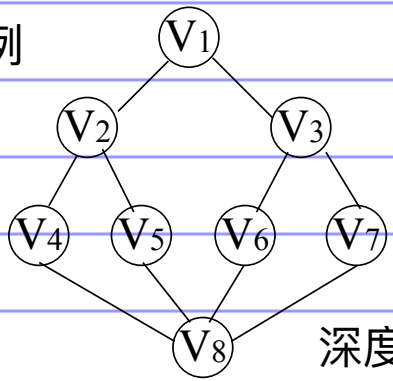
- 深度优先遍历(DFS)
  - 从图的某一顶点V_0出发，访问此顶点；然后依次从V_0的未被访问的邻接点出发，深度优先遍历图，直至图中所有和V_0相通的顶点都被访问到；若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止

例



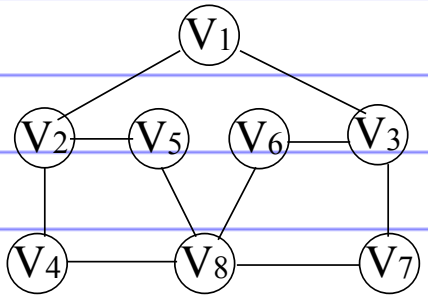
深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V3 ⇒V6 ⇒V7

例



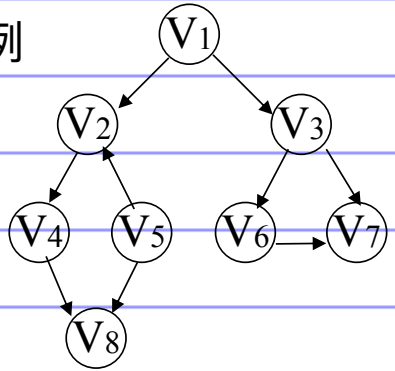
深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V6 ⇒V3 ⇒V7

例



深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V6 ⇒V3 ⇒V7

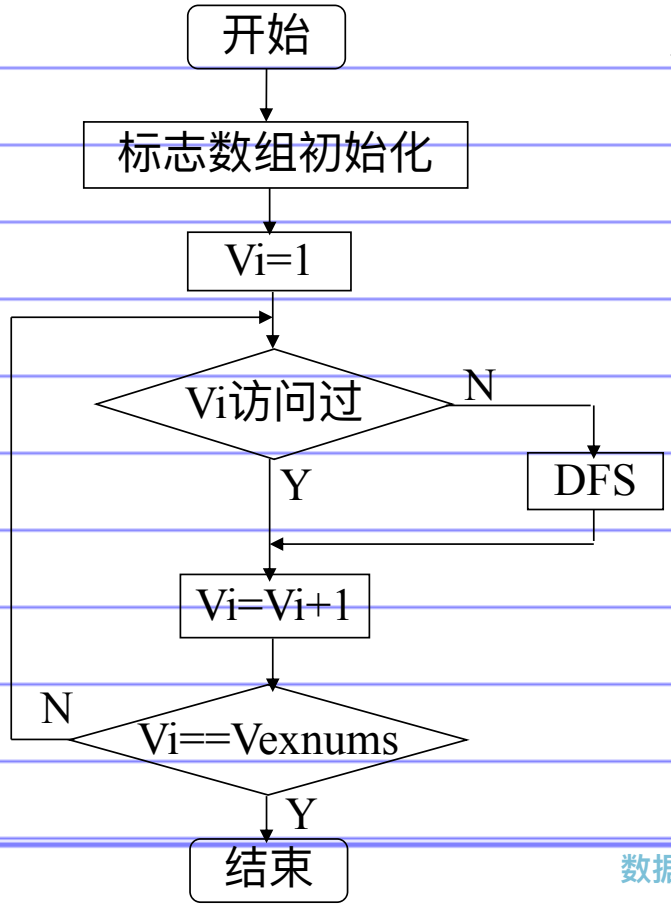
例

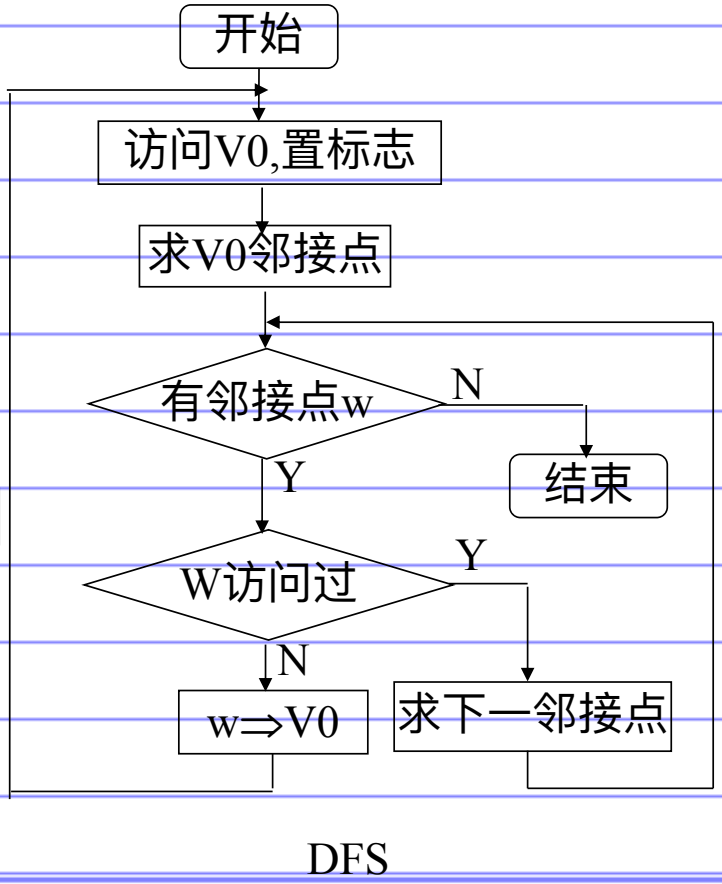


深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V3 ⇒V6 ⇒V7 ⇒V5

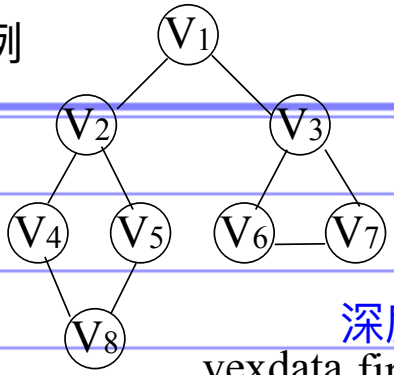
深度优先遍历算法

- **递归算法**

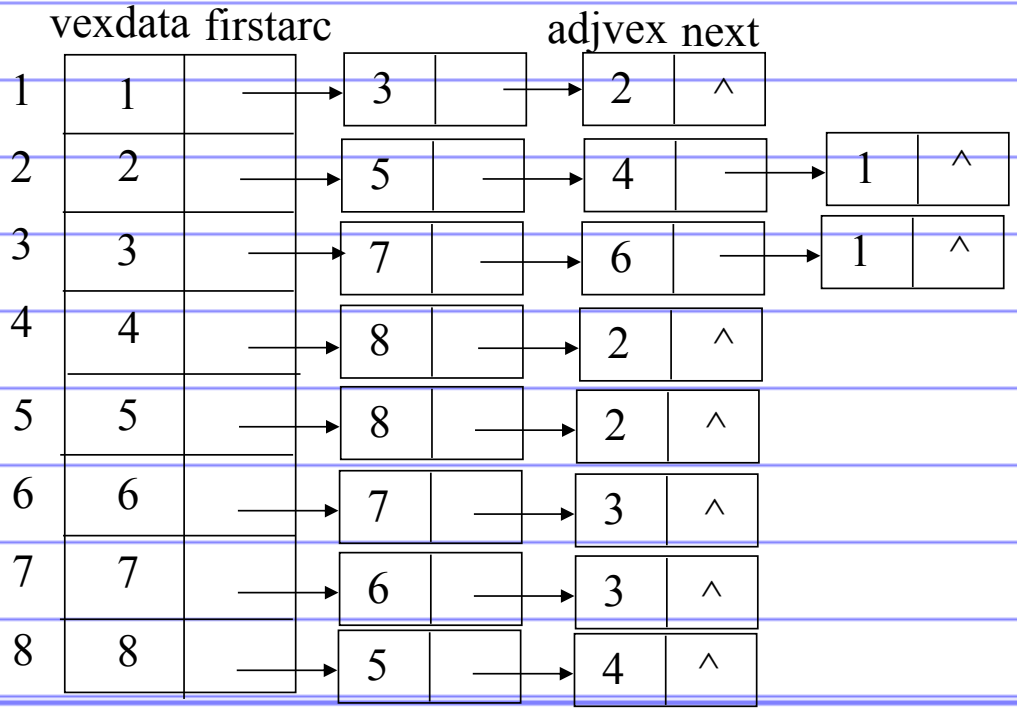




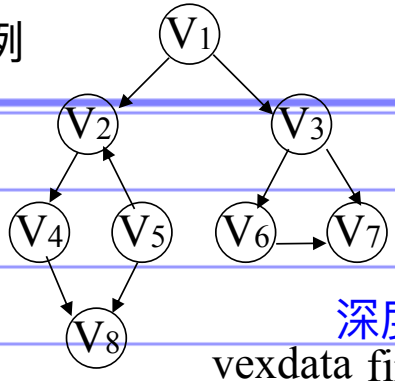
例



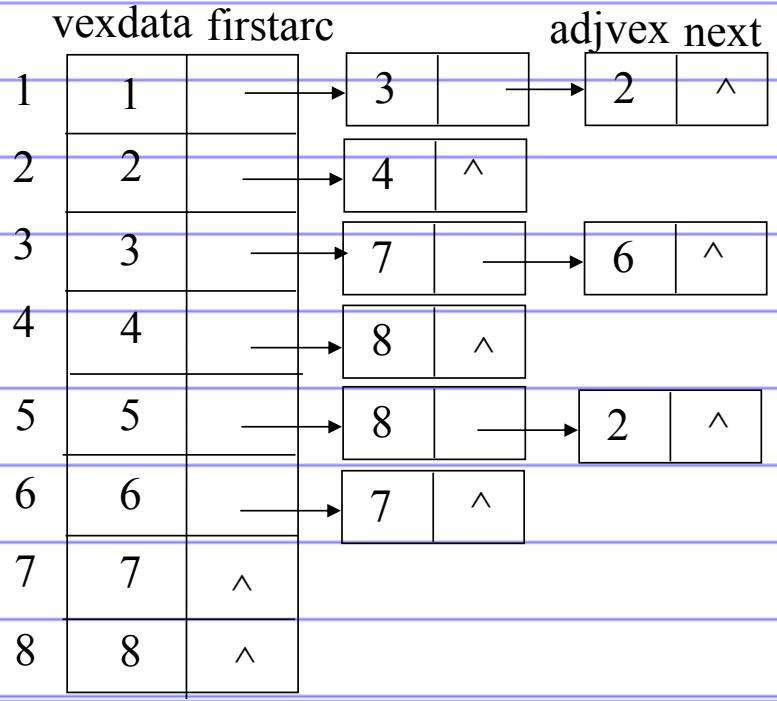
深度遍历：V1⇒V3 ⇒V7 ⇒V6 ⇒V2 ⇒V5 ⇒V8 ⇒V4



例



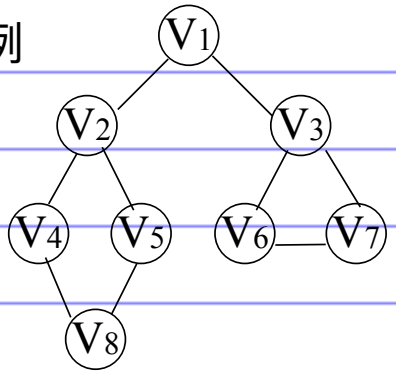
深度遍历：V1⇒V3 ⇒V7 ⇒V6 ⇒V2 ⇒V4 ⇒V8 ⇒V5



图的遍历

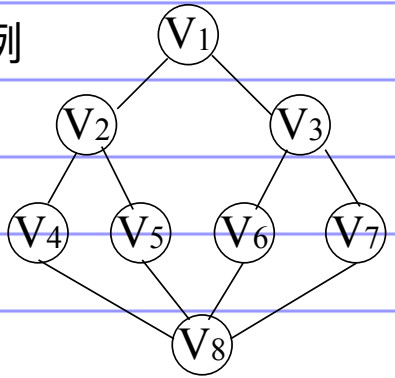
- 广度优先遍历(BFS)
  - 从图的某一顶点V0出发，访问此顶点后，依次访问V_0的各个未曾访问过的邻接点；然后分别从这些邻接点出发，广度优先遍历图，直至图中所有已被访问的顶点的邻接点都被访问到；若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止

例



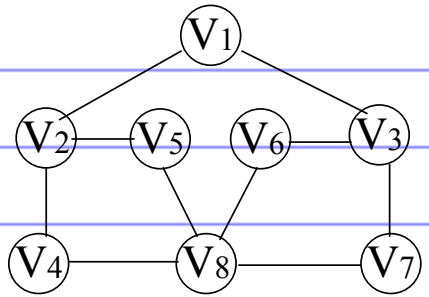
广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

例



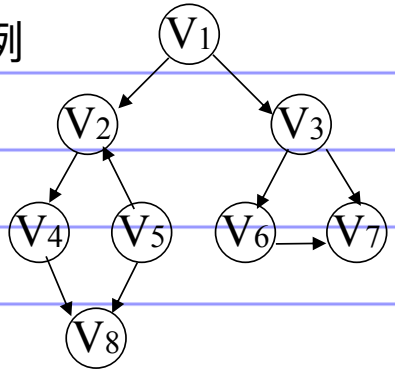
广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

例



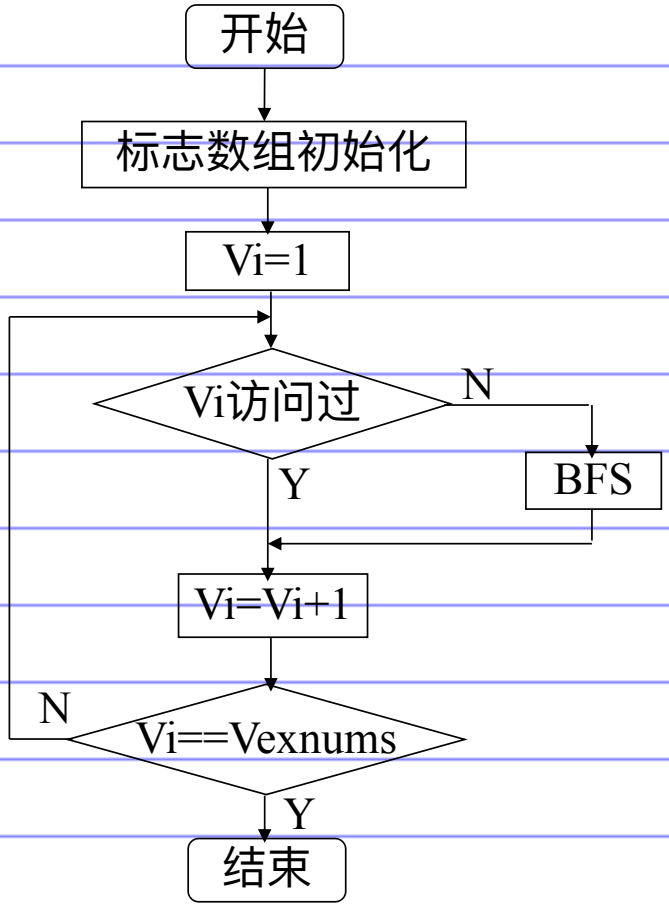
广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

例

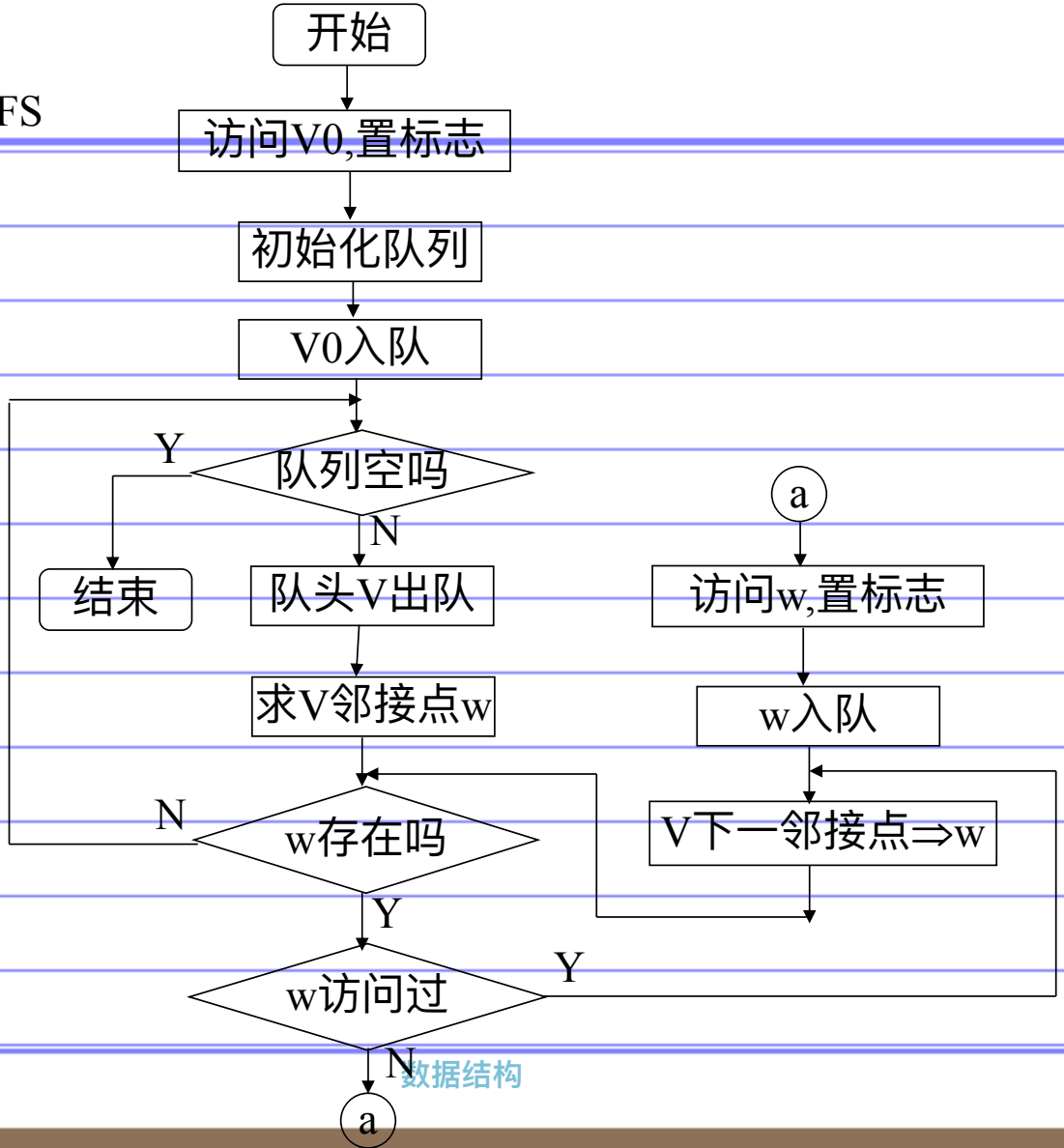


广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V6 ⇒V7 ⇒V8 ⇒V5

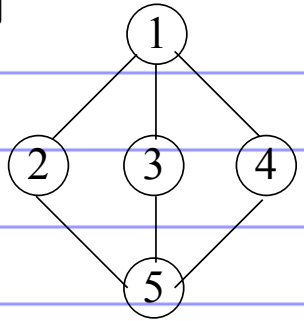
广度优先遍历算法

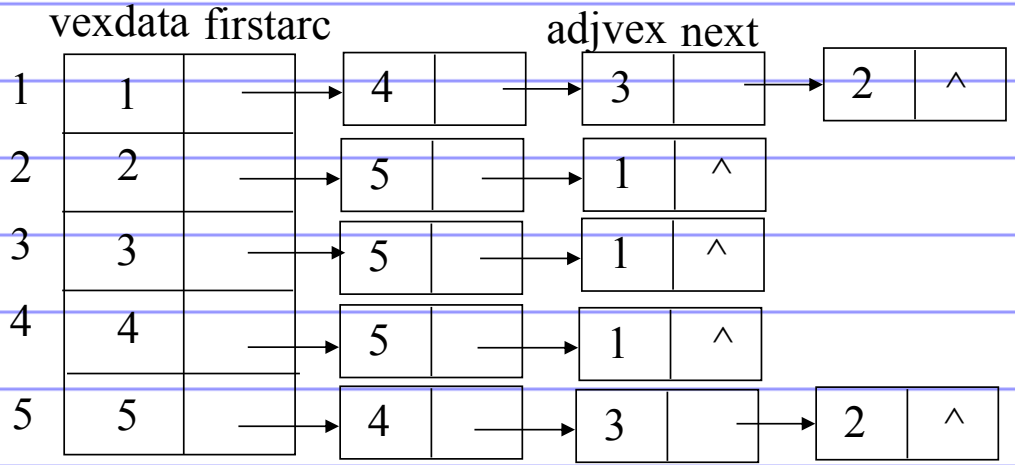


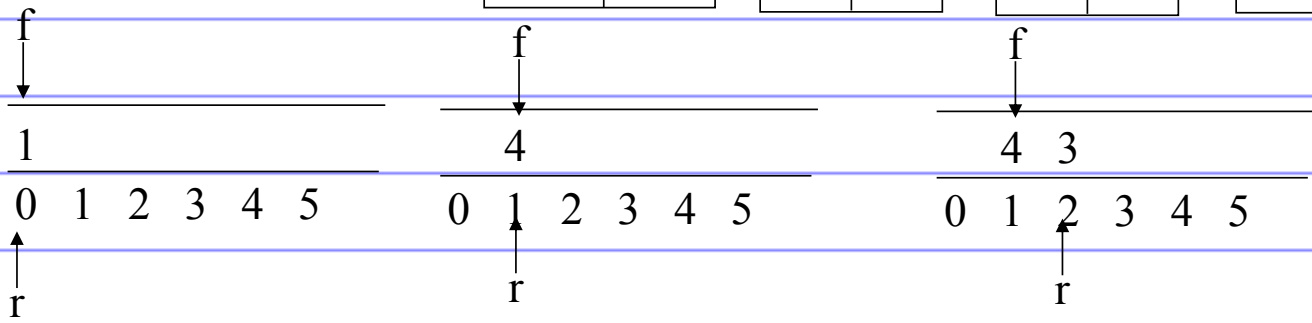
BFS


开始
访问V0,置标志
初始化队列
V0入队
队列空吗
Y
N
结束
队头V出队
求V邻接点w
w存在吗
N
Y
w访问过
Y
N
a
a
访问w,置标志
w入队
V下一邻接点⇒w

例





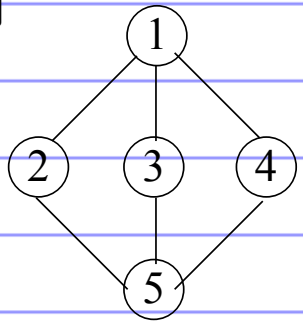


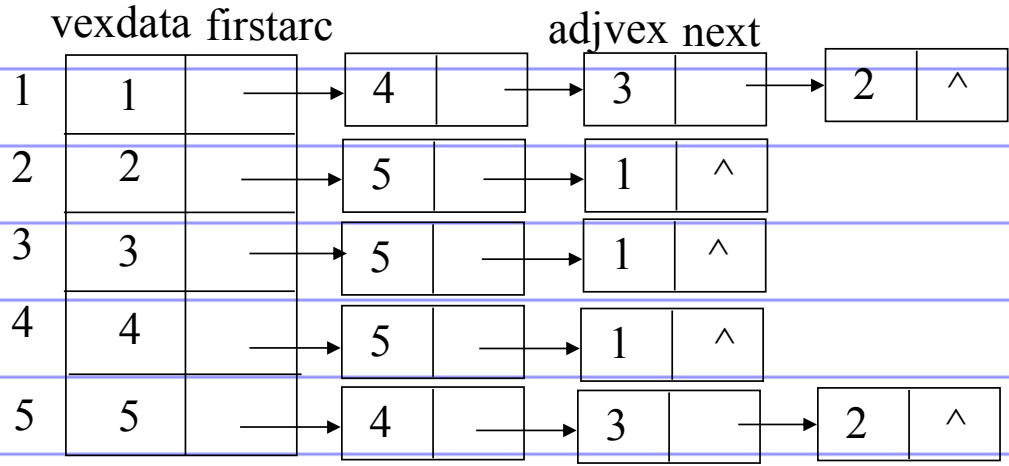
遍历序列：1

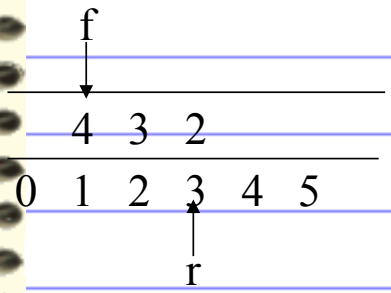
遍历序列：1 4

遍历序列：1 4 3

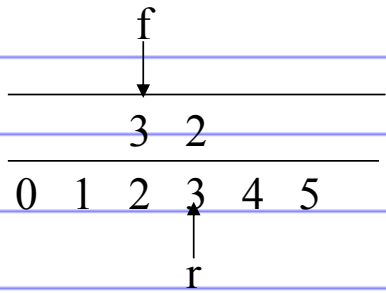
例



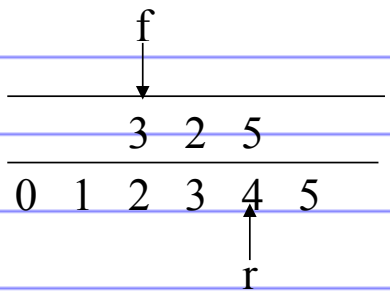




遍历序列：1 4 3 2

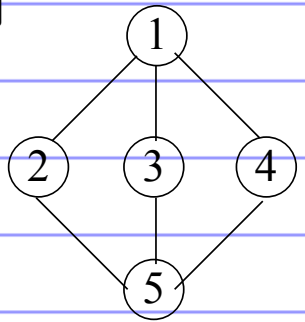


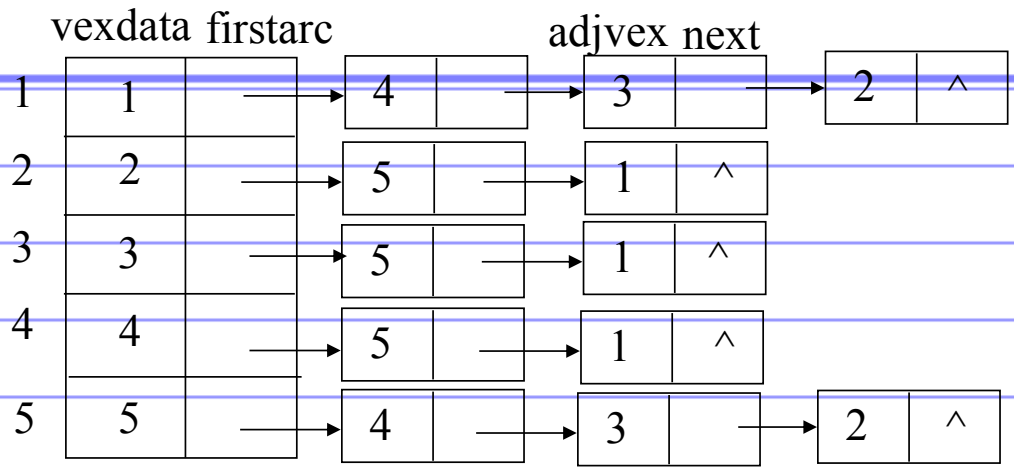
遍历序列：1 4 3 2

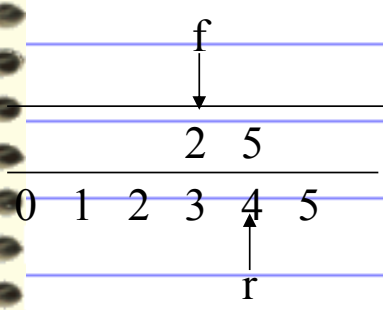


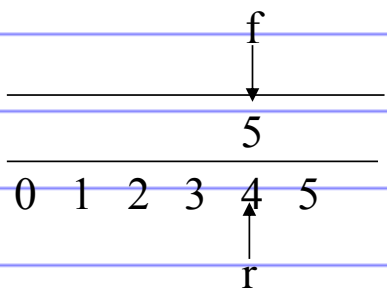
遍历序列：1 4 3 2 5

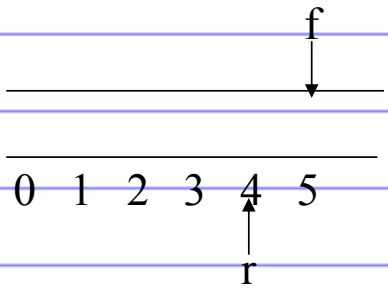
例











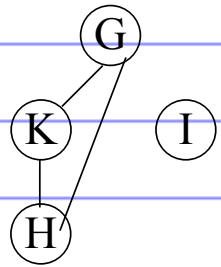
遍历序列：1 4 3 2 5

遍历序列：1 4 3 2 5

遍历序列：1 4 3 2 5

6.4 生成树

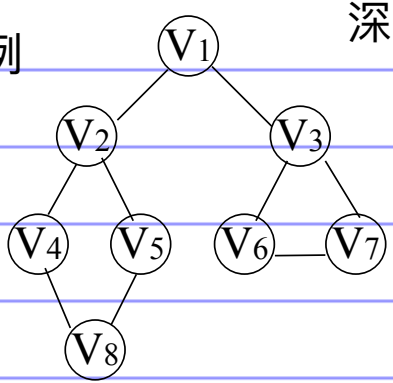
- 所有顶点均由边连接在一起，但不存在回路的图深度优先生成树与广度优先生成树
- 生成森林：非连通图每个连通分量的生成树一起组成非连通图的~
- 说明　一个图可以有许多棵不同的生成树
  - 所有生成树具有以下共同特点：
    - 生成树的顶点个数与图的顶点个数相同
    - 生成树是图的极小连通子图
    - 一个有n个顶点的连通图的生成树有n-1条边
    - 生成树中任意两个顶点间的路径是唯一的
    - 在生成树中再加一条边必然形成回路
  - 含n个顶点n-1条边的图不一定是生成树

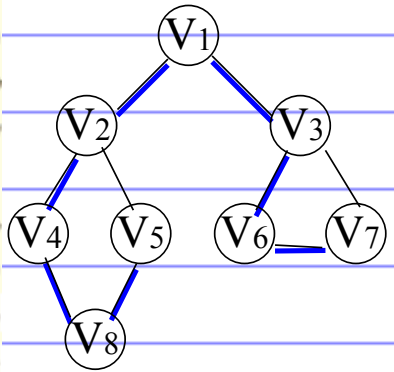


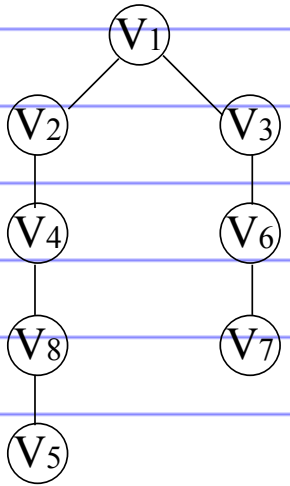
广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V3 ⇒V6 ⇒V7

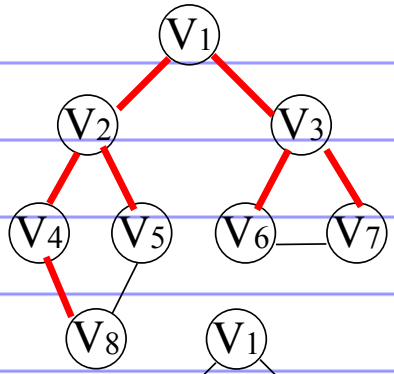
例

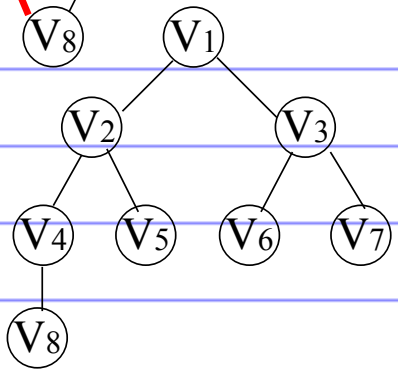






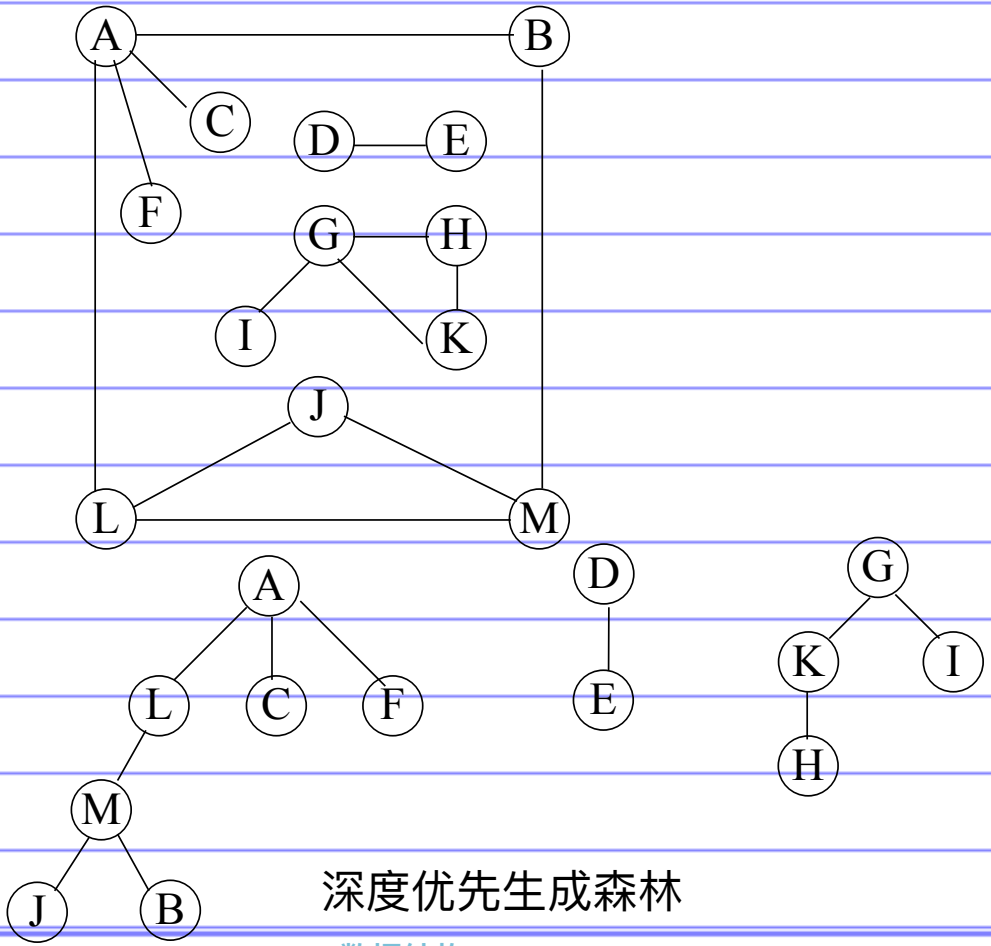
深度优先生成树





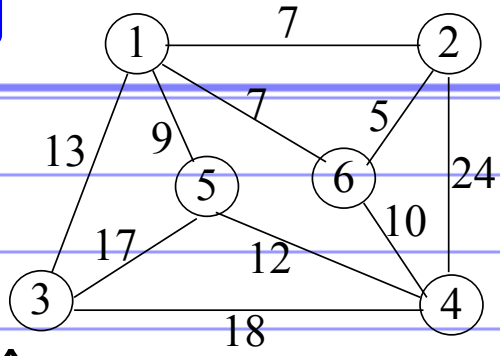
广度优先生成树

例



深度优先生成森林

最小生成树



问题提出

要在n个城市间建立通信联络网，

顶点——表示城市

权——城市间建立通信线路所需花费代价

希望找到一棵生成树，它的每条边上的权值之和（即建立该通信网所需花费的总代价）最小——最小代价生成树。

问题分析

n个城市间，最多可设置n(n-1)/2条线路

n个城市间建立通信网，只需n-1条线路

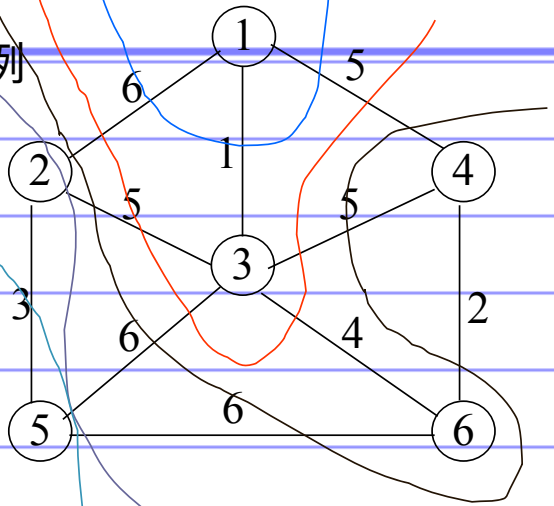
问题转化为：如何在可能的线路中选择n-1条，能把所有城市（顶点）均连起来，且总耗费（各边权值之和）最小。

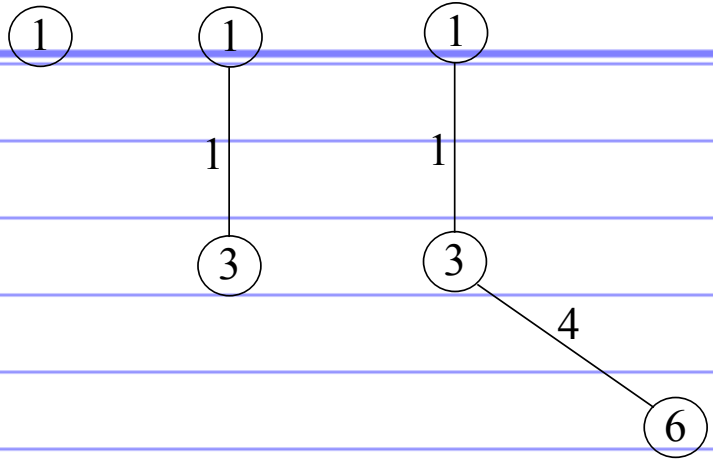
构造最小生成树方法

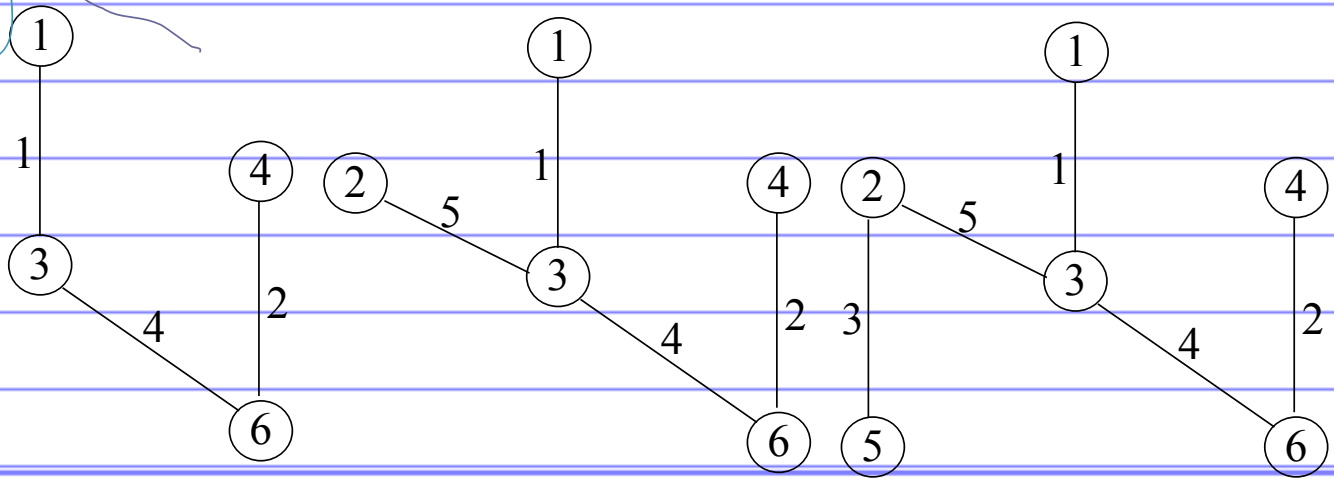
方法一：普里姆(Prim)算法

- 算法思想：设N=(V,{E})是连通网，TE是N上最小生成树中边的集合
  - 初始令U={u0},(u0∈V), TE=Φ
  - 在所有u∈U,v∈V-U的边(u,v)∈E中，找一条代价最小的边(u0,v0)
  - 将(u0,v0)并入集合TE，同时v0并入U
  - 重复上述操作直至U=V为止，则T=(V,{TE})为N的最小生成树
- 算法实现：图用邻接矩阵表示
- 算法描述
- 算法评价：$T(n)=O(n^2)$

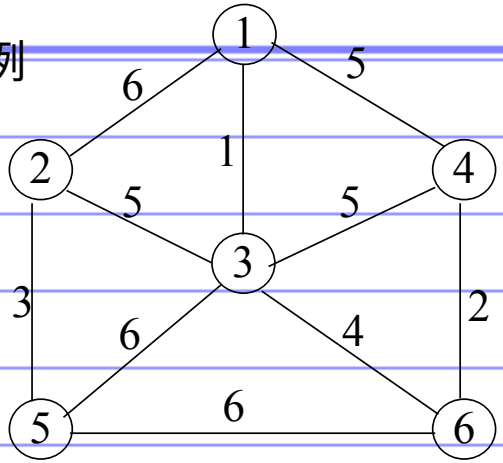
例


1
2
3
4
5
6
6
5
1
5
5
3
6
4
2
6

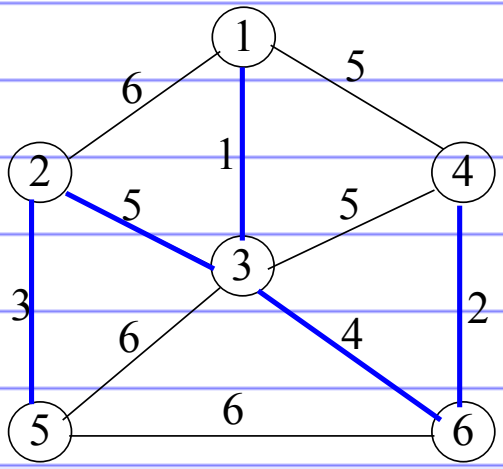

1
1
1
1
1
3
3
4
6

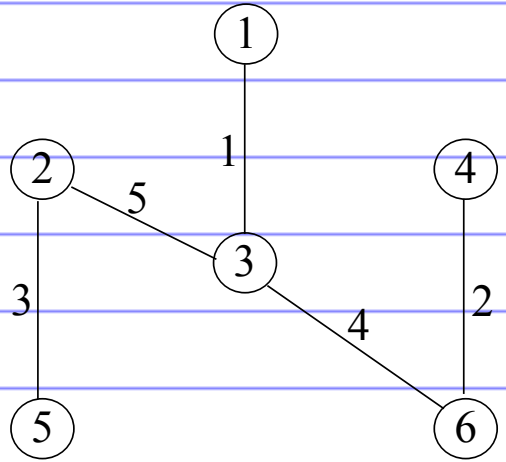

1
1
3
4
2
6
4
2
5
3
4
6
1
1
3
4
2
6
2
5
3
3
5
4
2
6
1
1
3
4
2
6

例



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

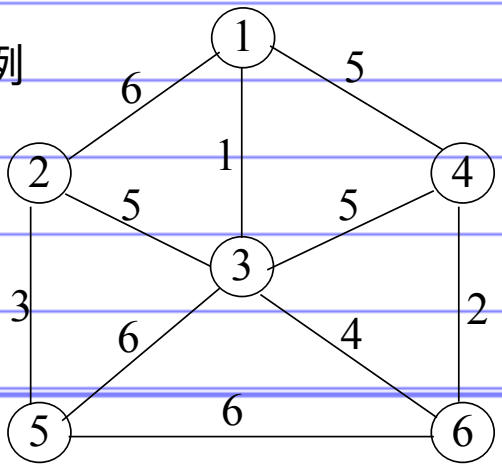


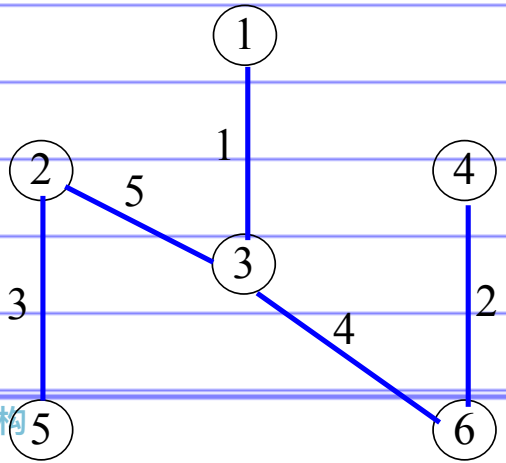


方法二：克鲁斯卡尔(Kruskal)算法

- 算法思想：设连通网N=(V,{E})，令最小生成树
  - 初始状态为只有n个顶点而无边的非连通图T=(V,{Φ})，每个顶点自成一个连通分量
  - 在E中选取代价最小的边，若该边依附的顶点落在T中不同的连通分量上，则将此边加入到T中；否则，舍去此边，选取下一条代价最小的边
  - 依此类推，直至T中所有顶点都在同一连通分量上为止

例





✧**算法实现：**

顶点结点：

```
typedef   struct
{    int data;    //顶点信息
     int  jihe;
}VEX;
```

边结点：

```
typedef   struct
{    int   vexh, vext;  //边依附的两顶点
     int  weight;       //边的权值
     int  flag;         //标志域
}EDGE;
```

0）用顶点数组和边数组存放顶点和边信息

1）初始时，令每个顶点的jihe互不相同；每个边的flag为0

2）选出权值最小且flag为0的边

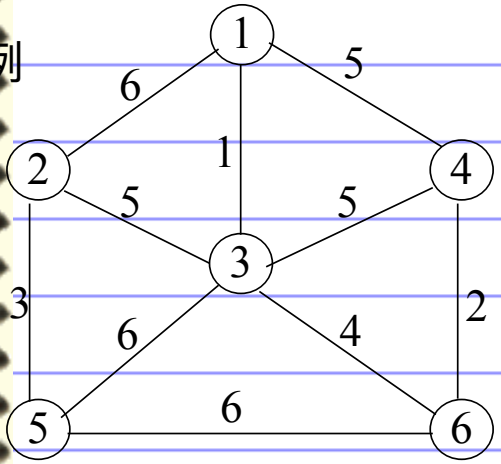
3）若该边依附的两个顶点的jihe值不同，即非连通，则令该边的flag=1，选中该边；再令该边依附的两顶点的jihe以及两集合中所有顶点的jihe 相同，若该边依附的两个顶点的jihe值相同，即连通，则令该边的flag=2，即舍去该边

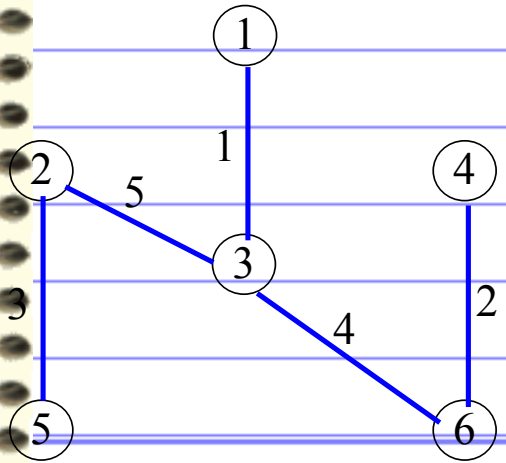
4）重复上述步骤，直到选出n-1条边为止

✧算法描述：

Ch6_30.txt

例





	data	jihe
1	1	1 2
2	2	2
3	3	3 1 2
4	4	4 1 2
5	5	5 2
6	6	6 4 1 2

	vexh	vext	weight	flag
0	1	2	6	0
1	1	3	1	0 1
2	1	4	5	0 2
3	2	3	5	0 1
4	2	5	3	0 1
5	3	4	5	0
6	3	5	6	0
7	3	6	4	0 1
8	4	6	2	0 1
9	5	6	6	0

Ch6_30.c

6.5 拓扑排序

- 问题提出：学生选修课程问题

  顶点——表示课程

  有向弧——表示先决条件，若课程i是课程j的先决条件，则图中有弧<i,j>

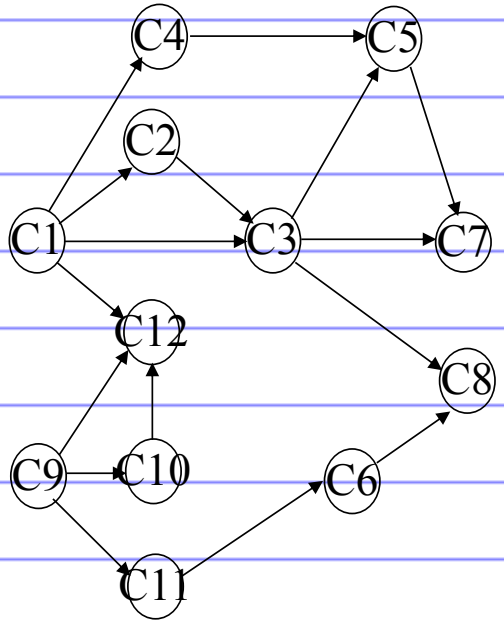
  学生应按怎样的顺序学习这些课程，才能无矛盾、顺利地完成学业——拓扑排序

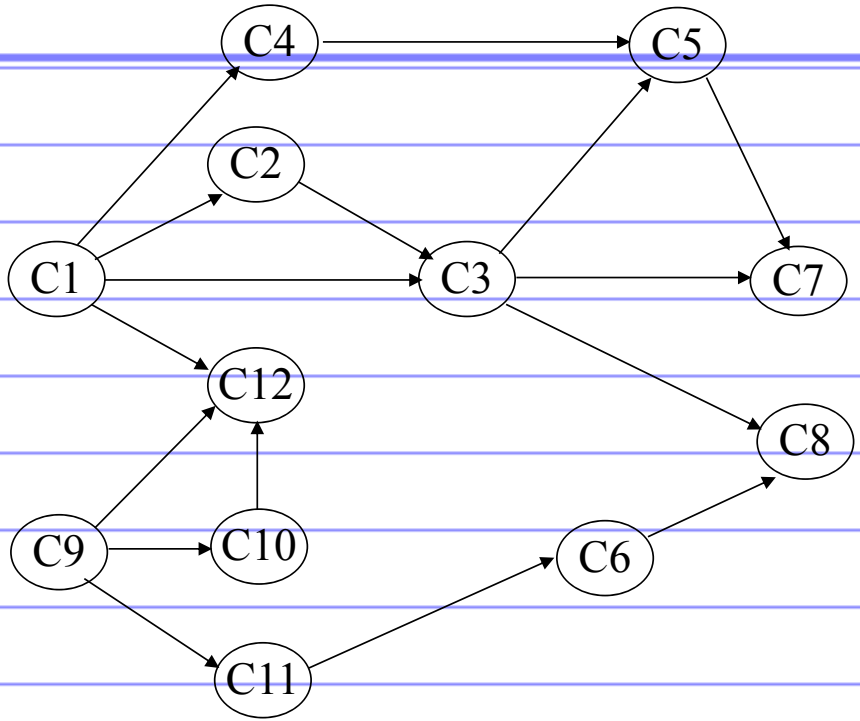
- 定义
  - AOV网——用顶点表示活动，用弧表示活动间优先关系的有向图称为顶点表示活动的网(Activity On Vertex network)，简称AOV网
  - 若<vi,vj>是图中有向边，则vi是vj的直接前驱；vj是vi的直接后继
  - AOV网中不允许有回路，即某项活动以自己为先决条件

- 拓扑排序——把AOV网络中各顶点按照它们相互之间的优先关系排列成一个线性序列的过程叫~
  - ✧检测AOV网中是否存在环方法：对有向图构造其顶点的拓扑有序序列，若网中所有顶点都在它的拓扑有序序列中，则该AOV网必定不存在环
- 拓扑排序的方法
  - ✧在有向图中选一个没有前驱的顶点且输出之
  - ✧从图中删除该顶点和所有以它为尾的弧
  - ✧重复上述两步，直至全部顶点均已输出；或者当图中不存在无前驱的顶点为止

例

课程代号	课程名称	先修棵
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	语言的设计和分析	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

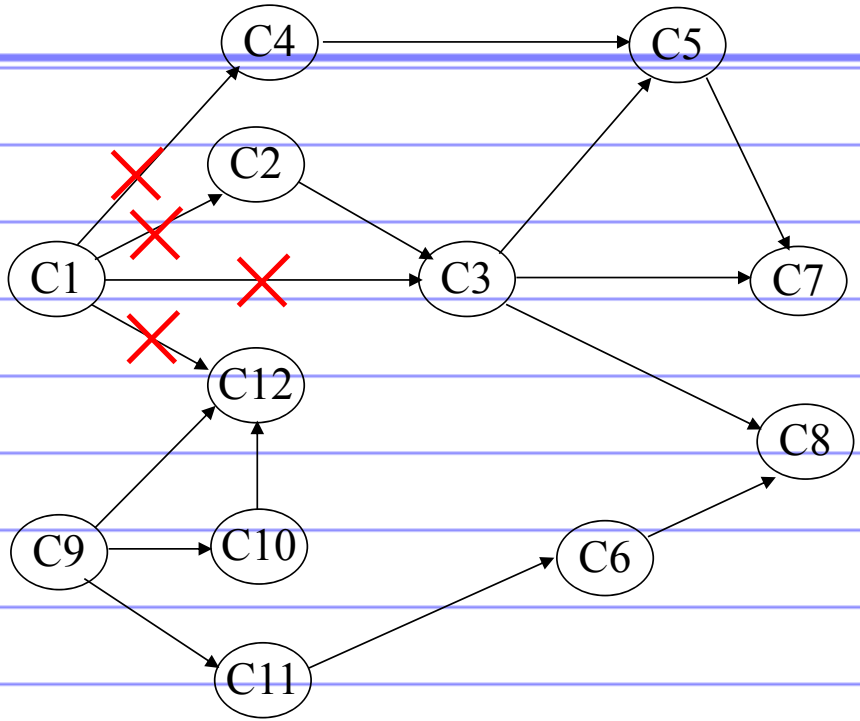


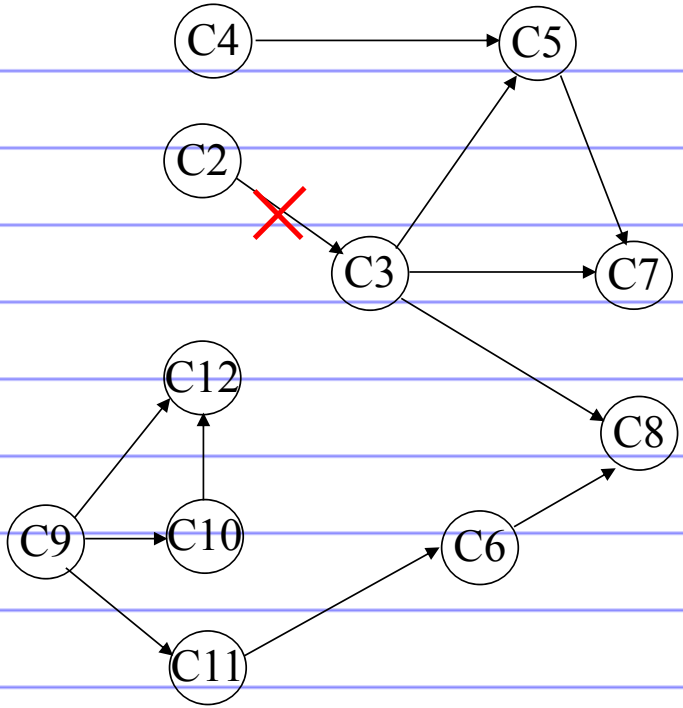


拓扑序列：C1--C2--C3--C4--C5--C7--C9--C10--C11--C6--C12--C8

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

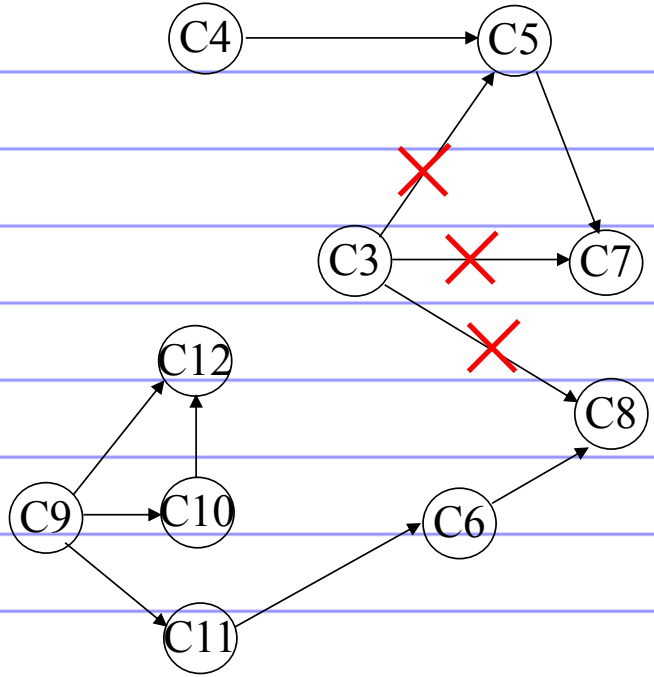
一个AOV网的拓扑序列不是唯一的


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



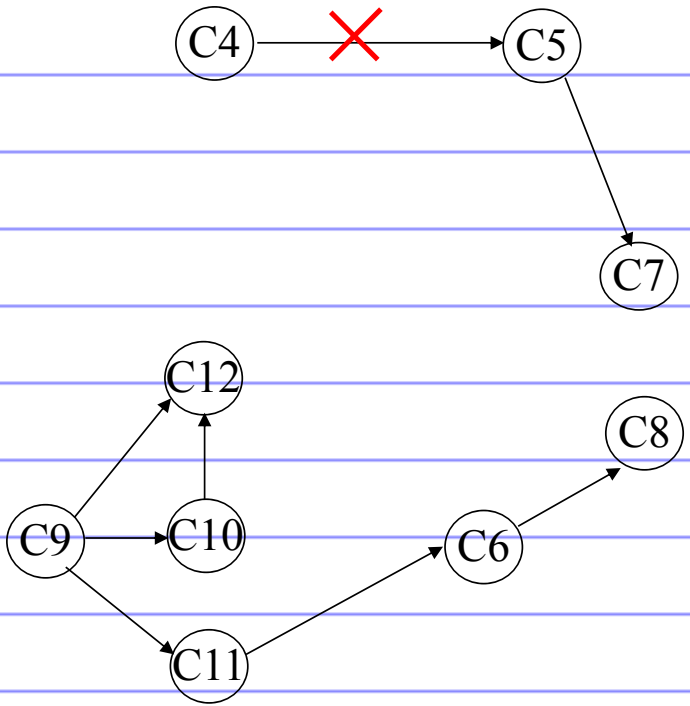
拓扑序列：C1

(1)



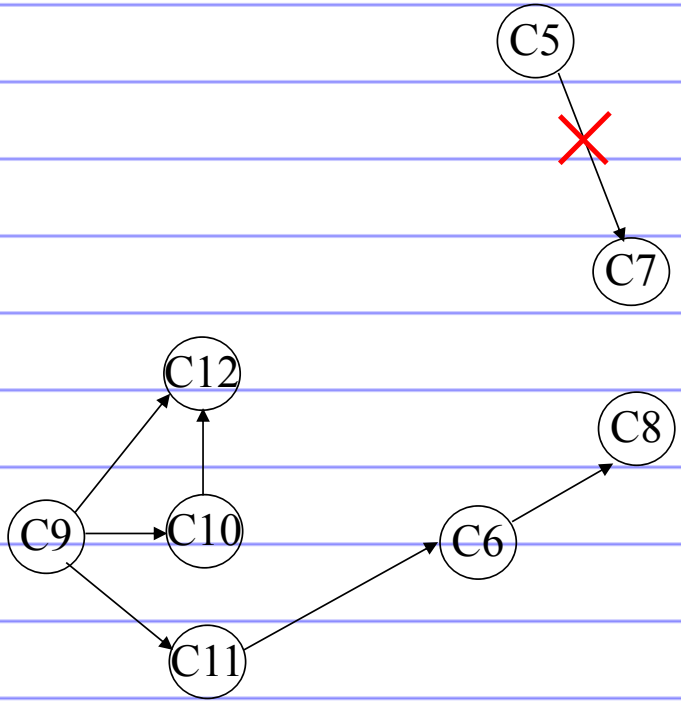
拓扑序列：C1--C2

(2)



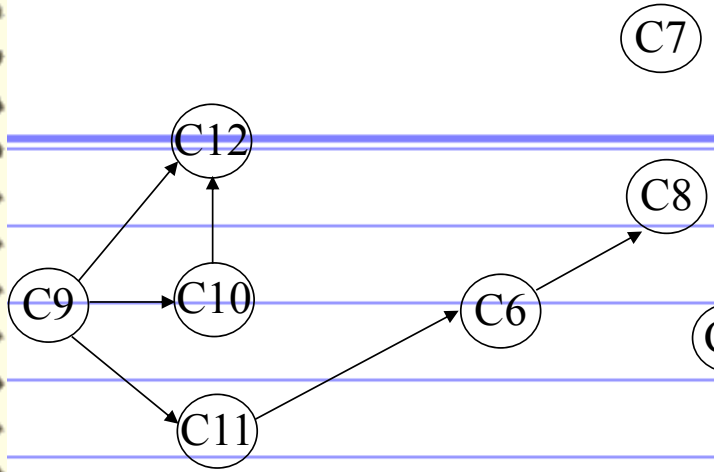
拓扑序列：C1--C2--C3

(3)



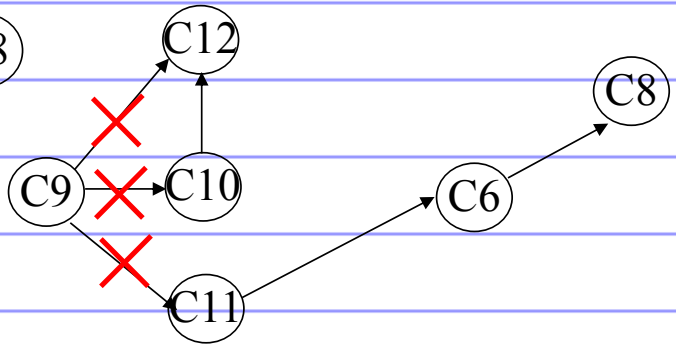
拓扑序列：C1--C2--C3--C4

(4)



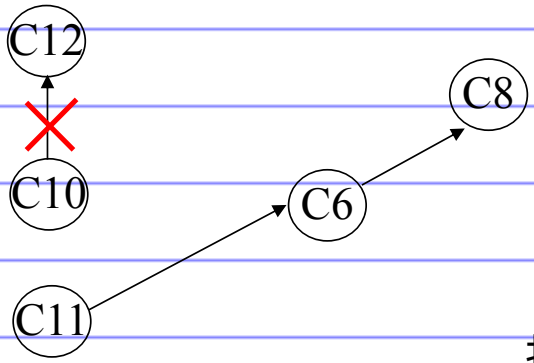
拓扑序列：C1--C2--C3--C4--C5

(5)

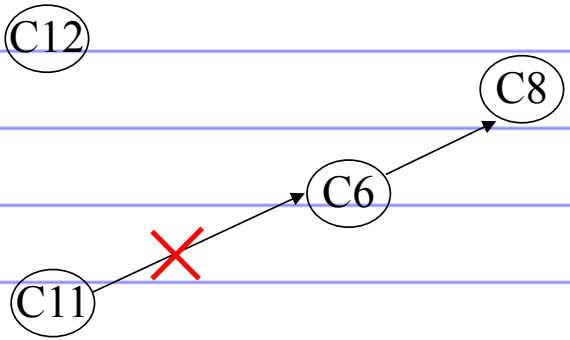


拓扑序列：C1--C2--C3--C4--C5--C7

(6)



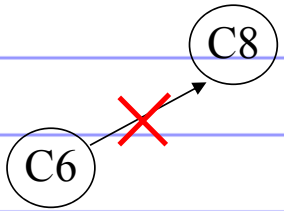
拓扑序列：C1--C2--C3--C4--C5--C7--C9



拓扑序列：C1--C2--C3--C4--C5--C7--C9

--C10

(8)



C12

C12

C8

(9)

(10)

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

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--C8

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

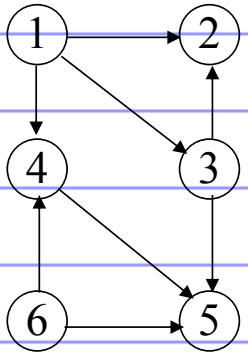
(12)

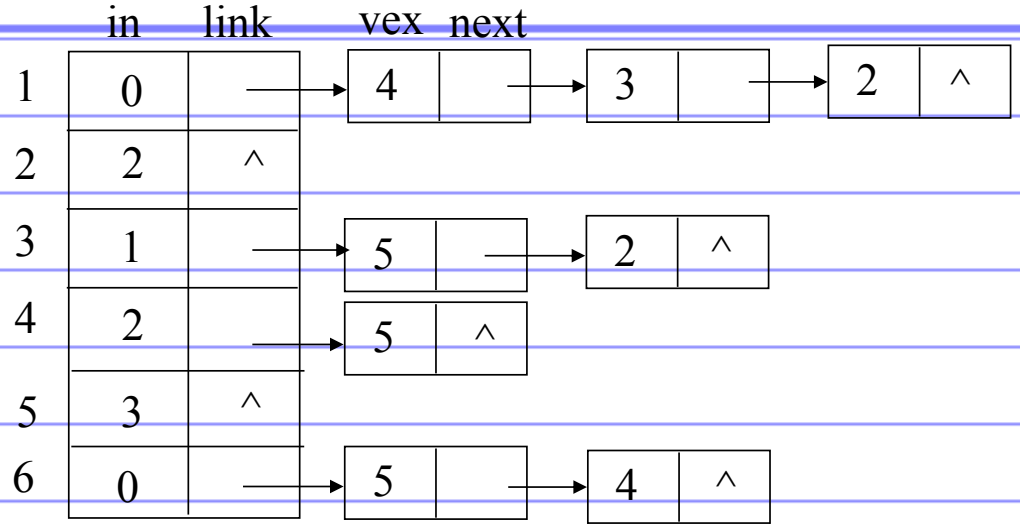
■ 算法实现

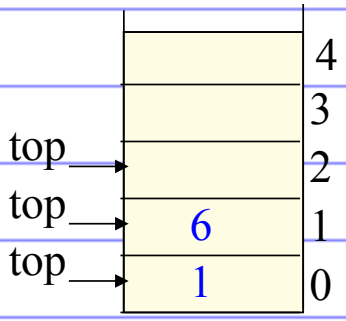
- ✧ 以邻接表作存储结构
- ✧ 把邻接表中所有入度为0的顶点进栈
- ✧ 栈非空时，输出栈顶元素V_j并退栈；在邻接表中查找V_j的直接后继V_k，把V_k的入度减1；若V_k的入度为0则进栈
- ✧ 重复上述操作直至栈空为止。若栈空时输出的顶点个数不是n，则有向图有环；否则，拓扑排序完毕

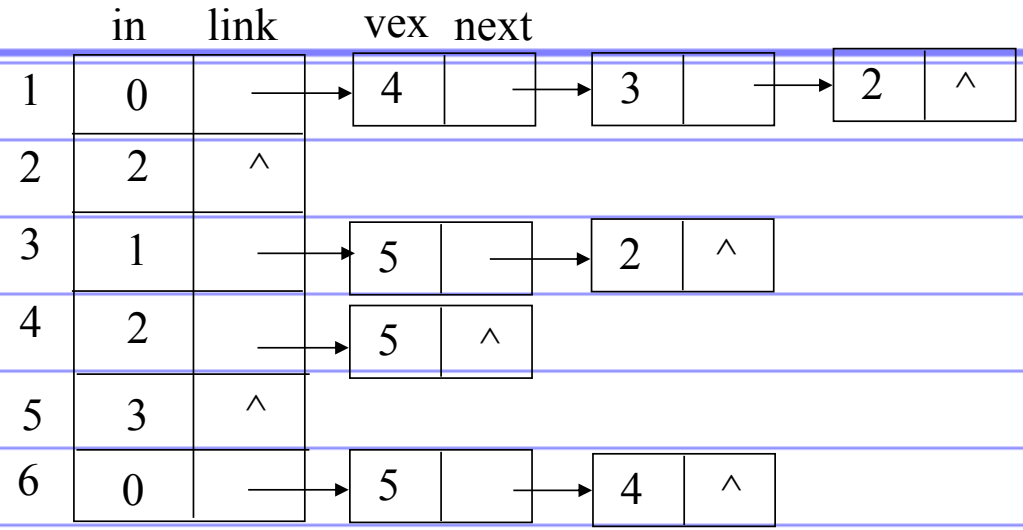
■ 算法描述

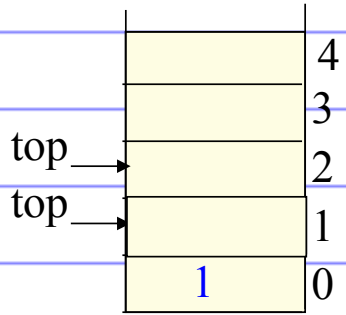
例


1
2
4
3
6
5

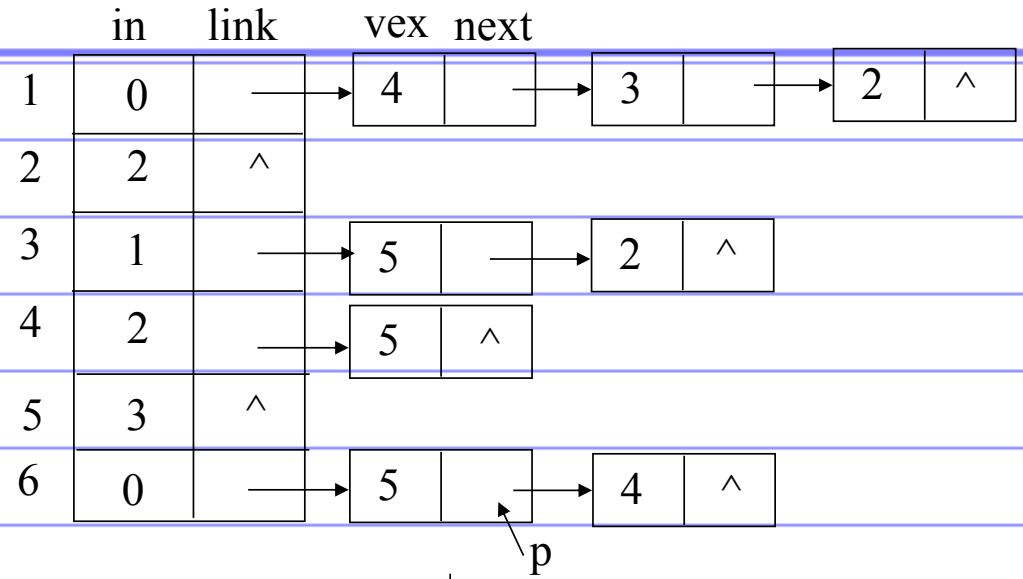

in
link
vex
next
1
0
4
3
2
^
2
2
^
3
1
5
2
^
4
2
5
^
5
3
^
6
0
5
4
^

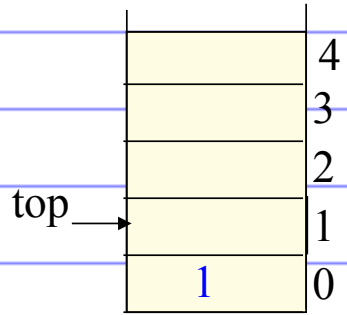

top
top
top
6
1
4
3
2
1
0



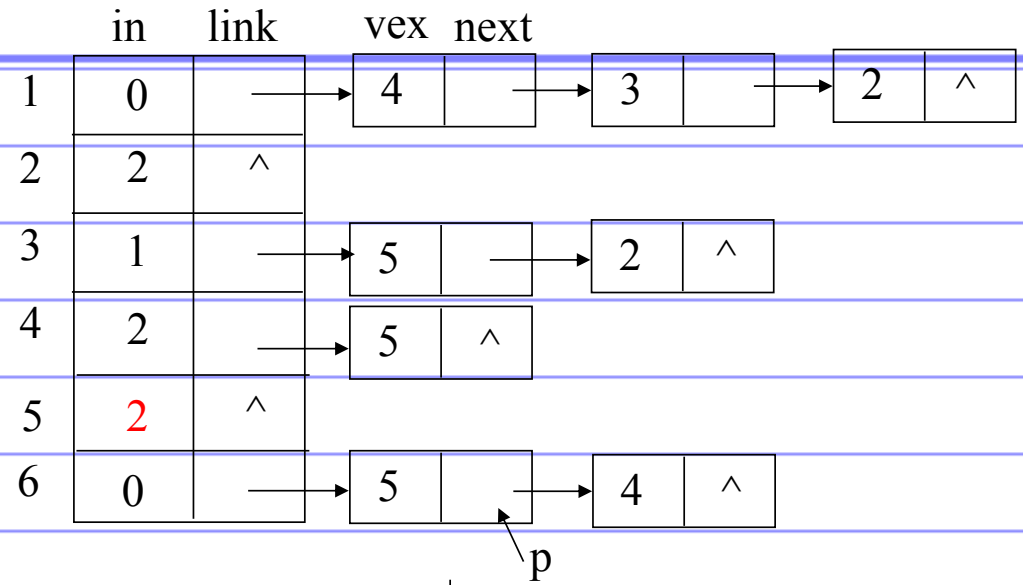


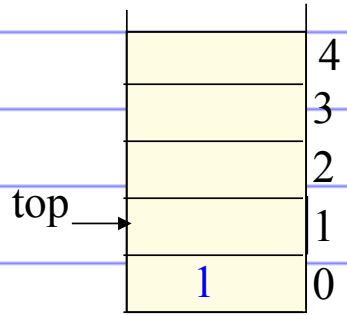
输出序列：6



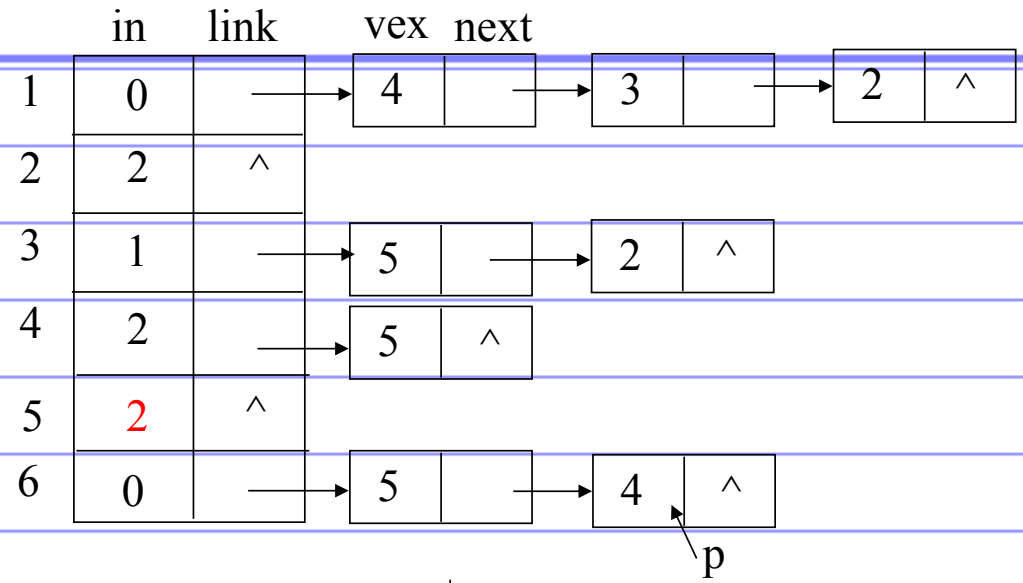


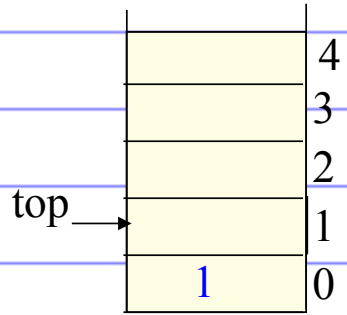
输出序列：6



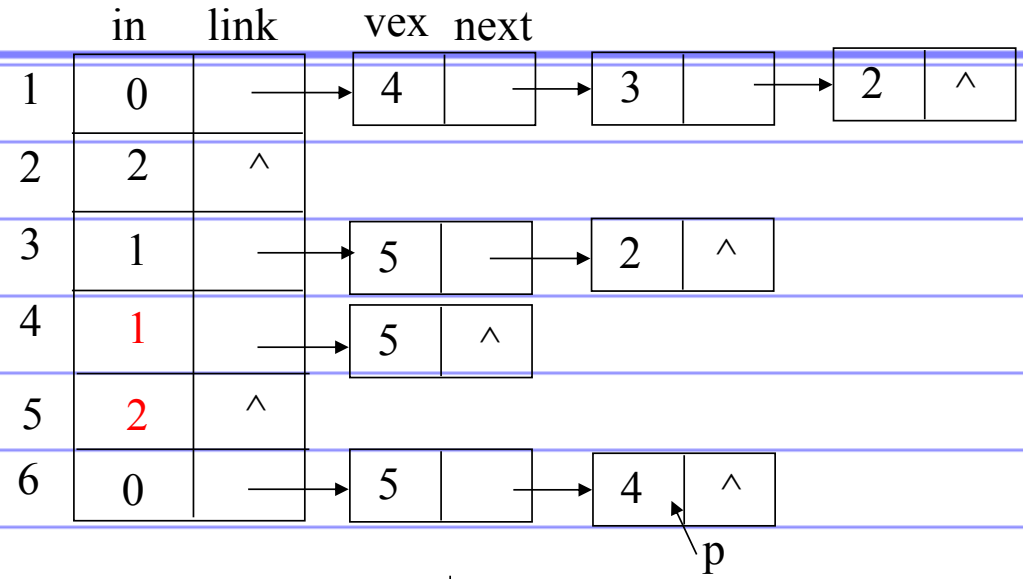


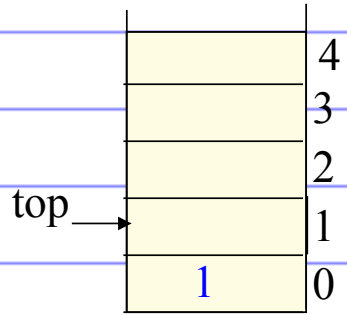
输出序列：6



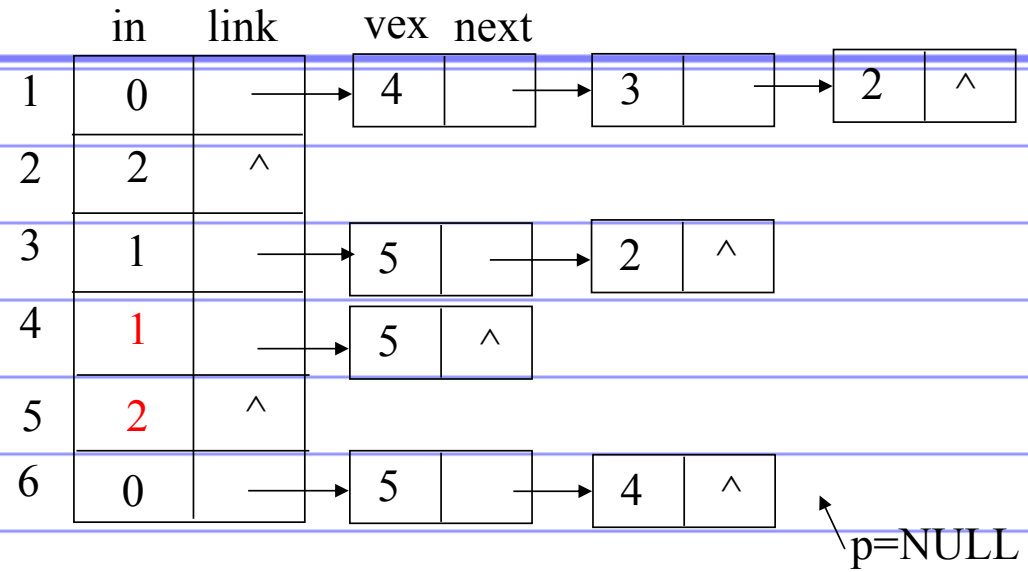


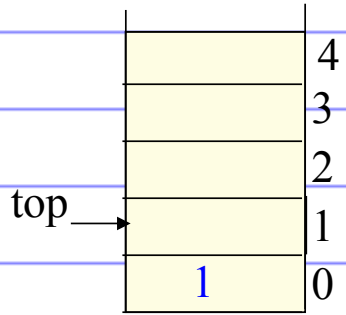
输出序列：6



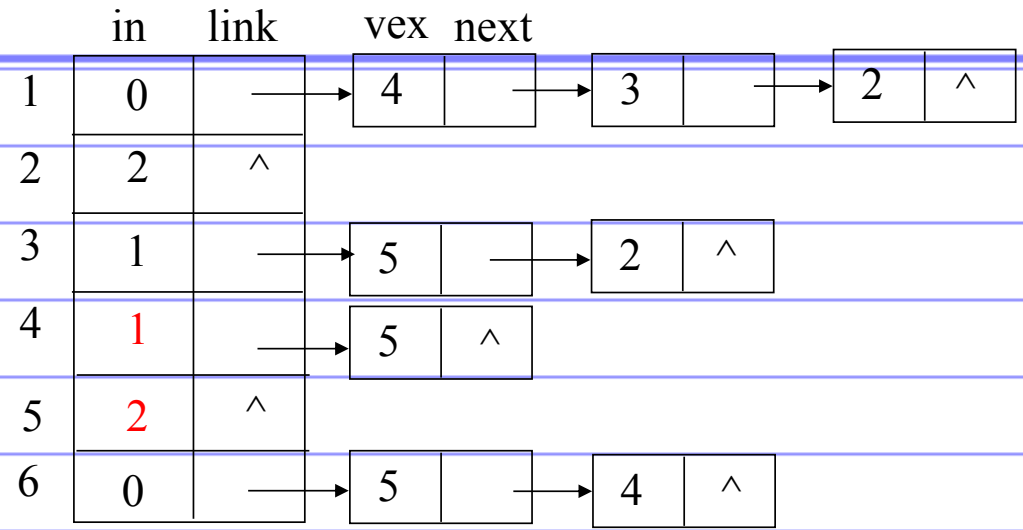


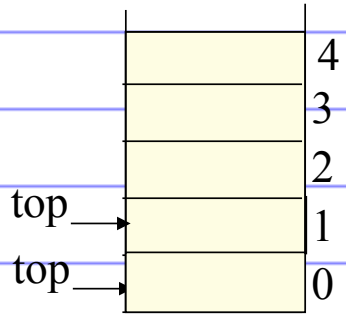
输出序列：6



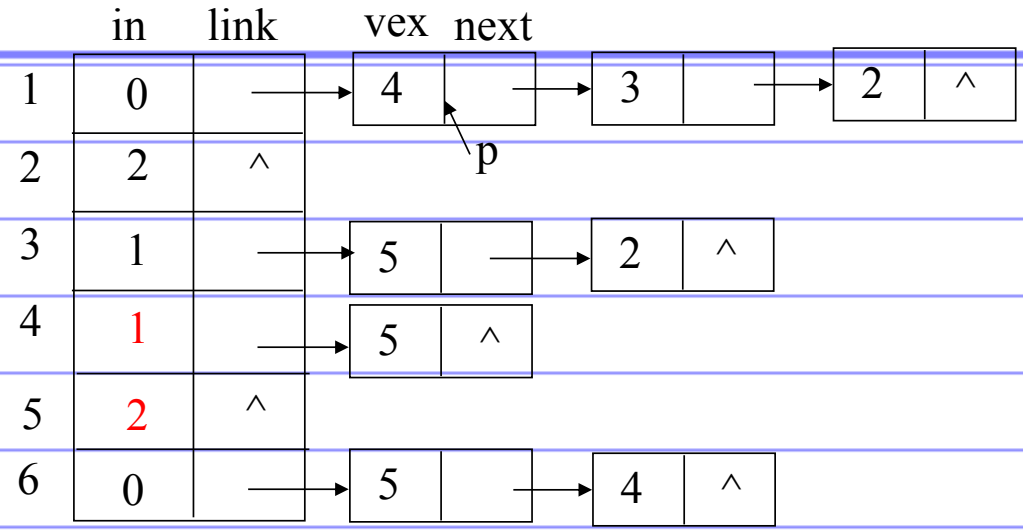


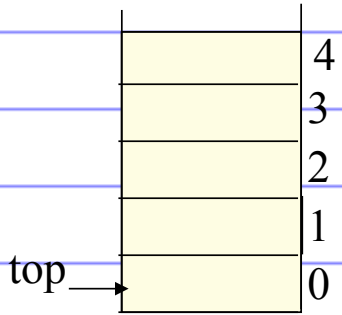
输出序列：6



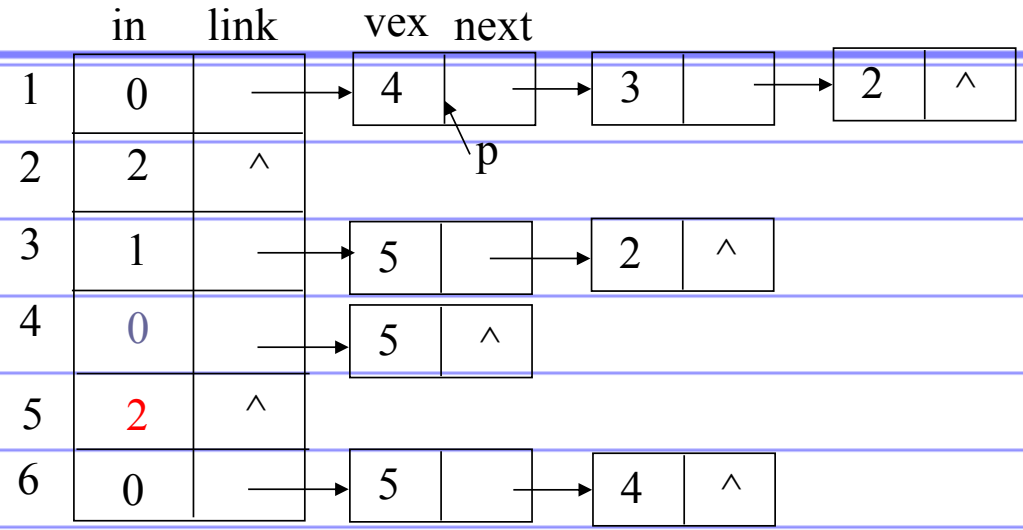


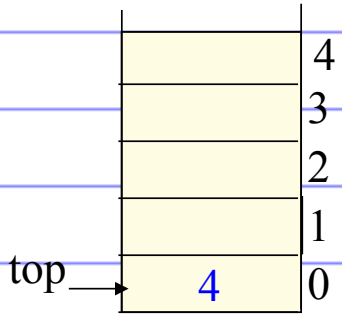
输出序列：6 1



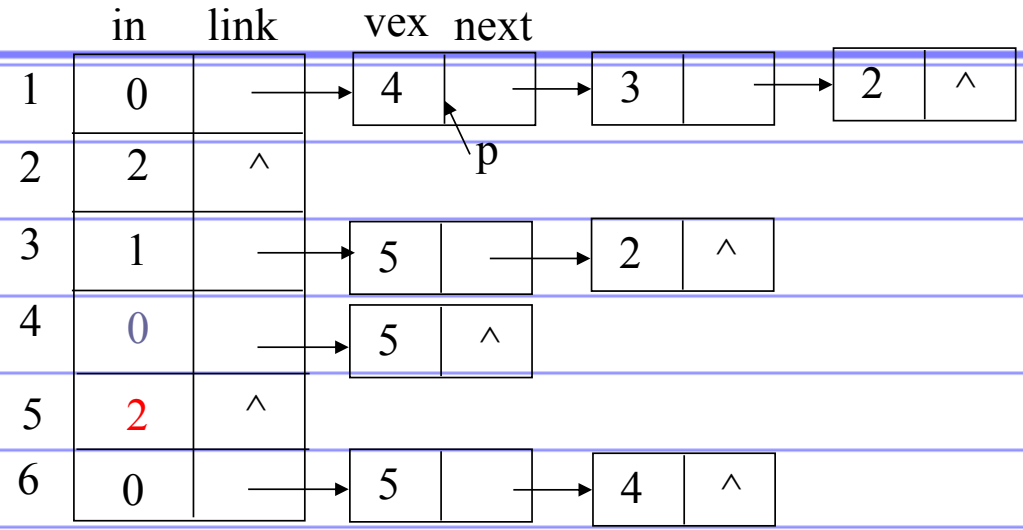


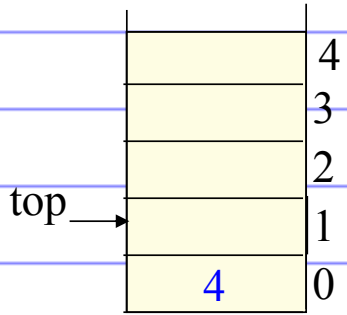
输出序列：6 1



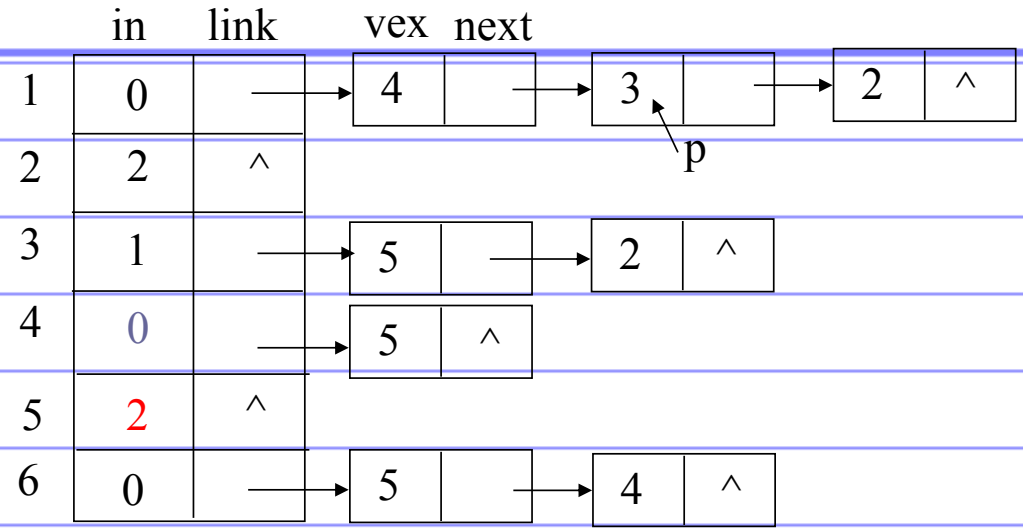


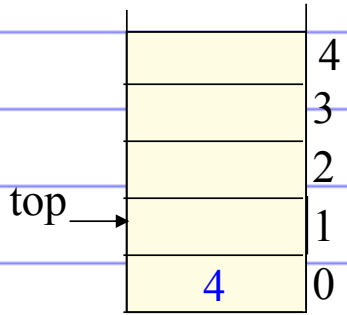
输出序列：6 1



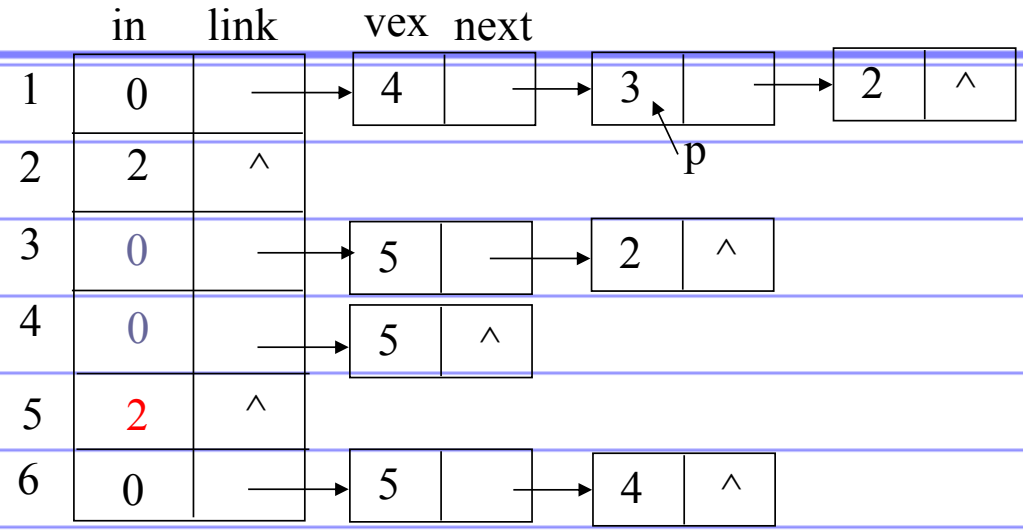


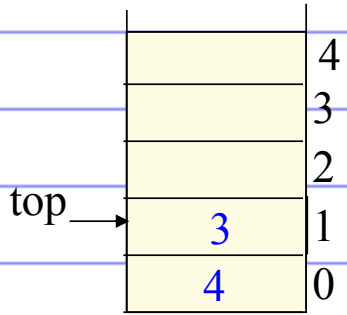
输出序列：6 1



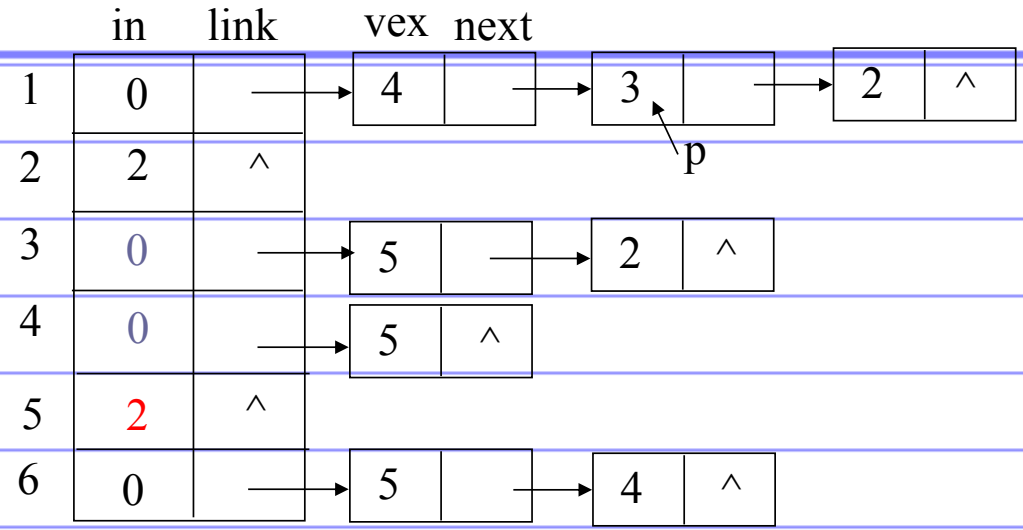


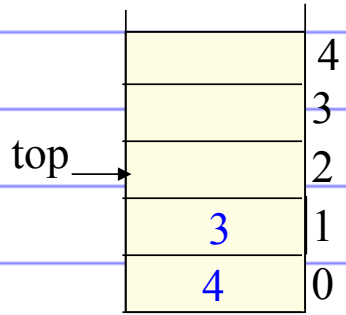
输出序列：6 1



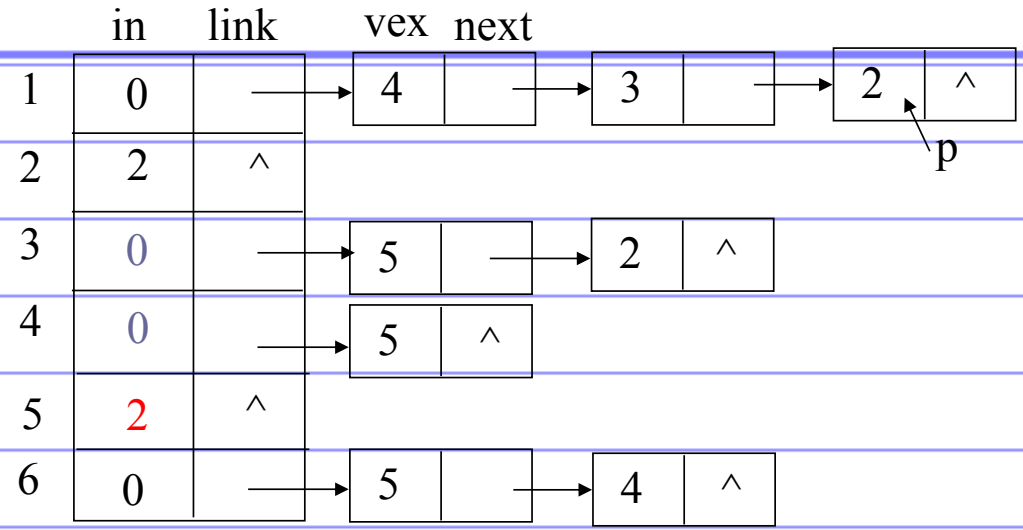


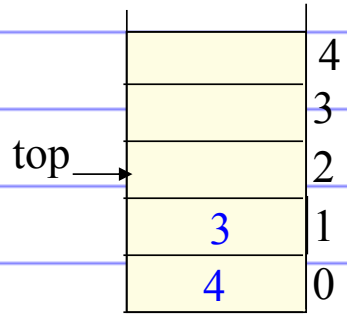
输出序列：6 1



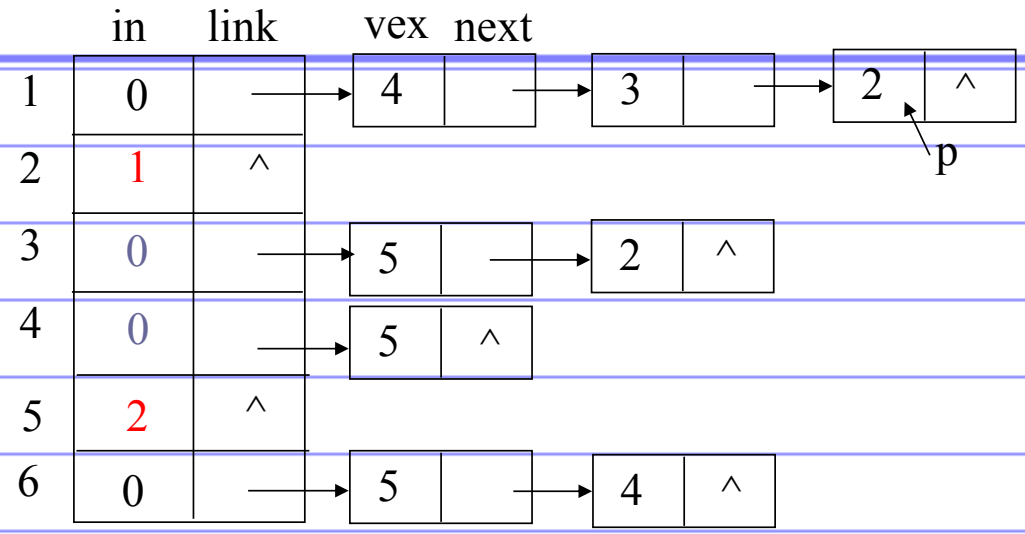


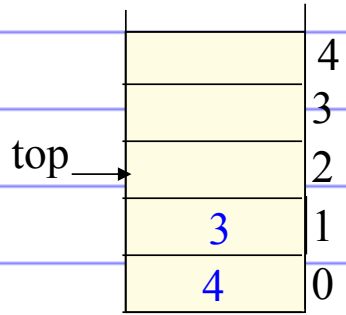
输出序列：6 1



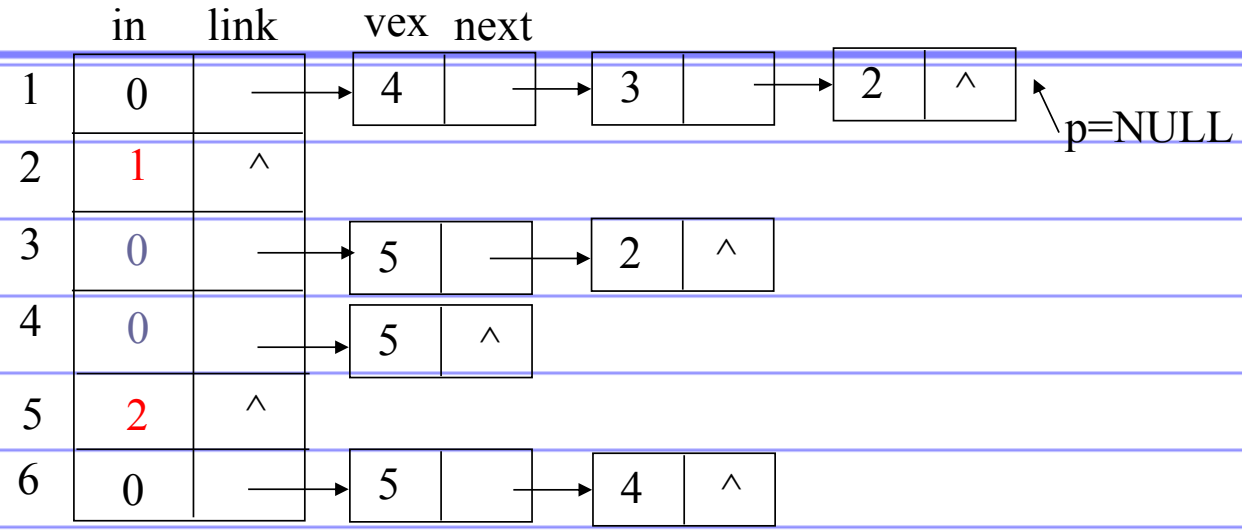


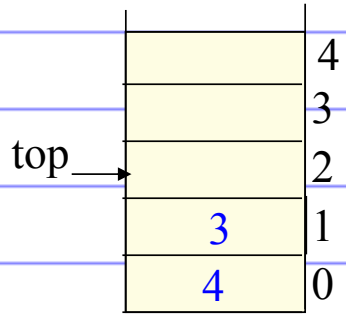
输出序列：6 1



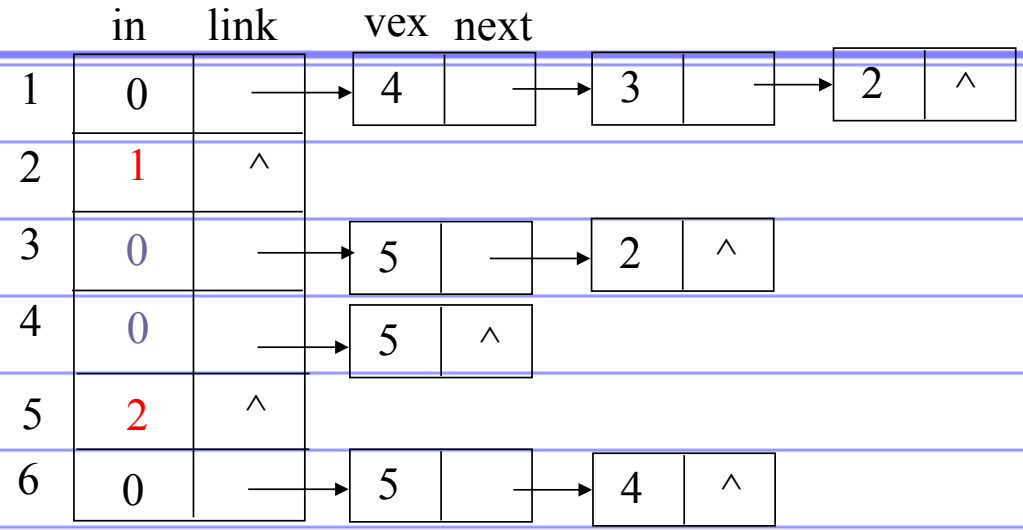


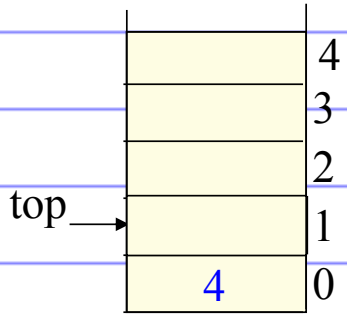
输出序列：6 1



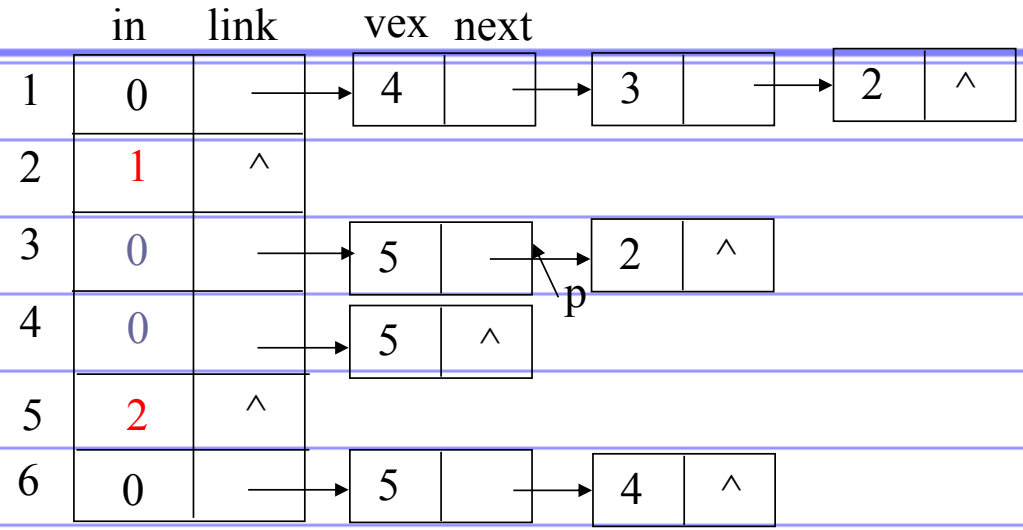


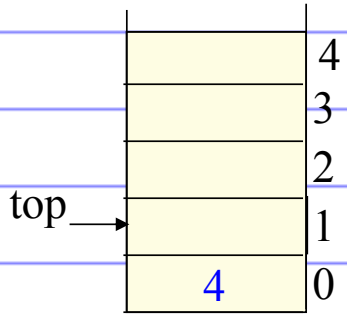
输出序列：6 1



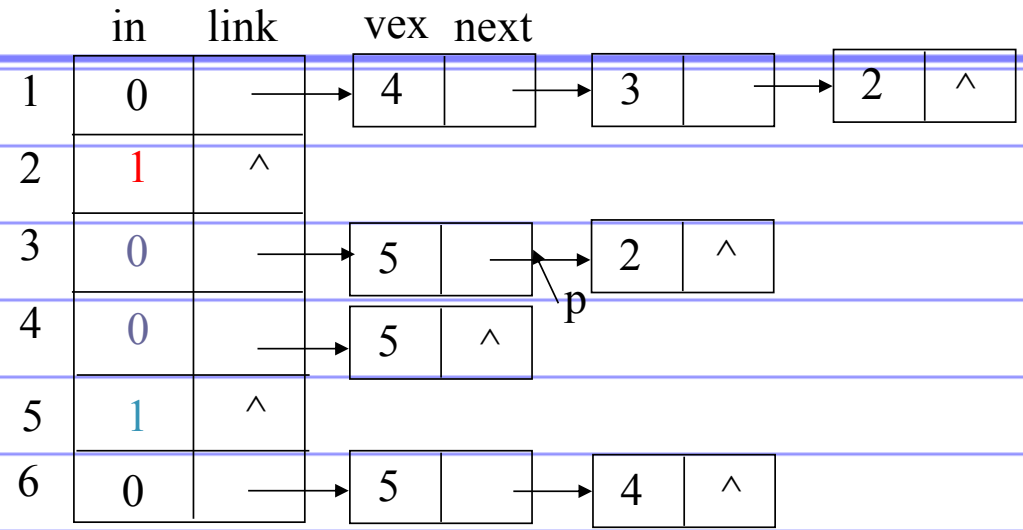


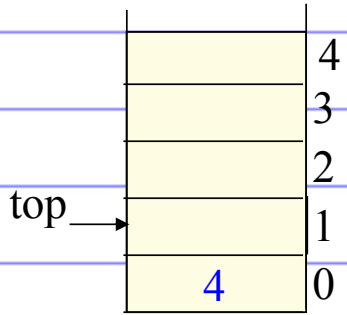
输出序列：6 1 3



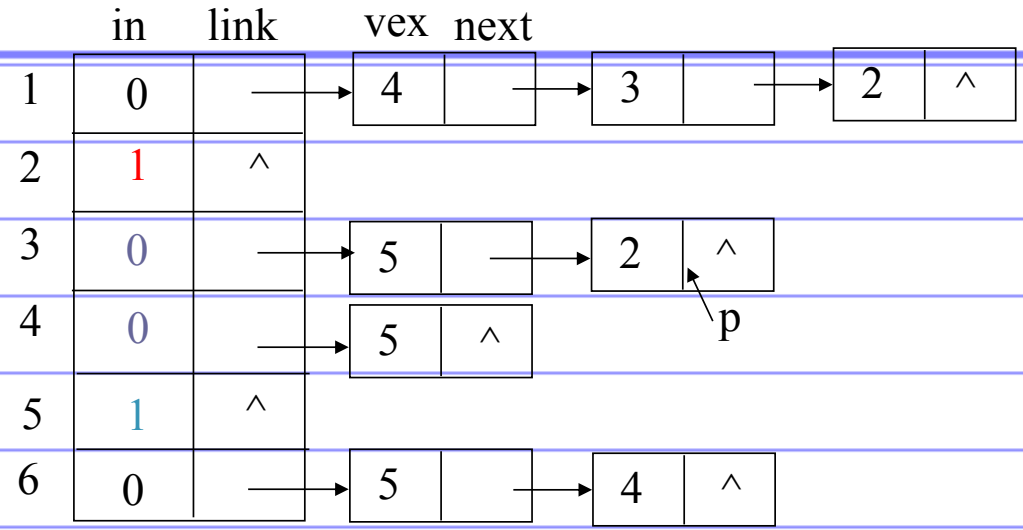


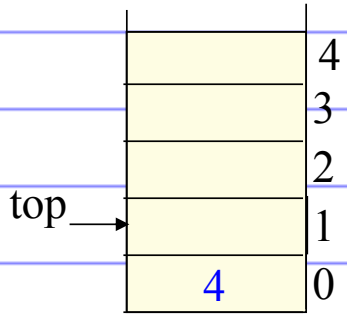
输出序列：6 1 3



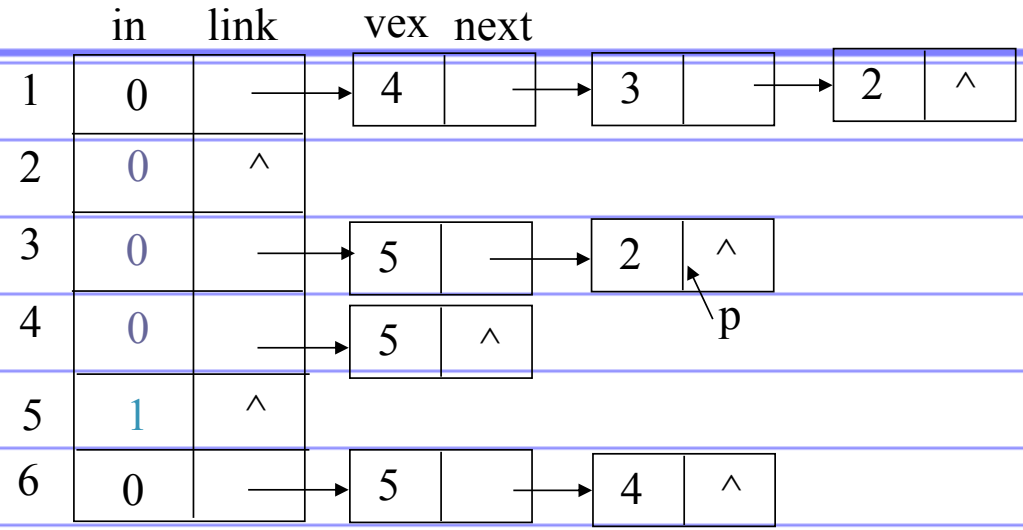


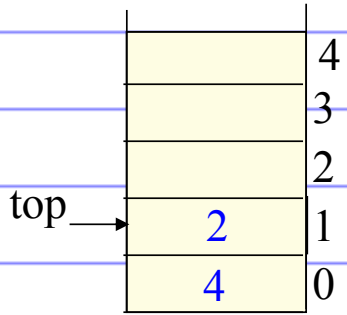
输出序列：6 1 3



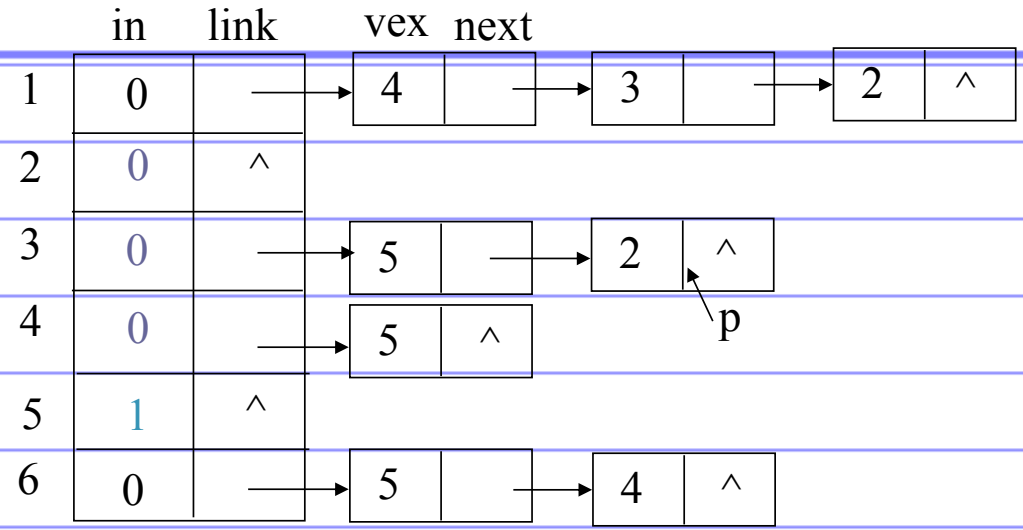


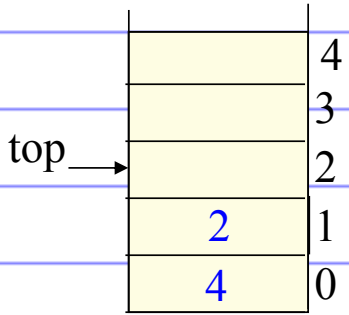
输出序列：6 1 3



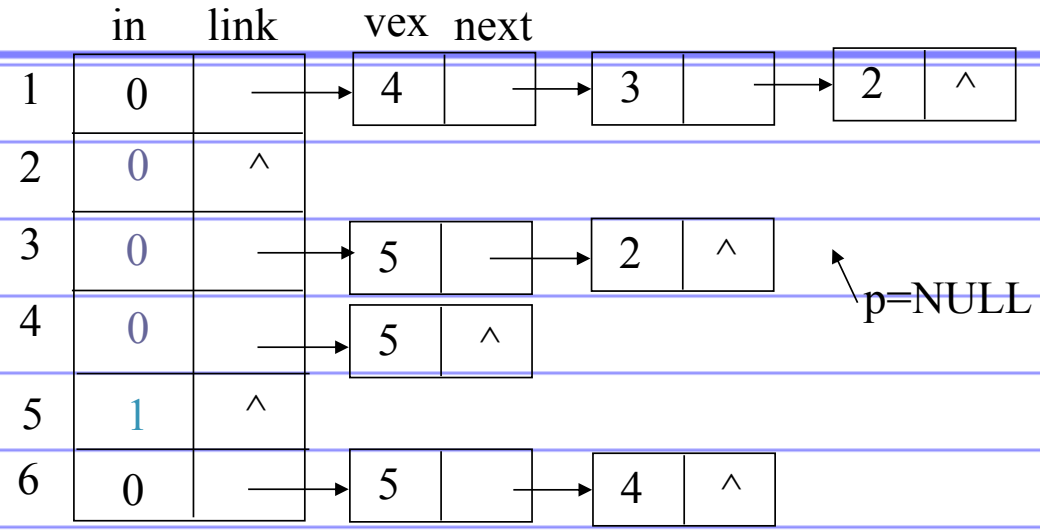


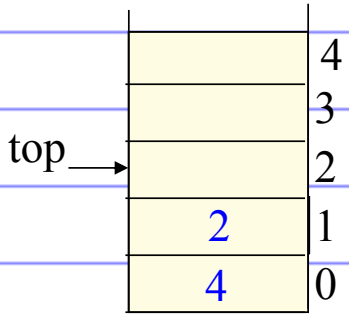
输出序列：6 1 3



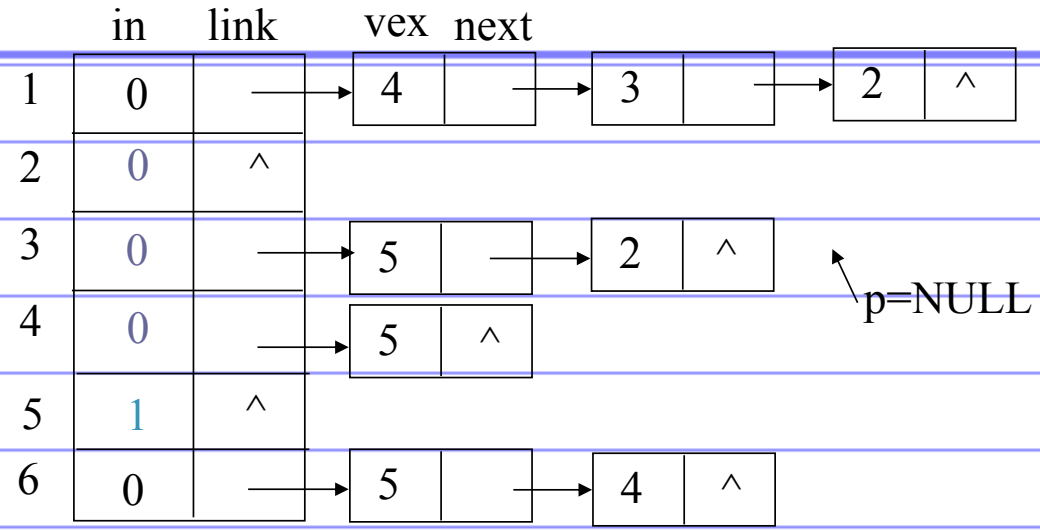


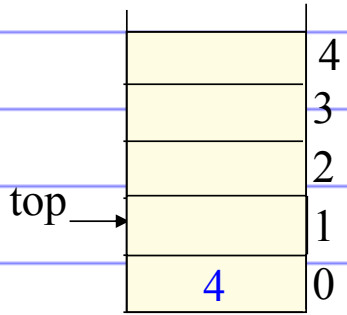
输出序列：6 1 3



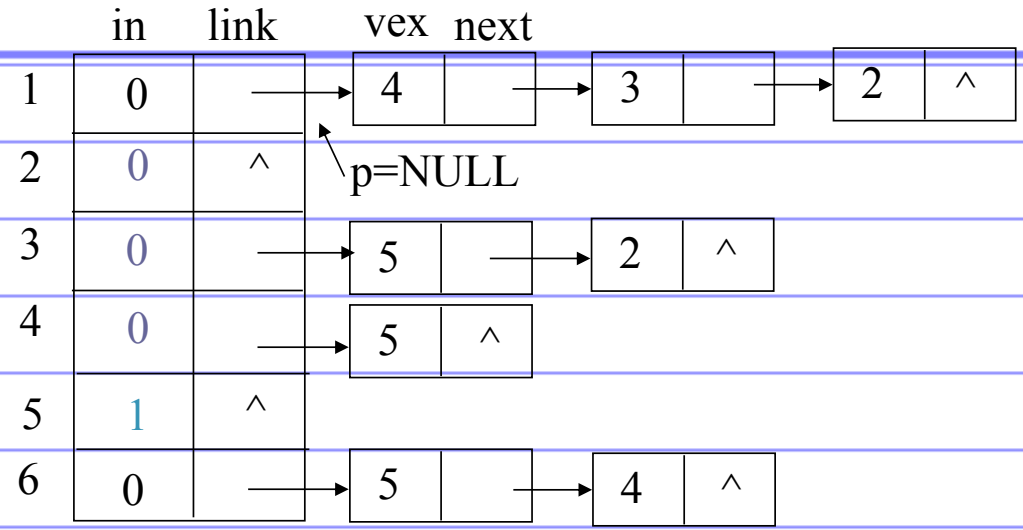


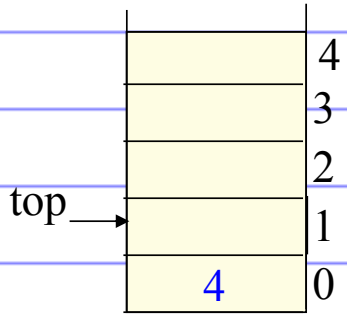
输出序列：6 1 3



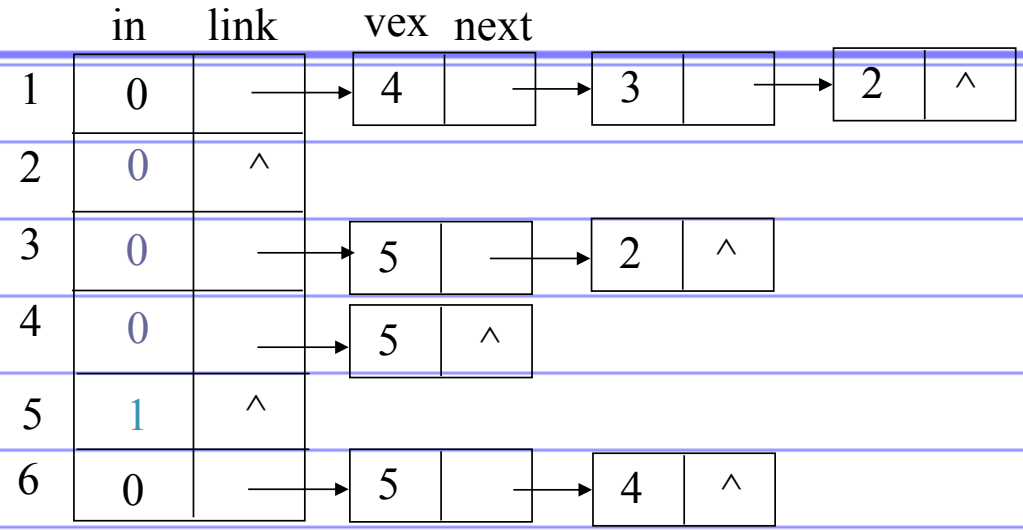


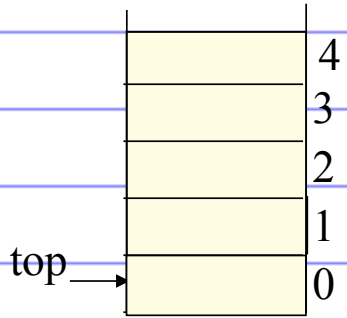
输出序列：6 1 3 2



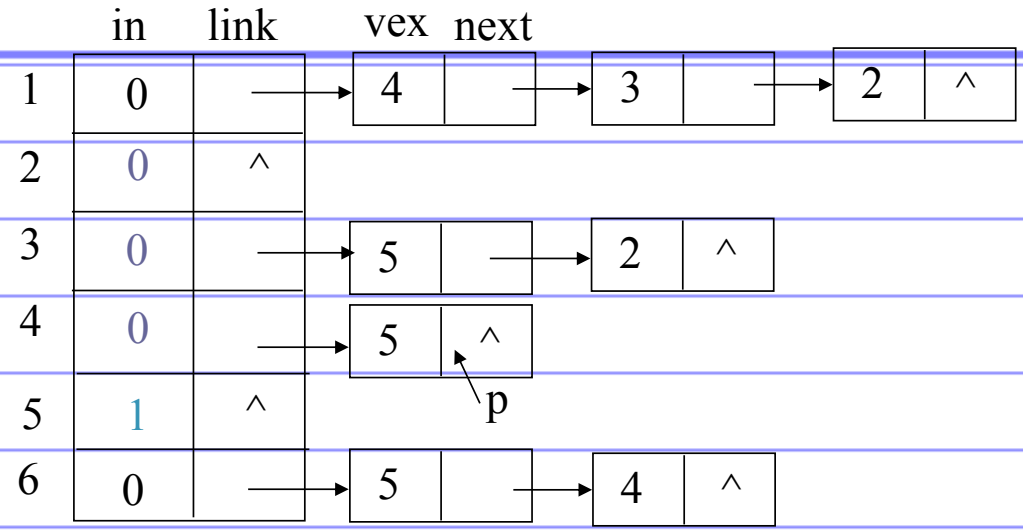


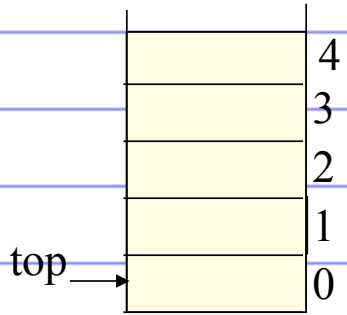
输出序列：6 1 3 2



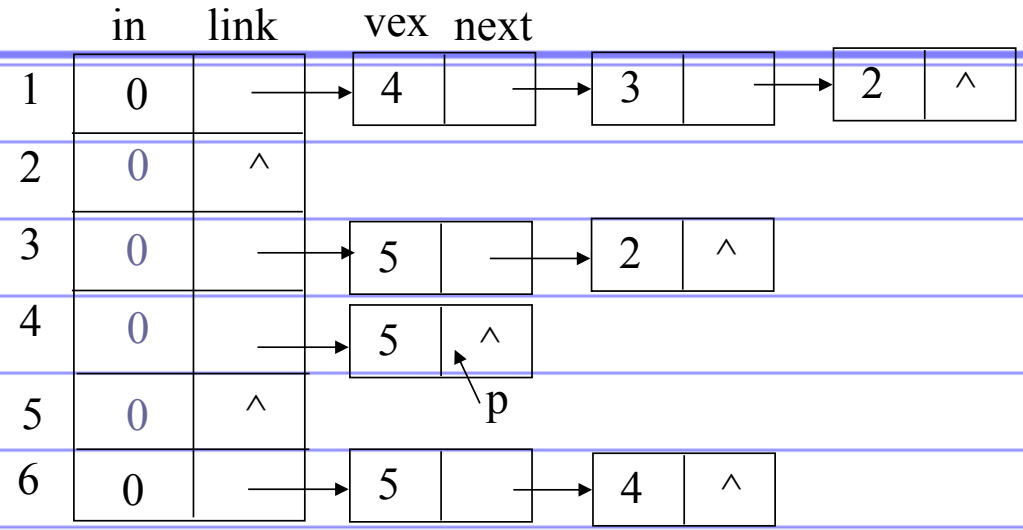


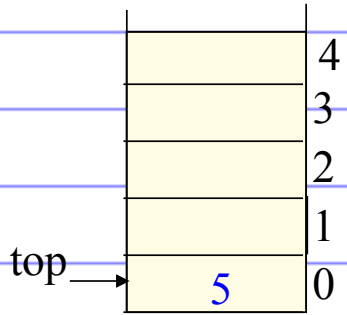
输出序列：6 1 3 2 4



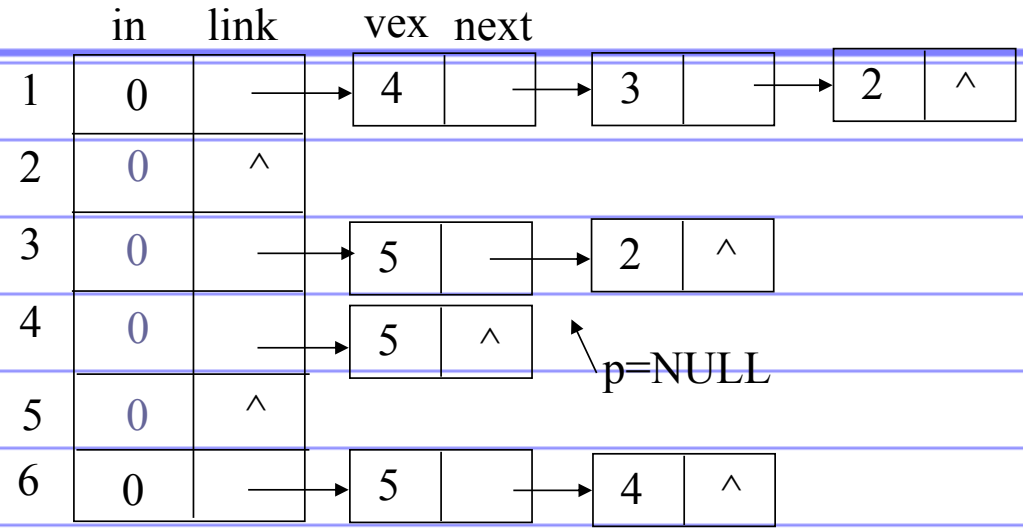


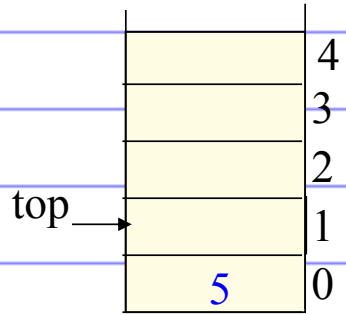
输出序列：6 1 3 2 4



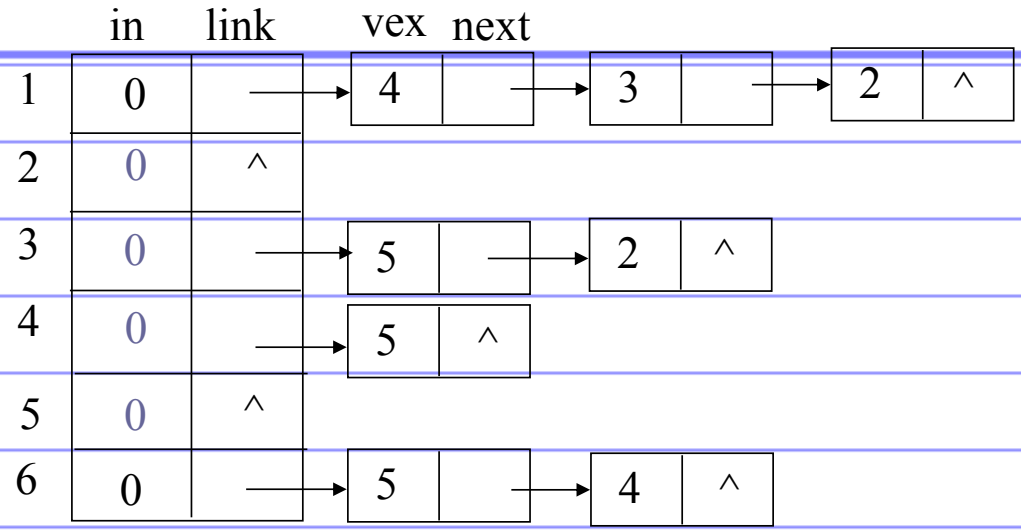


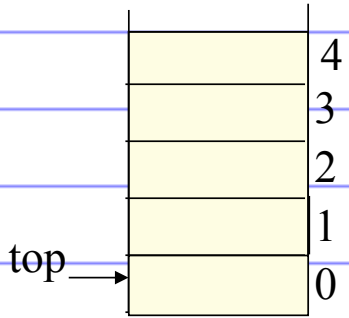
输出序列：6 1 3 2 4



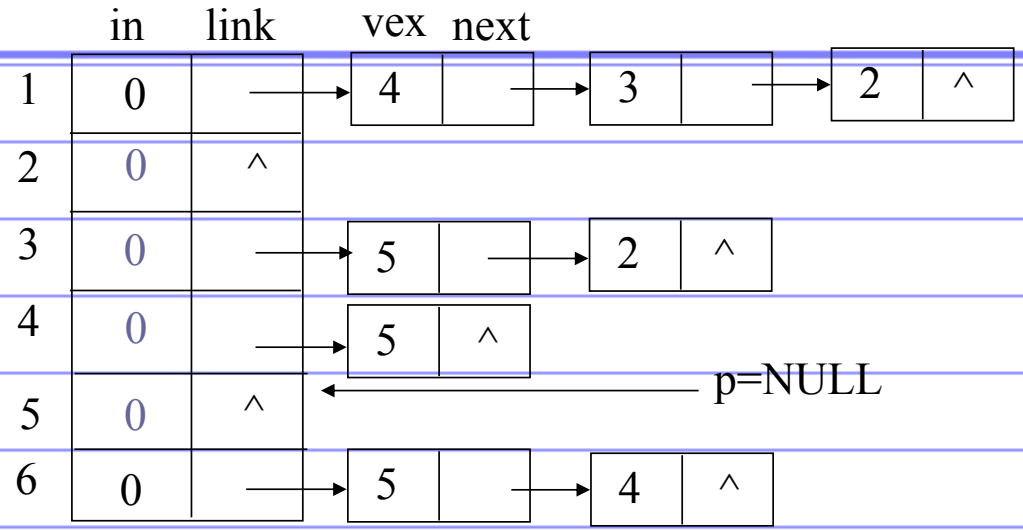


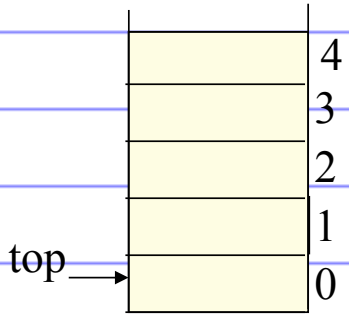
输出序列：6 1 3 2 4





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)

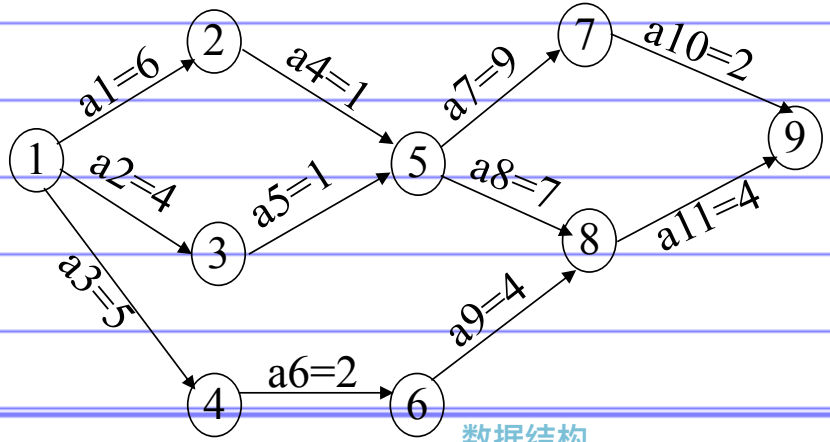
拓扑排序：T(n)=O(n+e)

6.6 关键路径

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
问题：（1）完成整项工程至少需要多少时间?
（2）哪些活动是影响工程进度的关键?



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V_j的最早发生时间
- Vl(j)——表示事件V_j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a_i的最迟开始时间
- l(i)-e(i)——表示完成活动a_i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

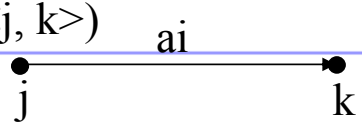
■ 问题分析

✧ 如何找e(i)=l(i)的关键活动?

设活动a_i用弧<j, k>表示，其持续时间记为：dut(<j, k>)

则有：　(1) e(i)=Ve(j)

(2) l(i)=Vl(k) - dut(<j,k>)



✧ 如何求Ve(j)和Vl(j)?

(1)从Ve(1)=0开始向前递推

$$Ve(j) = \max_{i}\{Ve(i) + dut(<i, j>)\}, <i, j> \in T, 2 \le j \le n$$

其中T是所有以j为头的弧的集合

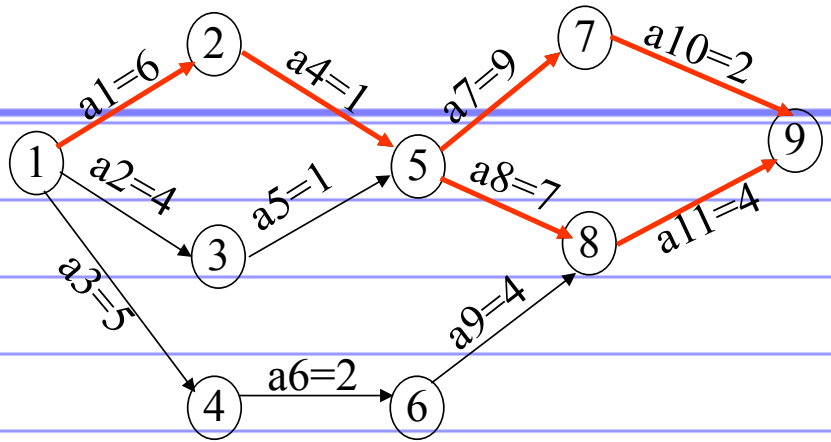
(2)从Vl(n)=Ve(n)开始向后递推

$$Vl(i) = \min_{j}\{Vl(j) - dut(<i, j>)\}, <i, j> \in S, 1 \le i \le n-1$$

其中S是所有以i为尾的弧的集合

■ 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)



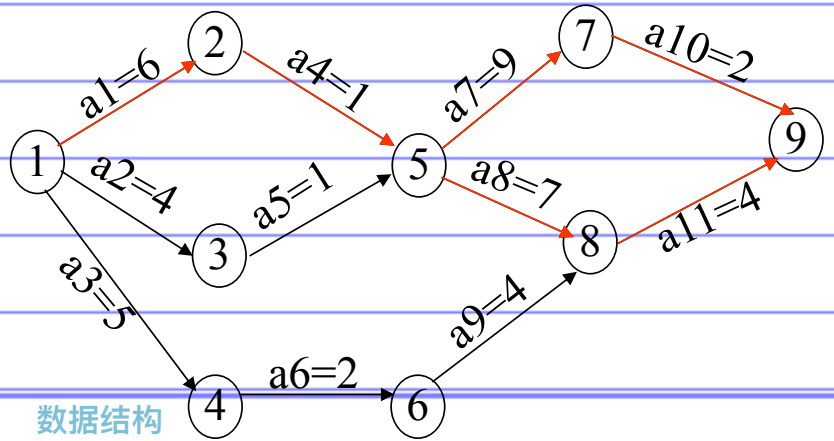
顶点	Ve	Vl
V1	0	0
V2	6	6
V3	4	6
V4	5	8
V5	7	7
V6	7	10
V7	16	16
V8	14	14
V9	18	18

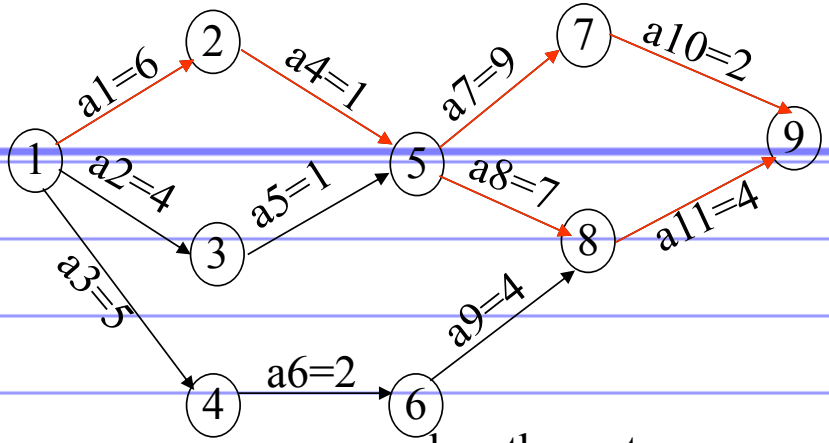
活动	e	l	l-e
a1	0	0	0 ✓
a2	0	2	2
a3	0	3	3
a4	6	6	0 ✓
a5	4	6	2
a6	5	8	3
a7	7	7	0 ✓
a8	7	7	0 ✓
a9	7	10	3
a10	16	16	0 ✓
a11	14	14	0 ✓

- 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





	vexdata	in	link
1	1	0	
2	2	1	
3	3	1	
4	4	1	
5	5	2	
6	6	1	
7	7	1	
8	8	2	
9	9	2	^

vex	length	next
2	6	
5	1	^
5	1	^
6	2	^
7	9	
8	4	^
9	10	^
9	4	^

3	4	

4	5	^

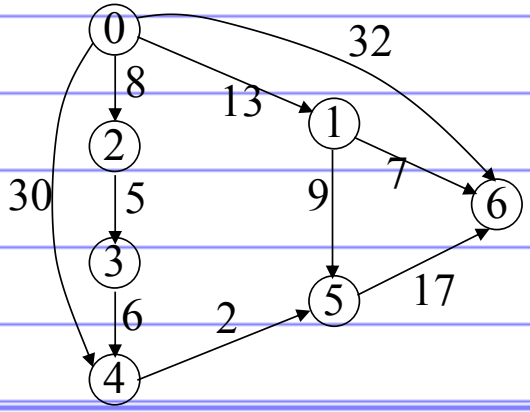
8	7	^

6.7 最短路径

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

迪杰斯特拉(Dijkstra)算法思想

按路径长度递增次序产生最短路径算法：

把V分成两组：

（1）S：已求出最短路径的顶点的集合

（2）V-S=T：尚未确定最短路径的顶点集合

将T中顶点按最短路径递增的次序加入到S中，

保证：（1）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

（2）每个顶点对应一个距离值

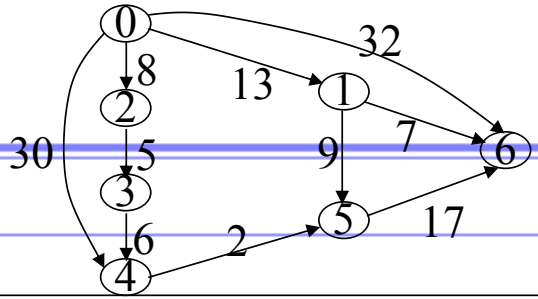
S中顶点：从V0到此顶点的最短路径长度

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，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

（反证法可证）

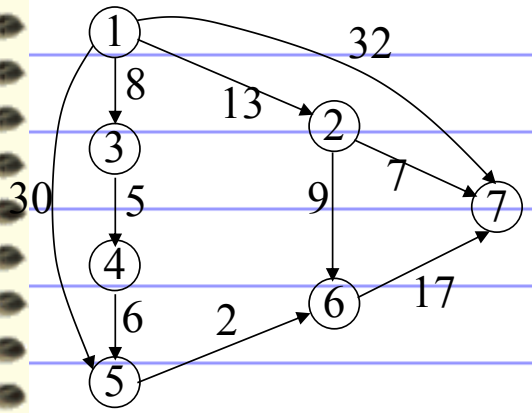
- 求最短路径步骤
  - 初使时令 S={V0},T={其余顶点}，T中顶点对应的距离值
    - 若存在<V0,Vi>，为<V0,Vi>弧上的权值
    - 若不存在<V0,Vi>，为∞
  - 从T中选取一个其距离值为最小的顶点W，加入S
  - 对T中顶点的距离值进行修改：若加进W作中间顶点，从V0到Vi的距离值比不加W的路径要短，则修改此距离值
  - 重复上述步骤，直到S中包含所有顶点，即S=V为止



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>

- 算法实现
  - 图用带权邻接矩阵存储ad[][]
  - 数组dist[]存放当前找到的从源点V0到每个终点的最短路径长度，其初态为图中直接路径权值
  - 数组pre[]表示从V0到各终点的最短路径上，此顶点的前一顶点的序号；若从V0到某终点无路径，则用0作为其前一顶点的序号

✧算法描述



$ad[][] =$

$$\begin{bmatrix} 1 & 13 & 8 & \infty & 30 & \infty & 32 \\ \infty & 0 & \infty & \infty & \infty & 9 & 7 \\ \infty & \infty & 0 & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 6 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 17 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

最短路径	长度
<V1,V2>	13
<V1,V3>	8
<V1,V3,V4>	13
<V1,V3,V4,V5>	19
<V1,V3,V4,V5,V6>	21
<V1,V2,V7>	20

	0	1	2	3	4	5	6
dist	0	13	8	13	19	21	20

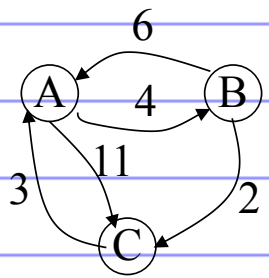
	0	1	2	3	4	5	6
pre	0	1	1	3	4	5	2

k=1 (1)

✧算法分析：T(n)=O(n²)

- 每一对顶点之间的最短路径
  - 方法一：每次以一个顶点为源点，重复执行Dijkstra算法n次——$T(n)=O(n^3)$
  - 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
      - 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∞
      - 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
      - 所有顶点试探完毕，算法结束

例



初始：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$ 路径：

	AB	AC
BA		BC
CA		

加入V1：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	AC
BA		BC
CA	CAB	

加入V2：$\begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	ABC
BA		BC
CA	CAB	

加入V3：$\begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	ABC
BCA		BC
CA	CAB	

✧算法实现

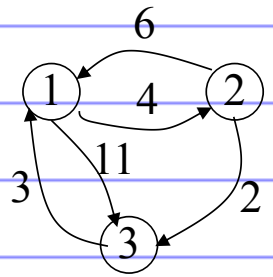
?图用邻接矩阵存储

?length[][]存放最短路径长度

?path[i][j]是从Vi到Vj的最短路径上Vj前一顶点序号

✧算法描述

例



初始：

$$length=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & \infty & 0\end{pmatrix}\quad path=\begin{pmatrix}0 & 1 & 1\\ 2 & 0 & 2\\ 3 & 0 & 0\end{pmatrix}$$

加入V1：

$$length=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}\quad path=\begin{pmatrix}0 & 1 & 1\\ 2 & 0 & 2\\ 3 & 1 & 0\end{pmatrix}$$

加入V2：

$$length=\begin{pmatrix}0 & 4 & 6\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}\quad path=\begin{pmatrix}0 & 1 & 2\\ 2 & 0 & 2\\ 3 & 1 & 0\end{pmatrix}$$

加入V3：

$$length=\begin{pmatrix}0 & 4 & 6\\ 5 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}\quad path=\begin{pmatrix}0 & 1 & 2\\ 3 & 0 & 2\\ 3 & 1 & 0\end{pmatrix}$$

?算法分析：$T(n)=O(n^3)$