第7章　图

王　勇
计算机/软件学院　大数据分析与信息安全团队
21#518　电　话 13604889411
Email：　wangyongcs@hrbeu.edu.cn

哈爾濱工程大學
Harbin Engineering University

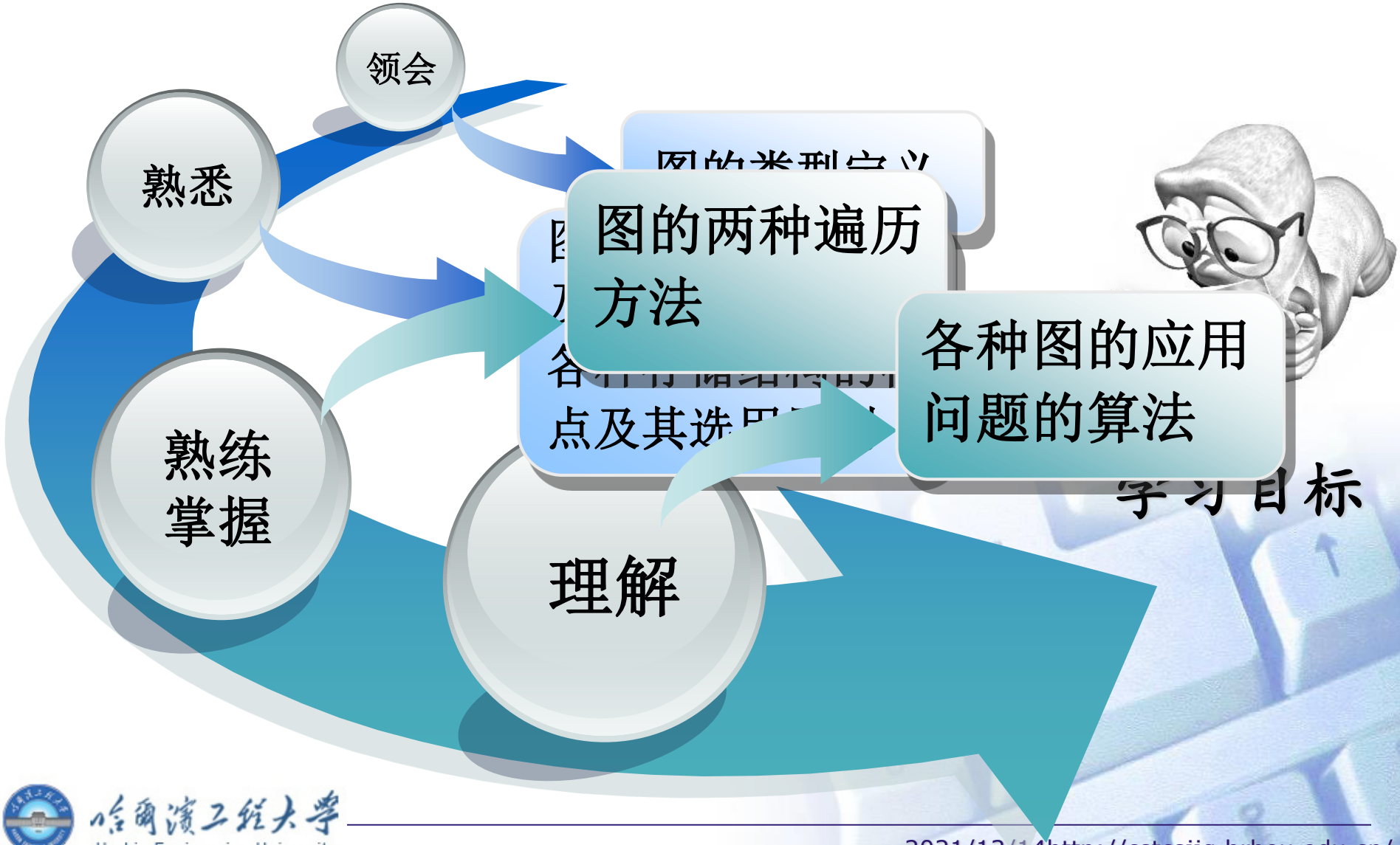
问题提出

- 城市间通信网络布局
- 社交网络关系
- 旅行家问题
- 商店位置设置
- 工程项目时间估算

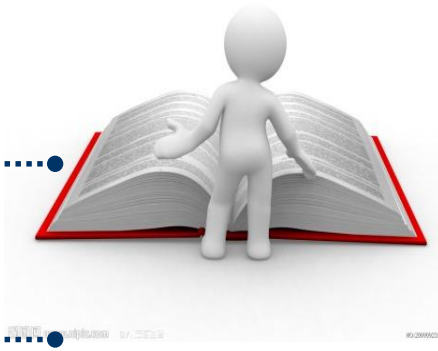
本章说明

知识点	重　点	难　点
图的类型定义 图的存储表示 图的搜索遍历 最小生成树 最短路径 拓扑排序 关键路径	理解各种图的算法及其应用场合	无

本章说明



本章内容



1. 图的定义和术语
2. 图的存储结构
3. 图的遍历
4. 生成树
5. 拓扑排序
6. 最短路径

图

图G是由两个集合V和VR组成的，记为G=(V,VR)

其中：V是顶点的有穷非空有限集

VR是两个顶点之间关系的有限集合，即边集合，边是顶点的无序对或有序对

有向图

G是由两个集合V和VR组成的

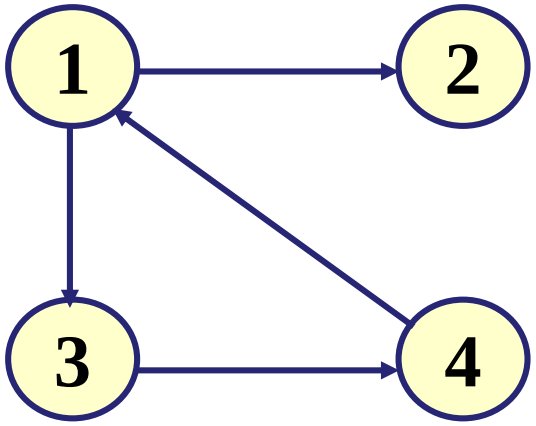
其中：V是顶点的有穷非空有限集

VR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

例如： $G_1 = (V_1, VR_1)$



有向图G1

其中

V_1={1, 2, 3, 4}

VR_1={<1,2>, <1,3>,
<3,4>, <4,1>}

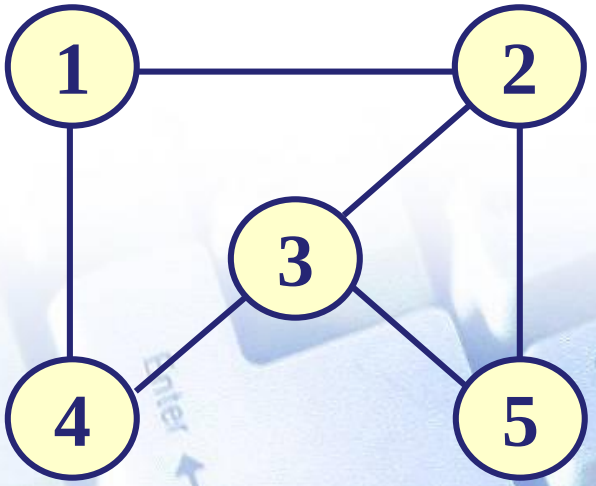
无向图 由顶点集和边集构成的图

若<v, w>∈VR 必有<w, v>∈VR,则称 (v,w) 为顶点 v 和顶点 w 之间存在一条边。

例如：$G_2=(V_2,VR_2)$

$V_2=\{1, 2, 3, 4, 5\}$

$VR_2=\{(1, 2), (1, 4), (2, 3), (2, 5), (3, 4), (3, 5)\}$

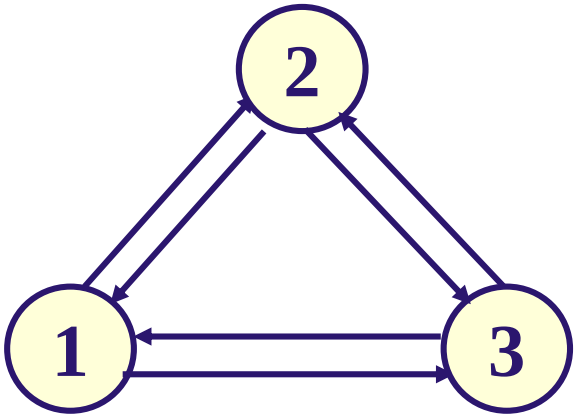


无向图G2

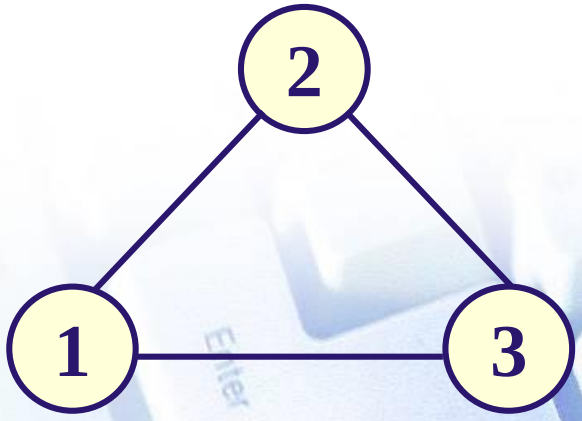
有向完全图 有n个顶点，n(n-1)弧的有向图

无向完全图 有n个顶点，n(n-1)/2条边的无向图

例



有向完全图



无向完全图

稀疏图 有很少条边和弧(e<nlogn)的图

稠密图 与稀疏图相反

权 与图的边或弧相关的数

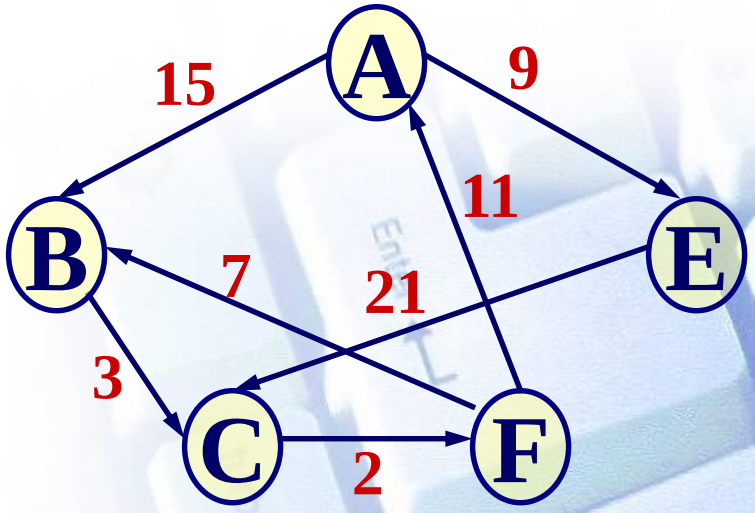
网 带权的图

有向网 弧带权的图

无向网 边带权的图

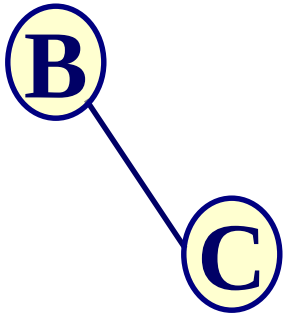
无向图 → 边 → (v,w)

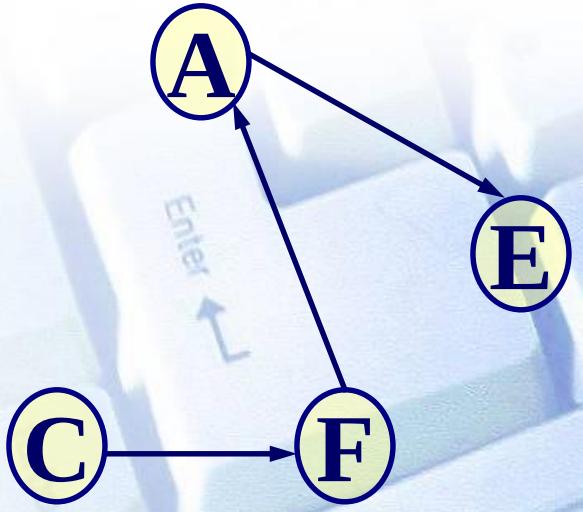
有向图 → 弧 → <v,w>



子 图 如果图G(V，E)和图G′(V′,E′)满足：V′⊆V 且 E′⊆E，则称G′为G的子图

邻接点 在无向图中一条边连起来的两个顶点(v, v′),互称邻接点，称边(v, v′) 依附于顶点v和v′,或相关联



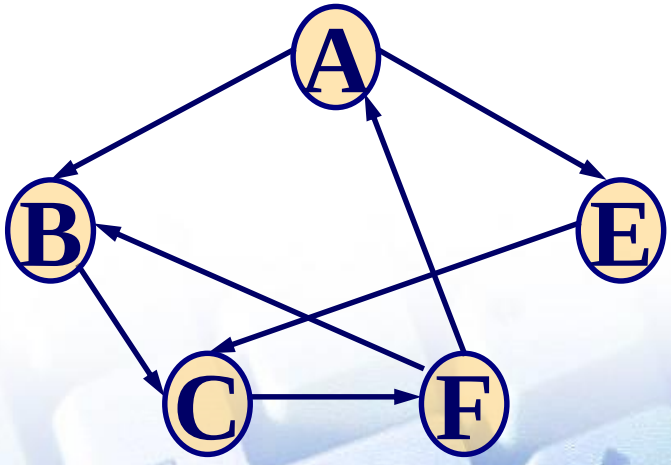


顶点的度

- 无向图中顶点的度为与每个顶点相连的边数
- 有向图中顶点的度为:

  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



例如: OD(B) = 1　ID(B) = 2　TD(B) = 3

顶点的度(TD)=出度(OD)+入度(ID)

路　径 从顶点v到v′的路径是一个顶点的序列

$v=\{v_{i,0},v_{i,1},\ldots\ldots v_{i,n}\}$

满足$(v_{i,j-1},v_{i,j})\in E$ 或 $\langle v_{i,j-1},v_{i,j}\rangle\in E,(1<j\leq n)$

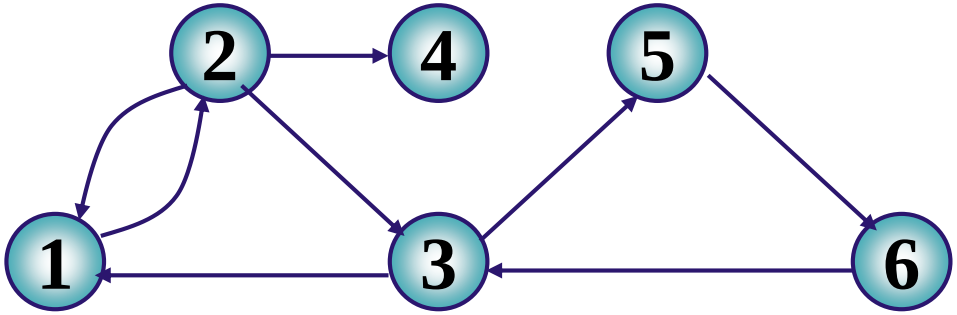
路径上边的数目称作“路径长度”

回路(环) 第一个顶点和最后一个顶点相同的路径

简单路径 序列中顶点不重复出现的路径

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

例 如



G1

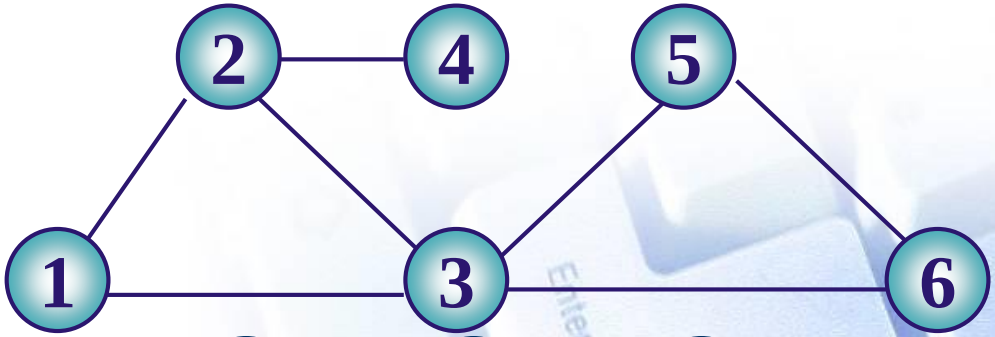
路径：1，2，3，5，6，3
路径长度：5
简单路径：1，2，3，5
回路：1，2，3，5，6，3，1
简单回路：3，5，6，3

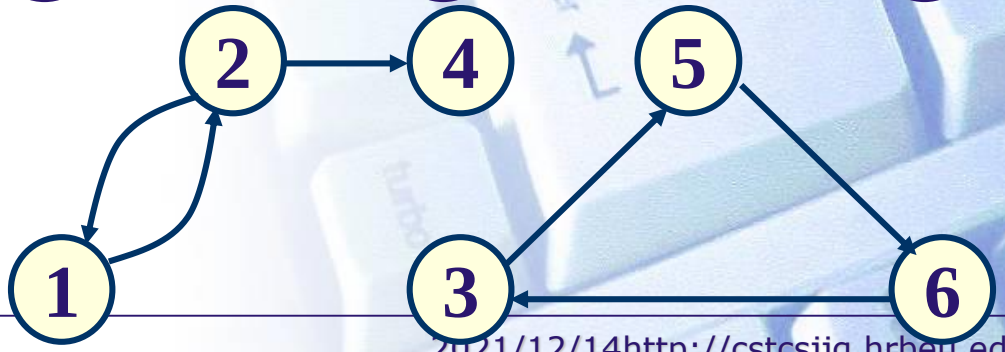
连 通

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

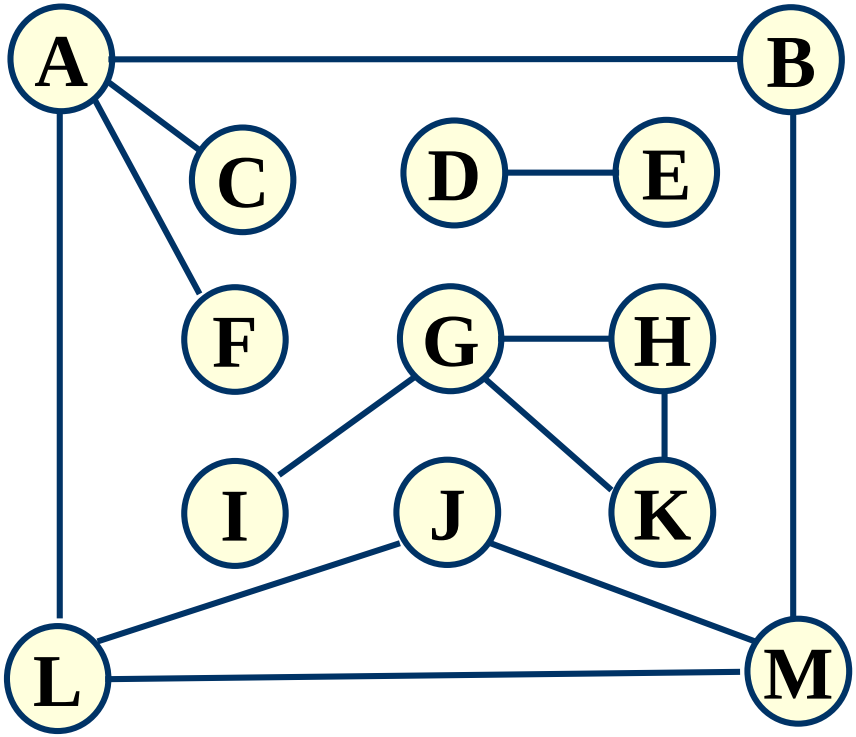
图中任意个顶点都是连通的



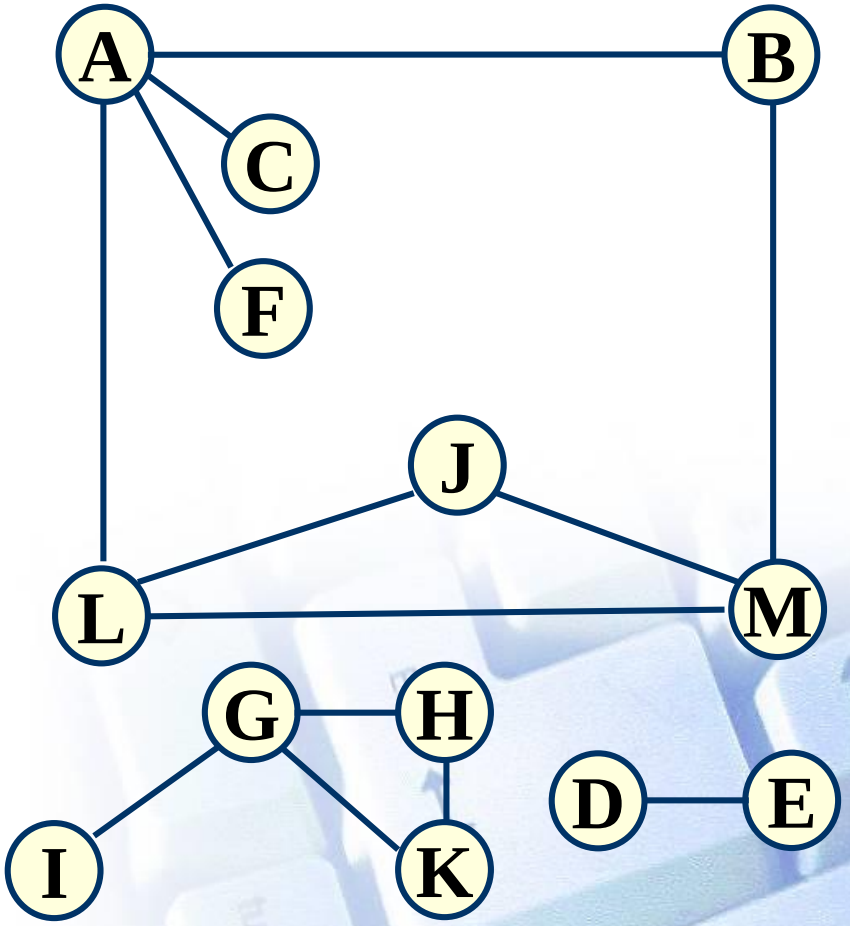


连通图非连通图

连通分量 指的是无向图中的极大连通子图



无向图G3

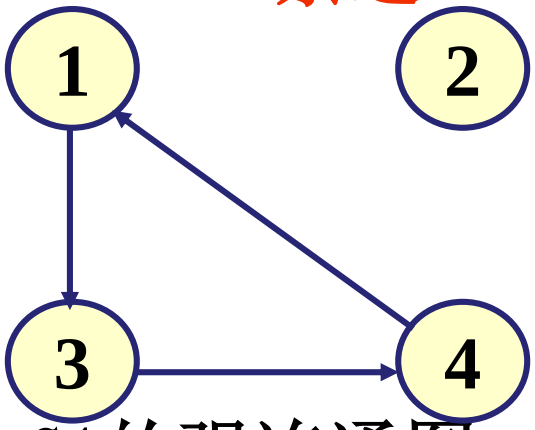


G3的3个连通分量

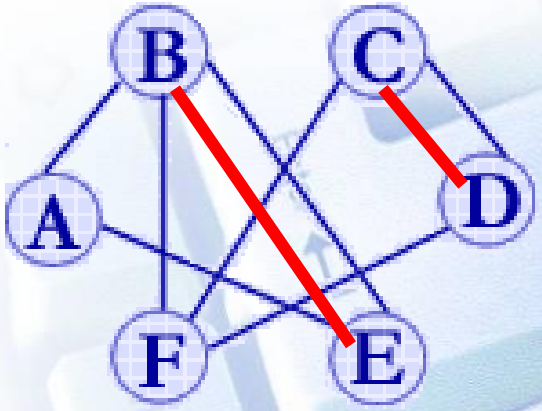
强连通图 有向图中，如果对每一对vi,vj∈V, vi≠vj,从vi到vj 和从vj到 vi都存在路径

强连通分量 指的是有向图中的极大强连通子图

生 成 树 是连通图的一个极小连通子图，它含有图的全部顶点，但只有足以构成一棵树的n-1条边



G1的强连通图

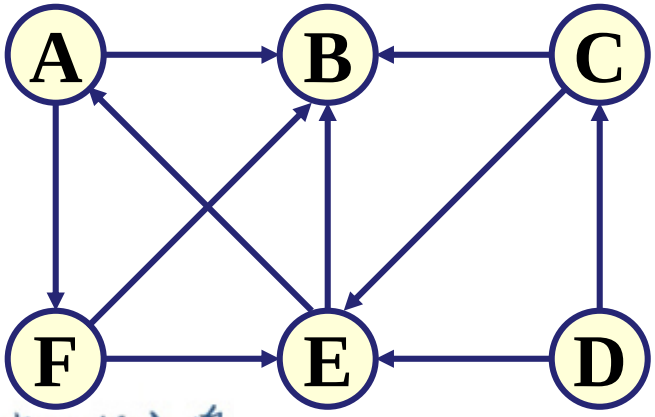


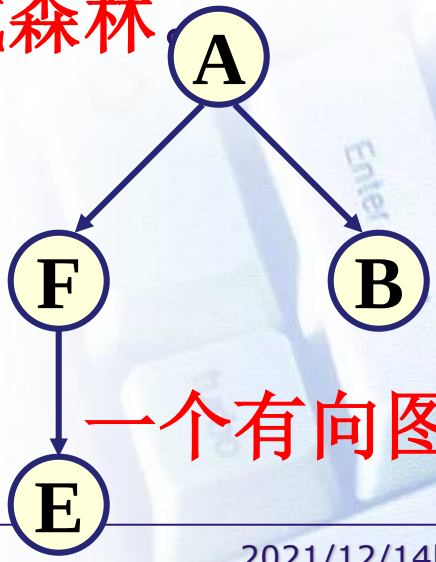
生成树

有向树 有向图恰有一个顶点入度为0，其余顶点入度均为1，则是一棵有向树

生成森林 有向图的生成森林由若干棵有向树组成含有图中全部顶点，但只有足以构成若干不相交的有向树的弧

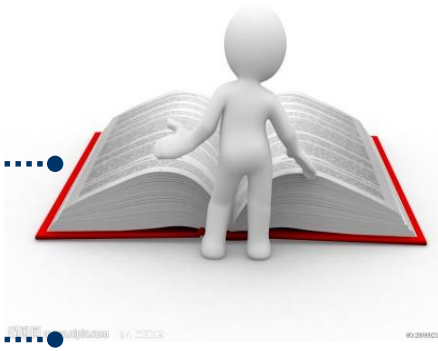
对于非连通图，对其每个连通分量可以构造一棵生成树，合成起来就是一个生成森林





一个有向图及其生成森林

本章内容



1. 图的定义和术语
2. 图的存储结构
3. 图的遍历
4. 生成树
5. 拓扑排序
6. 最短路径

1. 图的数组(邻接矩阵)存储表示
2. 图的邻接表存储表示
3. 有向图的十字链表存储表示
4. 无向图的邻接多重表存储表示

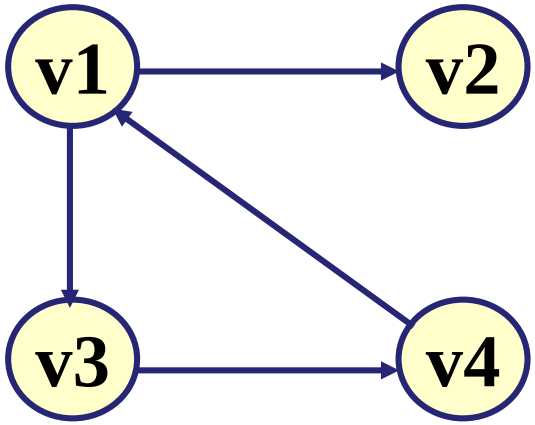
图的邻接矩阵

- 将图的*顶点信息*存储在一个一维数组中，并将它的*邻接矩阵*存储在一个二维数组中即构成图的数组表示
- 图的邻接矩阵——表示顶点间相联关系的矩阵

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

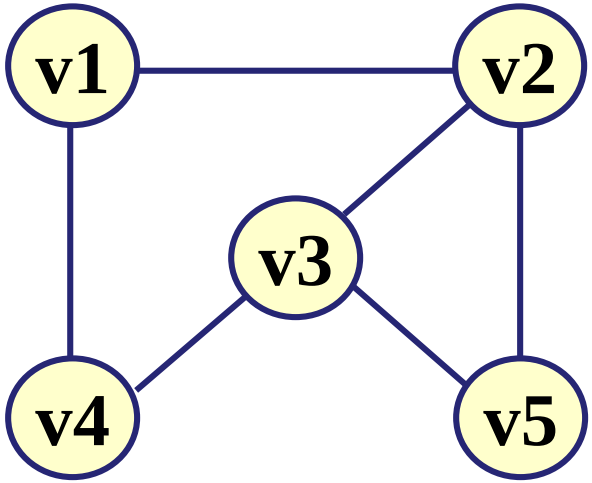
$$A[i,j]=\begin{cases}1, \text{若}(v_i,v_j)\text{或}<v_i,v_j>\in E(G)\\ 0, \quad \text{其它}\end{cases}$$

图的存储结构　　图的数组(邻接矩阵)存储表示



有向图G1

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



无向图G2

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

图的存储结构　　图的数组(邻接矩阵)存储表示

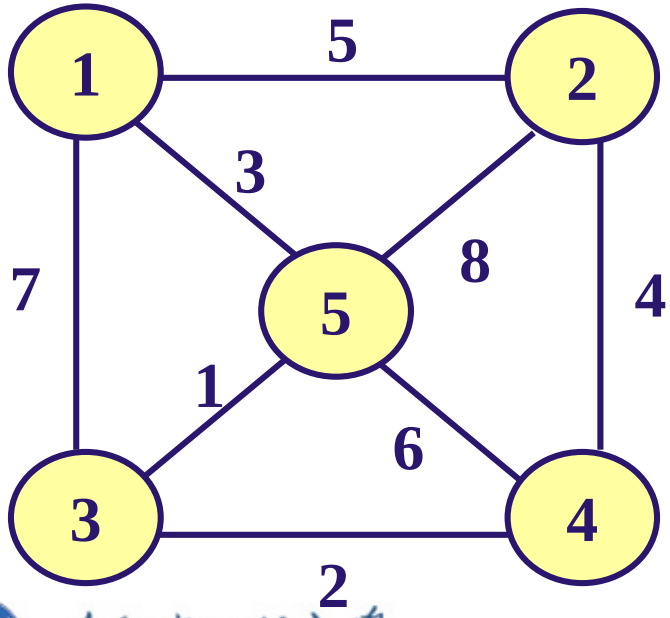
特　点

- 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Vi的度TD(Vi)是邻接矩阵A中第i行元素之和
- 有向图中：
  - 顶点Vi的出度是A中第i行元素之和
  - 顶点Vi的入度是A中第i列元素之和

网的邻接矩阵

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

例

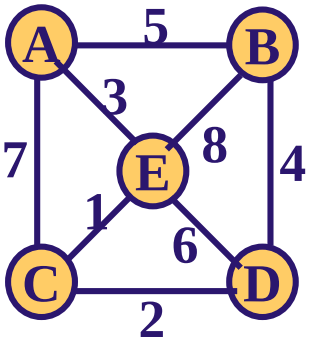


$$\begin{array}{c} \\ ① \\ ② \\ ③ \\ ④ \\ ⑤ \end{array}\begin{array}{c} \begin{array}{ccccc} ① & ② & ③ & ④ & ⑤ \end{array} \\ \begin{bmatrix} \infty & 5 & 7 & \infty & 3 \\ 5 & \infty & \infty & 4 & 8 \\ 7 & \infty & \infty & 2 & 1 \\ \infty & 4 & 2 & \infty & 6 \\ 3 & 8 & 1 & 6 & \infty \end{bmatrix} \end{array}$$

存储表示

```
#define  INFINITY  INT_MAX   //对无穷大预设值
#define  MAX_VERTEX_NUM  20
typedef enum {DG, DN, UDG, UDN} GraphKind;
                //{有向图，有向网，无向图，无向网}
typedef struct ArcCell{    //邻接矩阵
  VRType   adj;             //顶点关系类型，权值
  InfoType  *info;          //该弧相关信息的指针
}ArcCell, AdjMatrix[MAX_VERTEX_NUM]
                                [ MAX_VERTEX_NUM];
typedef struct{
  VertexType vexs[MAX_VERTEX_NUM];  //顶点向量
  AdjMatrix    arcs;                //邻接矩阵
  int              vexnum, arcnum;  //顶点数和弧数
  GraphKind  kind;                  //图的种类
}MGraph;
```

图的存储结构　　图的数组(邻接矩阵)存储表示



G3

G3.vexs=[A,B,C,D,E]

$$G3.arcs=\begin{bmatrix} \infty & 5 & 7 & \infty & 3 \\ 5 & \infty & \infty & 4 & 8 \\ 7 & \infty & \infty & 2 & 1 \\ \infty & 4 & 2 & \infty & 6 \\ 3 & 8 & 1 & 6 & \infty \end{bmatrix}$$

G3.vexnum=5　G3.arcnum=8　G3.kind=UDN

优缺点

- 优点：容易判定顶点间有无边（弧），容易计算顶点的度（出度、入度）
- 缺点：所占空间只和顶点个数有关，和边数无关，在边数较少时，空间浪费较大

- 建立无向网邻接矩阵的算法

  算法中：G.vexs[ ]，一维，顶点向量

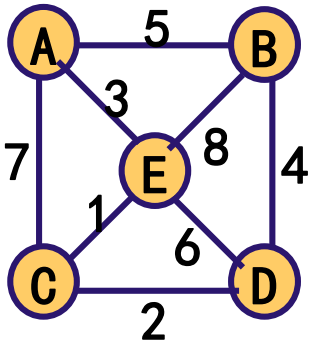
  .arcs[ ][ ]，二维，邻接矩阵

  .vexnum，顶点数

  .arcnum，边数


sf7.2

G3.vexs=[A,B,C,D,E]



G3

$$G3.arcs=\begin{bmatrix} \infty & 5 & 7 & \infty & 3 \\ 5 & \infty & \infty & 4 & 8 \\ 7 & \infty & \infty & 2 & 1 \\ \infty & 4 & 2 & \infty & 6 \\ 3 & 8 & 1 & 6 & \infty \end{bmatrix}$$

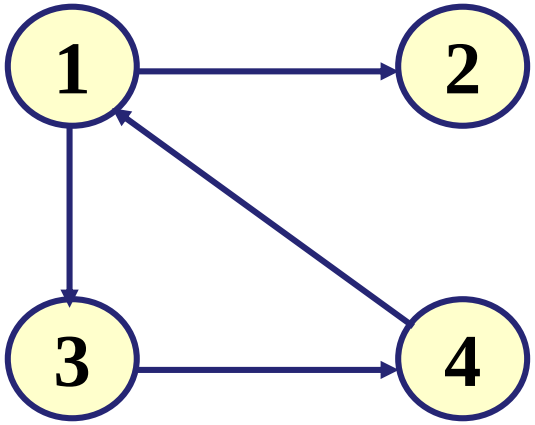
G3.vexnum=5　　G3.arcnum=8　　G3.kind=UDN

```
Status CreateUDN(MGraph &G)
{
   // 采用数组（邻接矩阵）表示法，构造无向网G。
   scanf("%d,%d,%d",&G.vexnum, &G.arcnum, &IncInfo);
   for (i=0; i<G.vexnum; i++ )
      scanf("%c",&G.vexs[i]);  // 构造顶点向量
   for (i=0; i<G.vexnum; ++i )  // 初始化邻接矩阵
      for (j=0; j<G.vexnum; ++j )
      {
         G.arcs[i][j].adj = INFINITY;
         G.arcs[i][j].info= NULL;
      }
```

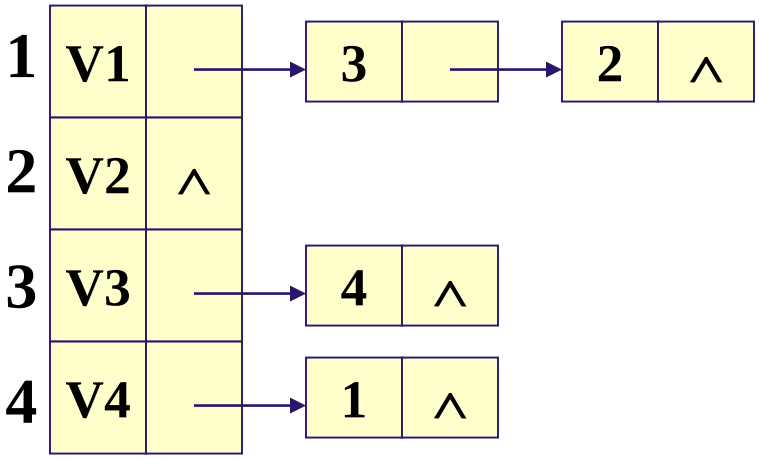
```
for (k=0; k<G.arcnum; ++k )
      {
         // 构造邻接矩阵
         scanf(&v1,&v2,&w);       // 输入一条边依附的顶点及权值
         i = LocateVex(G, v1);
         j = LocateVex(G, v2);  // 确定v1和v2在G中位置
         G.arcs[i][j].adj = w;              // 弧<v1,v2>的权值
         if (IncInfo)   scanf(G.arcs[i][j].info);
        // 根据前面的输入判断是否需要输入弧含有相关信息
         G.arcs[j][i].adj = G.arcs[i][j].adj; // 置<v1,v2>的对称弧<v2,v1>
      }
   return OK;
}
```

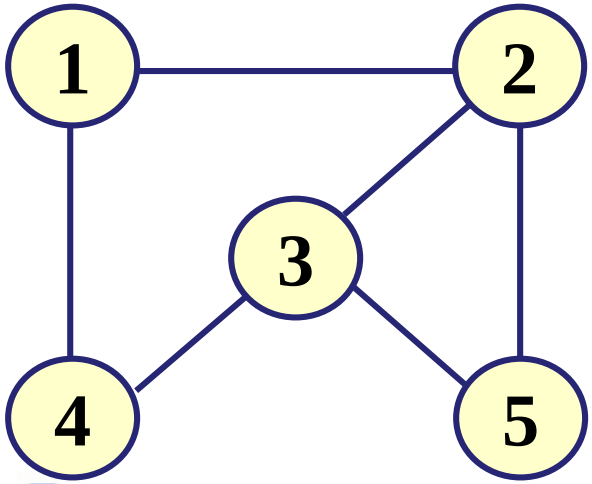
- 引入原因
  - 邻接矩阵在稀疏图时空间浪费较大
- 实现
  - 为图中每个顶点建立一个单链表，第i个单链表中的结点表示依附于顶点Vi的边（有向图中指以Vi为尾的弧）。
  - 每个链表附设一个*表头结点*（头向量）

图的存储结构　　图的邻接表存储表示

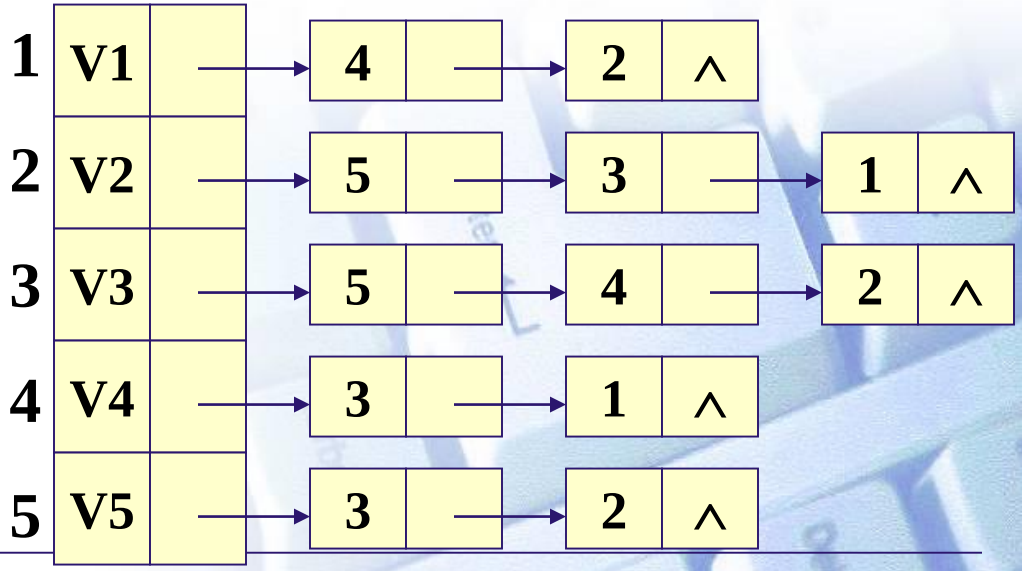


有向图G1





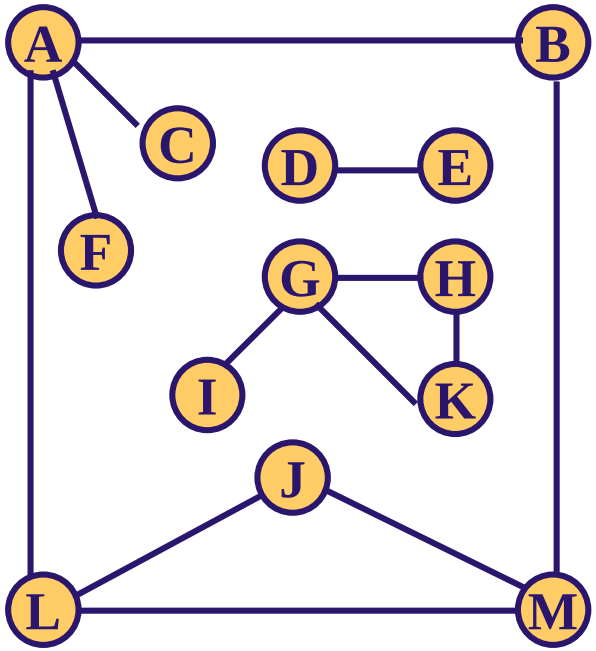
无向图G2



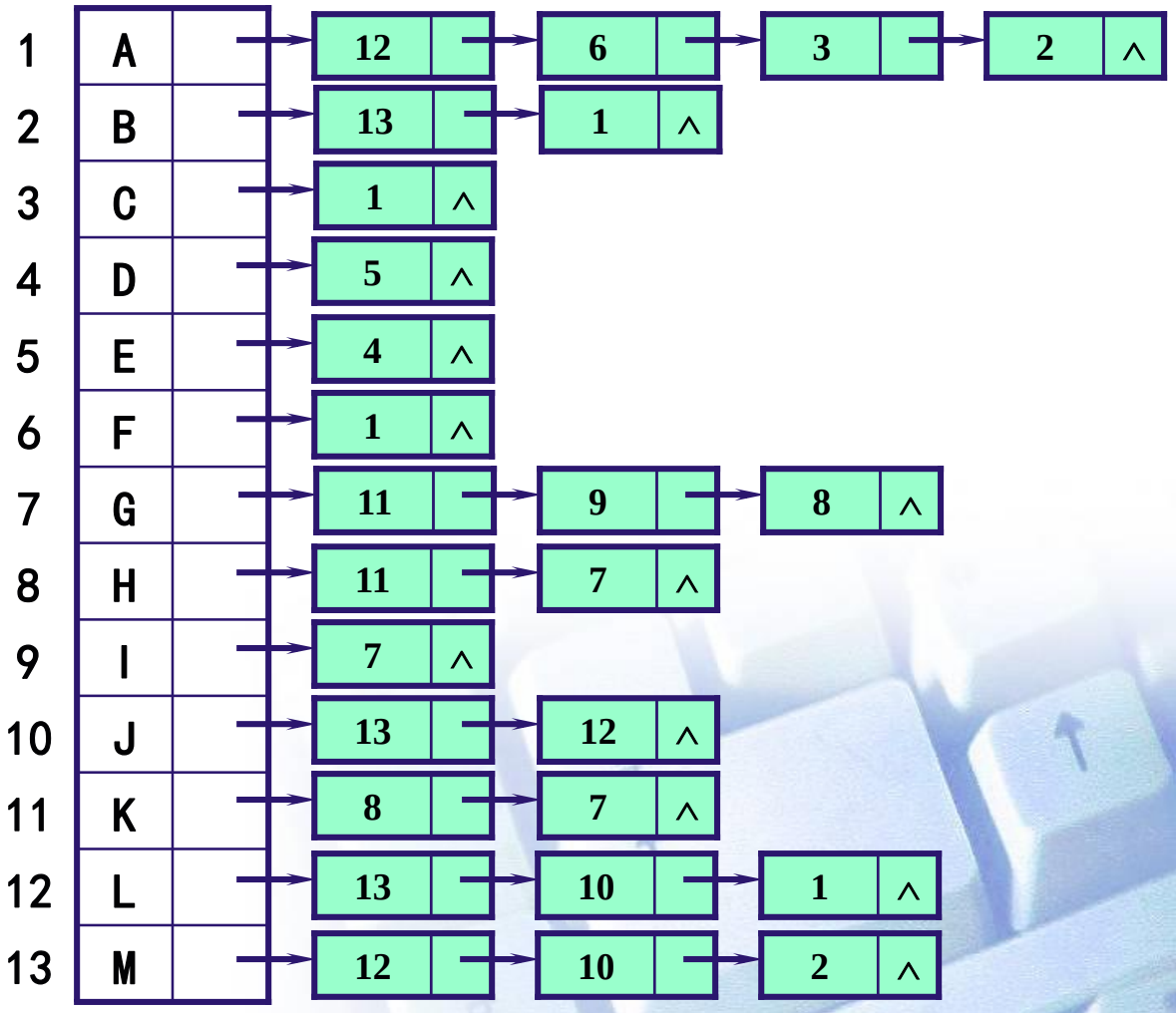
- 图的邻接表存储表示

```
#define MAX_VERTEX_NUM  20
Typedef struct ArcNode      //表结点
{   int                 Adjvex;     //顶点的位置
    struct ArcNode  *nextarc;   //指向下一条弧的指针
    InfoType            *info;      //该弧相关信息的指针
}//ArcNode;
Typedef struct Vnode            //头结点
{   VertexType    data;             //顶点信息
    ArcNode         *firstarc;      //指向第一条依附该顶点的弧
}Vnode, AdjList[MAX_VERTEX_NUM];
Typedef struct                      //图
{   AdjList  vertices;              //顶点数组（向量）
    int vexnum, arcnum, kind;  //图的当前顶点数和弧数，图的种类标志
}ALGraph;
```

图的存储结构　　图的邻接表存储表示



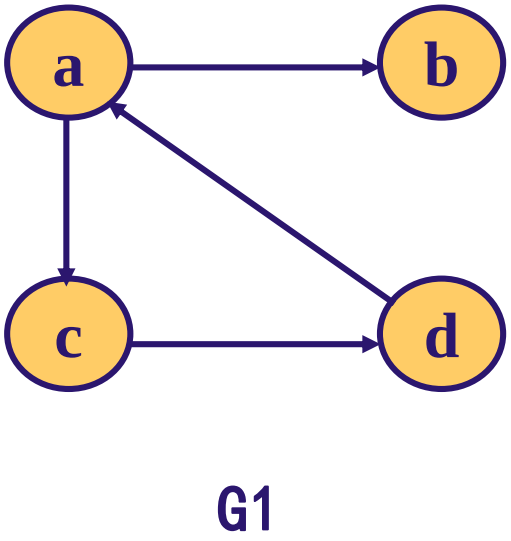
邻接表



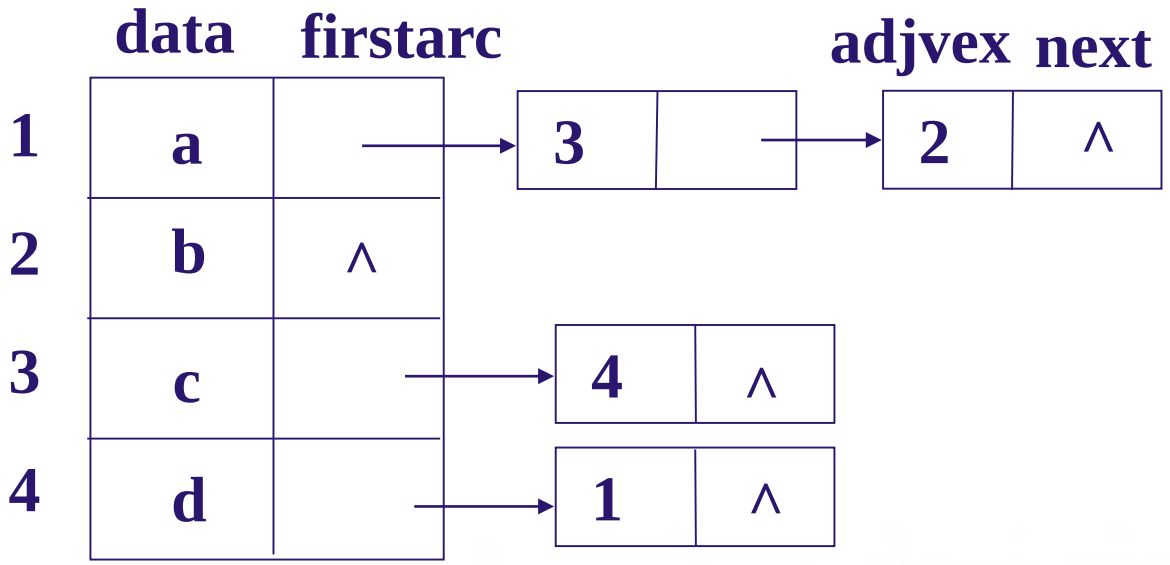
◆ 特点

- 无向图中顶点Vi的度为第i个单链表中的结点数

图的存储结构　　图的邻接表存储表示



G1

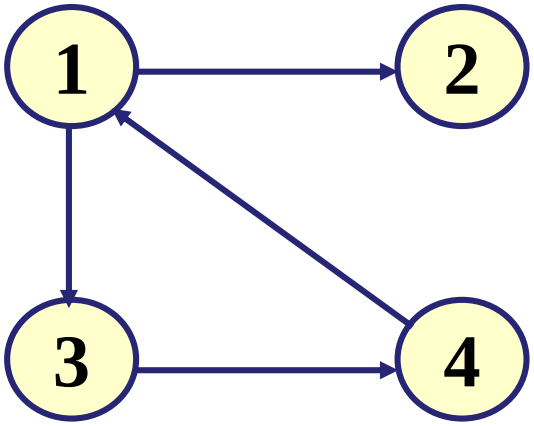


- 特点
  - 有向图中
    - 顶点Vi的出度为第i个单链表中的结点个数
    - 顶点Vi的入度为整个单链表中邻接点域值是i的结点个数

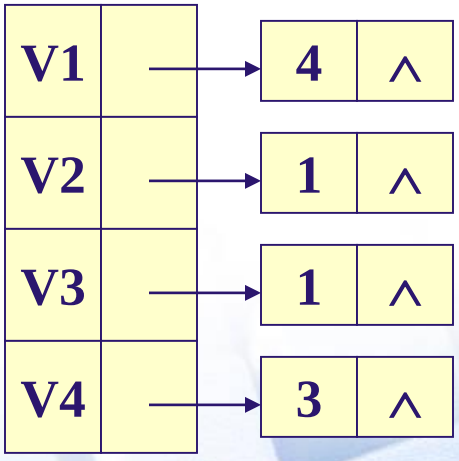
- 邻接表的优缺点
  - 优点：空间较省；无向图容易求各顶点的度；有向图容易求顶点的出度；
  - 缺点：有向图顶点的入度不容易求，要遍历整个表。
  - 为了求顶点的入度，有时可设逆邻接表（指向某顶点的邻接点链接成单链表）

- 逆邻接表

  第i(下标i-1)链表的结点个数即为Vi顶点的入度



有向图G1



- 建立邻接表的算法

sf7.jljb

建立邻接表的算法

```
void createUDN(ALGraph &G)
    //采用邻接表构造无向图
{
    scanf(&G.vexnum,&G.arcnum);  // 输入顶点数和边数
    for (i=1;i<=G.vexnum;i++)
        {
          scanf(G.vertices[i].data);   // 输入各顶点值
          G.vertices[i].firstarc=null;
        }
   for (i=1; i<=G.arcnum;i++)       //输入各边
        {
          scanf(&v1,&v2);      /* 输入顶点v1和v2
          i=LocateVex(G,v1);
          j=LocateVex(G,v2);
                  //确定两个顶点在图中的位置序号分别为i,j
```

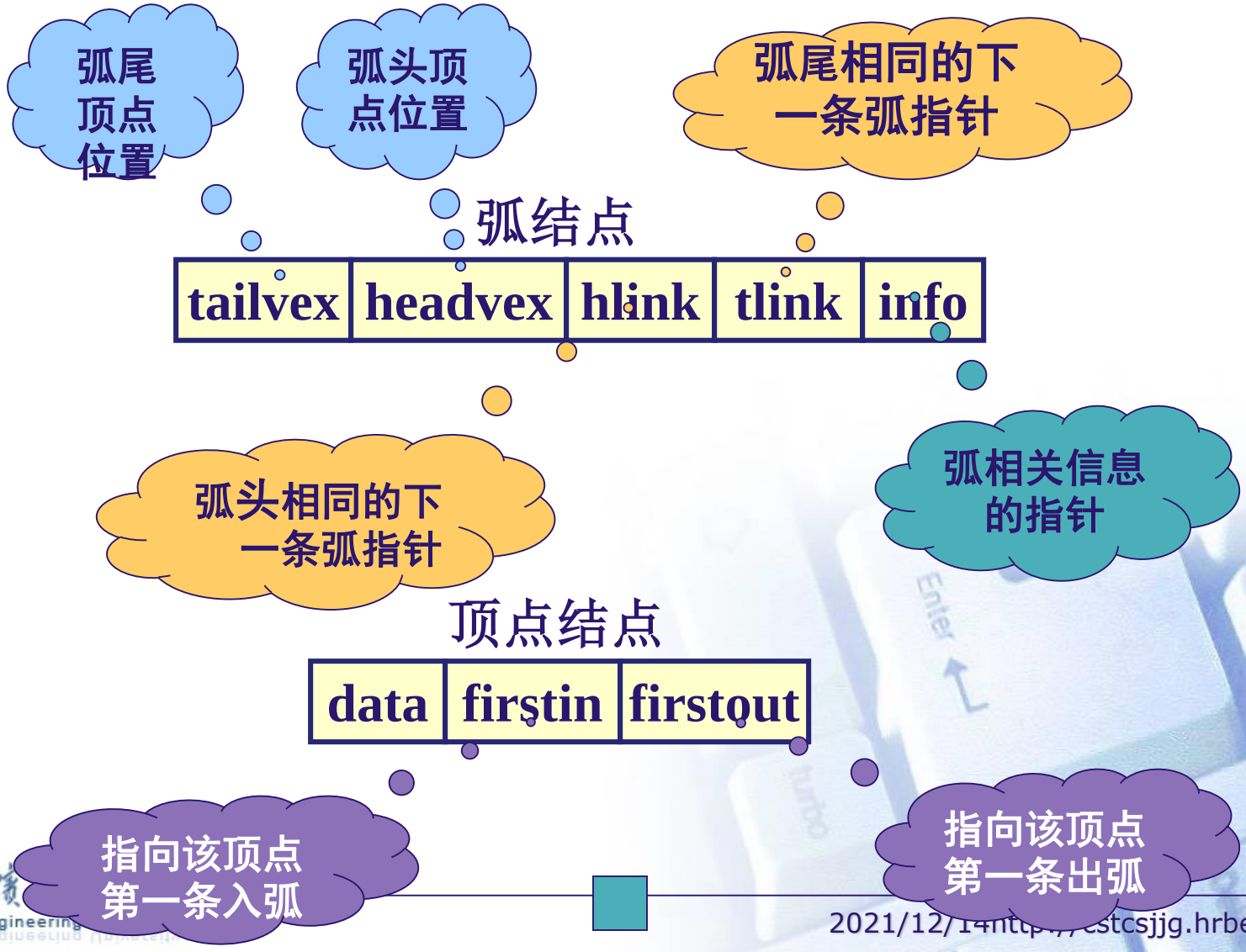
建立邻接表的算法

```
if (i!=0 && j!=0)   /* 如果输入的顶点在图中 */
        {
          r=(Vnode *)malloc(sizeof(Vnode));
          r->adjvex=j;
          r->nextarc=G.vertices[i]->firstarc; //链入i链表中
          G.vertices[i]->firstarc=r; //头插法
          r=(Vnode *)malloc(sizeof(Vnode));
          r->adjvex=i;
          r->nextarc=G.vertices[j]->firstarc; /*链入j链表中*/
          G.vertices[j]->firstarc=r;
        }
    }
}
```

- 引入原因
  - 对于同一个有向图需要同时用邻接表和逆邻接表时，不方便。
- 实现
  - 将在有向图的邻接表和逆邻接表中两次出现的同一条弧用一个结点表示，由于在邻接表和逆邻接表中的顶点数据是相同的，则在十字链表中只需要出现一次，但需保留分别指向第一条"出弧"和第一条"入弧"的指针。

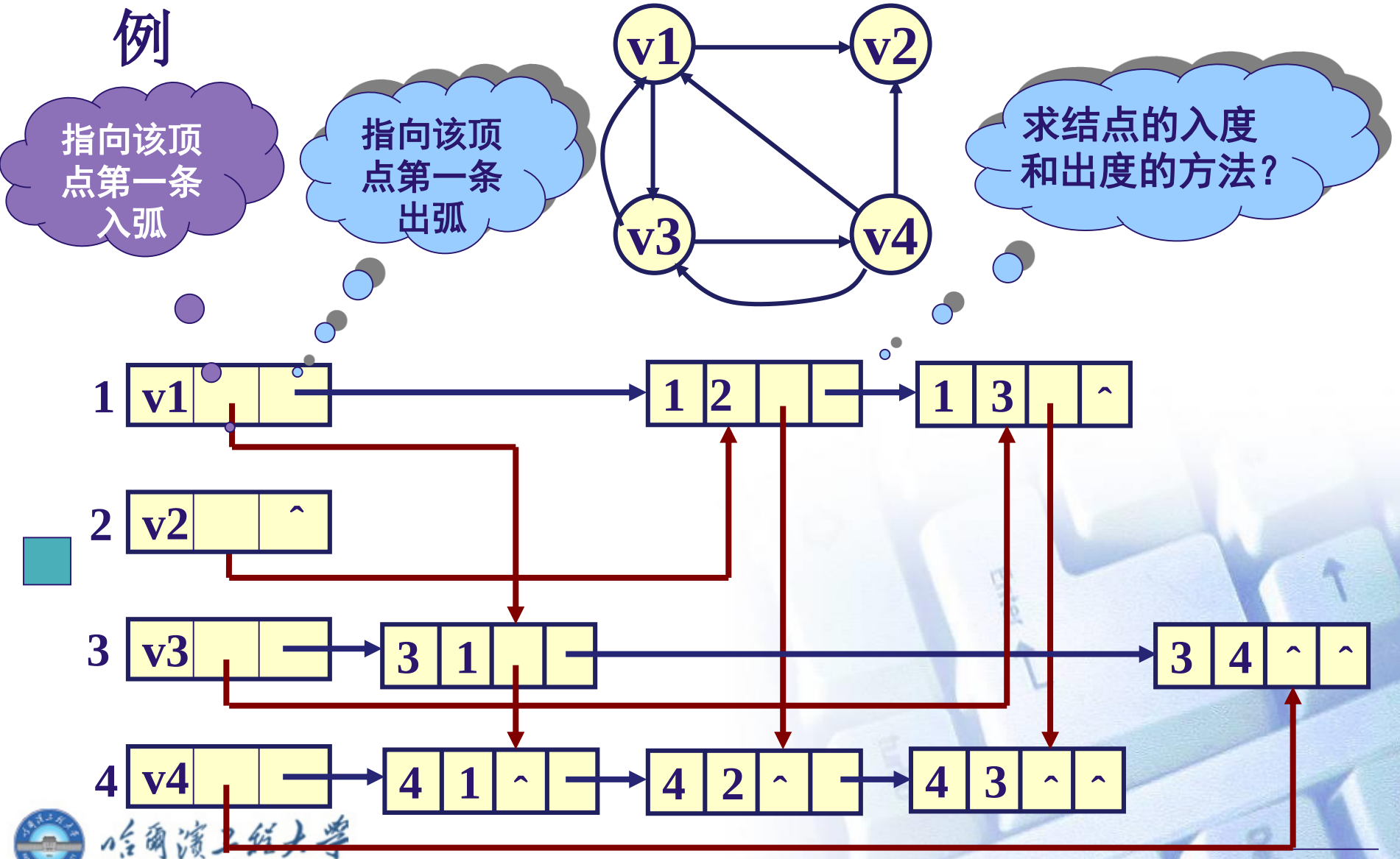
图的存储结构　　有向图的十字链表存储表示

- 十字链表结点结构



图的存储结构　　有向图的十字链表存储表示

例



- 有向图十字链表存储表示

```
#define MAX_VERTEX_NUM  20
typedef   struct  ArcBox          //弧结点
{   int                  tailvex,hesdvex;
    struct  ArcBox  *hlink,*tlink;
    infoType          *info;
}ArcBox;
Typedef struct VexNode          //顶点结点
{   VertexType  data;
    ArcBox          *firstin,*firstout;
}VexNode;
Typedef struct {                        //顶点表
    VexNode  xlist[MAX_VERTEX_NUM];//表头向量
    int              vexnum, arcnum;
}OLGraph;
```

- 建立有向图十字链表算法


sf7.3

思想：

（1）初始化表头向量、数据、指针

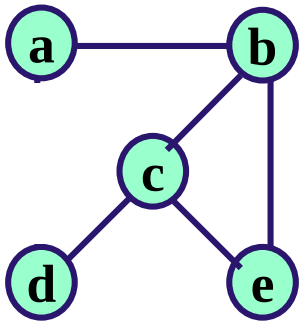
（2）输入一条弧，确定在G中位置(i,j)，申请结点空间，赋值

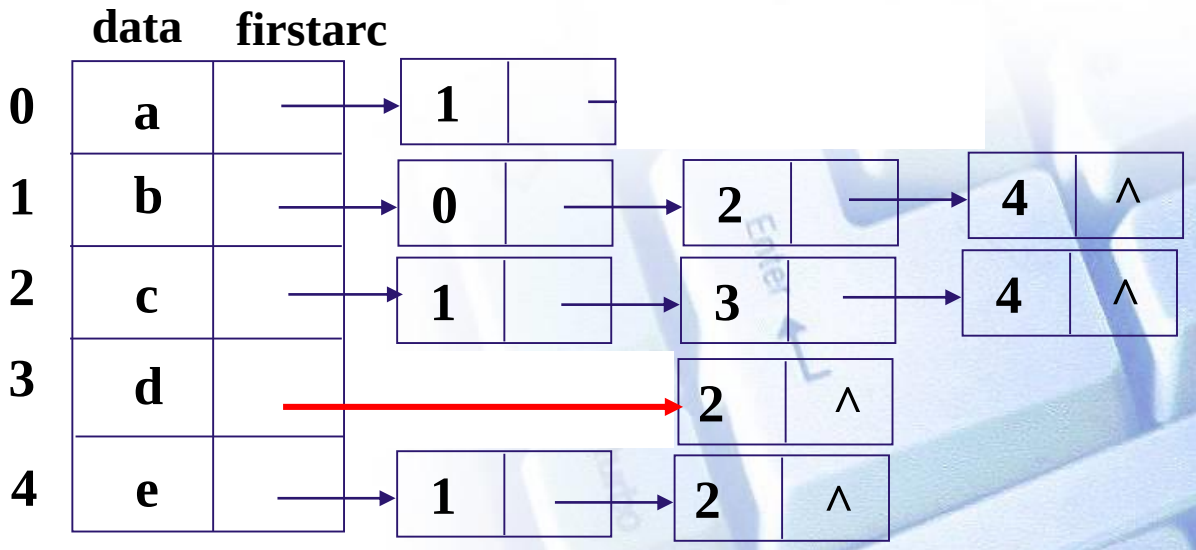
（3）插入到十字链表中（以i为弧尾和以j为弧头的链中，在链头插入）

（4）若InInfo=1，输入其信息

（5）重复（2）至（4），直到所有弧输入完为止

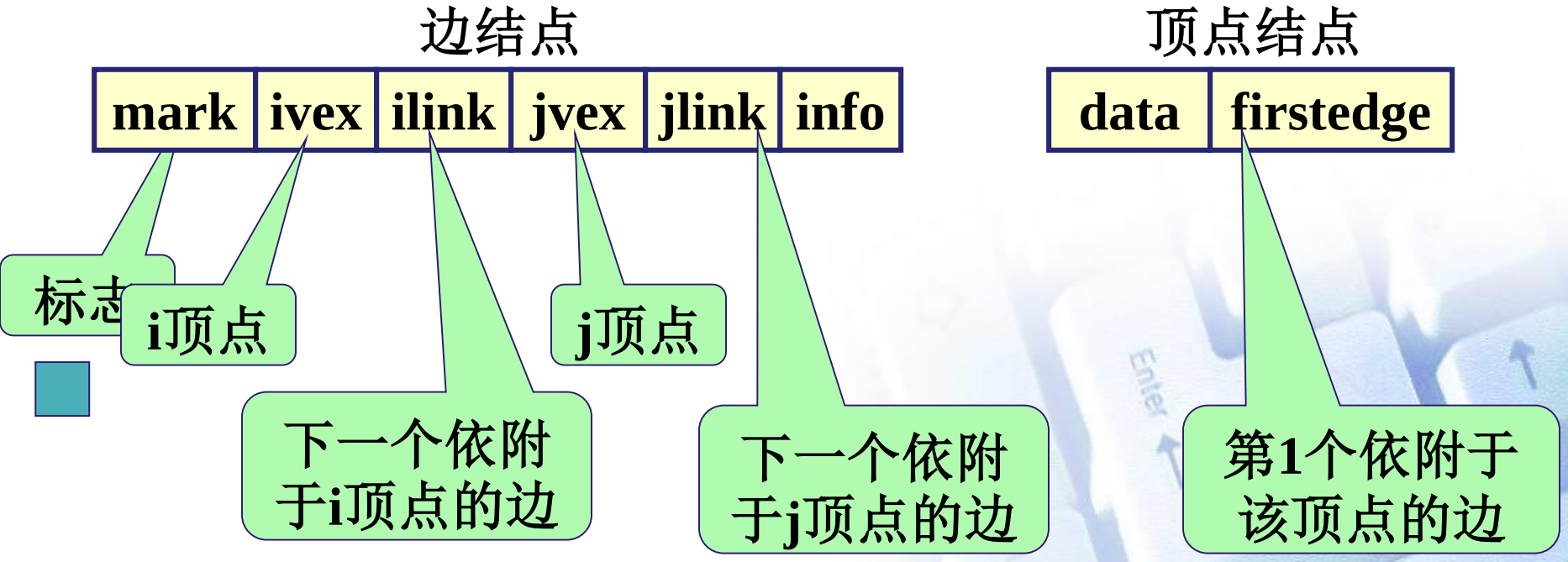
- 引入原因
  - 无向图的邻接表中每一条边有两个结点，给对图的边进行访问的操作带来不便。有些时候需要同时找到表示同一条边的两个结点（如删除一条边）





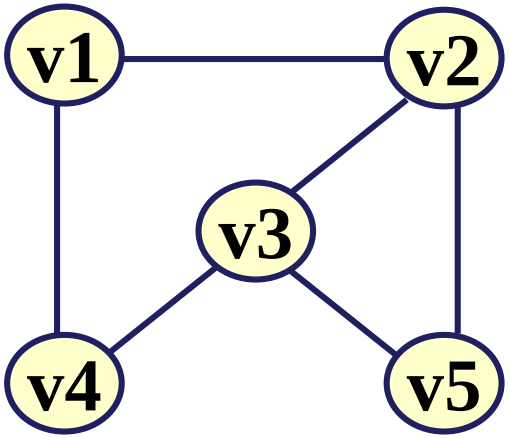
图的存储结构　　无向图的邻接多重表存储表示

- 实现
  - 每条边用一个结点表示



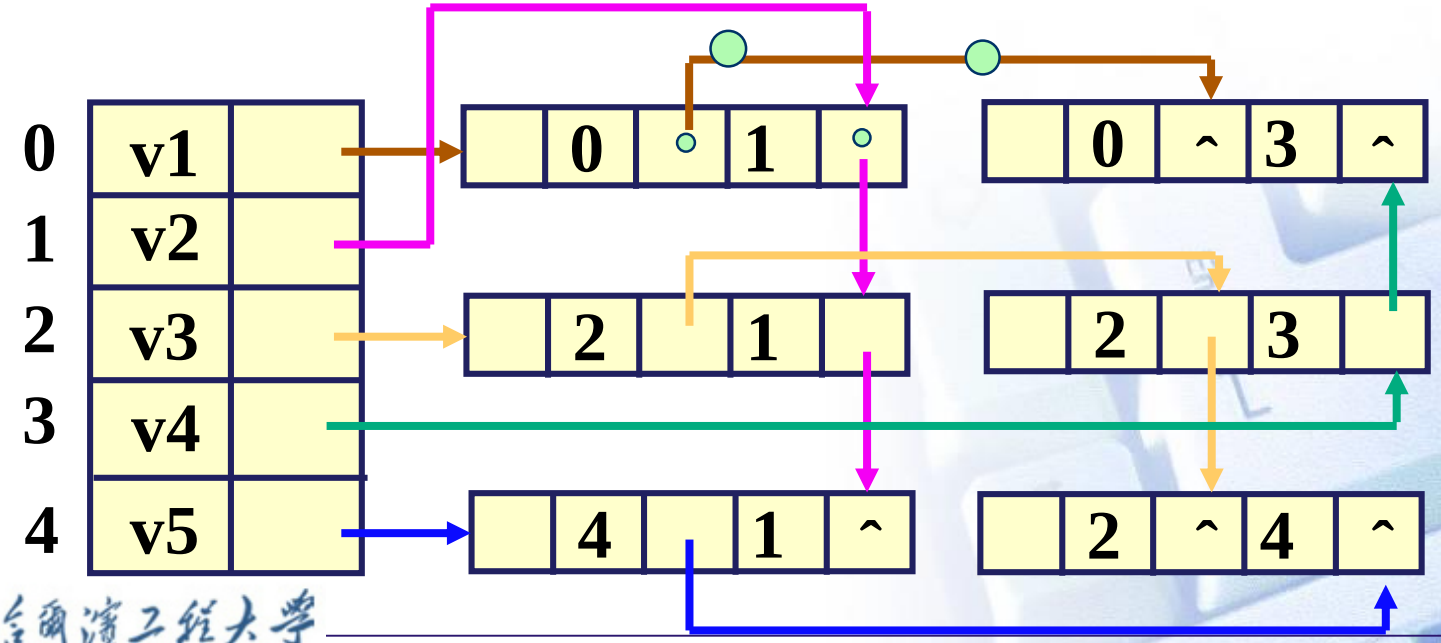
图的存储结构　　无向图的邻接多重表存储表示

例



指向下一个依附于v1的边

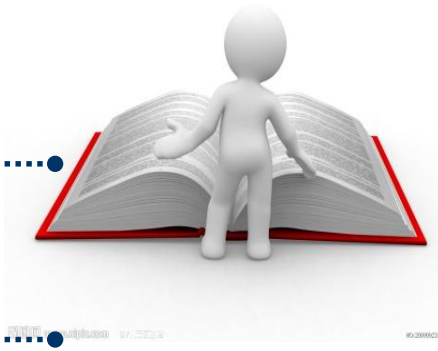
指向下一个依附于v2的边



- 无向图邻接多重表存储表示

```
#define MAX_VERTEX_NUM  20
Typedef emnu {unvisited, visited} VisitIf;
typedef   struct  ArcBox          //弧结点
{   VisitIf              mark;
    int                  ivex, jvex;
    struct  EBox         *ilink, *jlink;
    infoType             *info;
}EBox;
Typedef struct VexBox          //顶点结点
{   VertexType  data;
    EBox          *firstedge;  //指向第1条依附该顶点的边
}VexBox;
Typedef struct {                    //顶点表
    VexBox  adjlist[MAX_VERTEX_NUM];
    int          vexnum, arcnum;
}AMLGraph;
```

本章内容



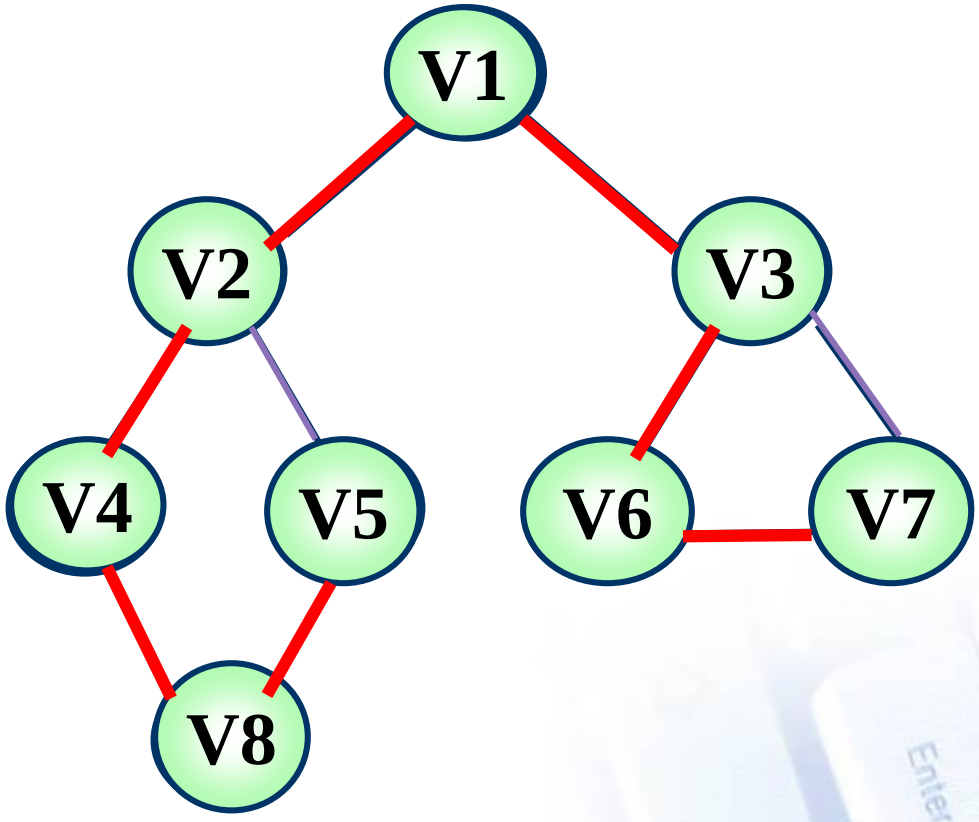
1. 图的定义和术语
2. 图的存储结构
3. 图的遍历
4. 生成树
5. 拓扑排序
6. 最短路径

- 从图的某顶点出发，对图中的每个顶点进行一次访问且使每个顶点仅被访问一次的过程
- 可以从图中任意一个顶点出发进行遍历
  - 遍历中需解决的问题
    - 确定一条搜索路径（按照一定原则）
    - 确保每个顶点被访问到
    - 确保每个顶点只能被访问一次

- 深度优先遍历
- 广度优先遍历

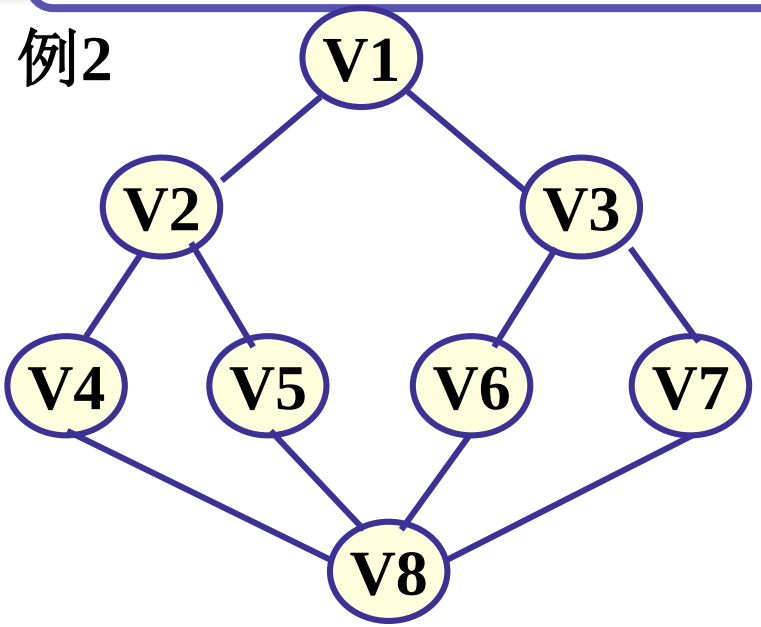
- 思想
  - 从图的某一顶点V_0出发，访问该顶点；然后依次从V_0的未被访问的邻接点出发，深度优先遍历图，直至图中所有和V0相通的顶点都被访问到
    - 访问任意一个与V_0邻接的顶点W_1，再从W_1出发；
    - 访问与W_1邻接且未被访问过的任意顶点W_2，再从W_2出发
    - 重复以上过程，直到一个所有邻接点都被访问过的顶点为止
    - 退回到尚有邻接点未被访问过的顶点，再从该顶点出发
    - 直到所有的被访问过的顶点的邻接点都已被访问过为止
  - 若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止

例1



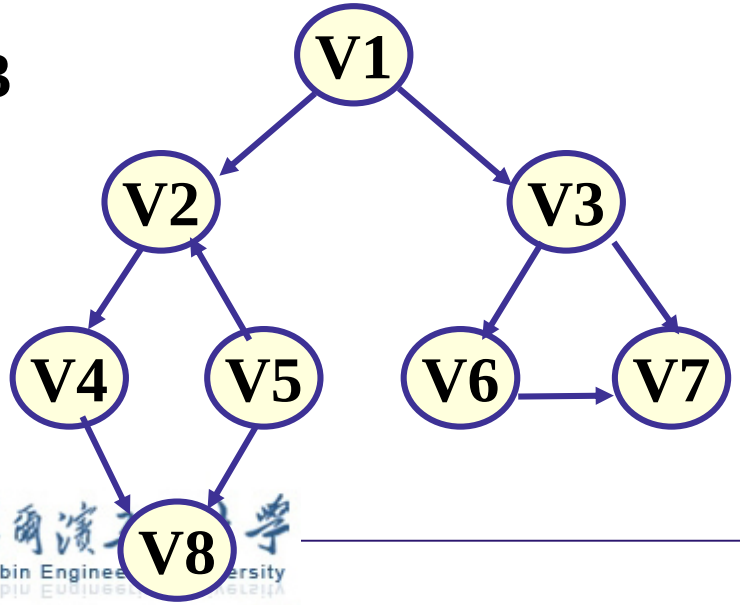
深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V3 ⇒V6 ⇒V7

例2



例2：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V6 ⇒V3 ⇒V7

例3



例3：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V3 ⇒V6 ⇒V7 ⇒V5

从上面例可见:

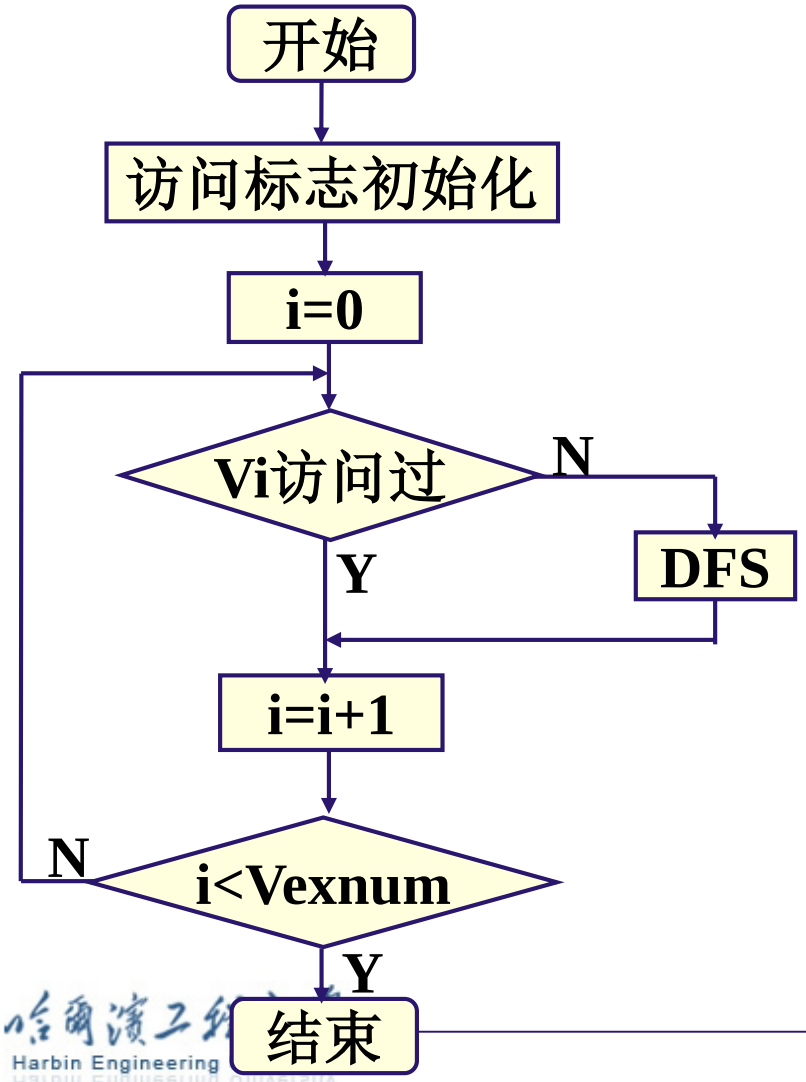
1. 从深度优先搜索遍历连通图的过程类似于树的先根遍历;

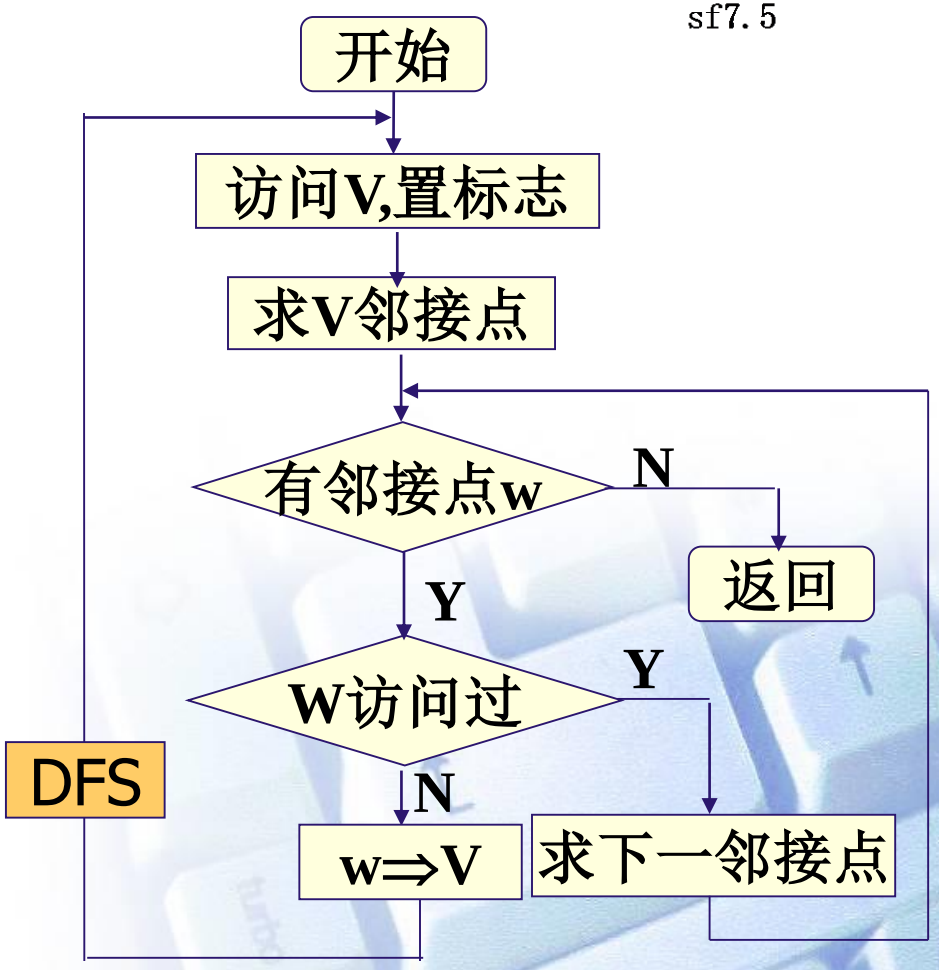
2. 如何判别V的邻接点是否被访问?

解决的办法是:为每个顶点设立一个“访问标志 visited[w]”;

- 深度优先遍历算法


sf7.5

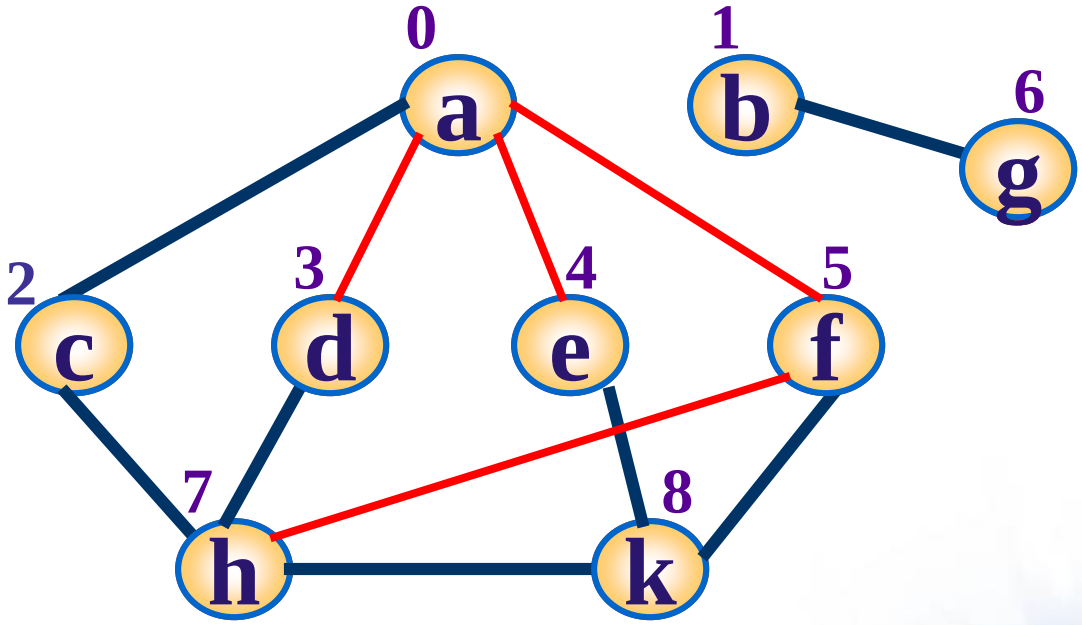

开始
访问标志初始化
i=0
Vi访问过
N
Y
DFS
i=i+1
i<Vexnum
N
Y
结束

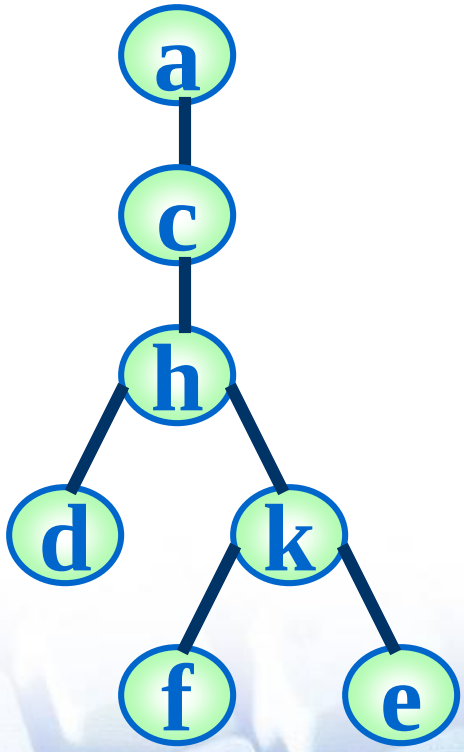

开始
访问V,置标志
求V邻接点
有邻接点w
N
返回
Y
W访问过
Y
N
w⇒V
求下一邻接点
DFS

深度优先遍历算法

```
void DFSTraverse(Graph G,   Status (*Visit)(int v))
{   // 对图 G 作深度优先遍历。
    VisitFunc = Visit;
    for (v=0; v<G.vexnum; ++v)
      visited[v] = FALSE;              // 访问标志数组初始化
    for (v=0; v<G.vexnum; ++v)
      if (!visited[v])  DFS(G, v);     // 对尚未访问的顶点调用DFS
}

void DFS(Graph G, int v)
{   // 从顶点v出发，深度优先搜索遍历连通图 G
    visited[v] = TRUE;   VisitFunc(v);
    for(w=FirstAdjVex(G, v); w!=0; w=NextAdjVex(G,v,w))
       if (!visited[w])  DFS(G, w);  //w成为新的v
            // 对v的尚未访问的邻接顶点w
            // 递归调用DFS，实现回退
} // DFS
```

例如:



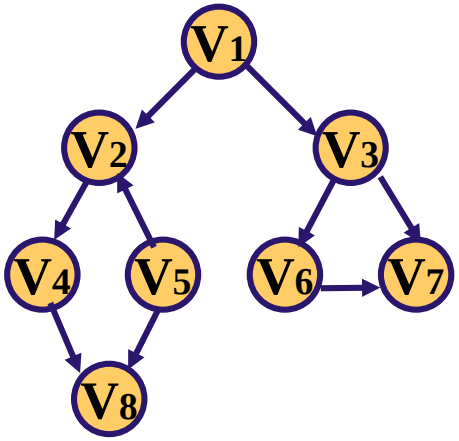


访问标志:

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

访问次序: a c h d k f e b g

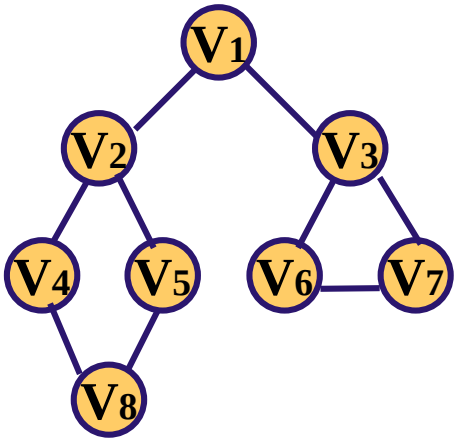
- 深度优先遍历



	data	firstarc	adjvex	next
1	V1	→	3	→ 2 ∧
2	V2	→	4 ∧	
3	V3	→	7	→ 6 ∧
4	V4	→	8 ∧	
5	V5	→	8	→ 2 ∧
6	V6	→	7 ∧	
7	V7	∧		
8	V8	∧		

深度遍历：V1⇒V3⇒ V7⇒ V6⇒ V2⇒ V4⇒ V8⇒ V5

- 深度优先遍历



	data	firstarc	adjvex	next
1	V1	→	3	→ 2 ∧
2	V2	→	5	→ 4 → 1 ∧
3	V3	→	7	→ 6 → 1 ∧
4	V4	→	8	→ 2 ∧
5	V5	→	8	→ 2 ∧
6	V6	→	7	→ 3 ∧
7	V7	→	6	→ 3 ∧
8	V8	→	5	→ 4 ∧

深度遍历：V1⇒V3⇒ V7⇒ V6⇒ V2⇒ V5⇒ V8⇒ V4

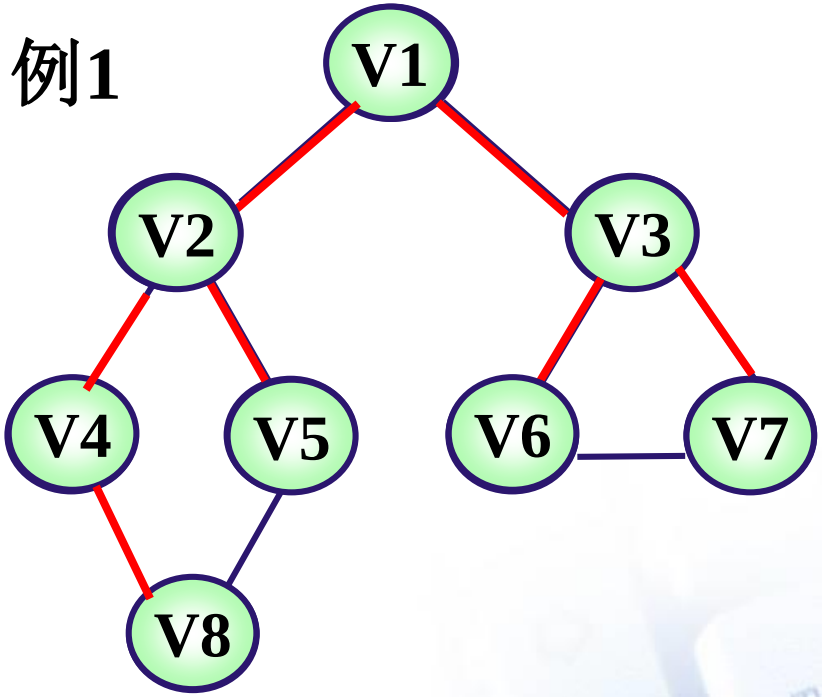
广度优先遍历

基本思想

Click

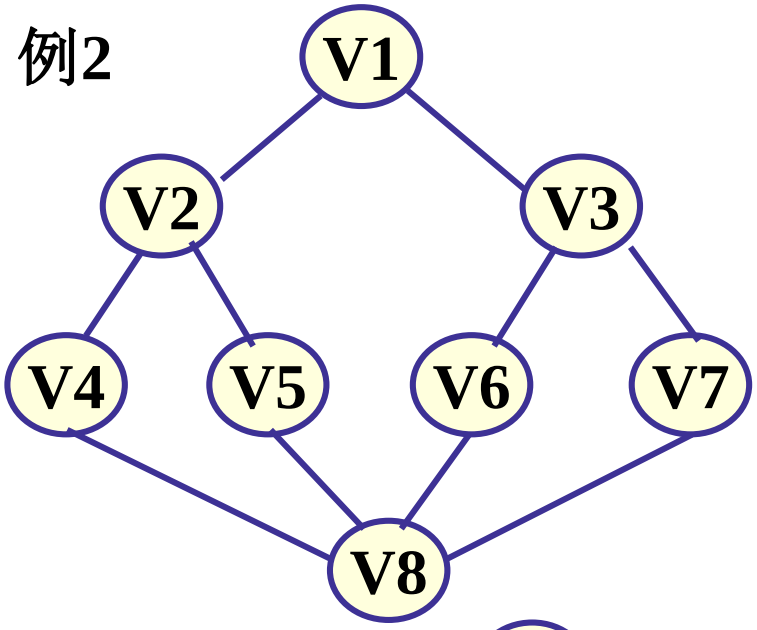
- 从图的某一顶点V_0出发，访问该顶点后，依次访问V_0的各个未被访问过的邻接点；然后分别从这些邻接点出发，广度优先遍历图，直至图中所有已被访问的顶点的邻接点都被访问到
- 若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止

例1



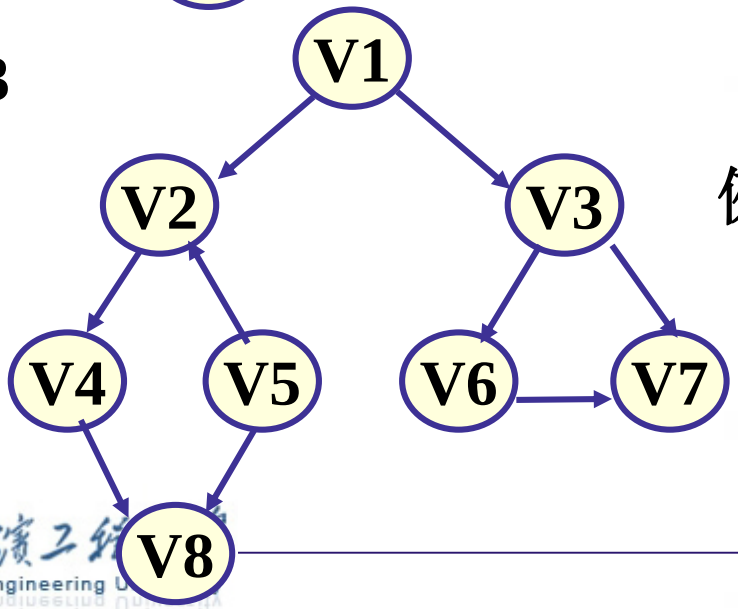
广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

例2



例2：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

例3

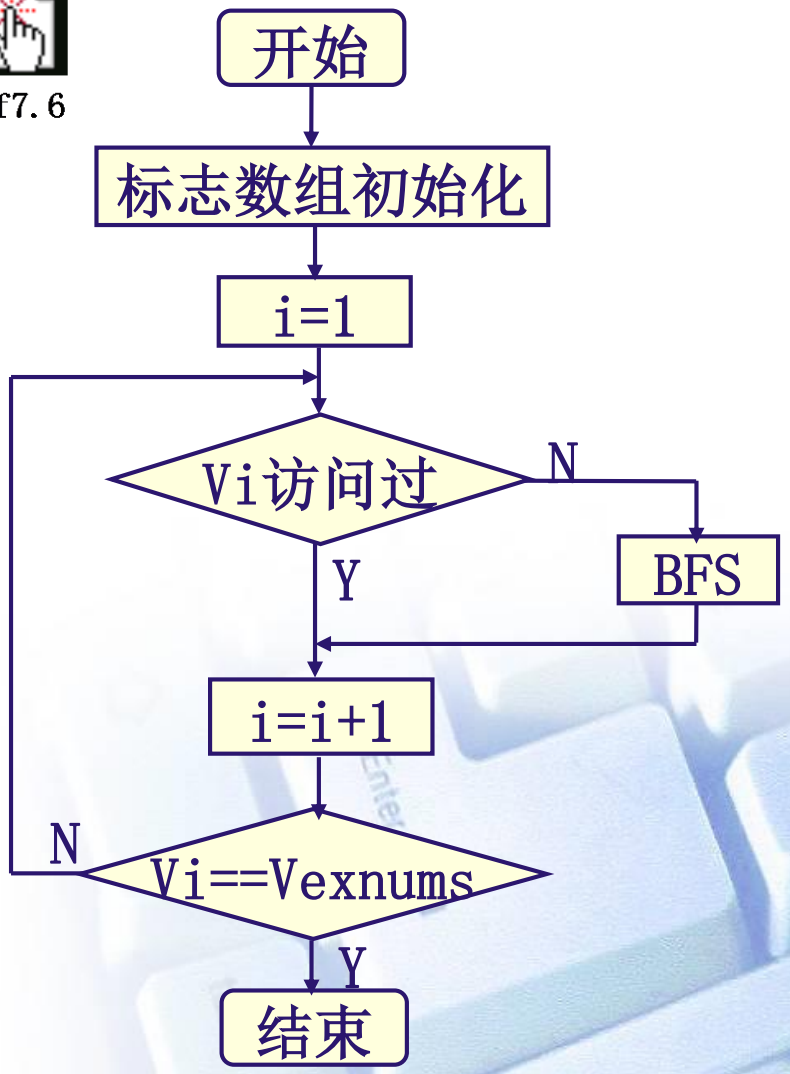


例3：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V6 ⇒V7 ⇒V8 ⇒V5

- 广度优先遍历算法

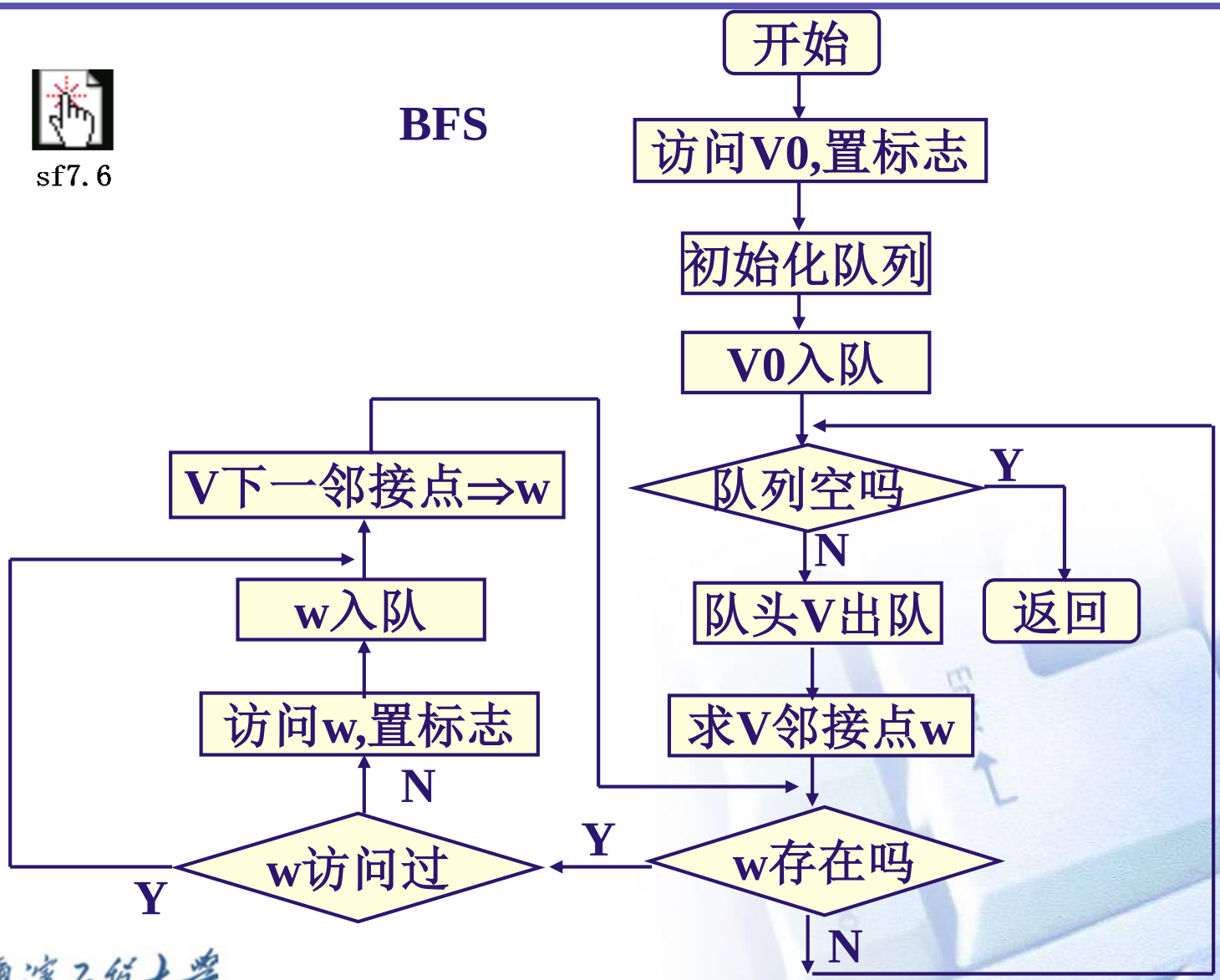
  思想：如何让每个结点仅访问一次，访问标志、借助队列

sf7.6



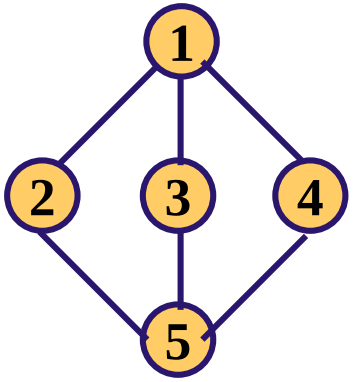
sf7.6

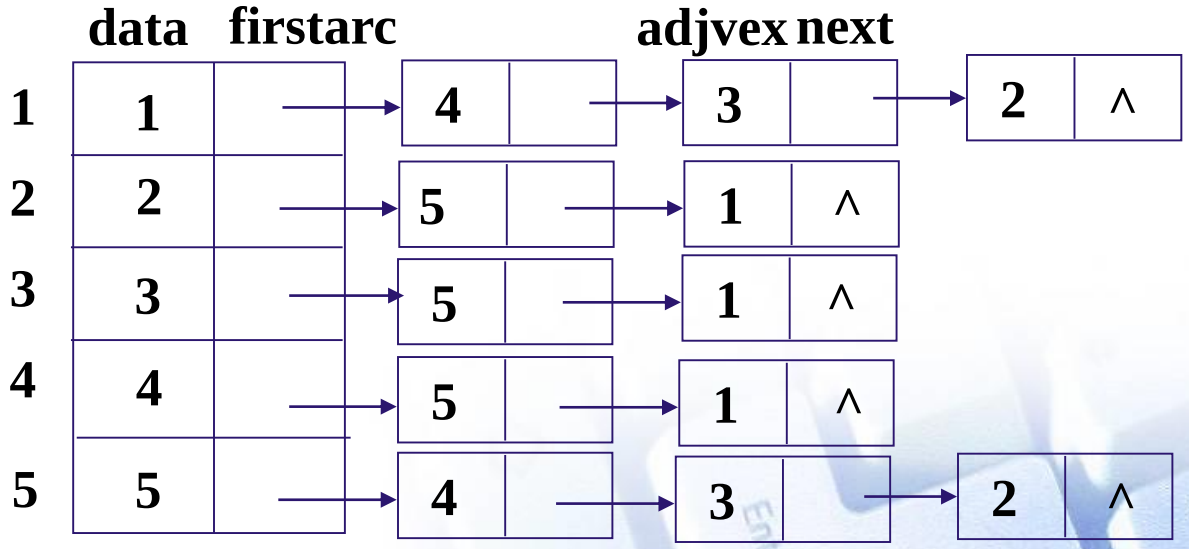
BFS


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

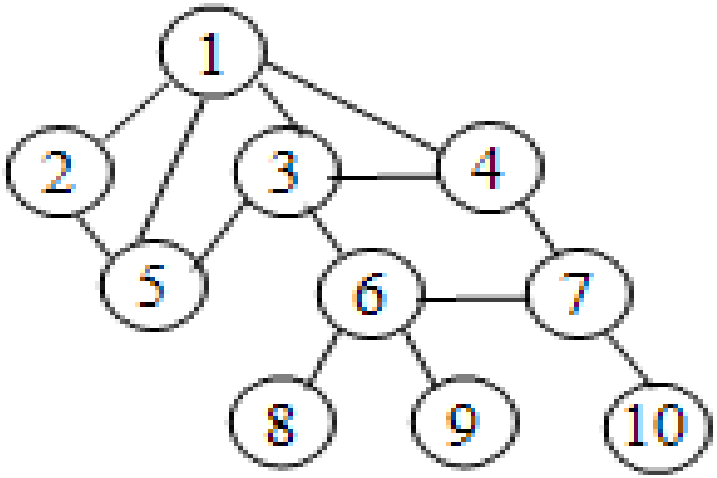
广度优先遍历算法

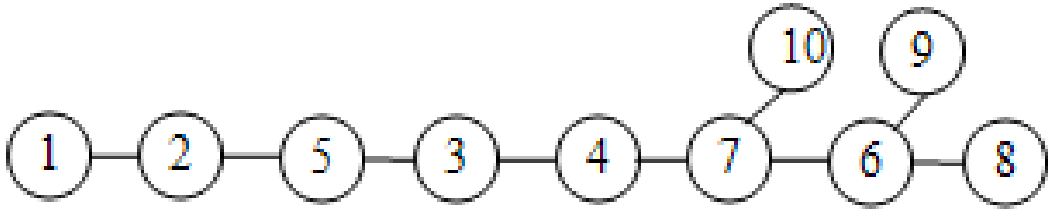
```
void BFSTraverse(Graph G,  Status (*Visit)(int v))
{   for (v=0; v<G.vexnum; ++v)
        visited[v] = FALSE;  //初始化访问标志
    InitQueue(Q);       // 置空的辅助队列Q
    for ( v=0;  v<G.vexnum;  ++v )
        if ( !visited[v])                          // v 尚未访问，保证访问所有节点
          {
            visited[v] = TRUE;  Visit(v);    // 访问u
            EnQueue(Q, v);                     // v入队列
            while (!QueueEmpty(Q))
                {  DeQueue(Q, u);              // 队头元素出队并置为u
                   for (w=FirstAdjVex(G, u); w!=0; w=NextAdjVex(G,u,w)) // 访问所有u未
                                                                        访问的邻接点
                       if ( ! visited[w])
                         { visited[w]=TRUE;  Visit(w);
                           EnQueue(Q, w); // 访问的顶点w入队列
                          } // if
                } // while
          } //if
  } // BFSTraverse
```

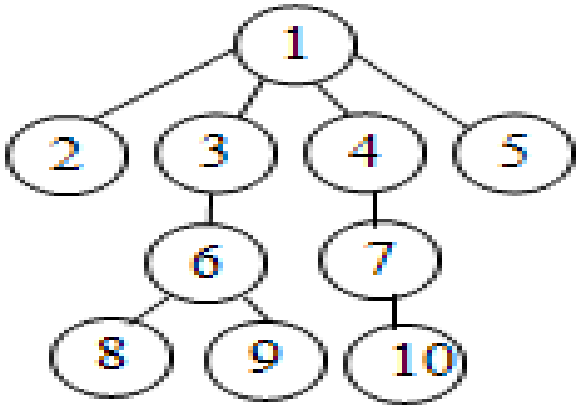

1
2
3
4
5


data
firstarc
adjvex
next
1
2
3
4
5
1
2
3
4
5
4
3
2
^
5
1
^
5
1
^
5
1
^
4
3
2
^

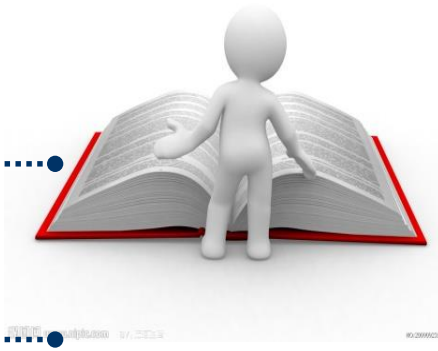
- 给出图G：以顶点1为根，画出G的深度优先生成树和广度优先生成树。



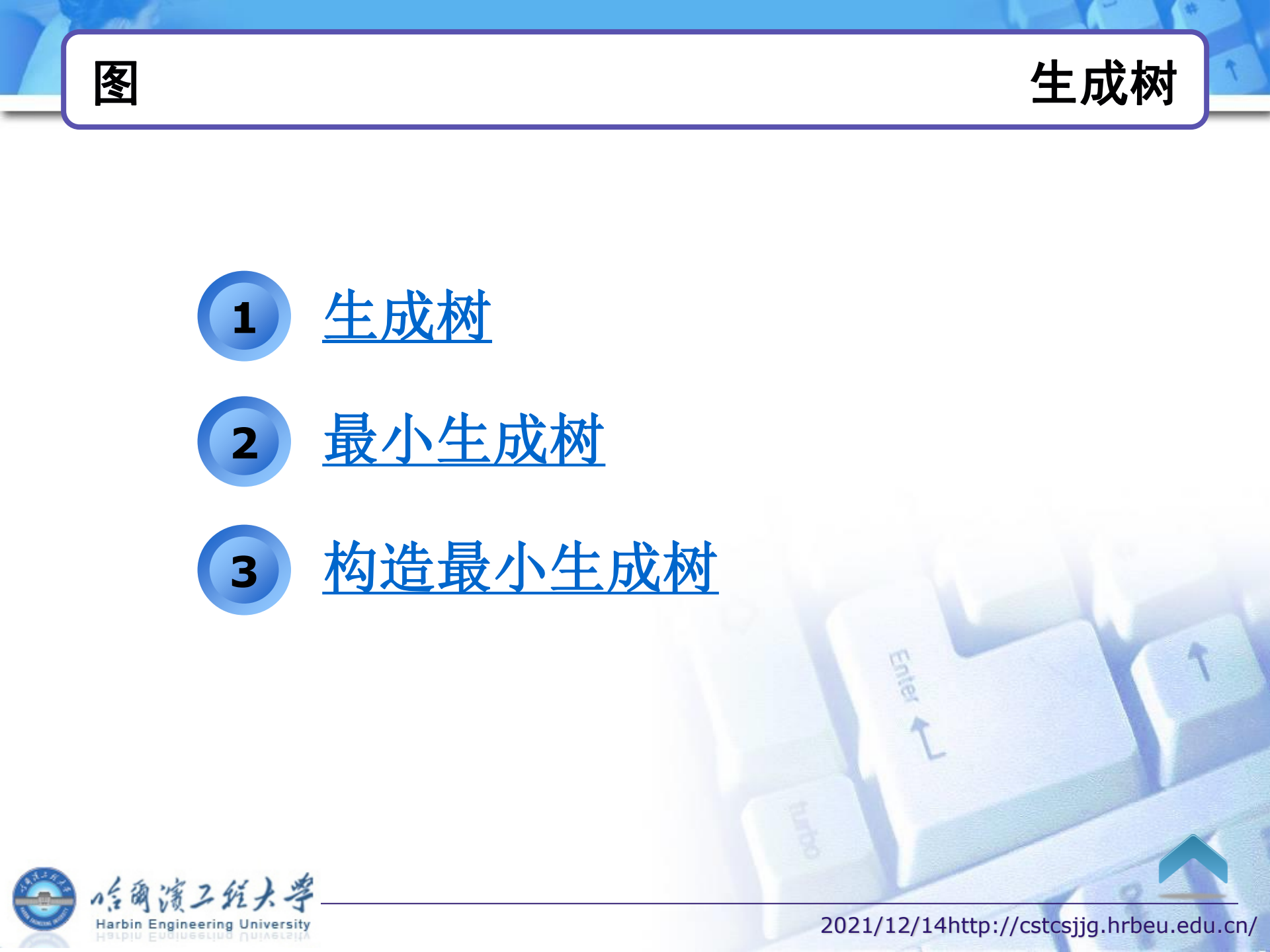

10
9
1
2
5
3
4
7
6
8


1
2
3
4
5
6
7
8
9
10

本章内容



1. 图的定义和术语
2. 图的存储结构
3. 图的遍历
4. 生成树
5. 拓扑排序
6. 最短路径



生成树 所有（n个）顶点均由边（n-1个）连接在一起，但不存在回路的图

深度优先生成树 连通图由深度优先遍历得到的生成树

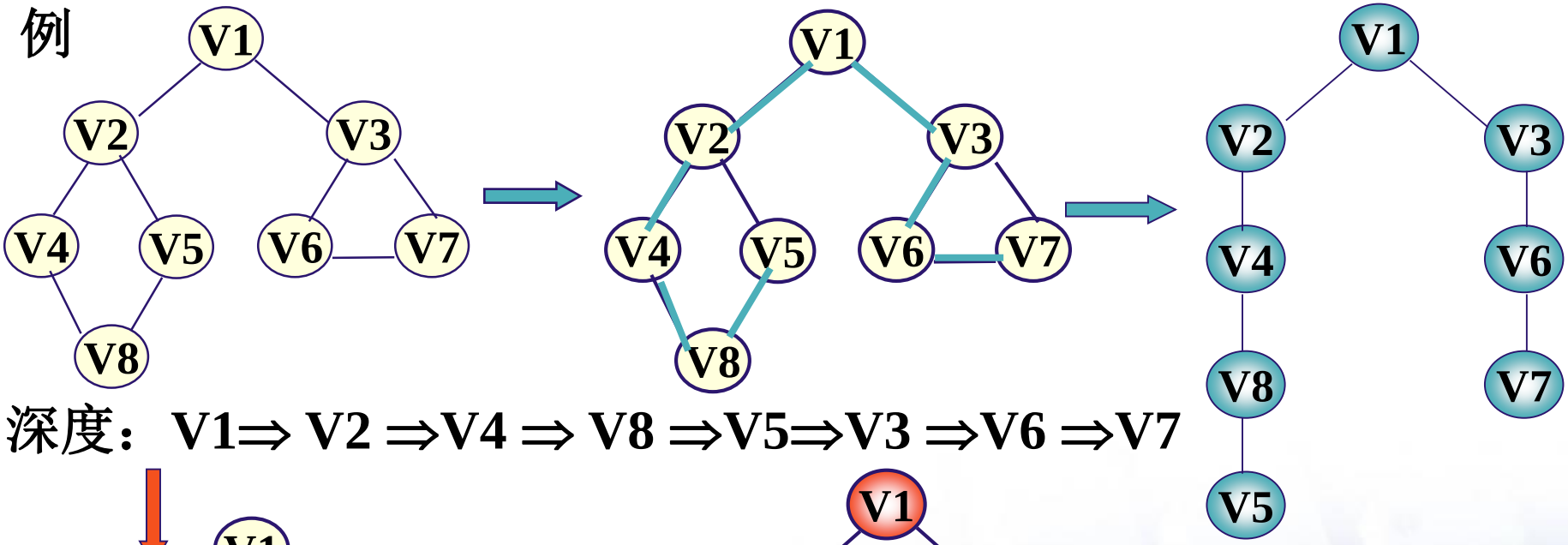
广度优先生成树 连通图由广度优先遍历得到的生成树

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生成森林

注 意

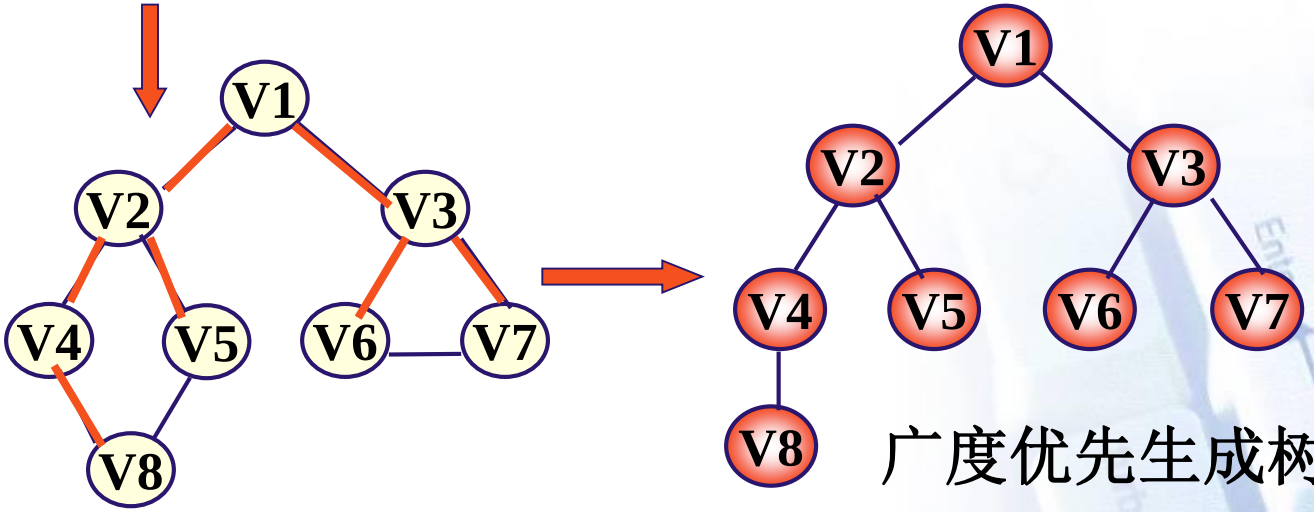
- 一个图可以有许多棵不同的生成树
- 所有生成树具有以下共同特点：
  - 生成树的顶点个数与图的顶点个数相同
  - 生成树是图的极小连通子图
  - 一个有n个顶点的连通图的生成树有n-1条边
  - 生成树中任意两个顶点间的路径是唯一的
  - 在生成树中再加一条边必然形成回路
- 含n个顶点n-1条边的图不一定是生成树

例



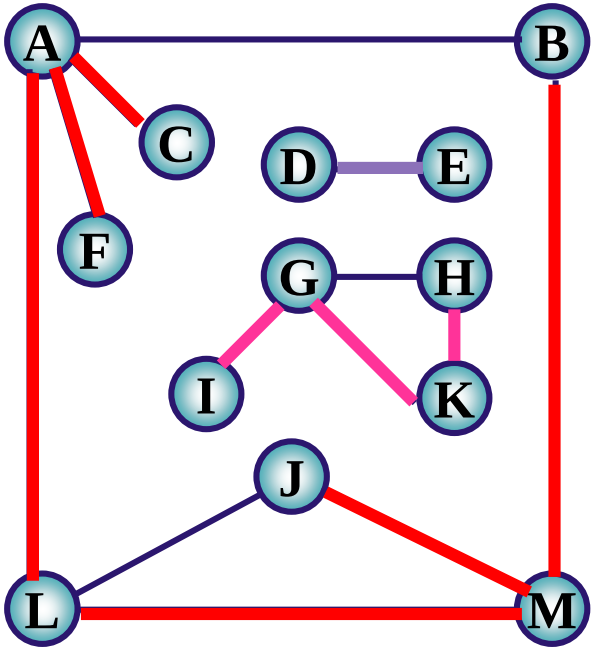
深度：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5⇒V3 ⇒V6 ⇒V7

深度优先生成树



广度优先生成树

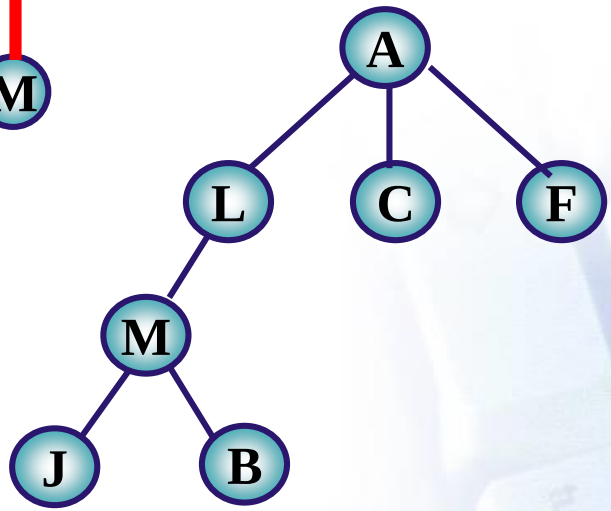
广度：V1⇒V2 ⇒V3 ⇒ V4 ⇒V5⇒V6 ⇒V7 ⇒V8

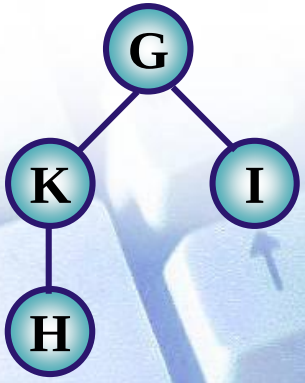


深度优先遍历：ALMJBFC

DE

GKHI





深度优先生成森林

问 题

假设要在 *n* 个城市之间建立通讯联络网，则连通 *n* 个城市只需要修建 *n-1*条线路，如何在最节省经费的前提下建立这个通讯网？

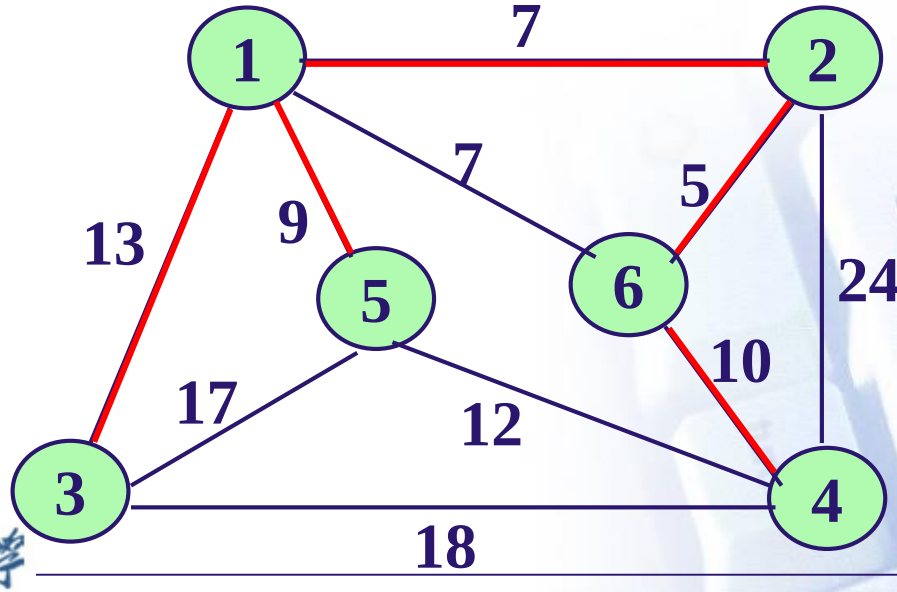
- 顶点——表示城市
- 权——城市间建立通信线路所需花费代价
- 最小（代价）生成树：希望找到一棵生成树，它的每条边上的权值之和(即建立该通信网所需花费的总代价）最小

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

该问题等价于：构造无向网的一棵最小生成树，即：在e条带权的边中选取n-1条边（不构成回路），使“权值之和”为最小。



方法一　克鲁斯卡尔算法（选边法）

思　想　为使生成树上总的权值之和达到最小，则应使每一条边上的权值尽可能地小，自然应从权值最小的边选起，直至选出 n-1 条互不构成回路的权值最小边为止。

方法：N=(V,{E})是连通图

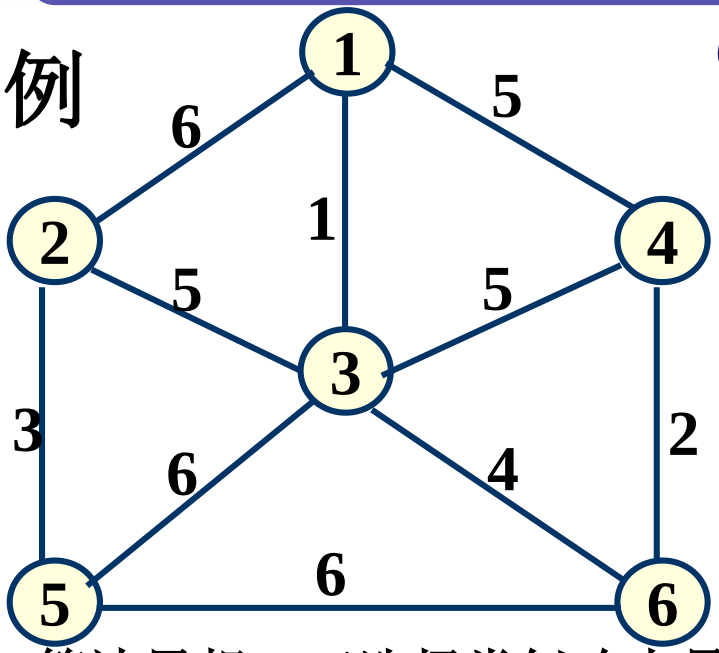
（1）初始状态为只有n个顶点而无边的非连通图T=(V,{Φ})，每个顶点自成一个连通分量

（2）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

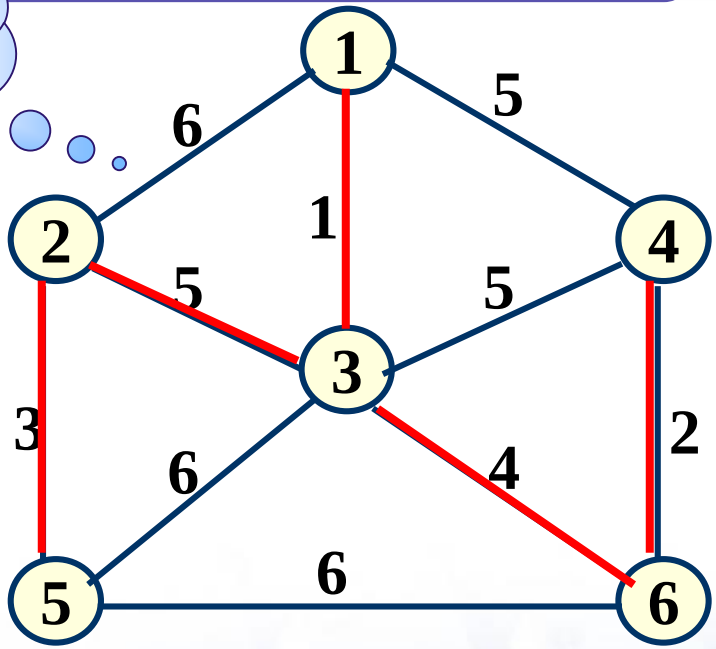
（3）依此类推，直至T中所有顶点都在同一连通分量上为止

生成树　　构造最小生成树

例



编程如何实现？



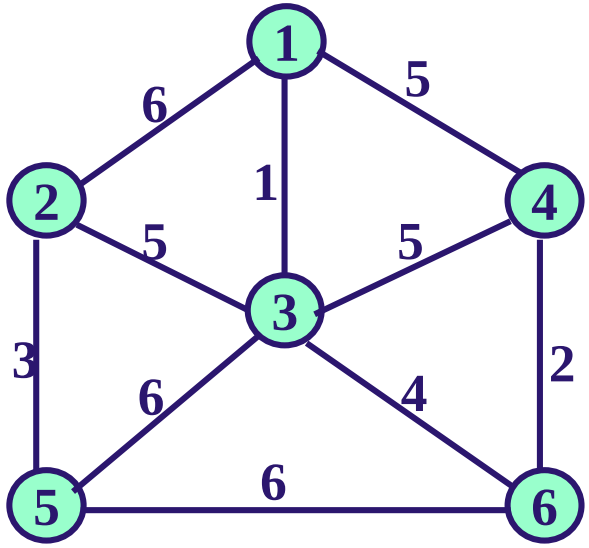
算法思想：（选择类似哈夫曼树存储结构）

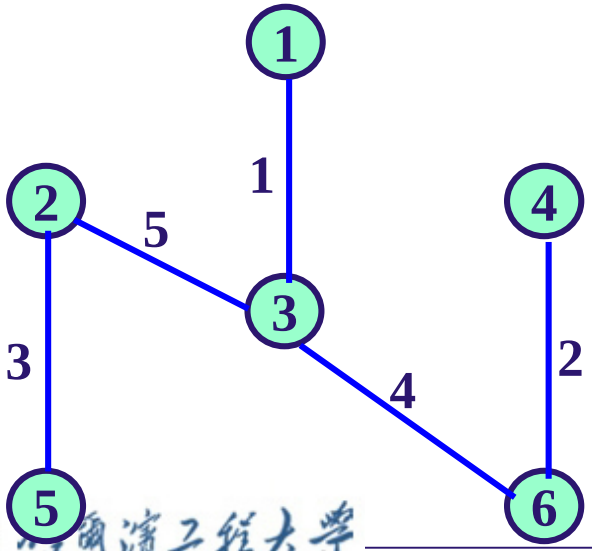
（1）将n个点当作n个子集{a}{b}{c}…，用t[]存储；边及权重用e[]存储

（2）选一条权最小的边(a,b)，若a和b∈不同子集，如a∈Vi，b ∈Vj，则将Vi和Vj合并，选中(a,b)，否则，舍去(a,b)，再选下一条权最小的边

（3）形成一个连通分量（集合）结束，否则转（2）

生成树　　构造最小生成树





t[]

	data	set
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

e[]

	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

```
void minitree_KRUSKAL(     )
{   int i, j, n,m,min,k;
    VEX t[M];
    EDGE  e[M];
    printf(“Input number of vertex and edge:”);
    scanf(“%d,%d”,&n,&m);//输入顶点数n、边数m
    for(i=1;i<=n;i++){  //初始化t[ ]
          printf(“t[%d].data=:”,i);
          scanf(“%d”,&t[i].data);
          t[i].set=i;}
    for(i=0;i<m;i++){   //初始化e[ ]
      printf("vexh,vext,weight:");
      scanf("%d,%d,%d",&e[i].vexh,&e[i].vext,&e[i].weight);
      e[i].flag=0;}
     i=1;  //计数最小生成树的边数
```

```
while(i<n) //边数应等于n-1，n为顶点
{   min=MAX;
      for(j=0;j<m;j++)  //选取当前未访问中代价最小的边
         if(e[j].weight<min && e[j].flag==0)
                { min=e[j].weight;  k=j; }   //找到代价最小的边，编号K
      if(t[e[k].vexh].set!=t[e[k].vext].set)
       { e[k].flag=1;

         for(j=1;j<=n;j++)  //合并可能形成新的连通分量
                if(t[j].set==t[e[k].vext].set) //合并与弧尾节点相同的集合
                    t[j].set=t[e[k].vexh].set; //以弧头节点所在集合为准
         i++;
       }
      else      e[k].flag=2; //标识访问过但不是需要的边
}
for(i=0;i<m;i++)
  if(e[i].flag==1)
 printf("%d,%d :%d\n",e[i].vexh,e[i].vext,e[i].weight);
}
```

时间复杂度为*O*（n×e）

方法二　普里姆(Prim)算法（选点法）

思　想　首先选取图中任意一个顶点 v 作为生成树的根，之后继续往生成树中添加顶点 w，则在顶点 w 和顶点 v 之间必须有边，且该边上的权值应在所有和 v 相邻接的边中最小且没有回路

方法：　TE是N上最小生成树中边的集合

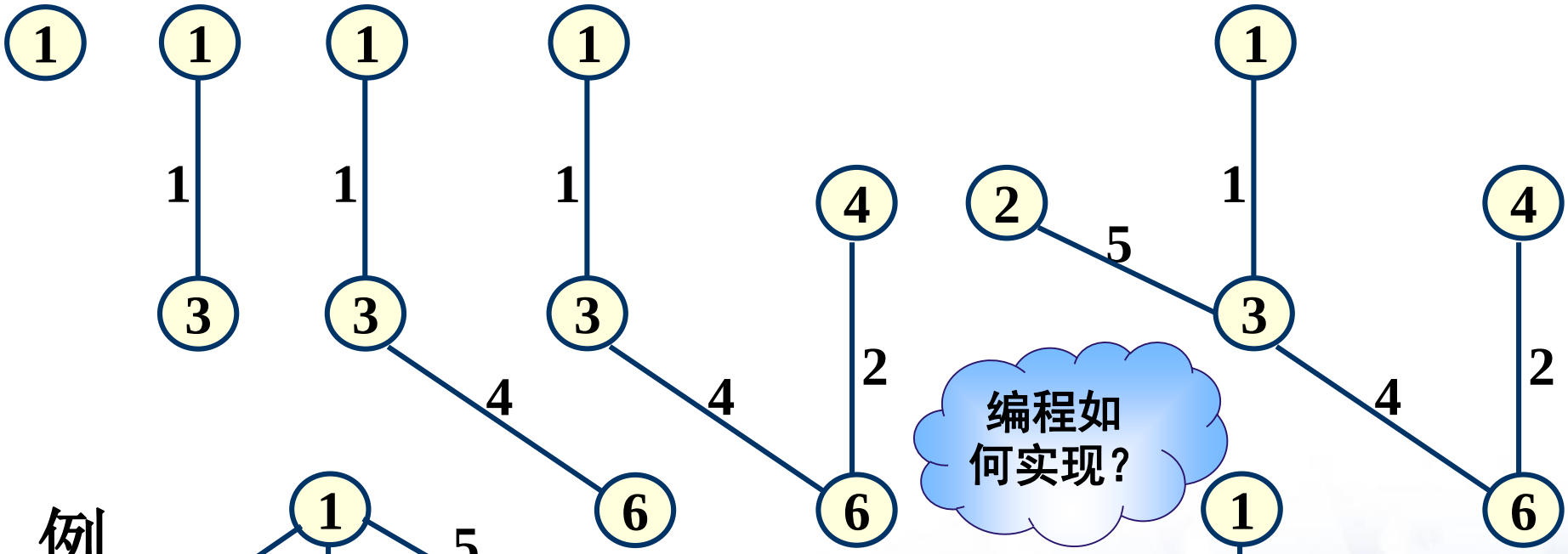
（1）初始令U={u_0},(u_0∈V), TE=Φ

（2）在所有u∈U,v∈V-U的边(u,v)∈E中，找一条代价最小的边(u_0,v_0)，且 没有回路的

（3）将(u_0,v_0)并入集合TE，同时v_0并入U

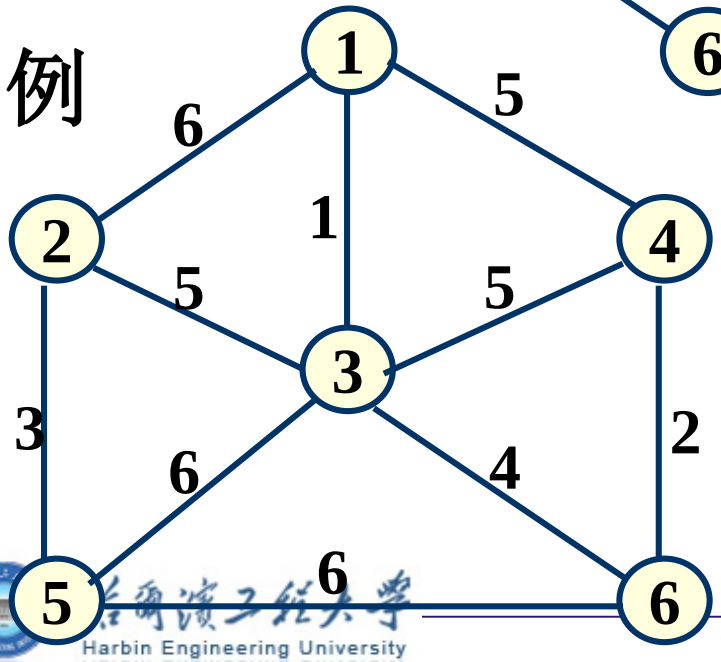
（4）重复上述操作直至U=V为止（选点），则 T=(V,{TE})为N的最小生成树

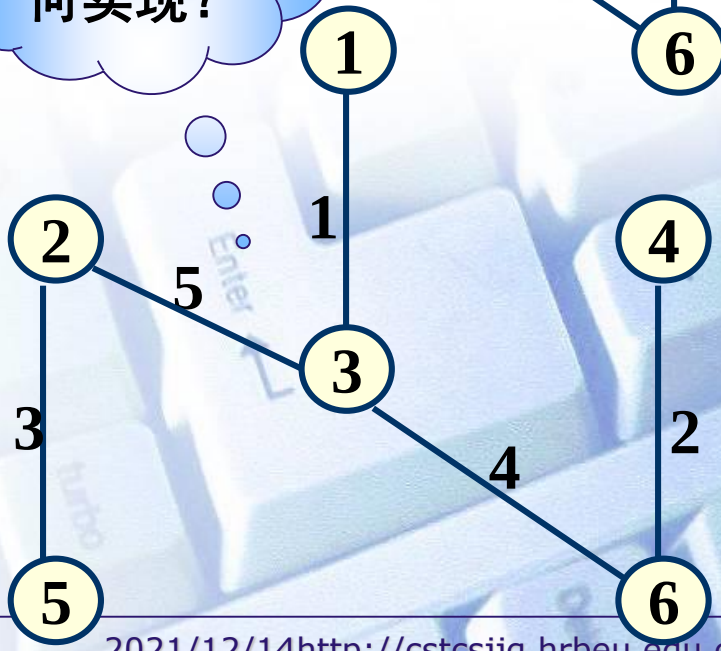
生成树　　构造最小生成树



编程如何实现？

例



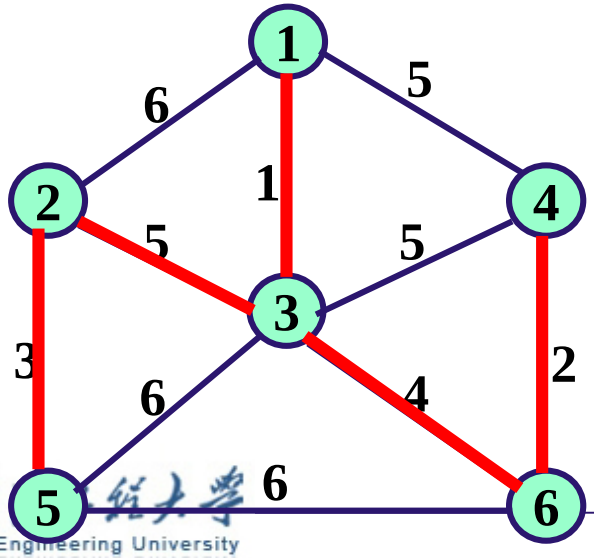


- 图的存储结构：邻接矩阵表示
- 设一辅助数组closedge，记录从U到V-U具有最小代价的边。每个顶点vi∈V-U，在辅助数组中存在一个相应的分量，closedge[i-1]，其中lowcost存储该边上的权

Closedge[i-1].lowcost=min{cost(u,vi)|u∈U}

Adjvex域存储该边依附的在U中的顶点

- 算法实现： sf7.9 //与vi相邻的最小边<u,vi>
- 算法评价： O(n²)



	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

生成树　　构造最小生成树

i-1	V2	V3	V4	V5	V6			
Closedge \ i	1	2	3	4	5	U	V−U	K
adjvex	V1	V1	V1			{V1}	{V2, V3, V4, V5, V6}	2
lowcast	6	1	5					
adjvex	V3		V1	V3	V3	{V1, V3}	{V2, V4, V5, V6}	5
lowcast	5	0	5	6	4			
adjvex	V3		V6	V3		{V1, V3, V6}	{V2, V4, V5}	3
lowcast	5	0	2	6	0			
adjvex	V3			V3		{V1, V3, V6. V4}	{V2, V5}	1
lowcast	5	0	0	6	0			
adjvex				V2		{V1, V3, V6. V4, V2}	{V5}	4
lowcast	0	0	0	3	0			
adjvex						{V1, V3, V6 . V4, V2, V5}	{}	
lowcast	0	0	0	0	0			

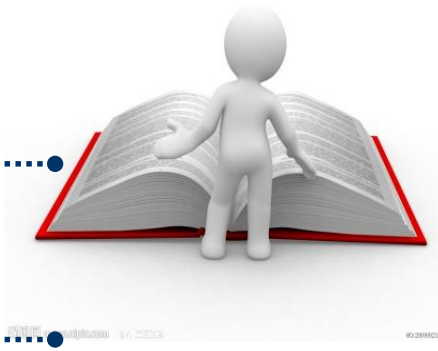
```
void MiniSpanTree_PRIM(MGraph G, VertexType u)
{  //用普里姆算法从第u个顶点出发构造网G的最小生成树T，输出T各边
   //记录从顶点集U到V-U的代价最小的边的辅助数组定义
   // struct
      {
   //     VertexType   adjvex;
   //     VRType       lowcost;
   //  }closedge[MAX_VERTEX_NUM];

    k=LocateVex(G,u);
   for (j=0; j<G.vexnum; j++)      //辅助数组初始化
       if (j!=k)  closedge[j]={u, G.arcs[k][j].adj};  //{adjvex, lowcost}
   closedge[k].lowcost=0;              //初始化，U={u}
```

```
for (i=0; i<G.vexnum; i++)     //选择其余G.vexnum-1个顶点
 {  k=minimum(closedge); //选权最小的边，对应顶点的位置号k
        //此时closedge[k].lowcost=
      //MIN{closedge[vi].lowcost | closedge[vi].lowcost>0,vi属V-U}
    printf(closedge[k].adjvex, G.vexs[k]);  //输出生成树的边
    closedge[k].lowcost=0;        //第k顶点并入U集，置标志0
      for (j=0; j<G.vexnum; j++)
        if (G.arcs[k][j].adj<closedge[j].lowcost) //新顶点并入后
            closedge[j]= { G.vexs[k],G.arcs[k][j].adj } ;
            //重选最小边（当前的顶点、权值）
       }
}//MiniSpanTree
```

比较两种算法

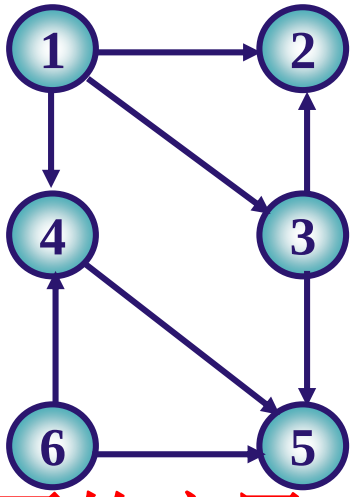
算法名	普里姆算法	克鲁斯卡尔算法
时间复杂度	$O(n^2)$	$O(e*n)$
适应范围	稠密图	稀疏图

本章内容



1. 图的定义和术语
2. 图的存储结构
3. 图的遍历
4. 生成树
5. 拓扑排序
6. 最短路径

- 有向无环图：一个无环的有向图（DAG）



- 应用
  - 在工程计划和管理方面有着广泛而重要的应用
  - 描述一项工程或系统的进行进程的有效工具
  - 对整个工程和系统，人们关心的是两方面的问题：一是工程能否顺利进行；二是完成整个工程所必须的最短时间。对应到有向图即为进行拓扑排序和求关键路径。

- 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，这意味着某项活动不以自己为先决条件

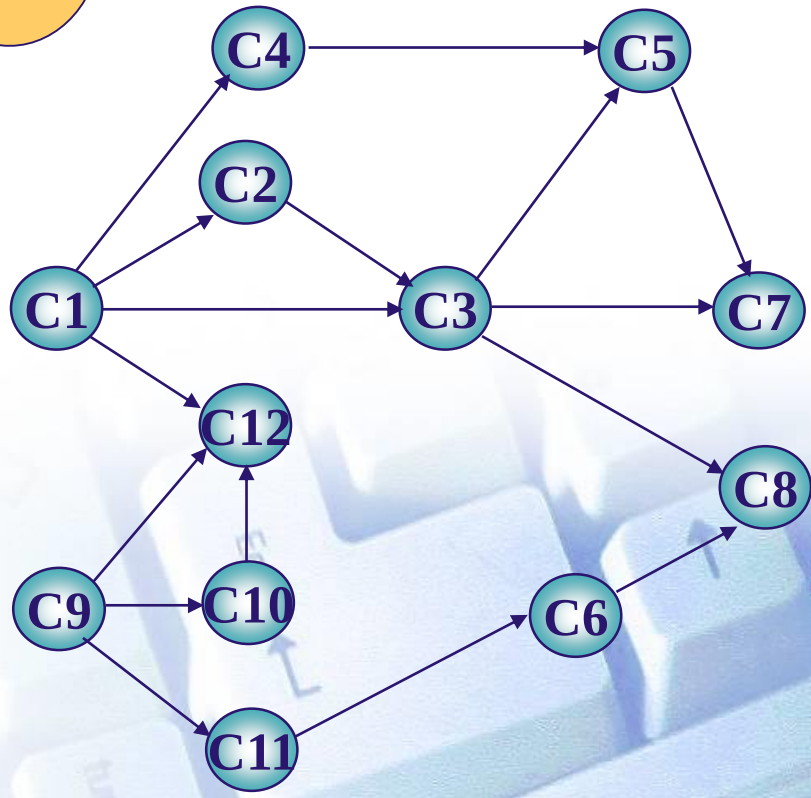
- 拓扑排序：把AOV网络中各顶点按照它们相互之间的优先关系排列成一个线性序列的过程
  - 如何检测AOV网中是否存在环方法?
  - 不存在环，一个工程可顺利完成；否则，该工程无法顺利完成
- 拓扑排序的方法
  - 在有向图中选一个入度为0的顶点且输出它
  - 从图中删除该顶点和所有以它为尾的弧
  - 重复上述两步，直至全部顶点均已输出；拓扑排序顺利完成。否则，若剩有入度非0的顶点，说明图中有环，拓扑排序不能进行

拓扑排序

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？

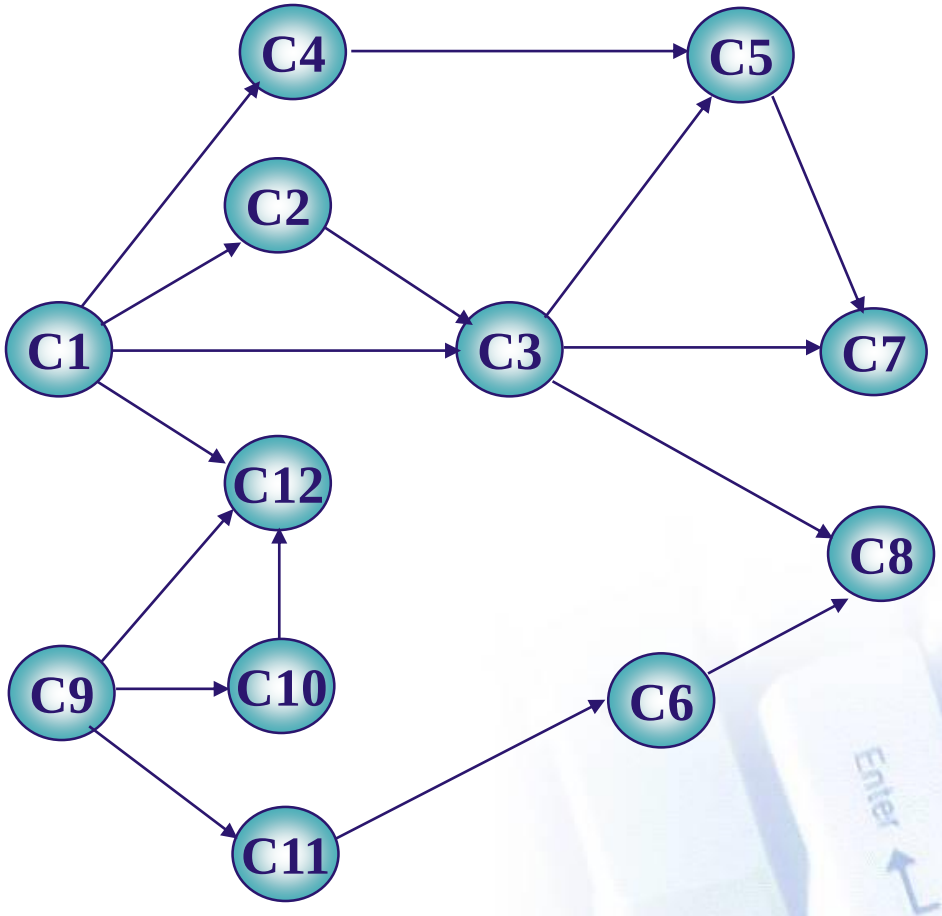
例

课程代号	课程名称	先
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



拓扑排序

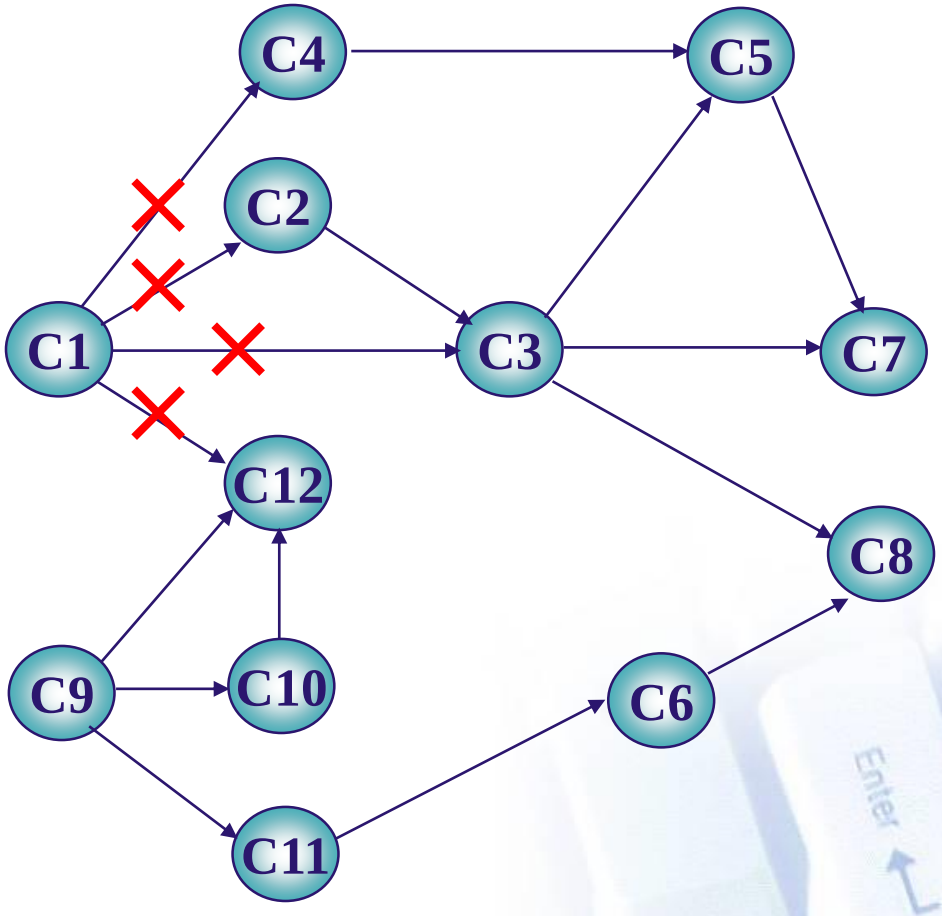
拓扑排序



拓扑序列：C1

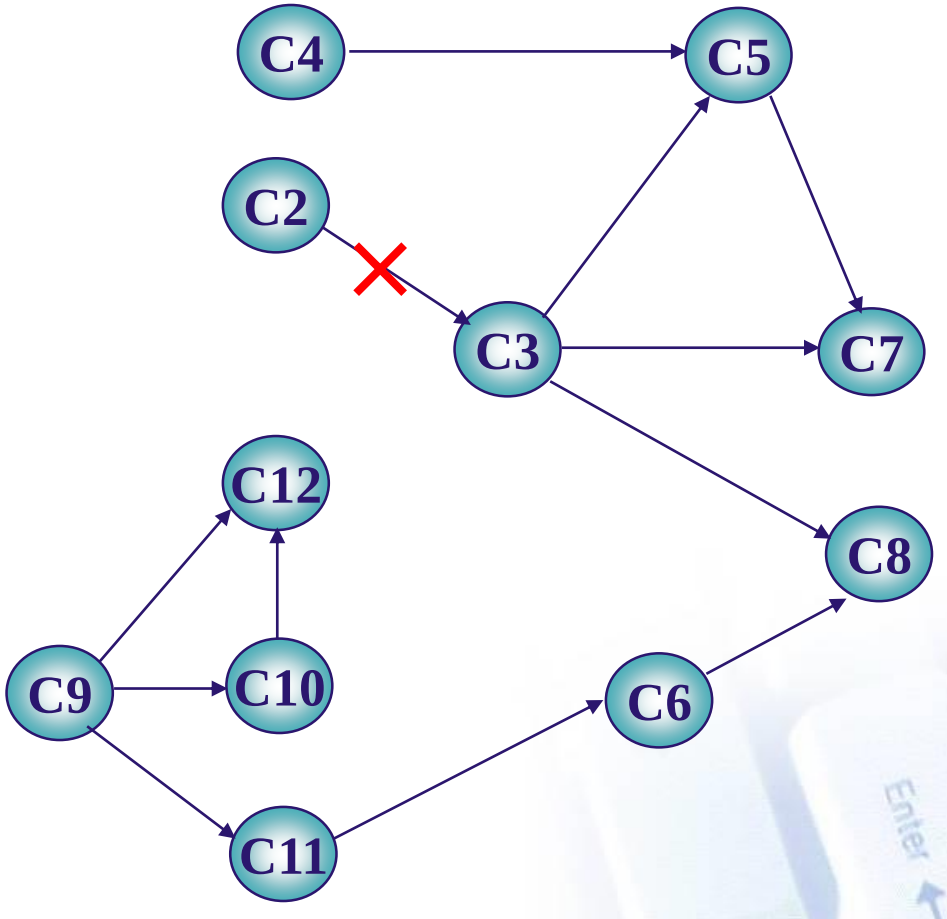
拓扑排序

拓扑排序



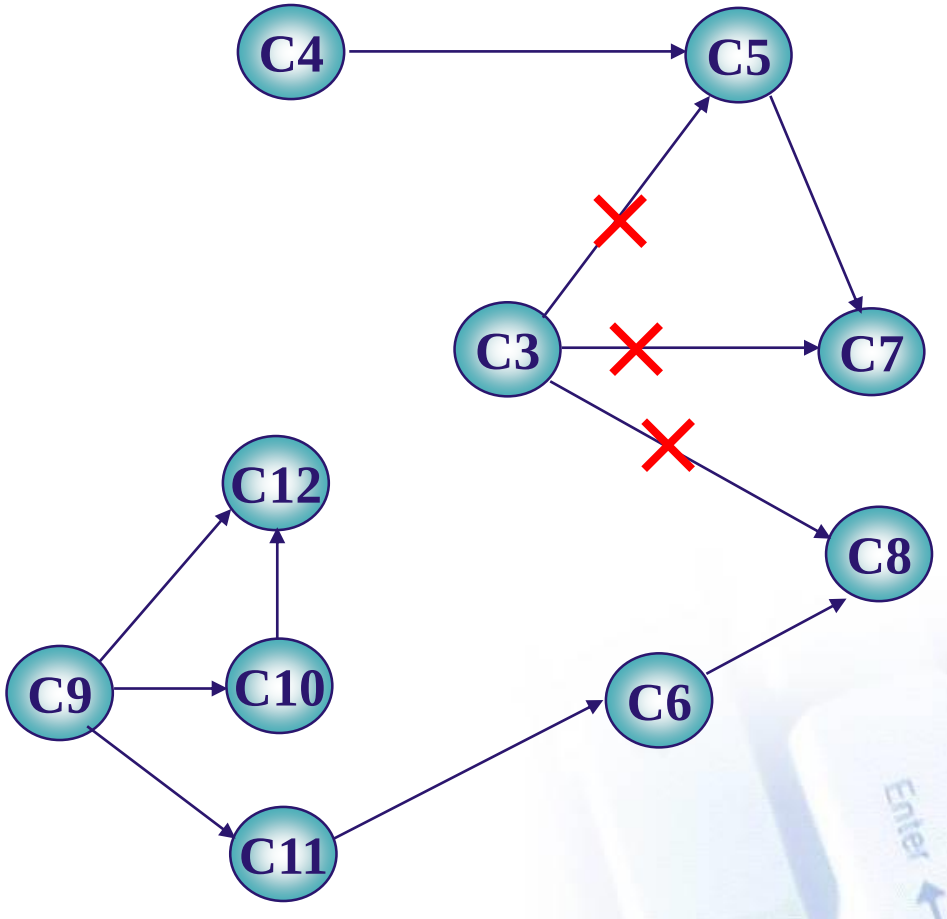
拓扑序列：C1

拓扑排序



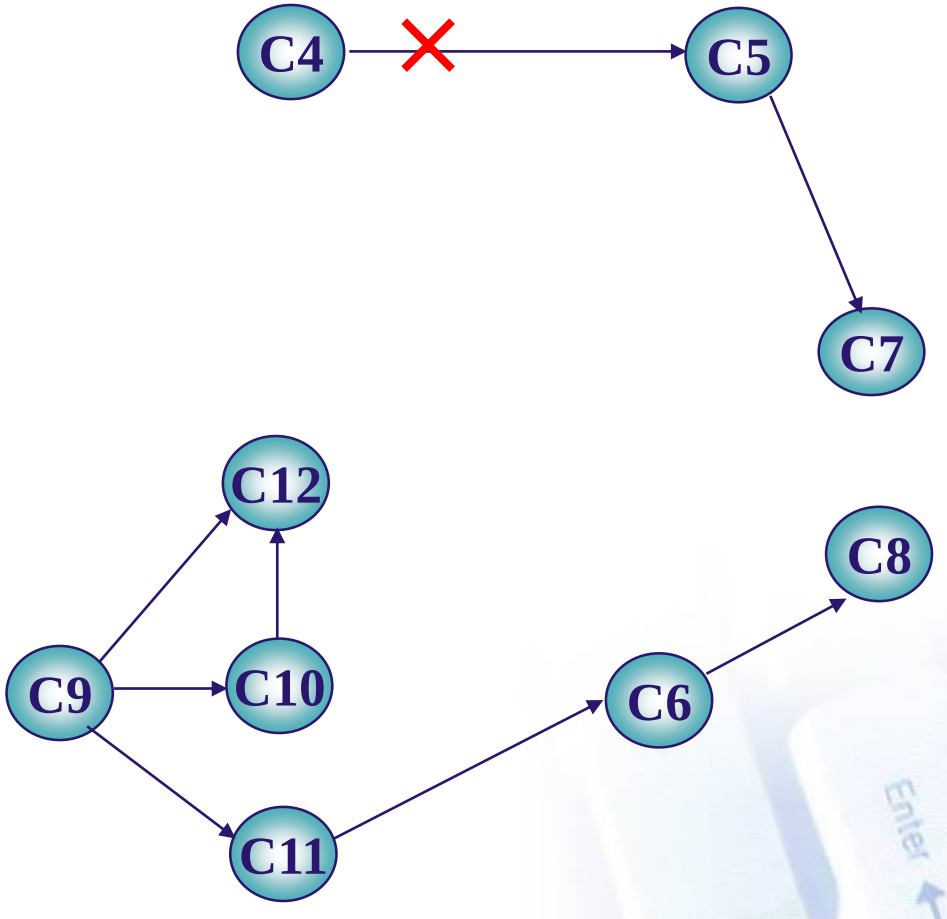
拓扑序列：C1 C2

拓扑排序



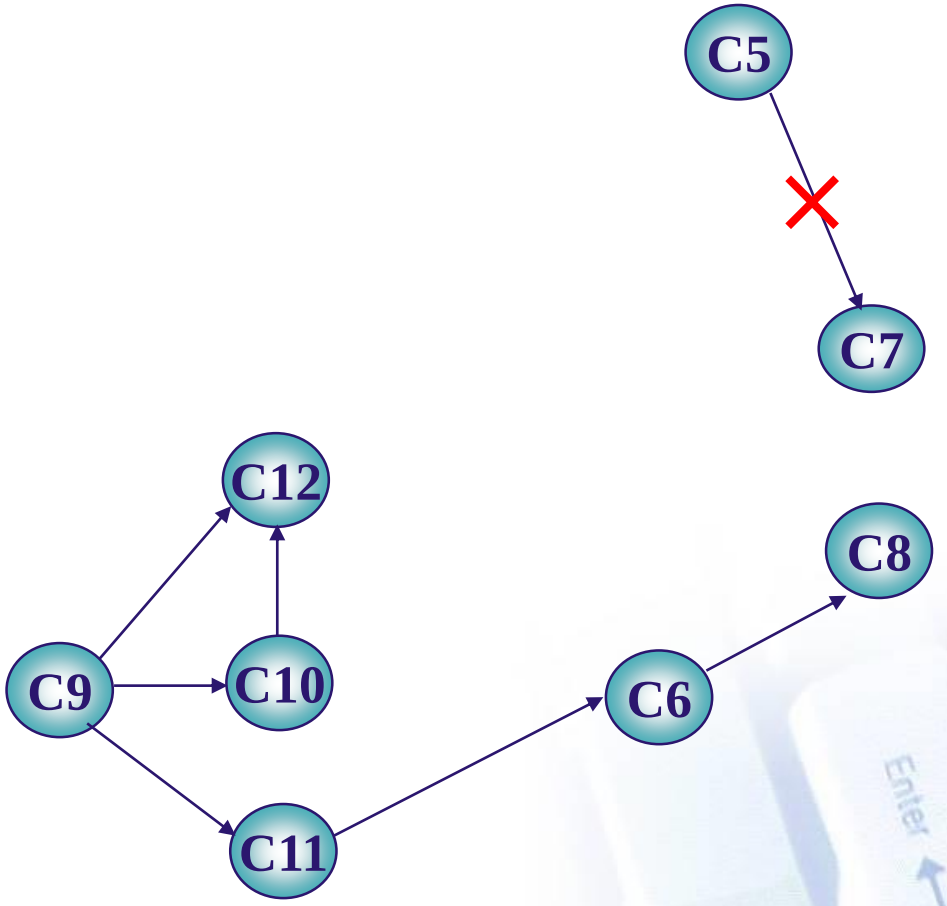
拓扑序列：C1 C2 C3

拓扑排序



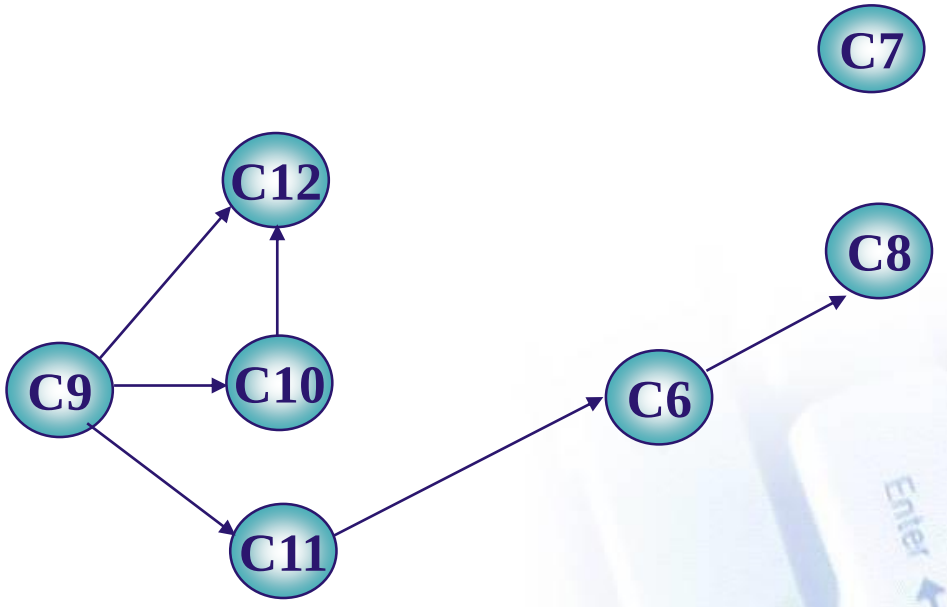
拓扑序列：C1 C2 C3 C4

拓扑排序



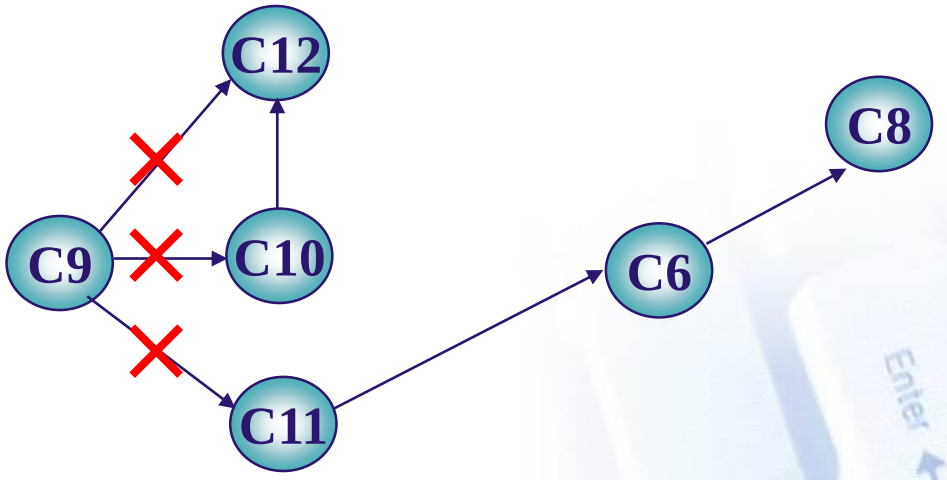
拓扑序列：C1 C2 C3 C4 C5

拓扑排序



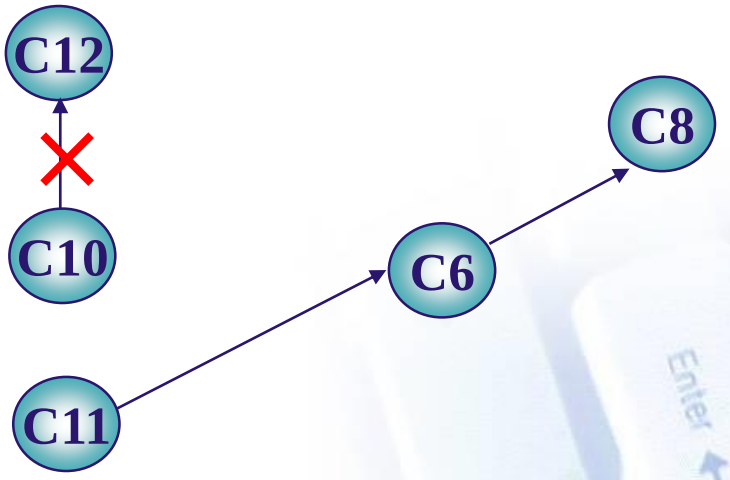
拓扑序列：C1 C2 C3 C4 C5 C7

拓扑排序



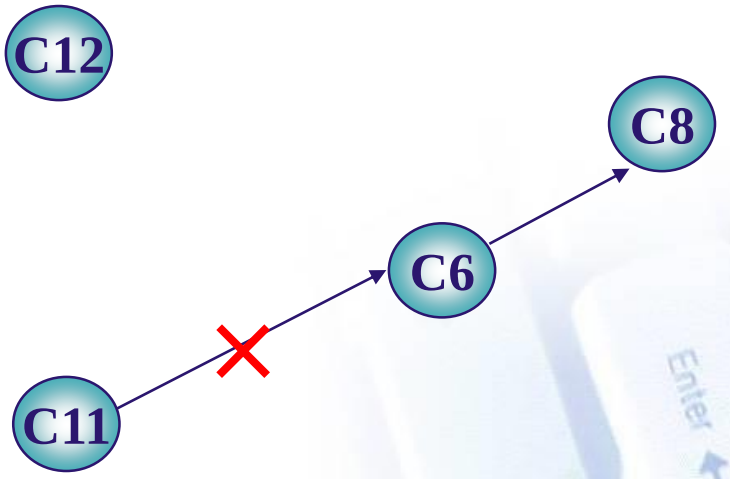
拓扑序列：C1 C2 C3 C4 C5 C7 C9

拓扑排序



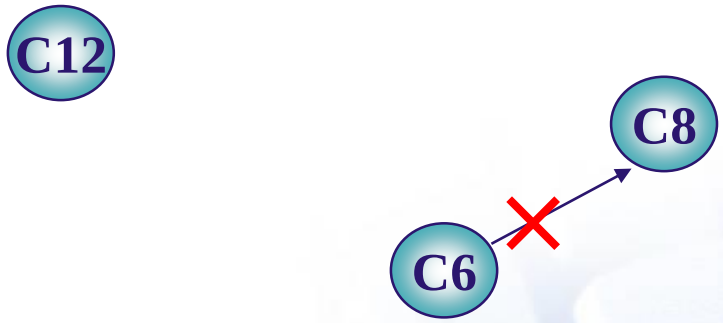
拓扑序列：C1 C2 C3 C4 C5 C7 C9 C10

拓扑排序



拓扑序列：C1 C2 C3 C4 C5 C7 C9 C10 C11

拓扑排序



拓扑序列：C1 C2 C3 C4 C5 C7 C9 C10 C11 C6

拓扑排序

C12

C8

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

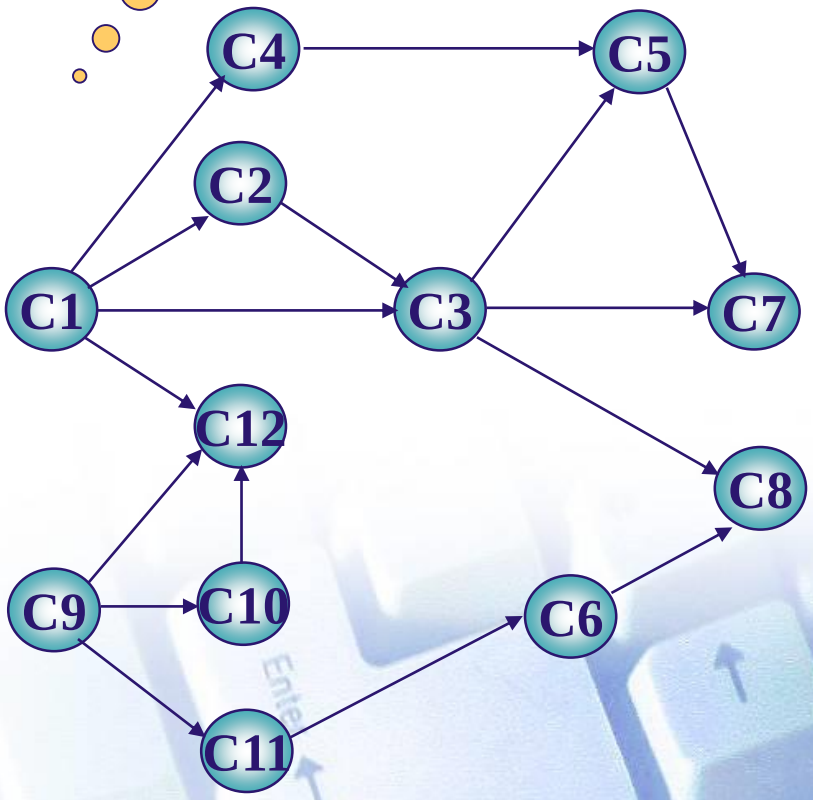
拓扑排序

拓扑排序

编程如何实现？

例

课程代号	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

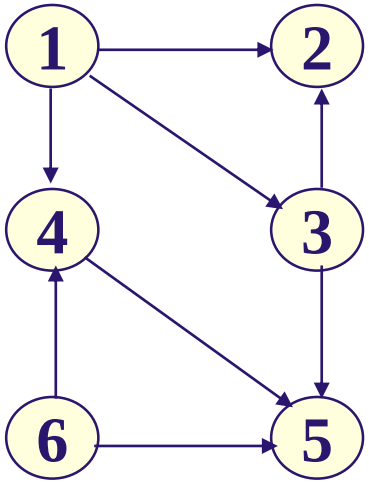


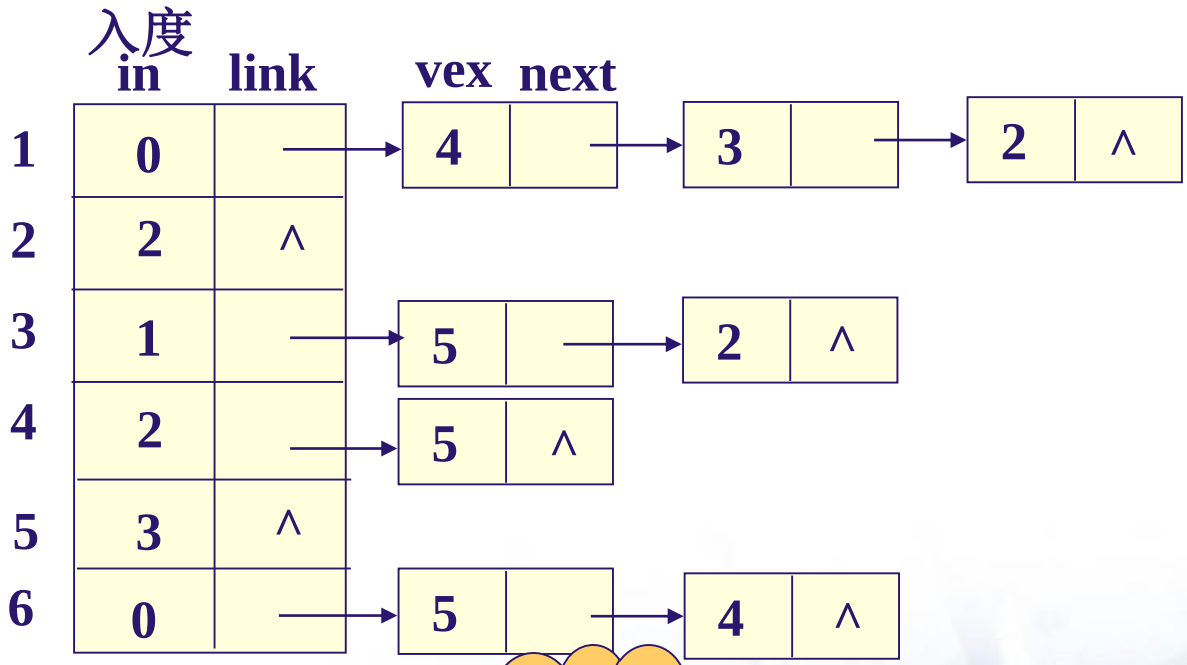
拓扑序列：C1 C2 C3 C4 C5 C7 C9 C10 C11 C6 C8 C12

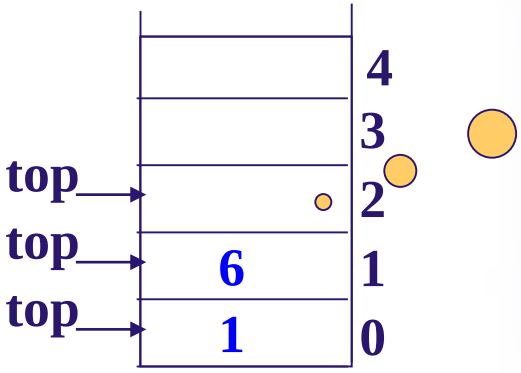
拓扑排序

- 算法思想

例

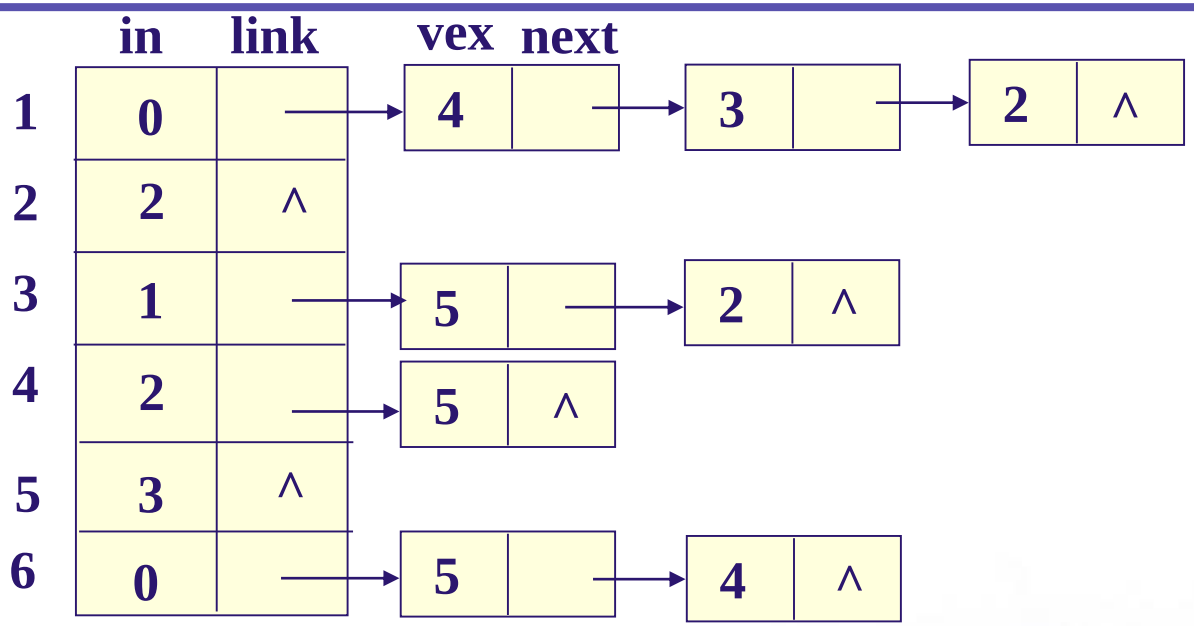


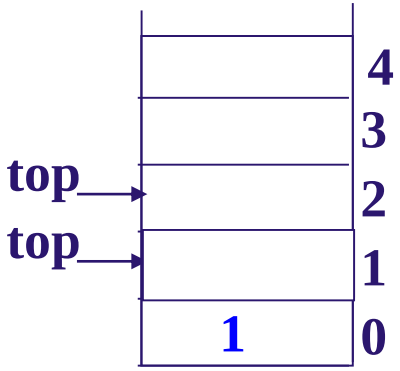


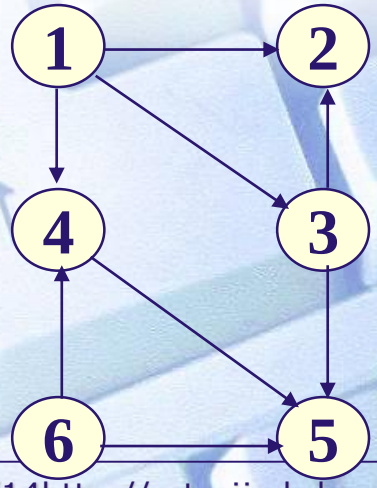


入度为0的顶点入栈

拓扑排序


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

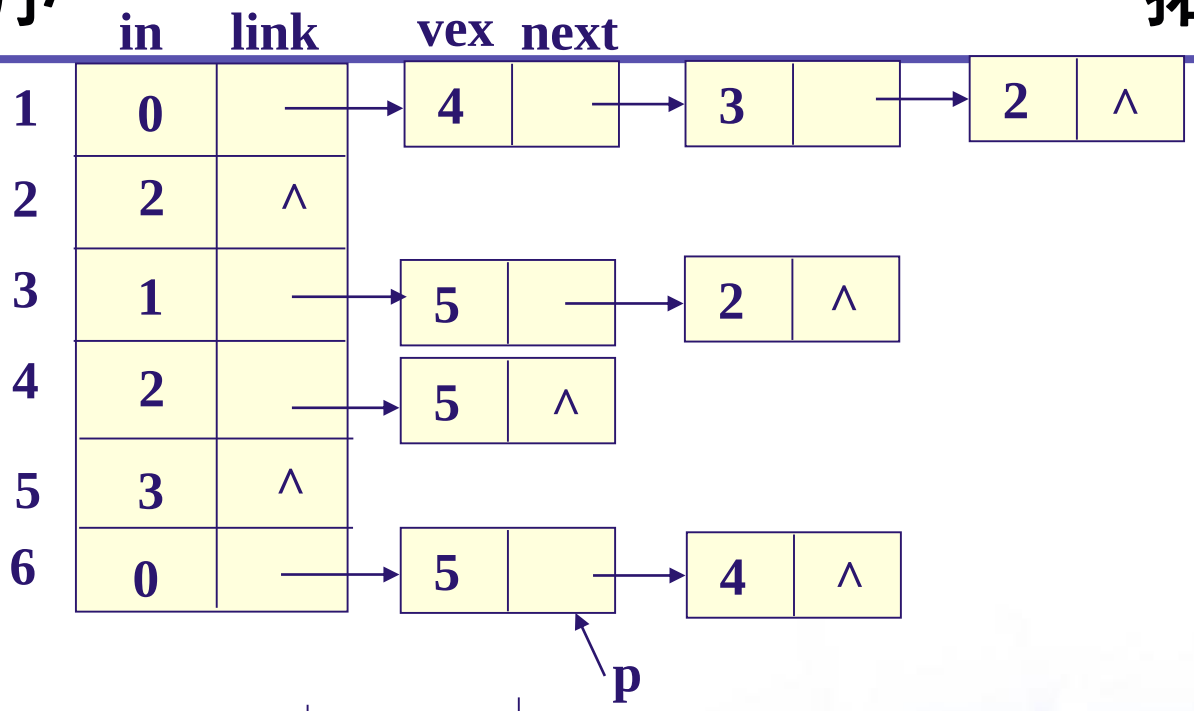

top
top
4
3
2
1
0
1

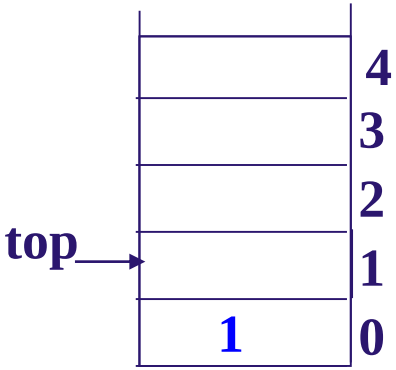

1
2
4
3
6
5

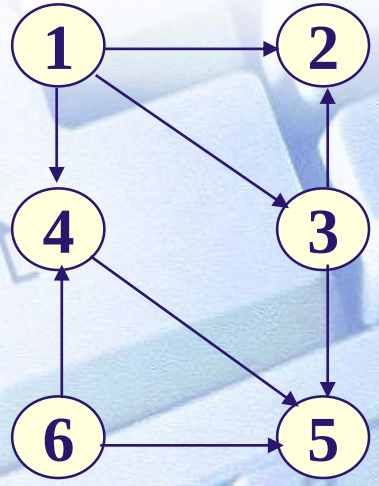
输出序列：6

拓扑排序

拓扑排序



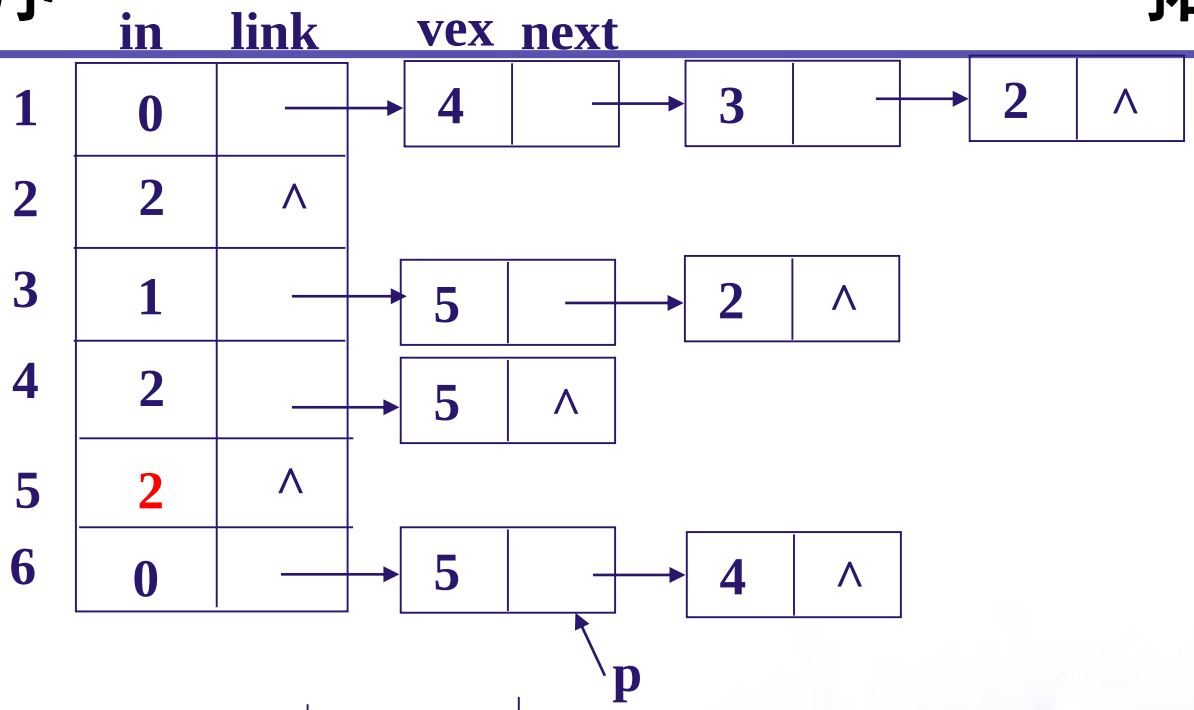


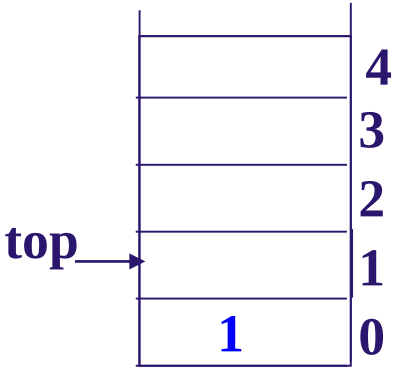


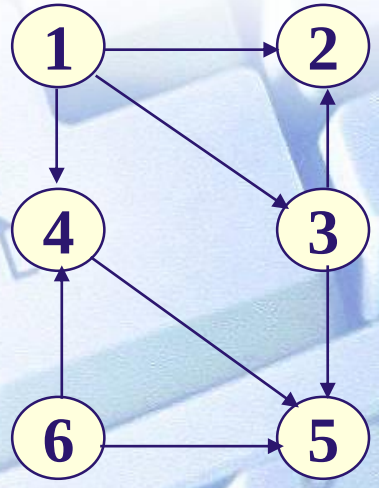
输出序列：6

拓扑排序

拓扑排序



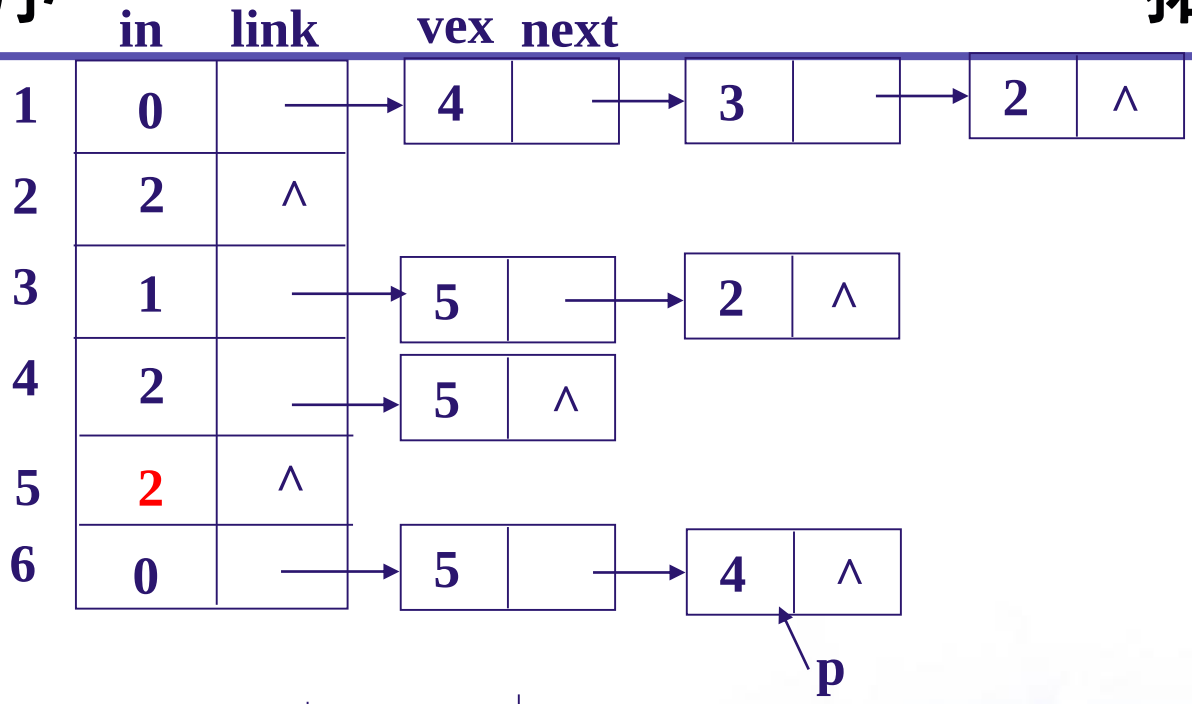


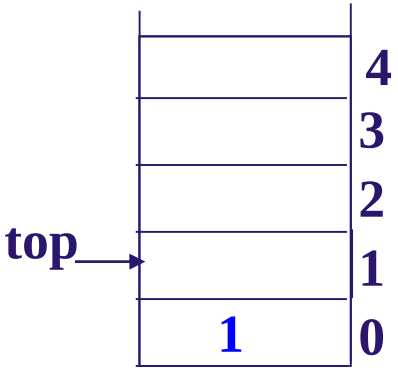


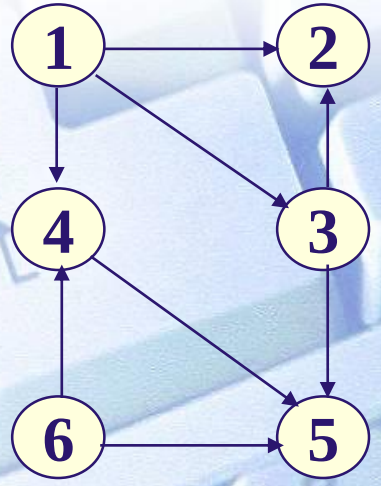
输出序列：6

拓扑排序

拓扑排序



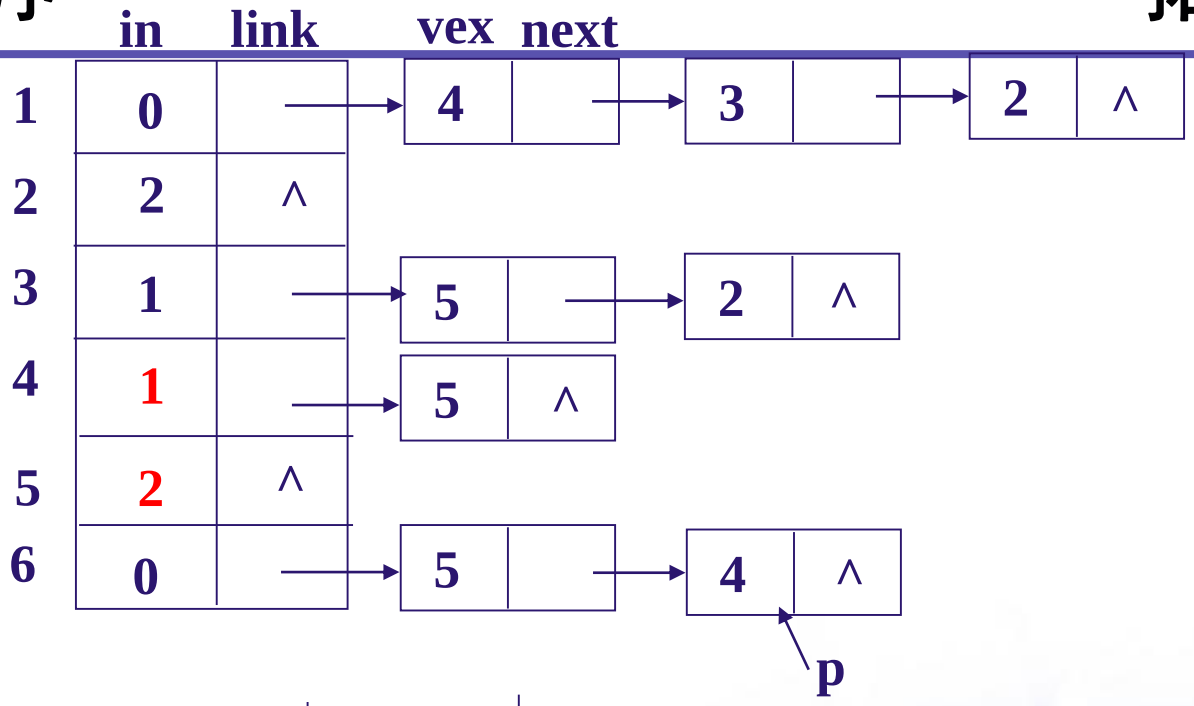


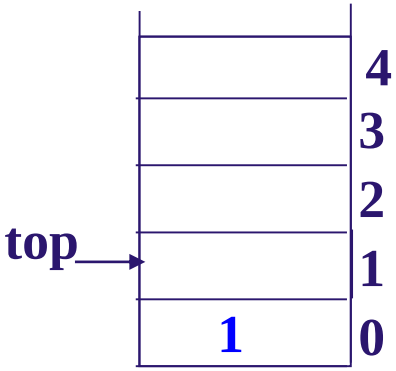


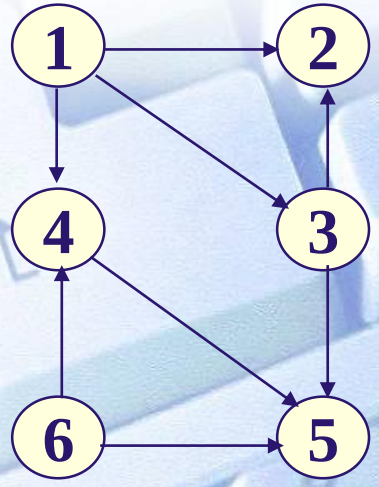
输出序列：6

拓扑排序

拓扑排序



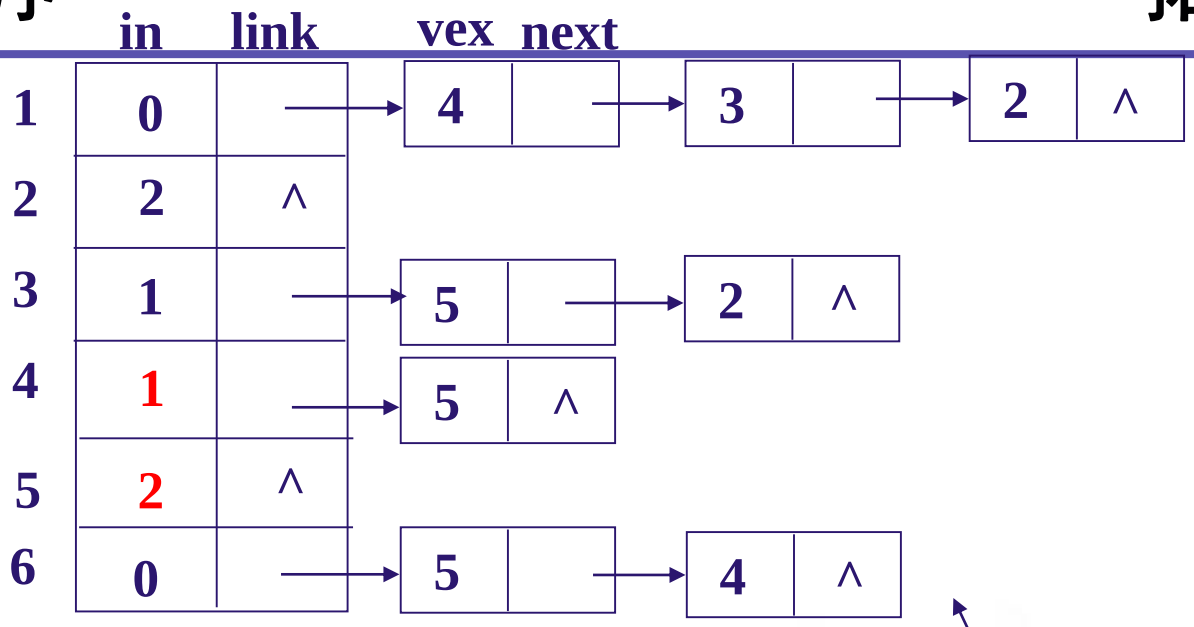


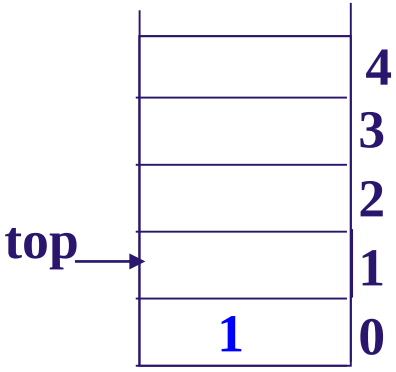


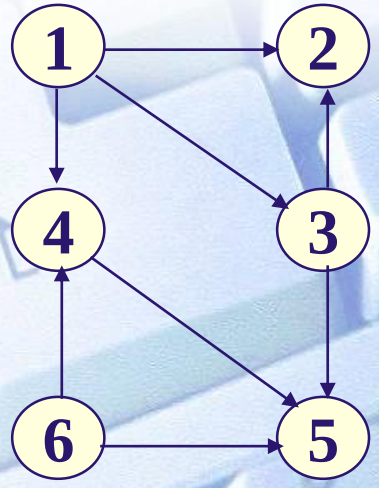
输出序列：6

拓扑排序

拓扑排序



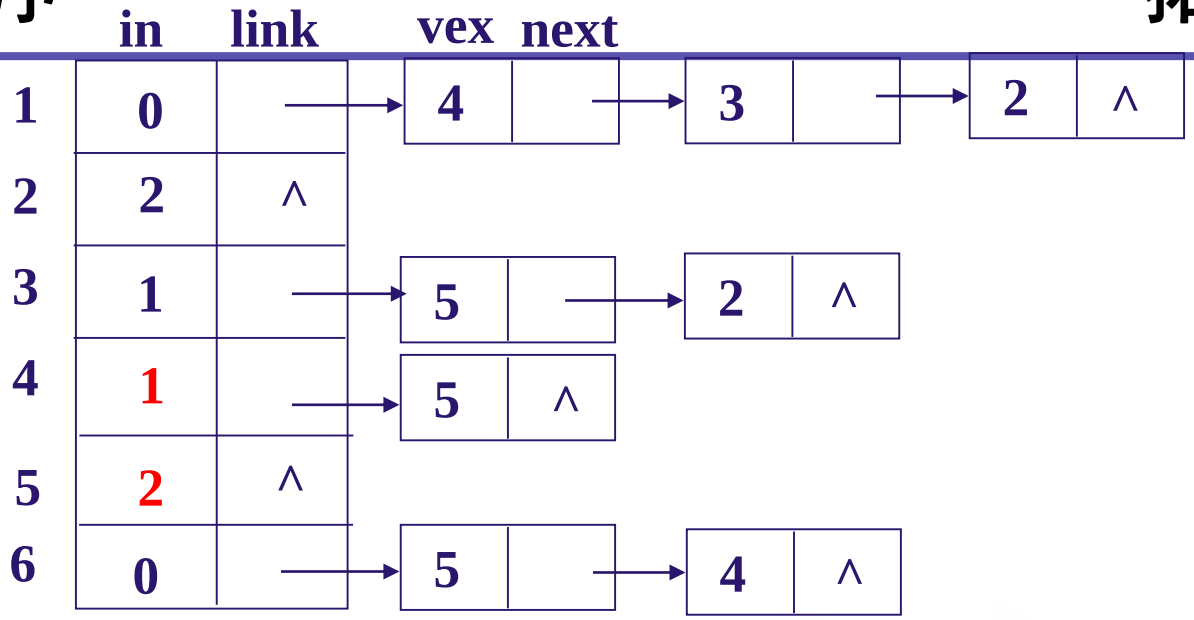


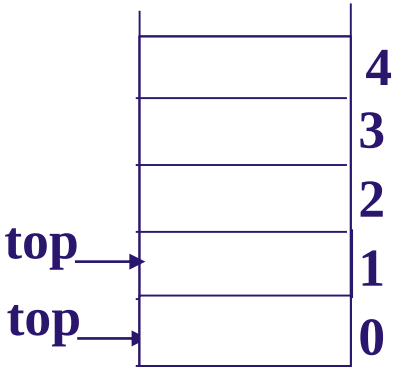


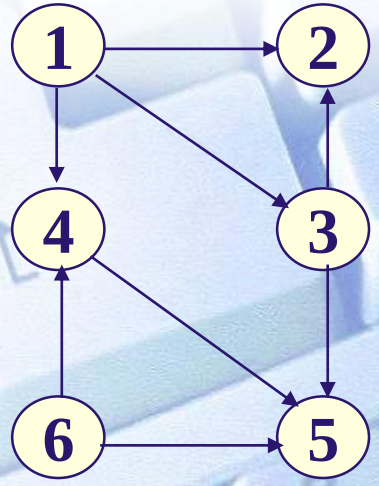
输出序列：6

拓扑排序

拓扑排序



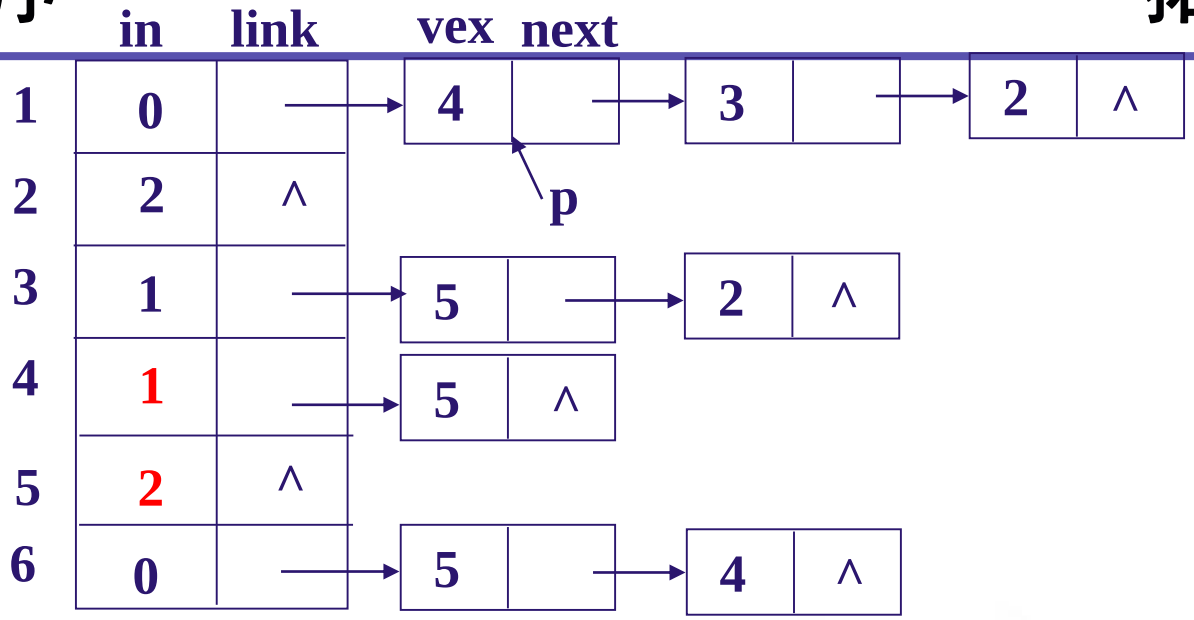


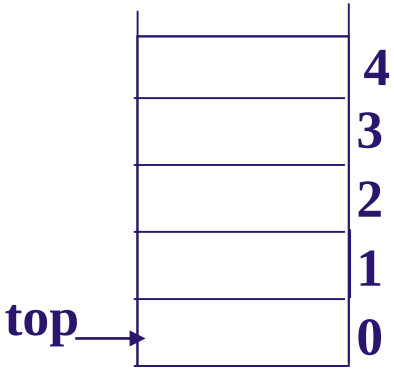


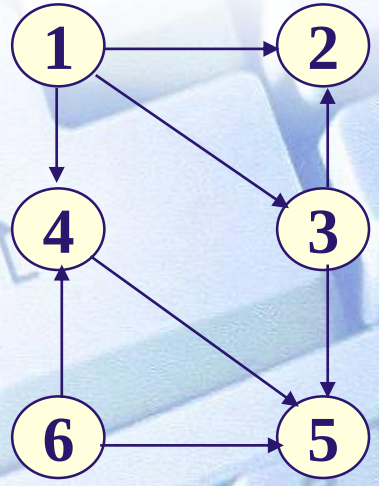
输出序列：6 1

拓扑排序

拓扑排序



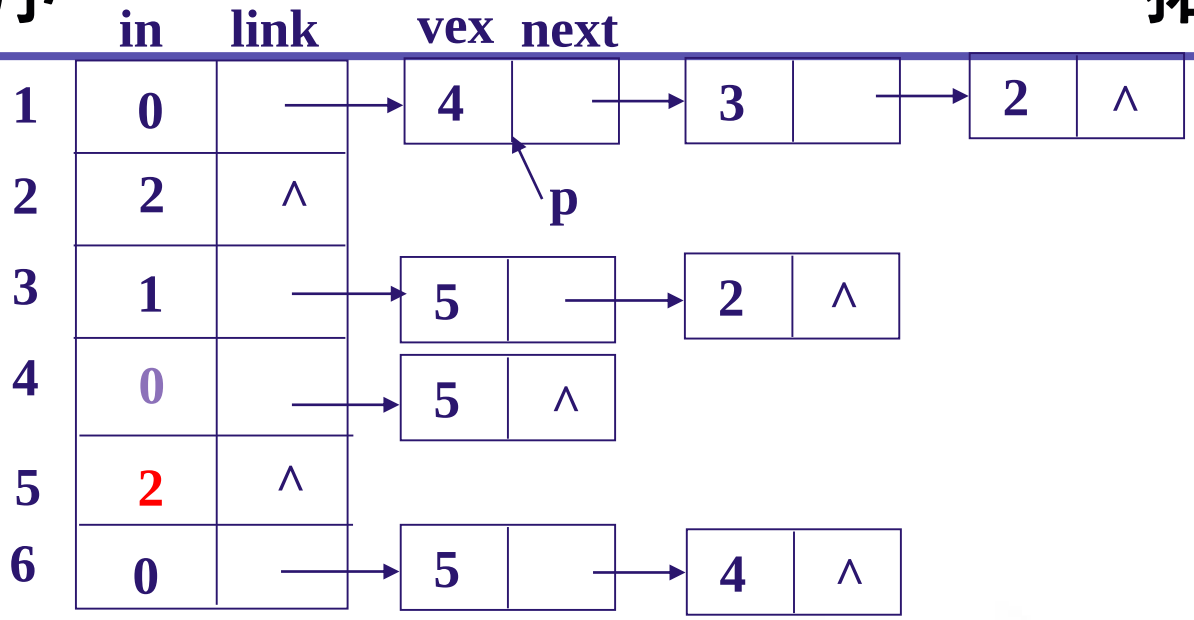


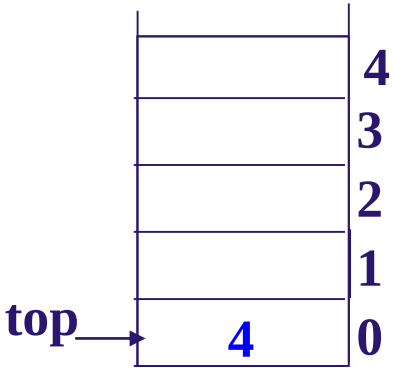


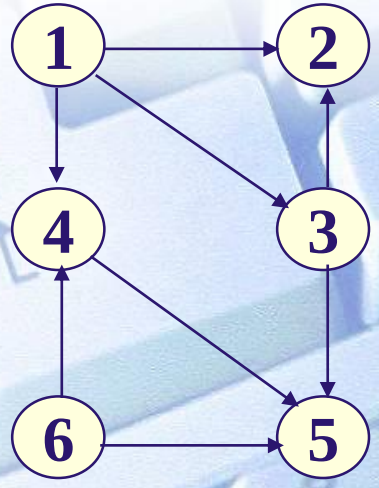
输出序列：6 1

拓扑排序

拓扑排序



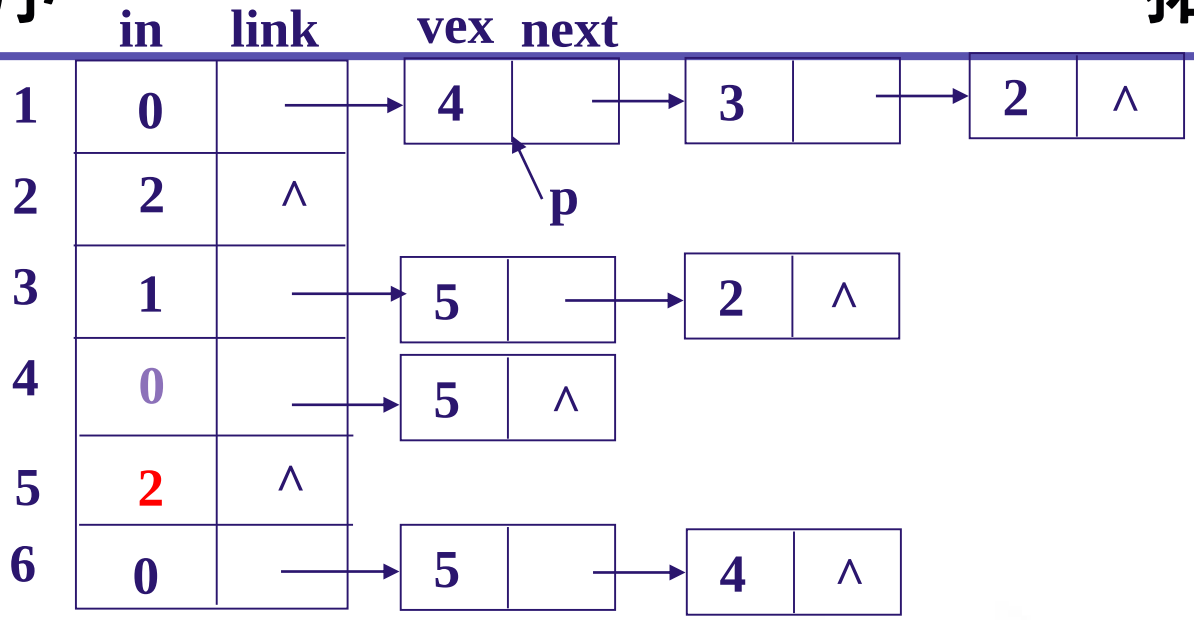


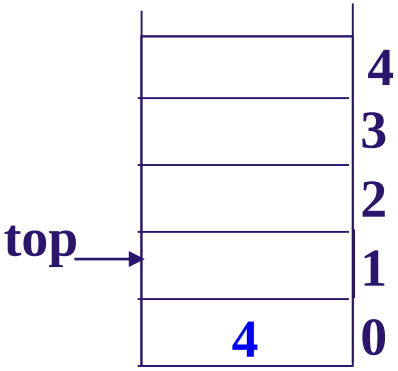


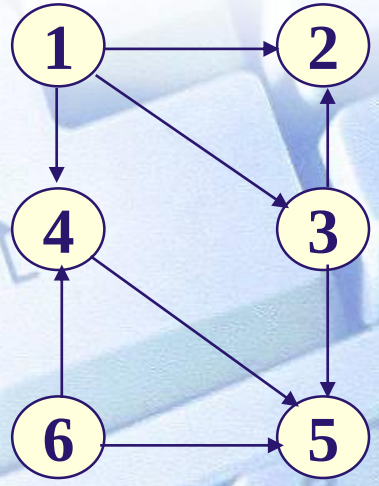
输出序列：6 1

拓扑排序

拓扑排序



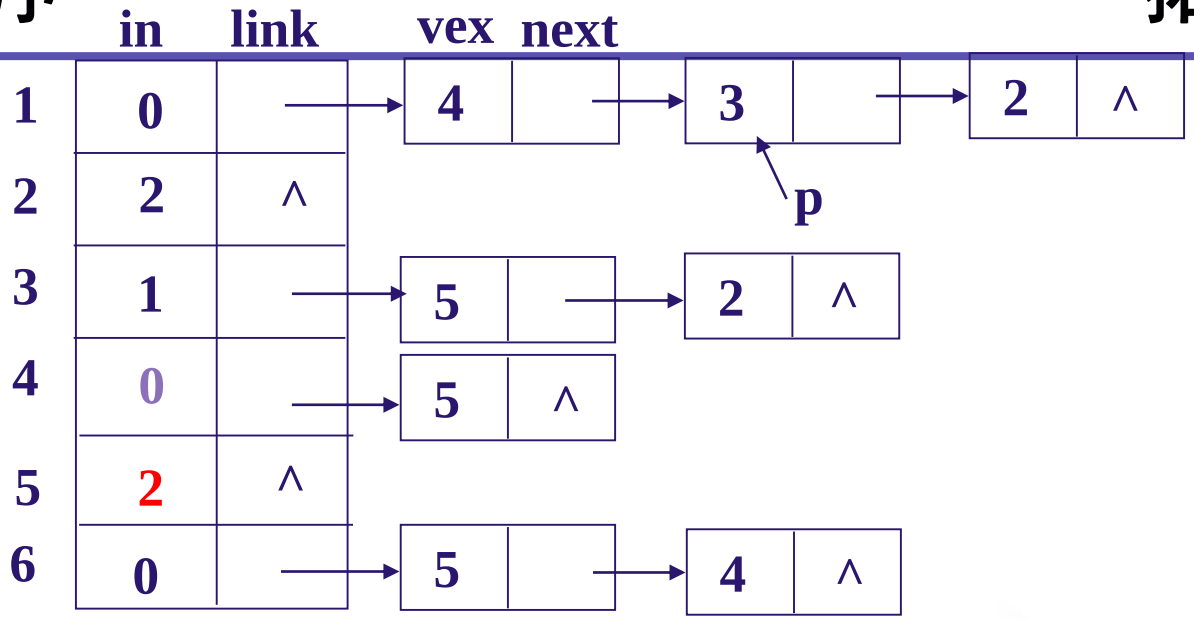


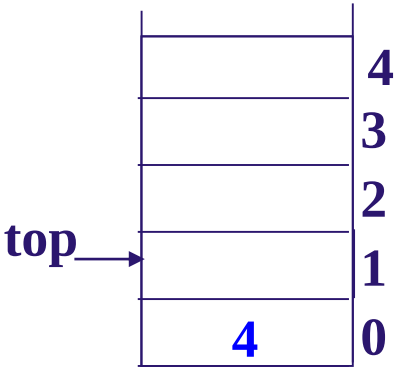


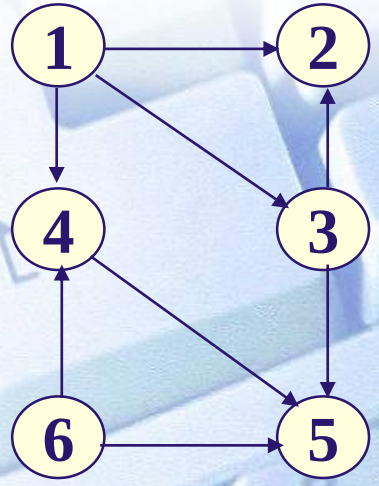
输出序列：6 1

拓扑排序

拓扑排序



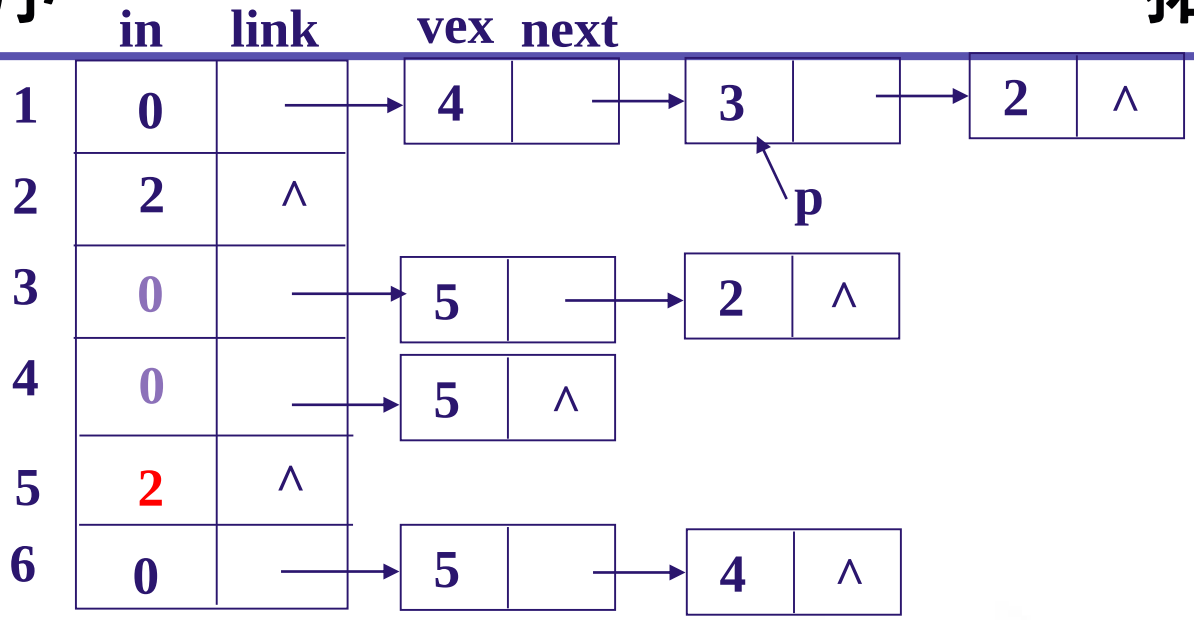


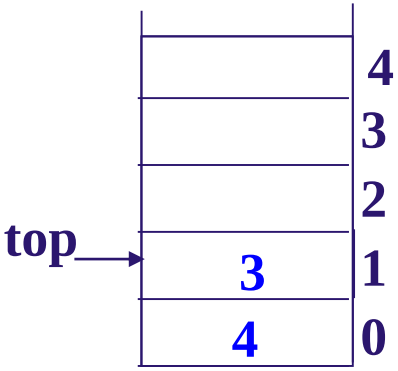


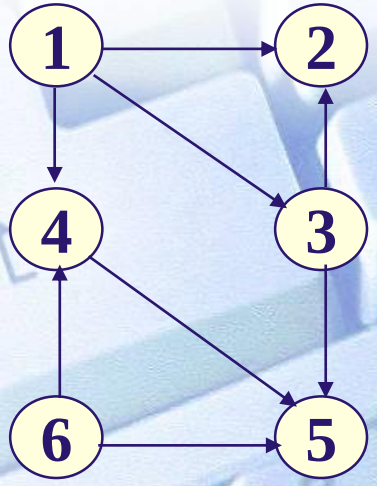
输出序列：6 1

拓扑排序

拓扑排序



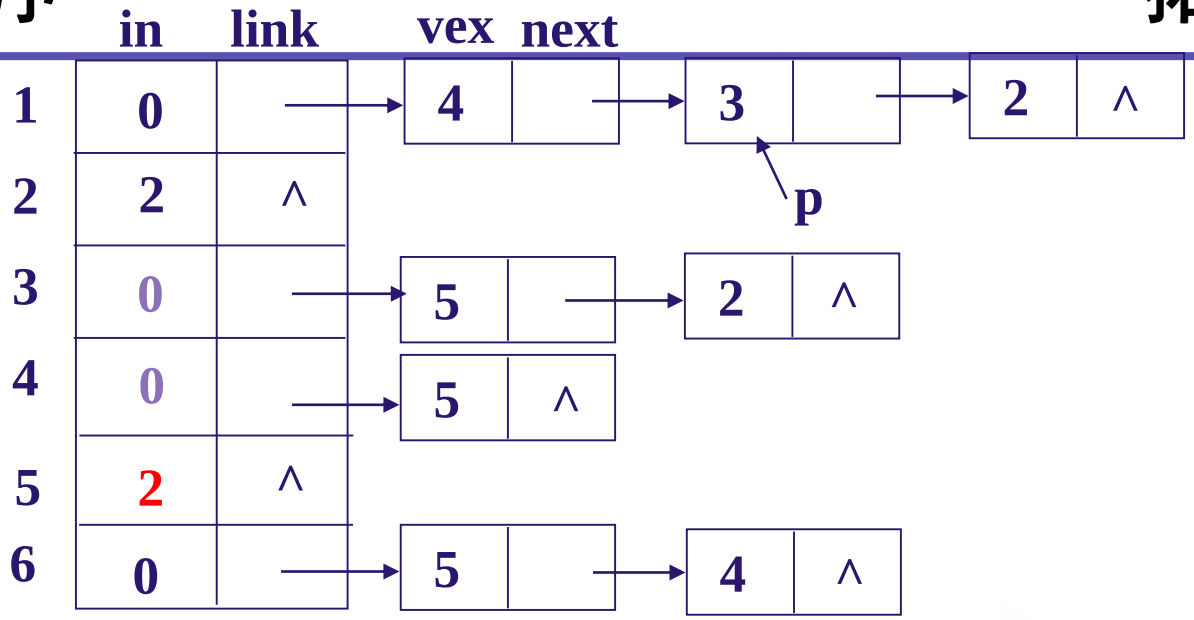


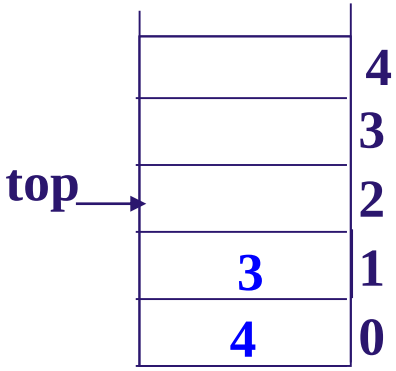


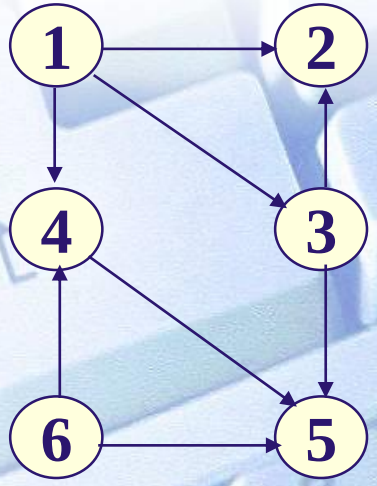
输出序列：6 1

拓扑排序

拓扑排序



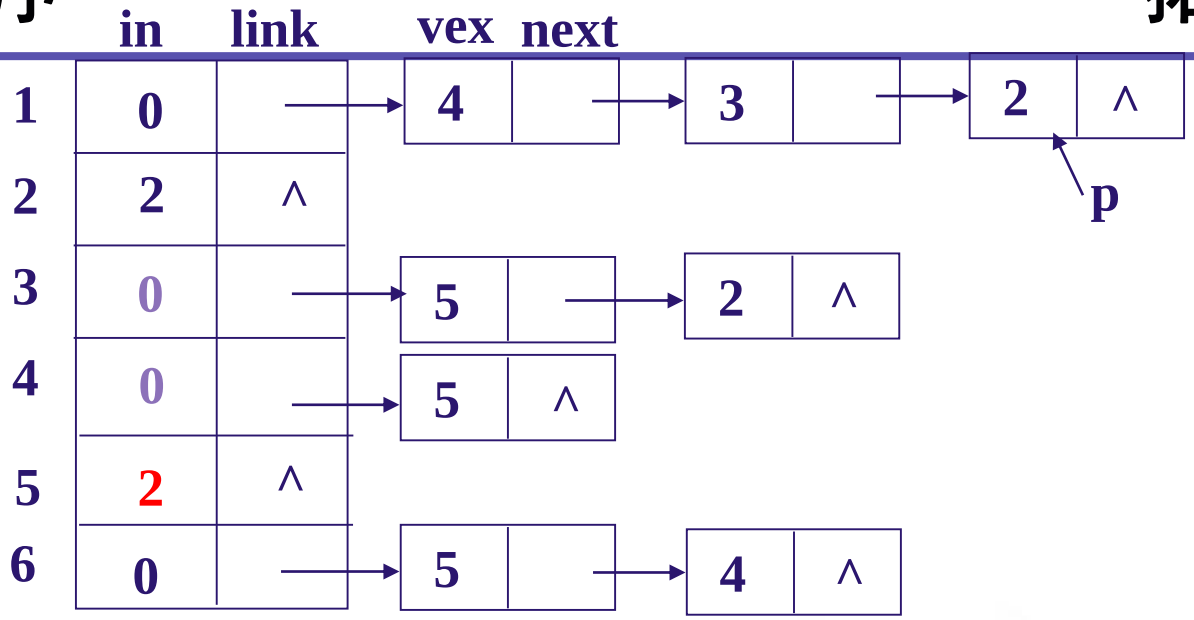


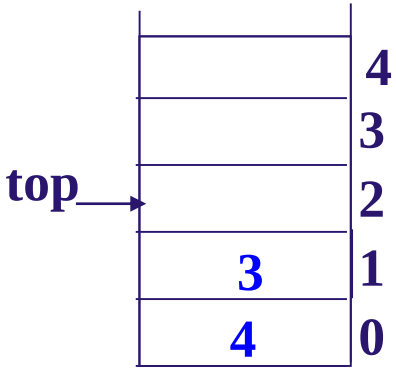


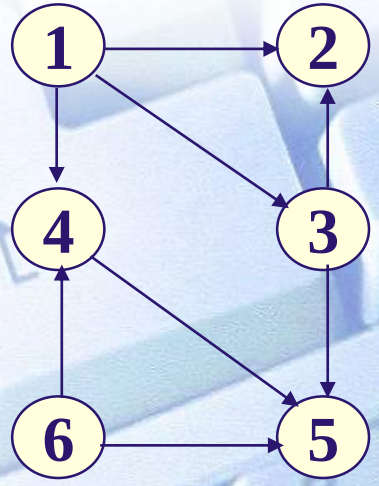
输出序列：6 1

拓扑排序

拓扑排序



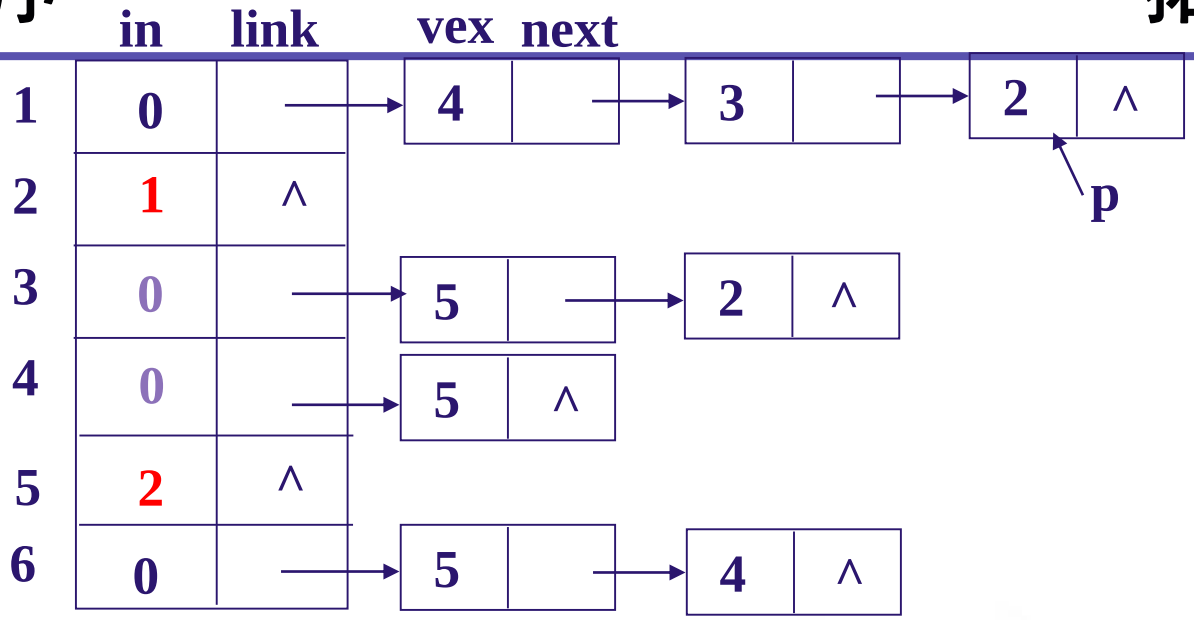


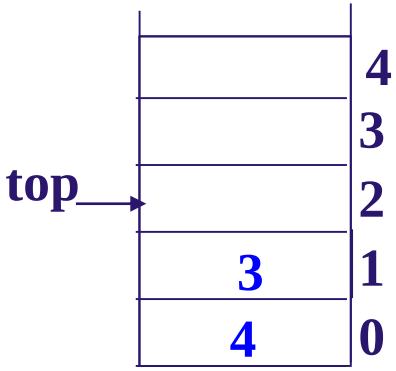


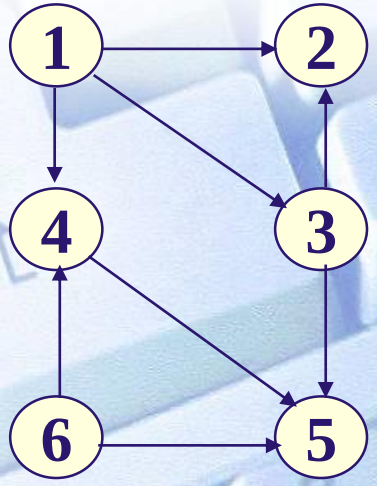
输出序列：6 1

拓扑排序

拓扑排序



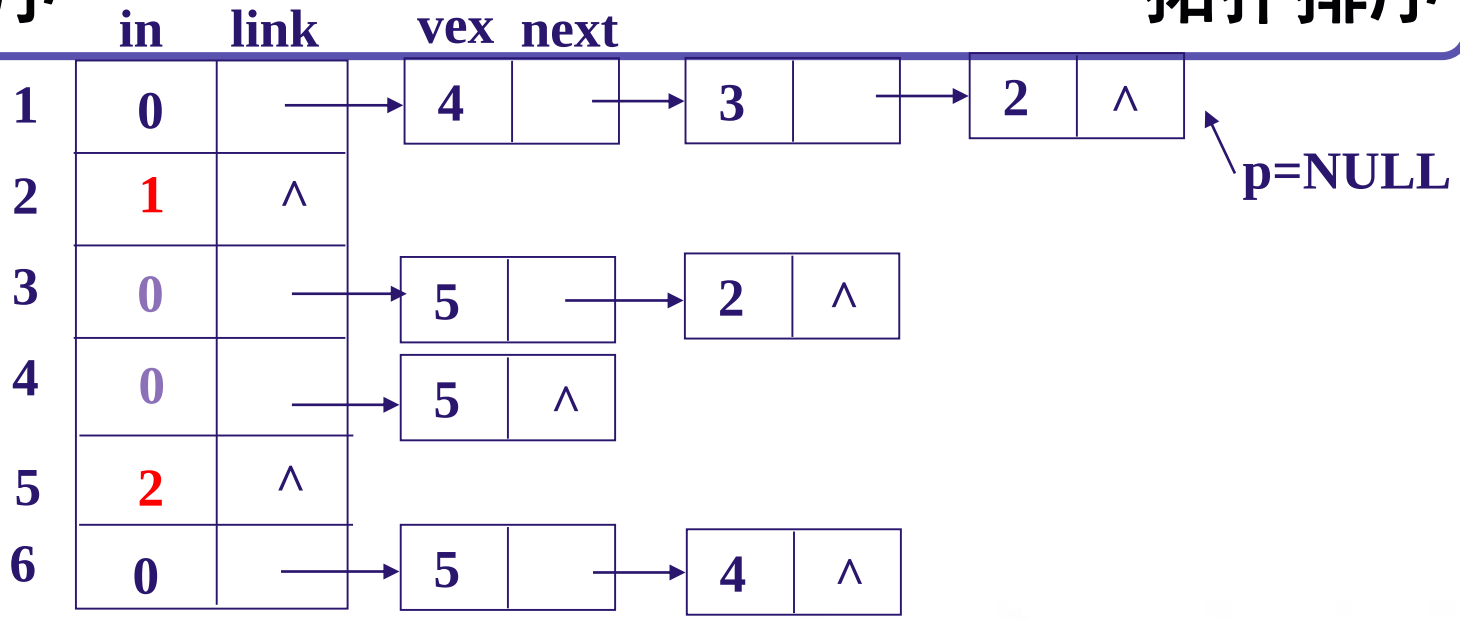


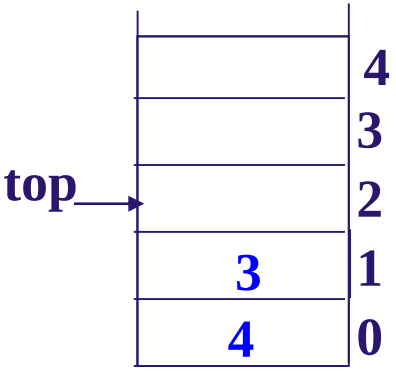


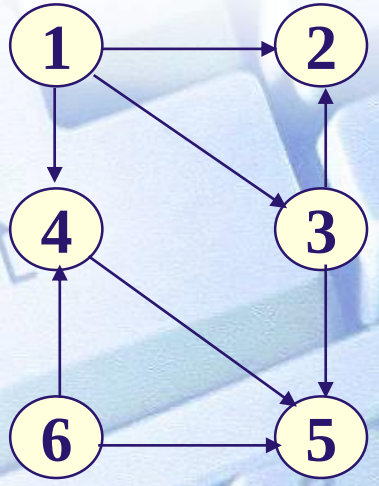
输出序列：6 1

拓扑排序

拓扑排序



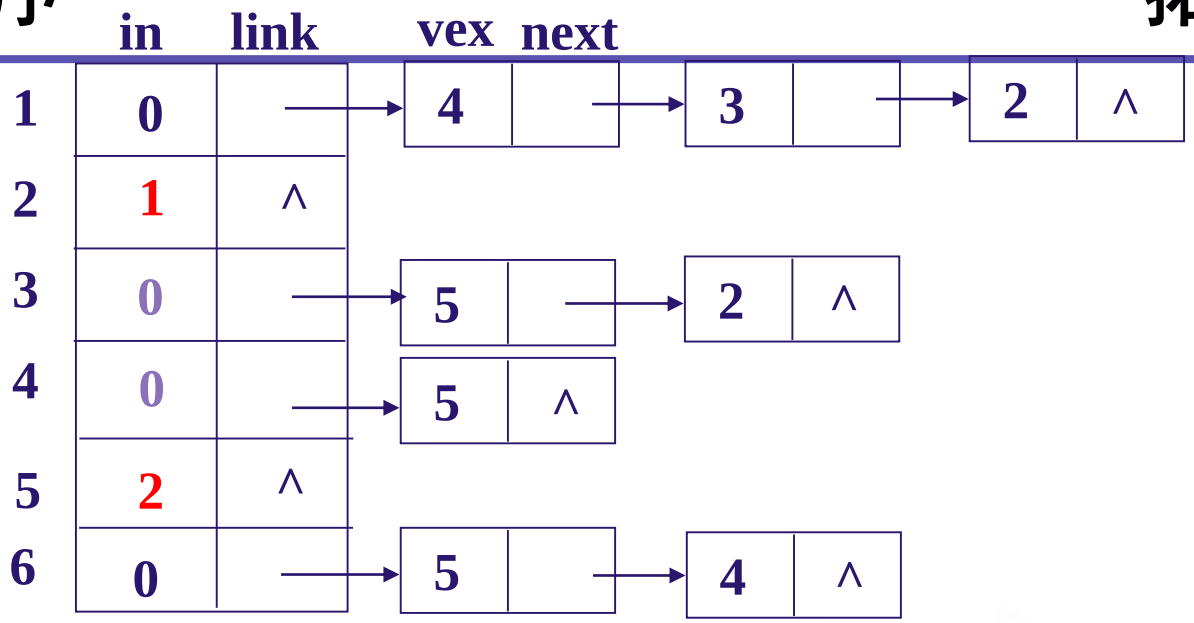


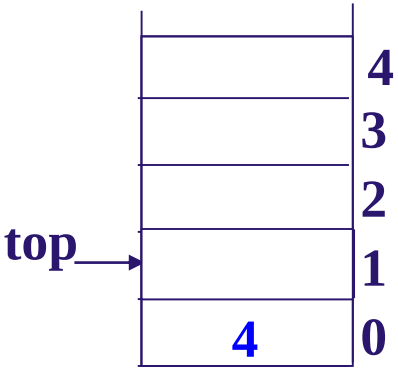


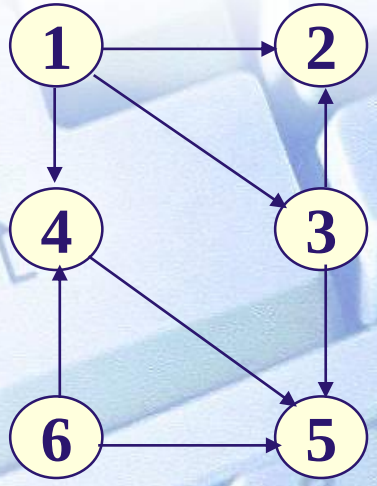
输出序列：6 1

拓扑排序

拓扑排序



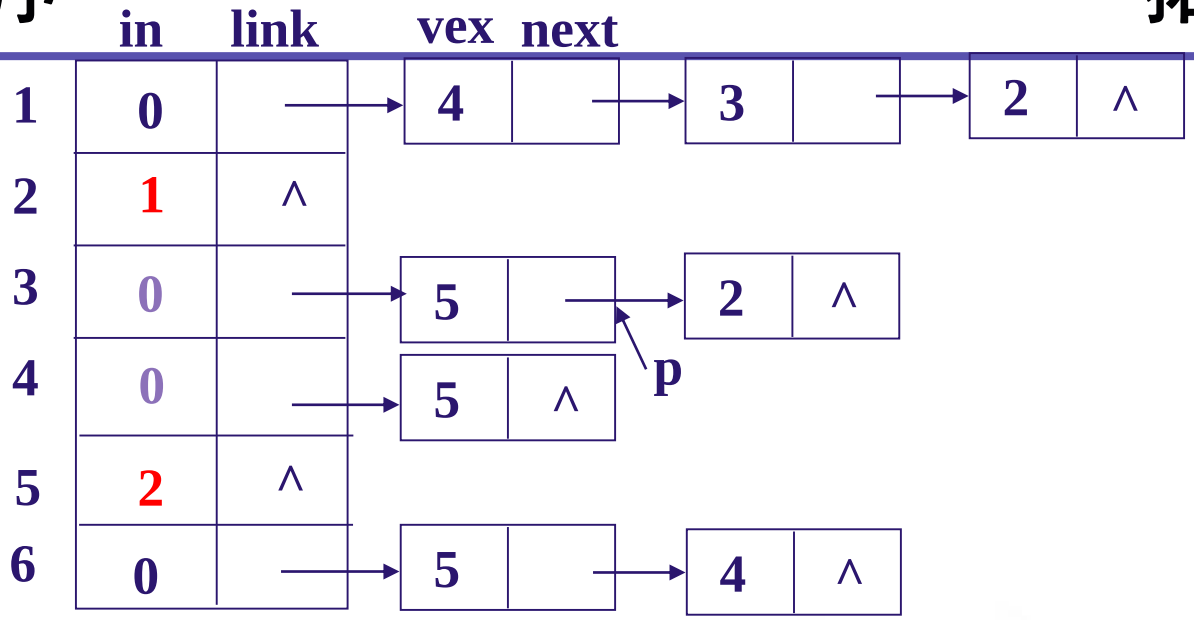


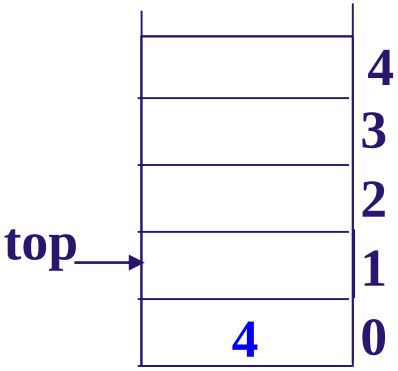


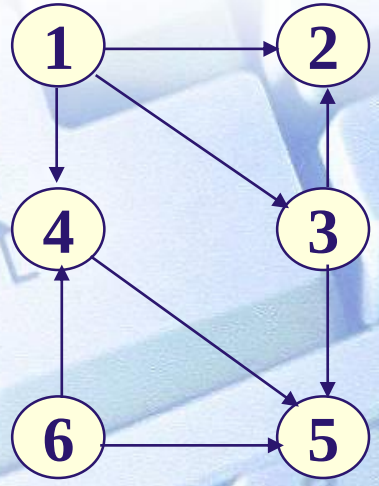
输出序列：6 1 3

拓扑排序

拓扑排序



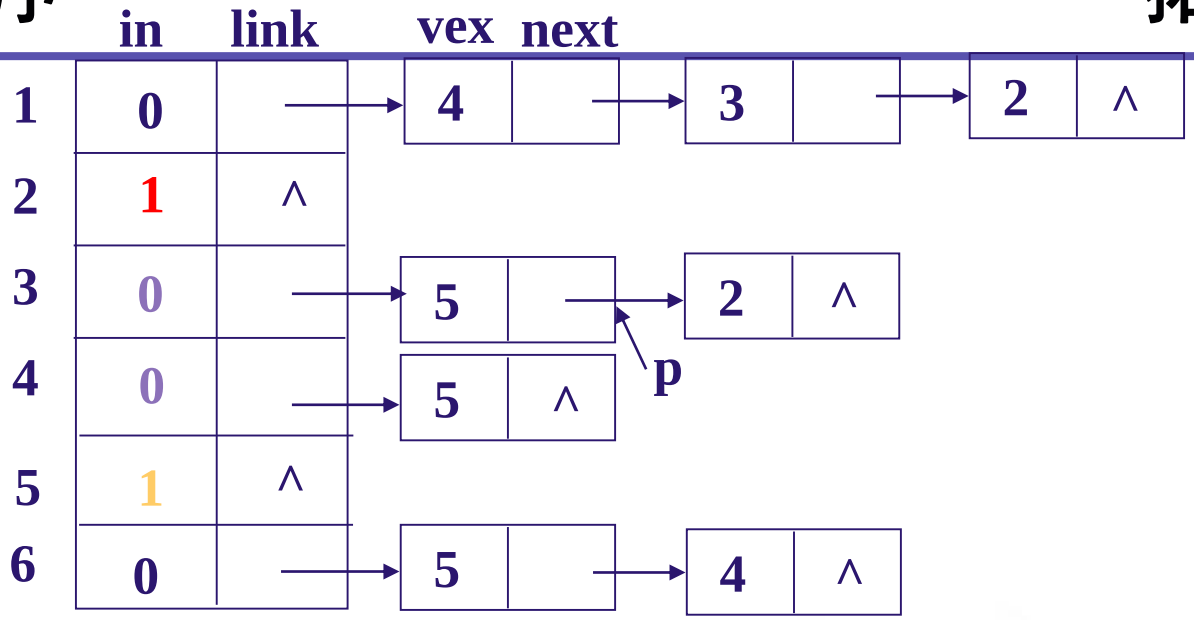


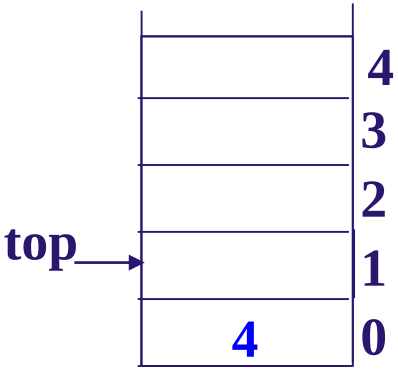


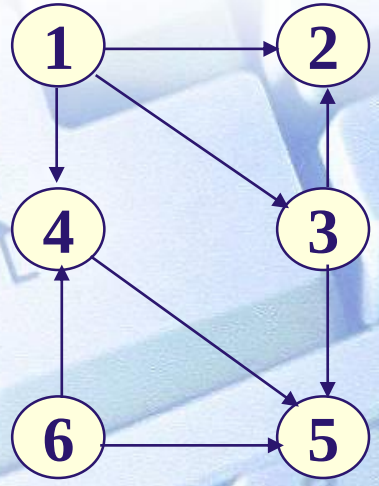
输出序列：6 1 3

拓扑排序

拓扑排序



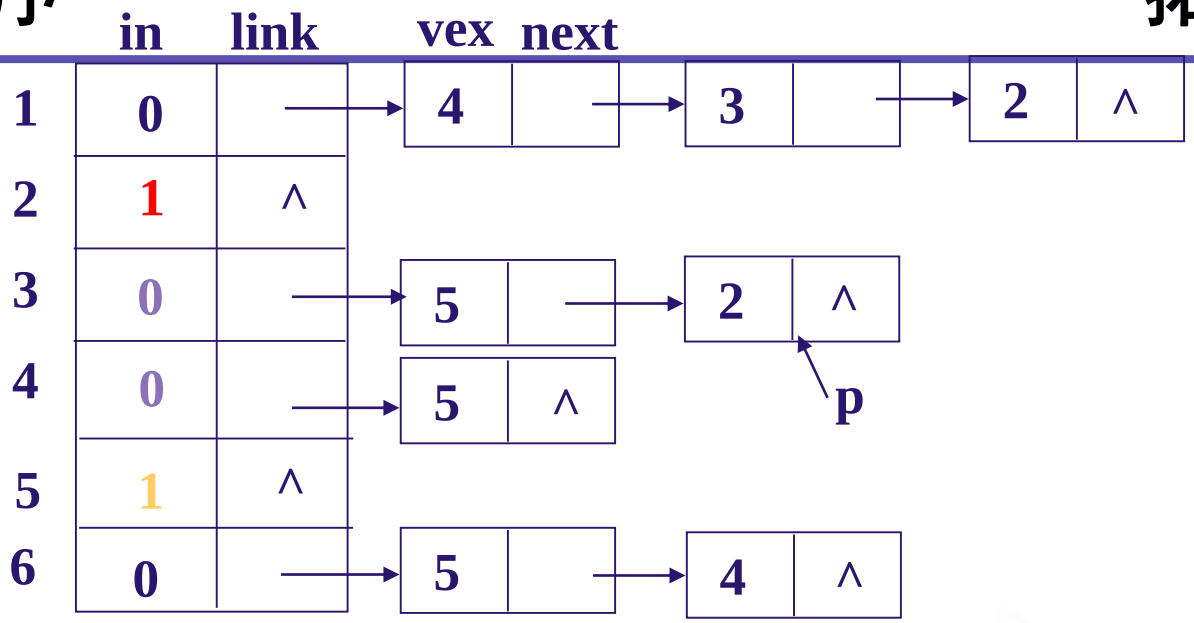


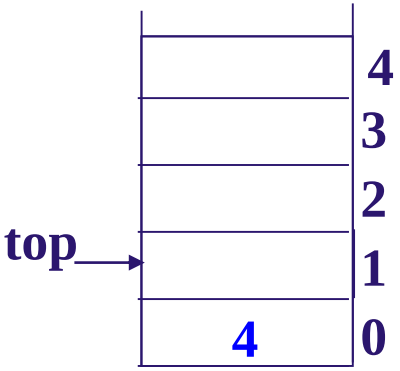


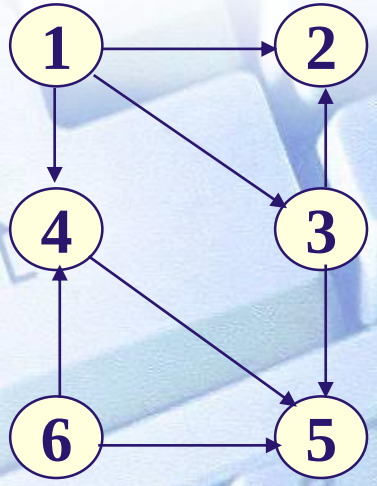
输出序列：6 1 3

拓扑排序

拓扑排序



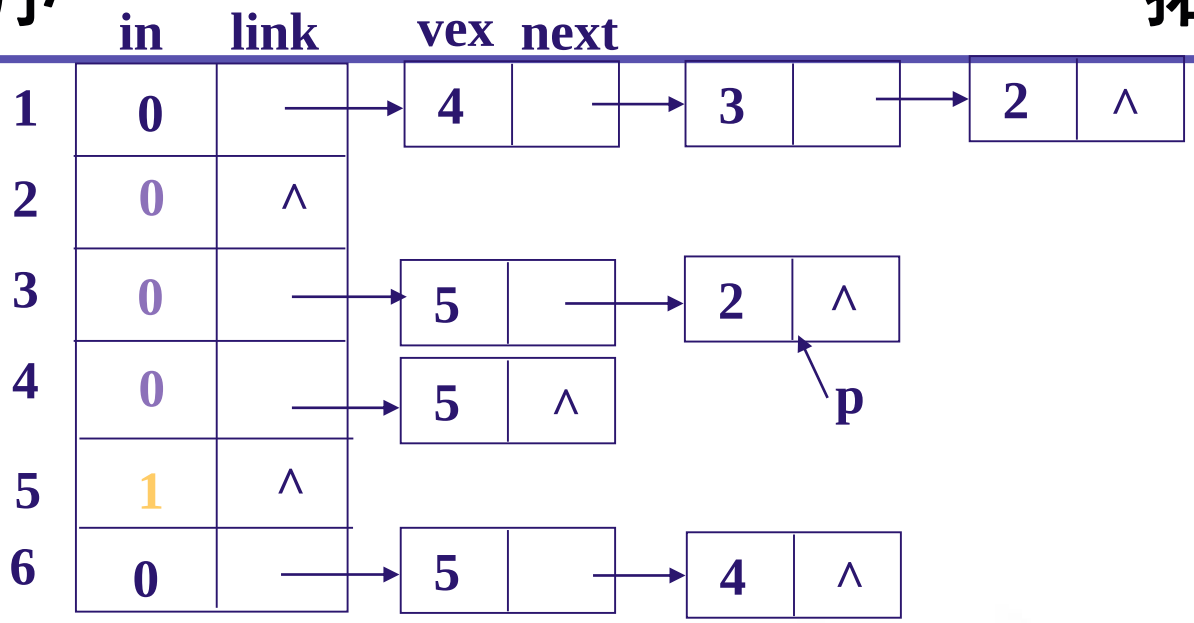


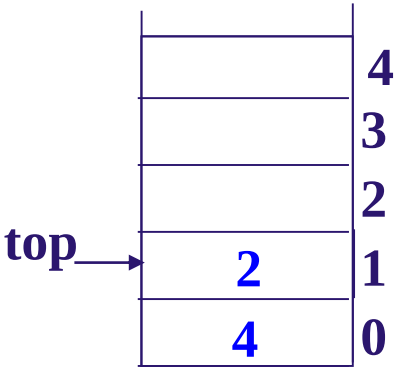


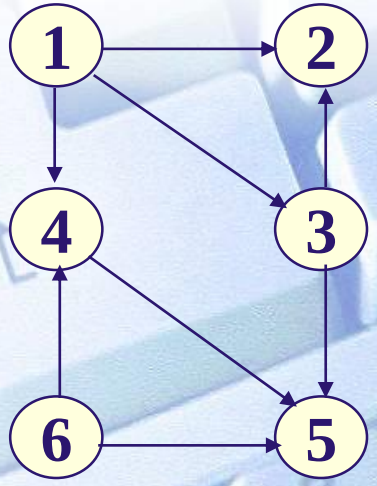
输出序列：6 1 3

拓扑排序

拓扑排序



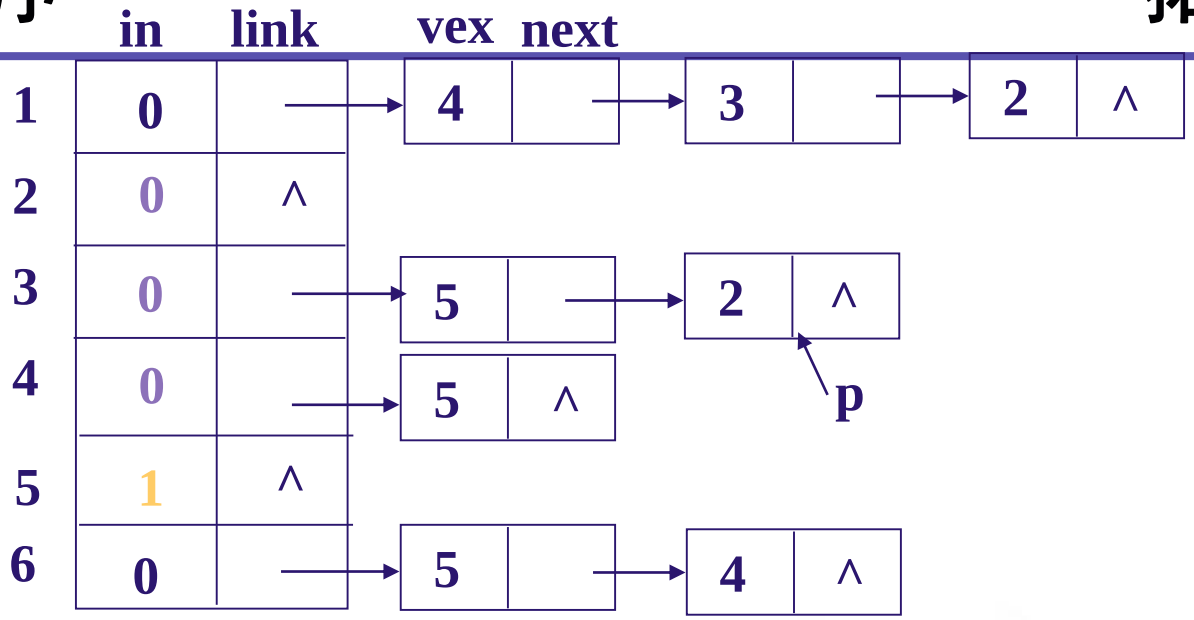


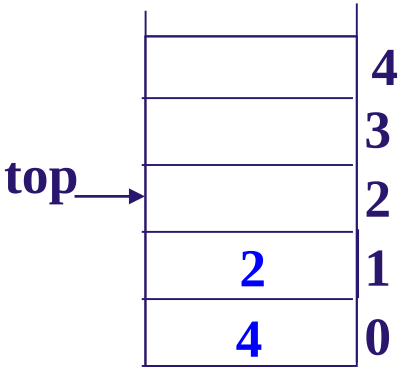


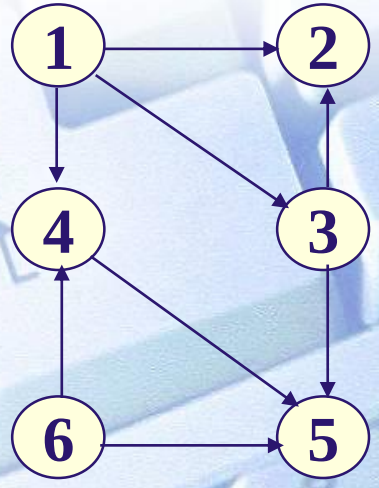
输出序列：6 1 3

拓扑排序

拓扑排序

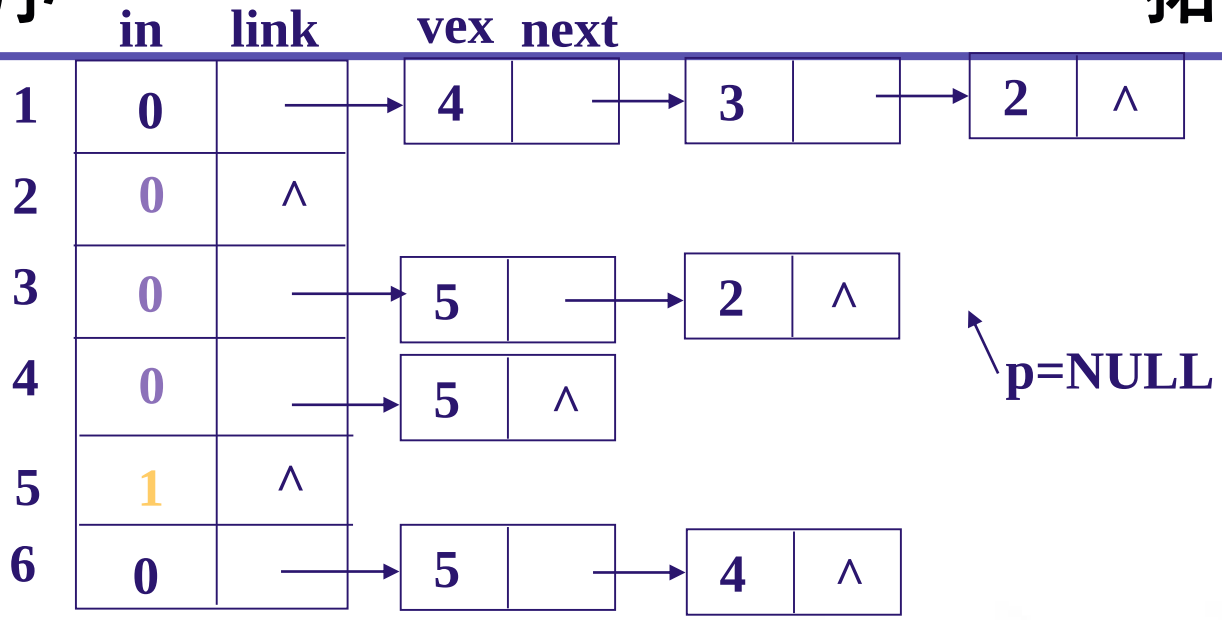


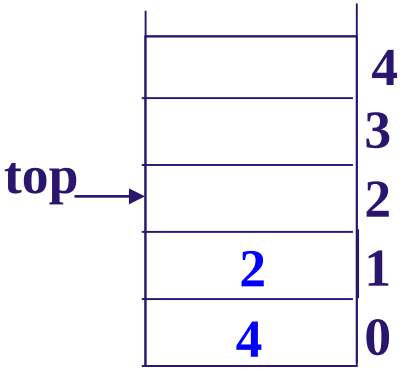


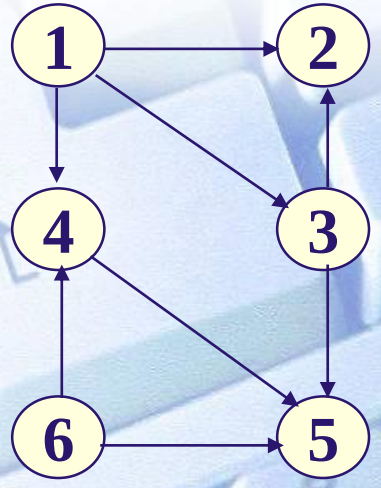


输出序列：6 1 3

拓扑排序



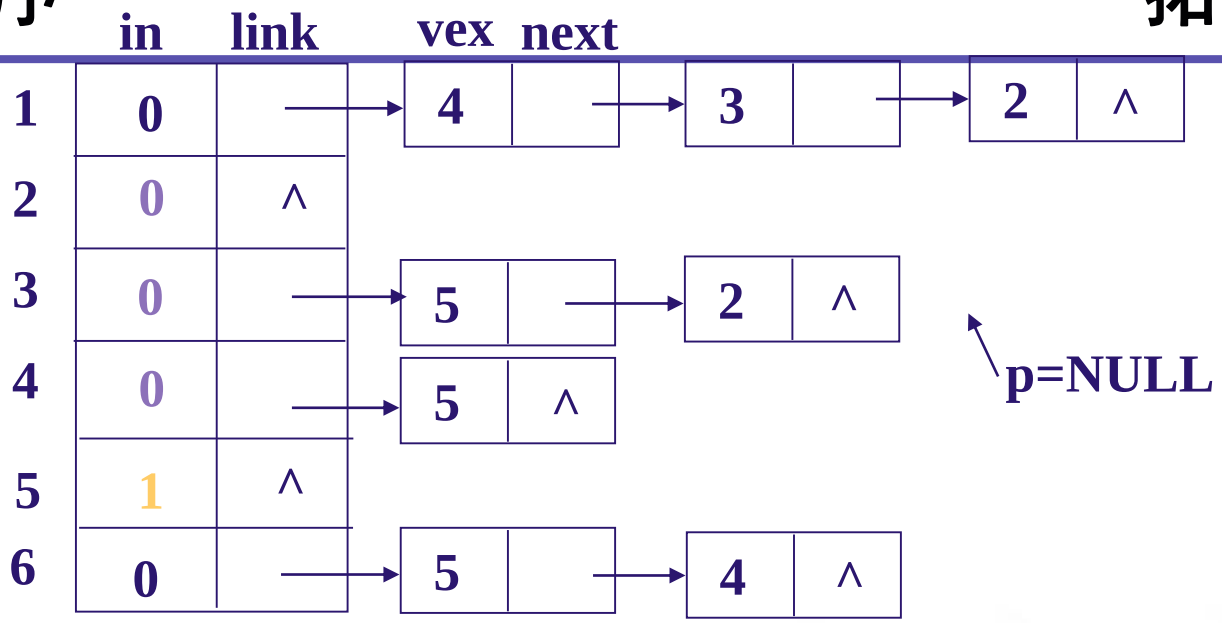


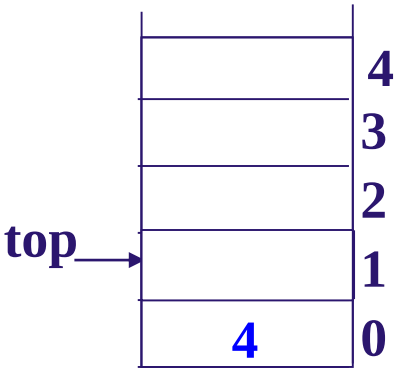


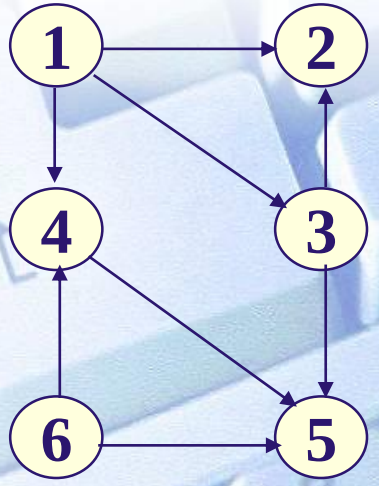
输出序列：6 1 3

拓扑排序

拓扑排序



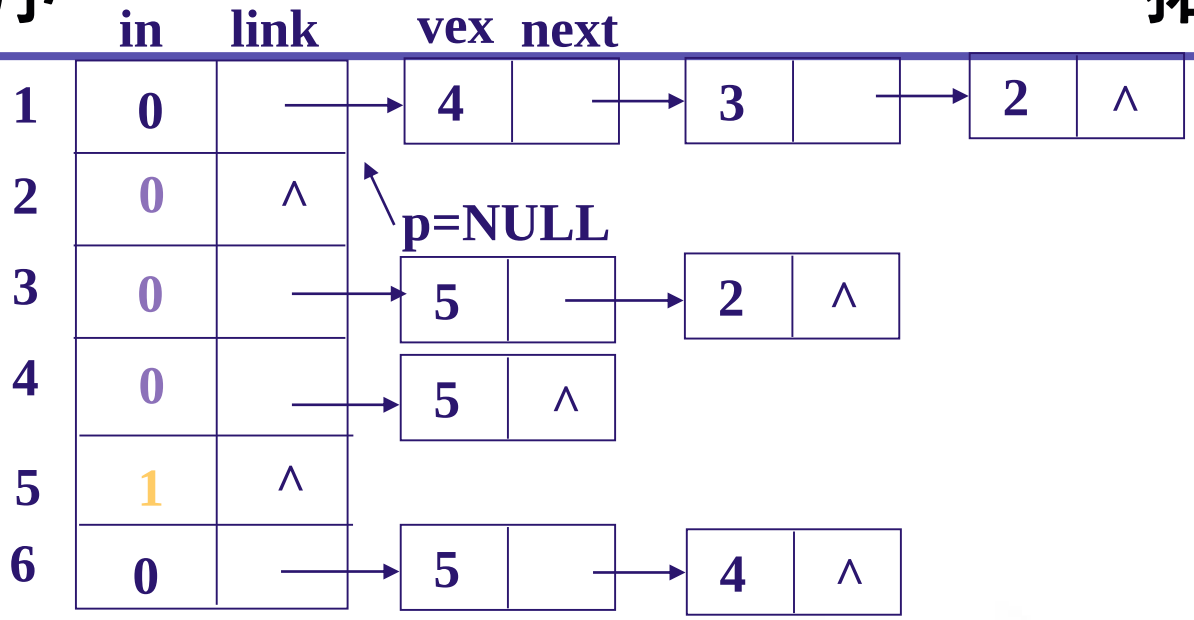


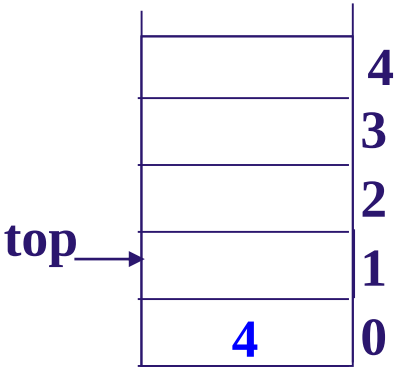


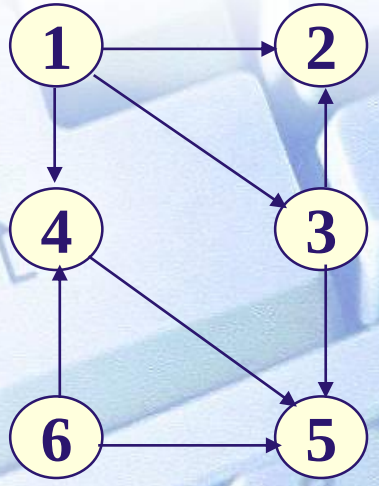
输出序列：6 1 3 2

拓扑排序

拓扑排序



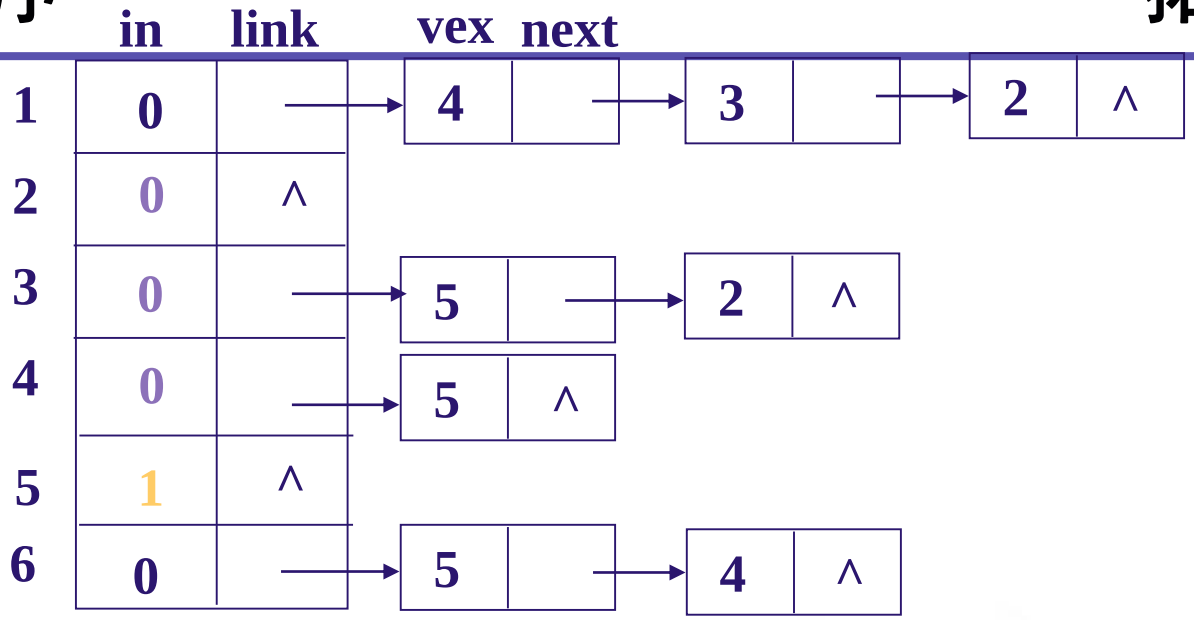


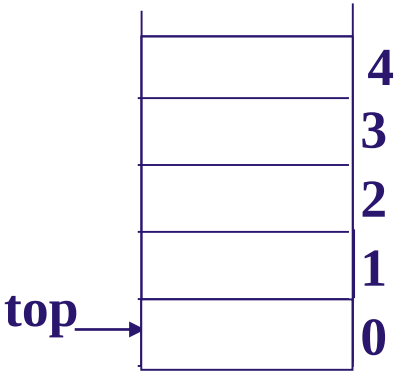


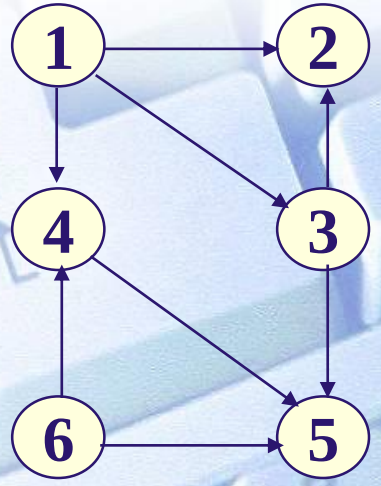
输出序列：6 1 3 2

拓扑排序

拓扑排序



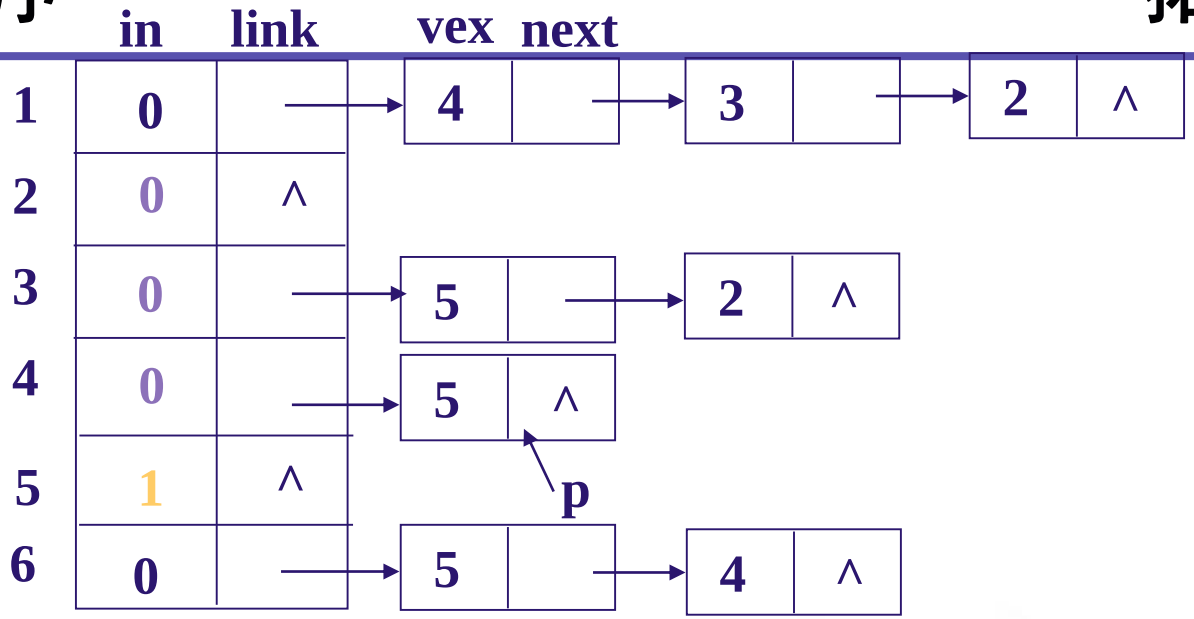


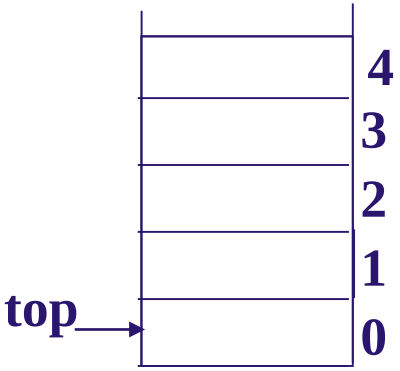


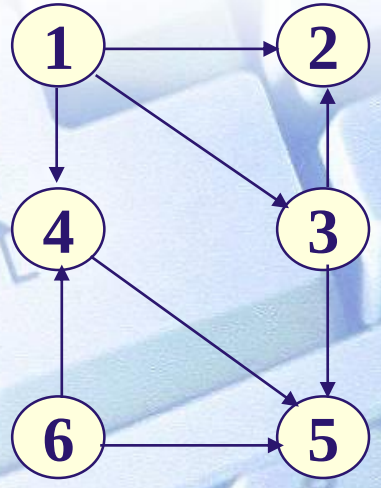
输出序列：6 1 3 2 4

拓扑排序

拓扑排序



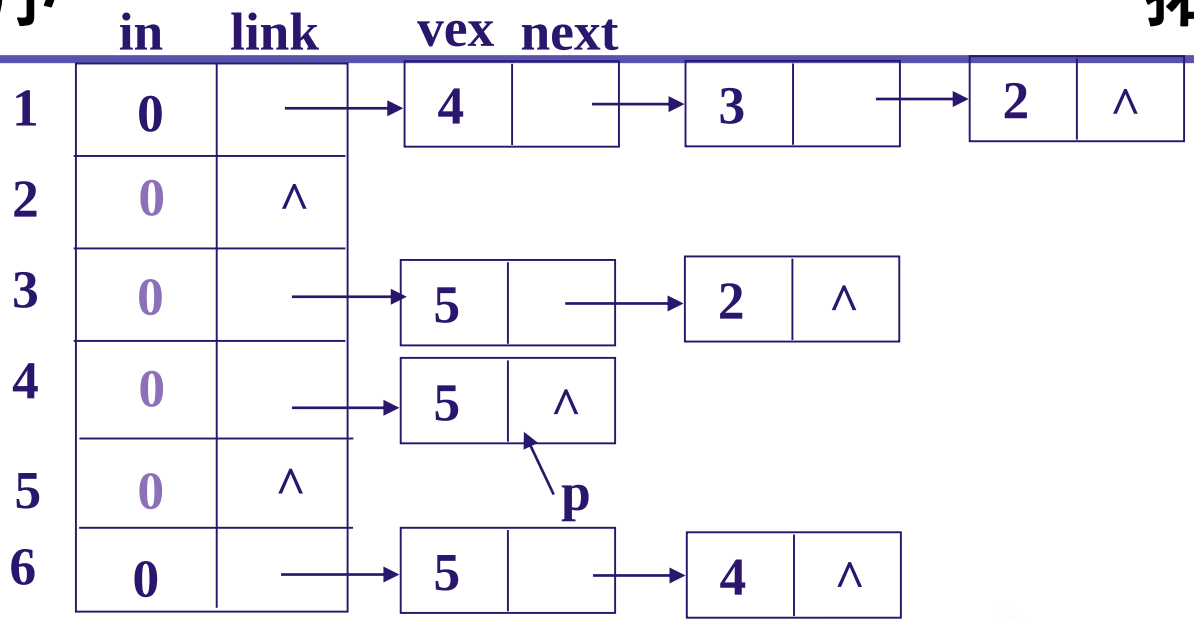


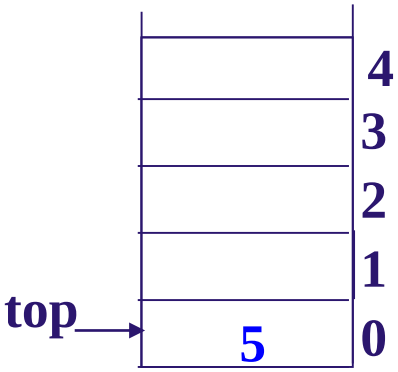


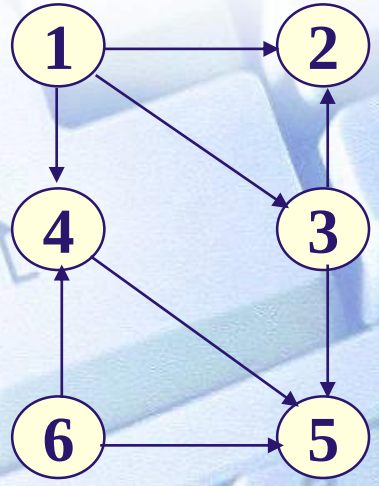
输出序列：6 1 3 2 4

拓扑排序

拓扑排序



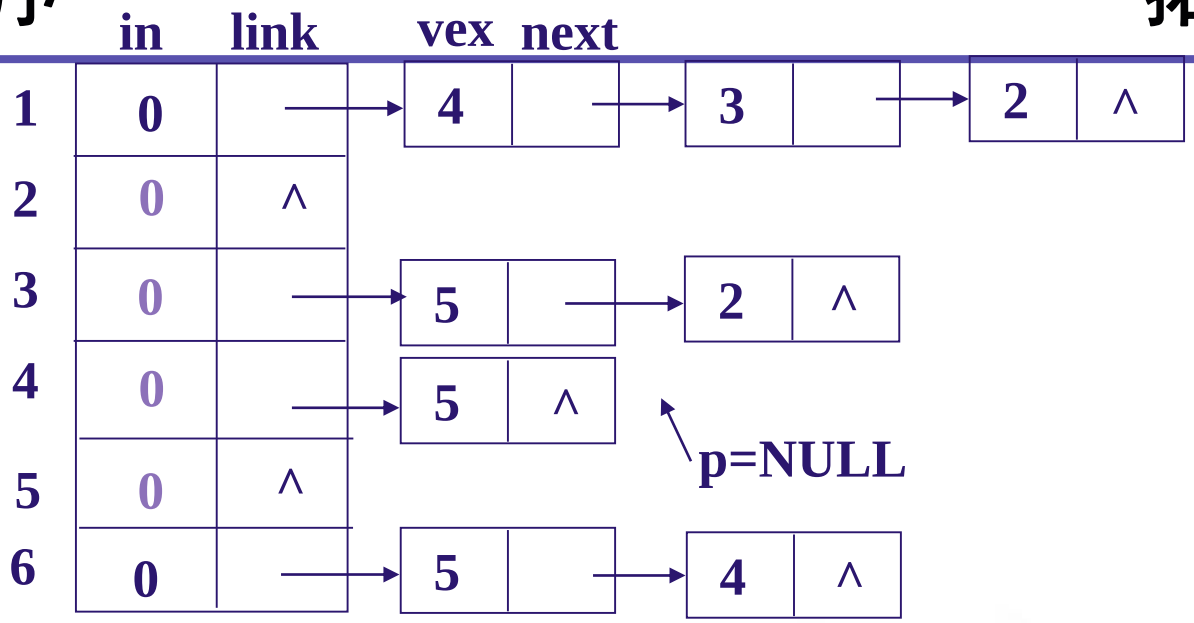


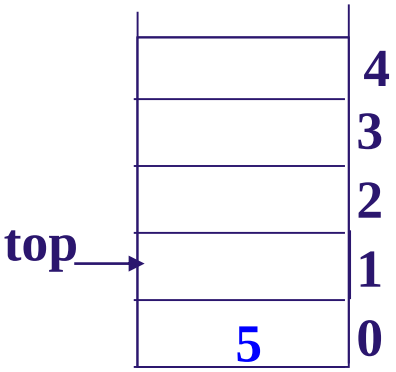


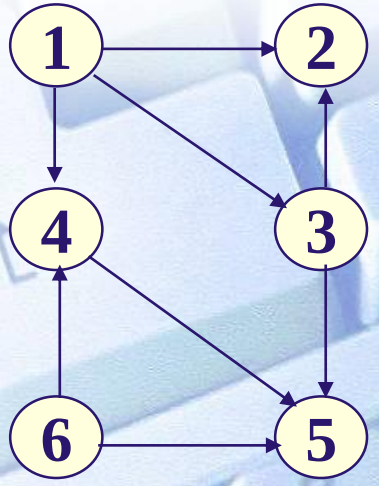
输出序列：6 1 3 2 4

拓扑排序

拓扑排序



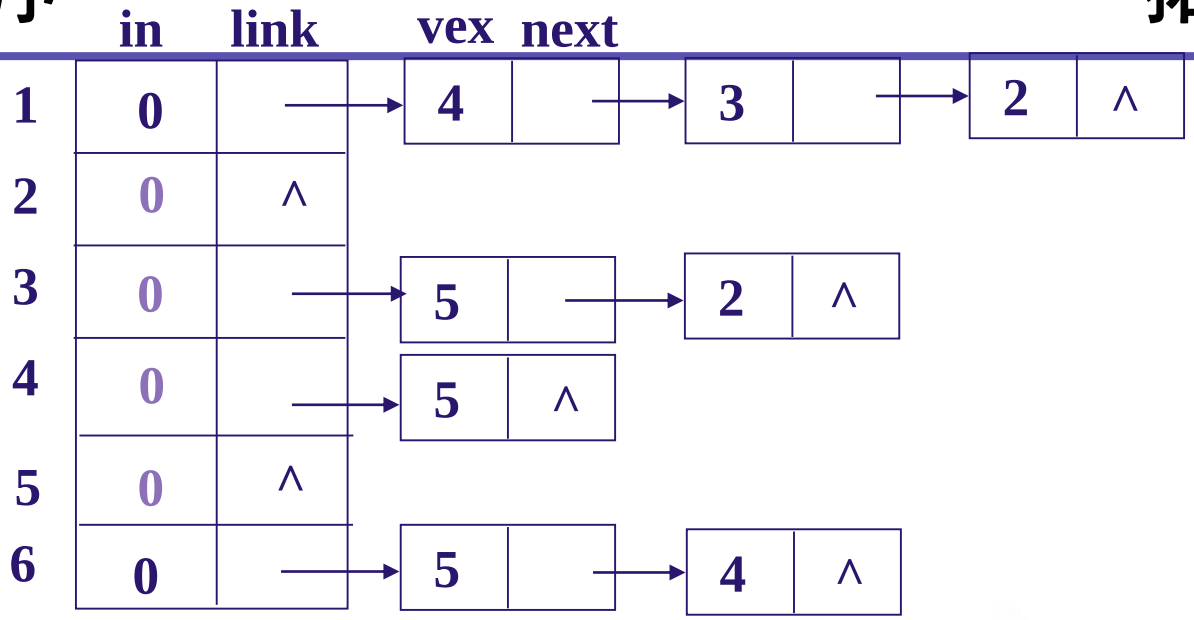


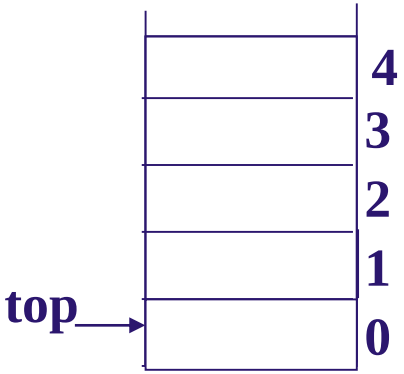


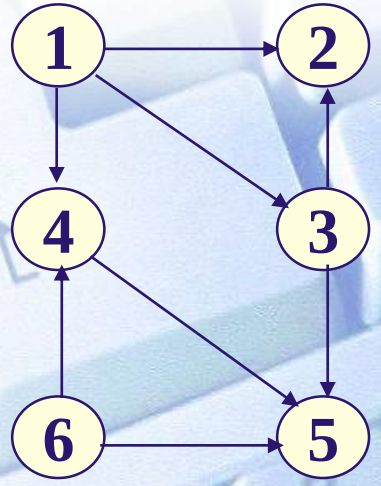
输出序列：6 1 3 2 4

拓扑排序

拓扑排序



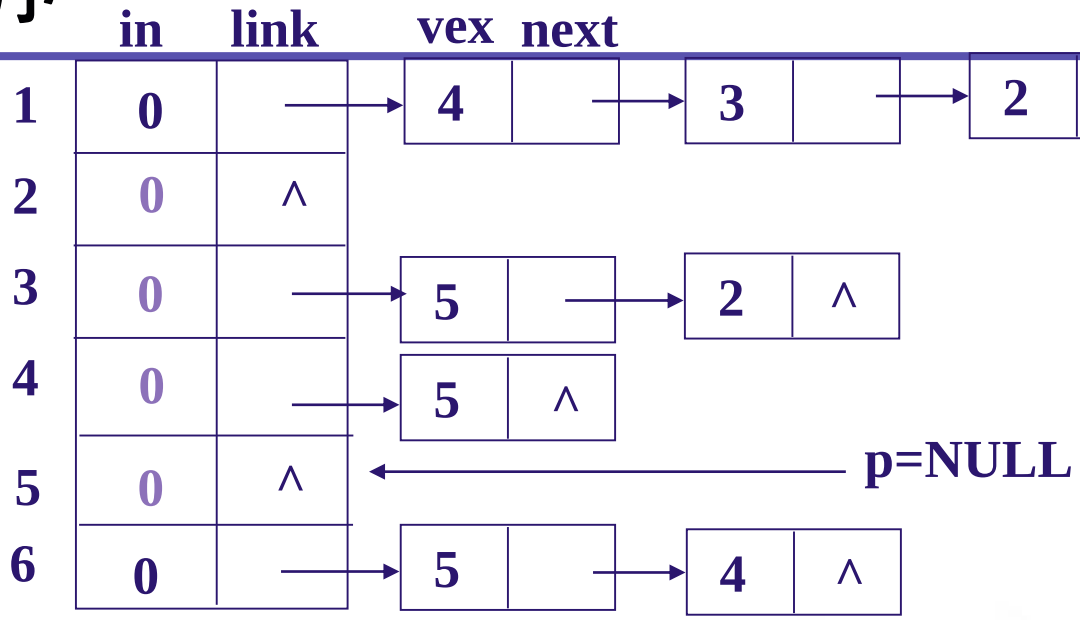


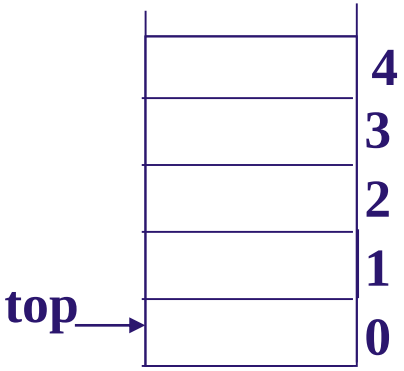


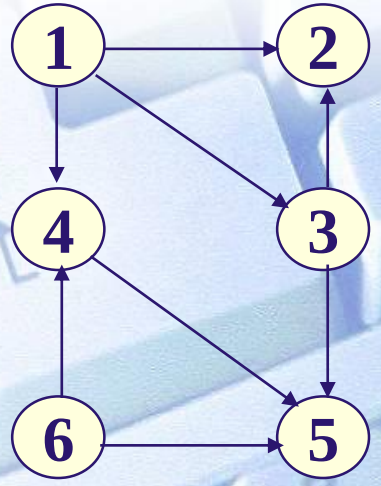
输出序列：6 1 3 2 4 5

拓扑排序

拓扑排序







输出序列：6 1 3 2 4 5

- 算法思想
  - 以邻接表作存储结构，count =0（计数输出顶点个数）
  - 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Vj并退栈，count++；在邻接表中查找Vj的直接后继Vk，把Vk的入度减1；若减1 后Vk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(count)不是n(G.vexnum)，则有向图有环；否则，拓扑排序完毕

- 算法实现


sf7. 12

- 算法分析

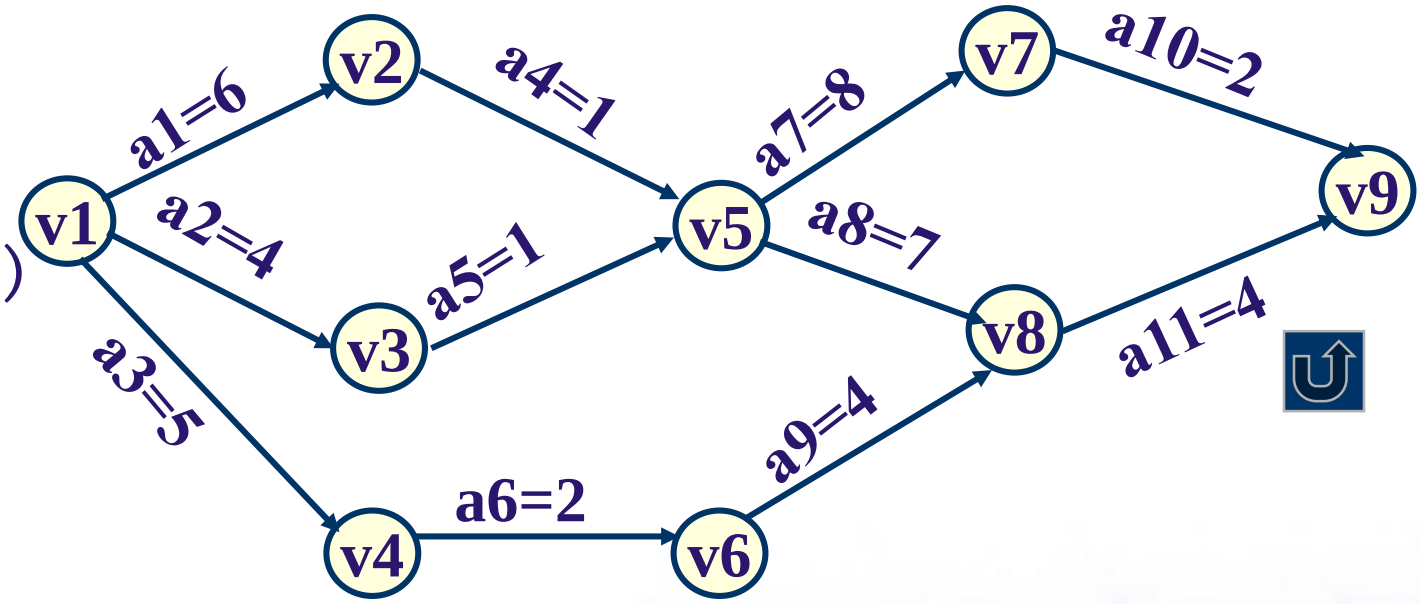
建邻接表：T(n)=O(e)
搜索入度为0的顶点的时间：T(n)=O(n)
拓扑排序：T(n)=O(n+e)

如何检测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，且该拓扑序列就是一个工程顺利完成的可行方案；否则，该工程无法顺利完成

```
Status ToplogicalSort(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
   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;
   while(!StackEmpty(S))   //处理入度为0的顶点
        {  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;
                   if(!(--indegree[k])) Push(S,k);
                }//for
        }//while
   if(count<G.vexnum) return ERROR;  //该有向图有回路
   else return OK;
}//ToplogicalSort
```

◆ 问题提出

例如：有一个工程有11项活动，9个事件(v1--v9)
v1--表示整个工程开始
v9--表示整个工程结束



假设以有向网表示一个施工流图，弧上的权值表示完成该项子工程所需时间

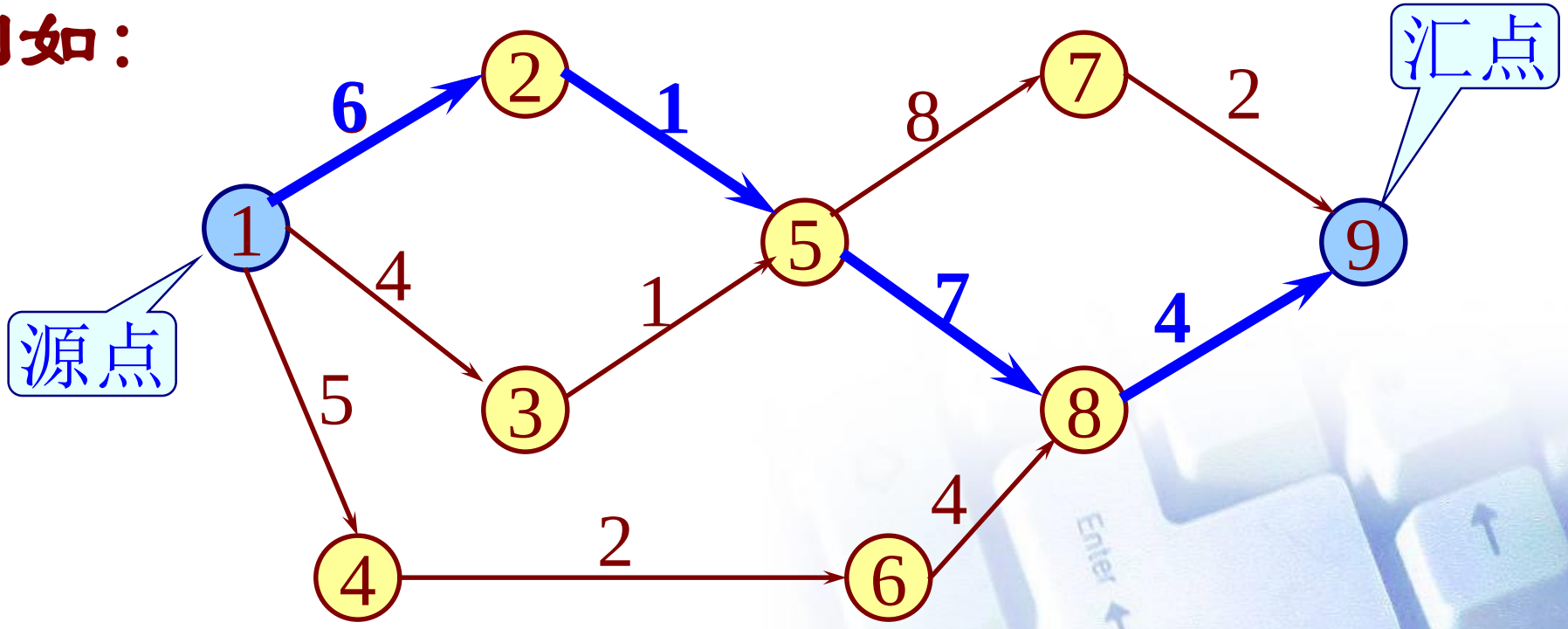
问题：（1）完成整项工程至少需要多少时间？
（2）哪些活动是影响工程进度的关键？

- 应用：1956年，美国杜邦公司提出关键路径法，并于1957年首先用于1000万美元进行化工厂建设，工期比原计划缩短了4个月。杜邦公司在采用关键路径法的一年中，节省了100万美元

- 定义
  - AOE网(Activity On Edge)——即边表示活动的网。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
  - 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ai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，即l(i)=e(i)的活动

整个工程完成的时间为：从有向图的源点到汇点的最长路径

例如：



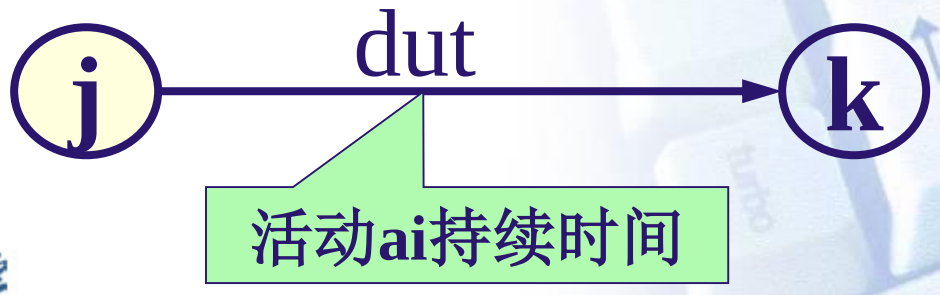
“关键活动”指的是：该弧上的权值增加 将使有向图上的最长路径的长度增加。

- 问题分析

若时间余量l(i)-e(i)=0，则该活动为关键活动，所在的路径为关键路径。

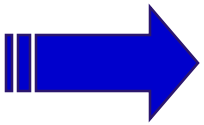
该活动的最早开始时间

= 该活动的最迟开始时间



“事件(顶点)”的最早发生时间 *ve*(j)

ve(j) = 从源点到顶点j的最长路径长度;



“事件(顶点)”的最迟发生时间 *vl*(k)

vl(k) = 从顶点k到汇点的最短路径长度;

假设第 i（ai） 条弧为 <j, k>

则 对第 i 项活动

“活动(弧)”的 最早开始时间 *ee*(i)

ee(i) = ve(j);

“活动(弧)”的 最迟开始时间 *el*(i)

el(i) = vl(k) – dut(<j,k>);

事件发生时间的计算公式:

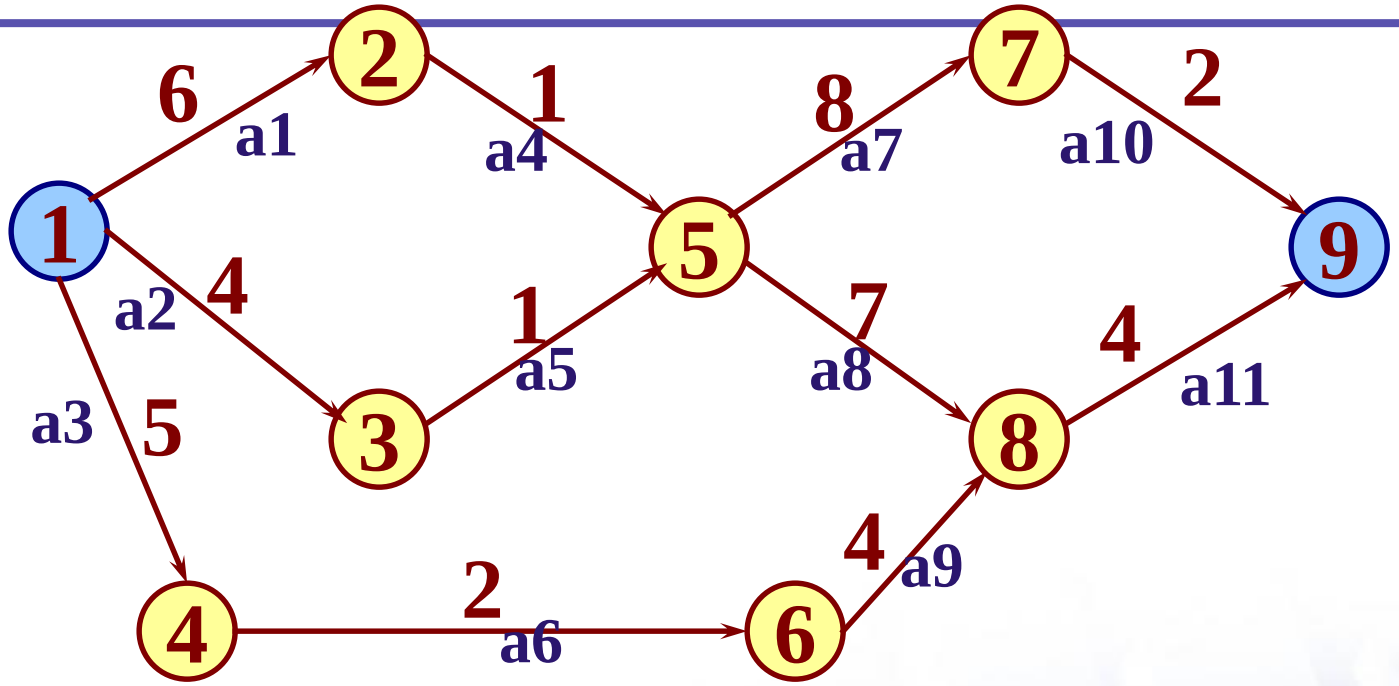
ve(源点) = 0;

ve(k) = Max{ve(j) + dut(<j, k>)}

vl(汇点) = ve(汇点);

vl(j) = Min{vl(k) – dut(<j, k>)}

拓扑排序　　关键路径



	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	15	14	18
vl	0	6	6	8	7	10	16	14	18

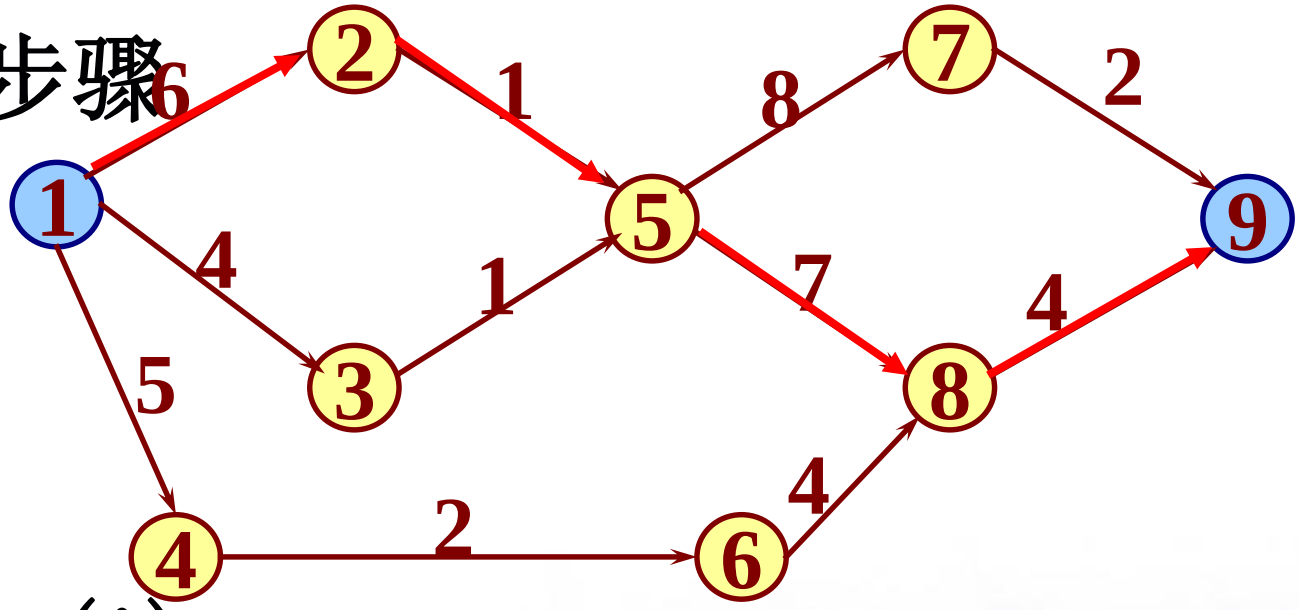
拓扑有序序列：1 - 4 - 6- 3 - 2 - 5 - 8 - 7 - 9

拓扑排序 关键路径

	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	15	14	18
vl	0	6	6	8	7	10	16	14	18

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11
权	6	4	5	1	1	2	8	7	4	2	4
e	0	0	0	6	4	5	7	7	7	15	14
l	0	2	3	6	6	8	8	7	10	16	14
l-e =0	✓			✓				✓			✓

- 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)



	1	2	3	4	5	6	7	8	9
ve	0	6	4	5	7	7	15	14	18
vl	0	6	6	8	7	10	16	14	18

- 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

要点：显然，求ve的顺序应该是按拓扑有序的次序；

而　求vl的顺序应该是按拓扑逆序的次序；

因为　拓扑逆序序列即为拓扑有序序列的逆序列，

因此　应该在拓扑排序的过程中，另设一个“栈”记下

- 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


sf7.13


sf7.14

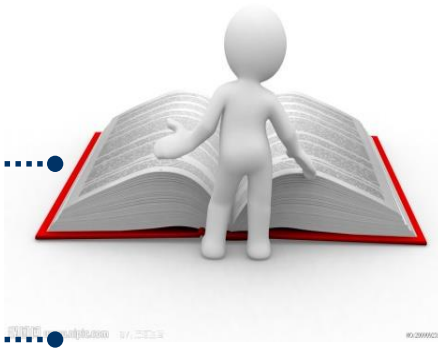
拓扑排序

```
Status ToplogicalOrder(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
   //T为拓扑序列顶点栈，S为零入度顶点栈
   //若G无回路，则栈T返回G的一个拓扑序列，且函数值为OK，否ERR
   FindInDegree(G, indegree);  //对各顶点求入度indegree[0..vernum-1]
   InitStack(S);                        //建零入度顶点栈S
   for (i=0; i<G.vexnum; ++i)
       if (!indegree[i]) push(S, i);
   InitStack(T); count=0; ve[0..G.vexnum-1]=0; //初始化各顶点最早开始时间
   while (!StackEmpty(S))
       {  Pop(S, j); Push(T, j); ++count;              //j号顶点入T栈并计数
          for (p=G.vertices[j].firstarc; p; p=p->nextarc)  //p指向j的链头结点
              {  k=p->adjvex;  //邻接点下标,对j号顶点的每个邻接点的入度减1
                 if(--indegree[k]==0)  Push(S, k);  //如入度减为0，则入栈
                 if(ve[j]+*(p->info)>ve[k])  ve[k]=ve[j]+*(p->info);//求最长路径
              }//for  *(p->info)=dut(<j,k>)//利用弧信息
       }//while
   if(count<G.vexnum) return ERROR;   //该有向网有回路
   else return OK;
}//ToplogicalOrder
```

关键路径算法

```
Status CriticalPath(ALGraph G)    //算法7.14
{  //G为有向网，输出G的各关键活动
   if (!ToplogicalOrder(G, T)) return ERROR;  //调用算法7.13判有环否
   vl[0..G.vexnum-1]=ve[0..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>    ai
            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);  //单链表即为边，依次输出每行
            ee=ve[j];  el=vl[k]-dut;           //以K变化为遍历
            tag=(ee==el)?'*':'';
            printf(j, k,dut, ee, el, tag);   //输出关键活动，*表示为关键路径
         }//for
}//CriticalPath
```

本章内容



1. 图的定义和术语
2. 图的存储结构
3. 图的遍历
4. 生成树
5. 拓扑排序
6. 最短路径

假设建立一交通咨询系统，图顶点表示城市，边表示费用或距离、时间等

- 问题

从A城市到B城市选择
一条中转次数最少的路线

从A城市到B城市
交通成本最少的路线

从A城市到B城市
交通路线最短的路线

- 广度优先搜索

  广度优先生成树

- 从某个源点到

  其余各点的最短路径

- 每一对顶点

  之间的最短路径

◆ 问题提出

用带权的有向图表示一个交通运输网

图中（有向图）：

顶点——表示城市

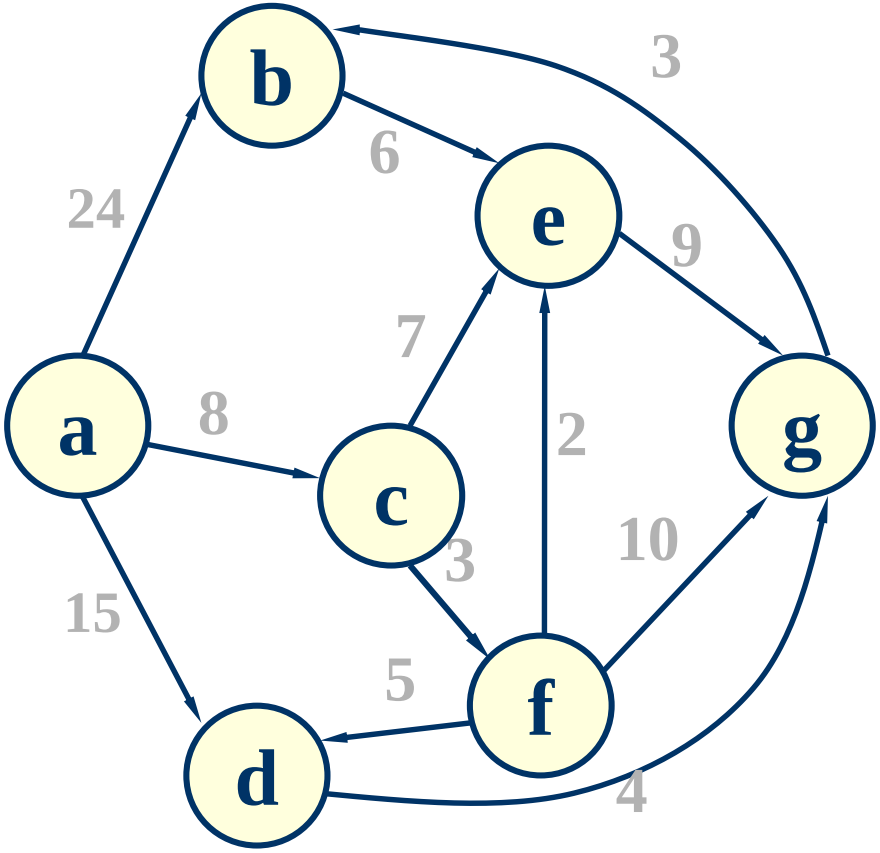
边——表示城市间的交通联系

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

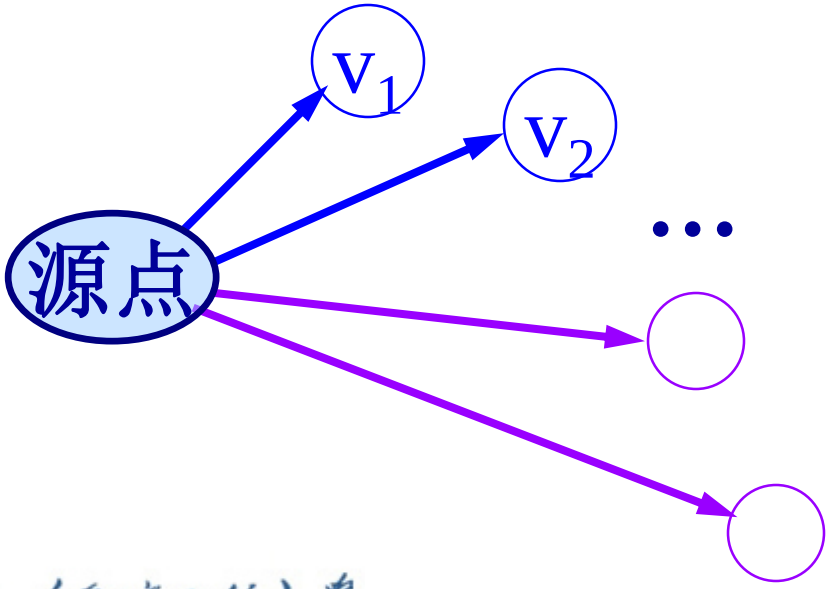
- 从某个源点到其余各顶点的最短路径



最短路径	长度
<a,d,g,b>	22
<a,c>	8
<a,d>	15
<a,c,f,e>	13
<a,c,f>	11
<a,g>	19

求从源点到其余各点的最短路径的算法的基本思想：

依路径长度递增的次序求得最短路径



其中，从源点到顶点v_1的最短路径是所有路径中长度最短者

- 路径长度最短的最短路径的特点:

在这条路径上，必定只含一条弧，并且这条弧是所有从源点出发的弧中权值最小

- 下一条路径长度次短的最短路径的特点:

它只可能有两种情况:或者是直接从源点到该点(只含一条弧)； 或者是，从源点经过顶点v_1，再到达该顶点(由两条弧组成)

- 再下一条路径长度次短的最短路径的特点：

它可能有三种情况：或者是，直接从源点到该点(只含一条弧)； 或者是，从源点经过顶点v_1，再到达该顶点(由两条弧组成)；或者是，从源点经过顶点V_1、顶点v_2，再到达该顶点

- 其余最短路径的特点：

它或者是直接从源点到该点(只含一条弧)； 或者是，从源点经过已求得最短路径的顶点，再到达该顶点。

求最短路径的迪杰斯特拉算法：

首先设置辅助数组Dist，其中每个分量Dist[k] 表示当前所求得的从源点v0到其余各顶点 k 的最短路径,其初值为：

Dist[k] = <源点v0到顶点 k 的弧上的权值>

一般情况下：

Dist[k] = <源点到其它顶点的最短路径长度>

+<其它顶点到顶点 k 的弧上的权值>

其次，v0加入S集合

再其次，设一数组Path[i]记录从源点到终点vi的当前最短路径上vi的直接前驱顶点序号。其初值为：如果从v0到vi有弧，则Path[i]为V0，否则为-1；

1）在所有从源点出发的弧中选取一条权值最小的弧，即为第一条最短路径。

$$Dist[k]=\begin{cases} G.arcs[v0][k] & \text{V0和k之间存在弧} \\ INFINITY & \text{V0和k之间不存在弧} \end{cases}$$

其中的最小值即为最短路径的长度。

2）每当加入一个新的顶点到S，修改v0到其它各顶点的*Dist*[*k*]值。

假设求得最短路径的顶点为u，u加入S

若 *Dist*[*u*]+*G.arcs*[*u*][*k*]<*Dist*[*k*]

则做如下：

将 *Dist*[*k*] 改为 *Dist*[*u*]+*G.arcs*[*u*][*k*]

Path[k]=u，u为k的直接前驱

其中u可能为多个顶点序列号的组合

算法思想：

(1) 初始化，v0加到S，即final[0]=true

v0到各顶点有弧则D[i]=G.arcs[v0][vi](vi∈V-S)

如果从v0到vi有弧，则Path[i]为V0，否则为-1；

(2) 选择下一条路径最短路径的终点vk，使

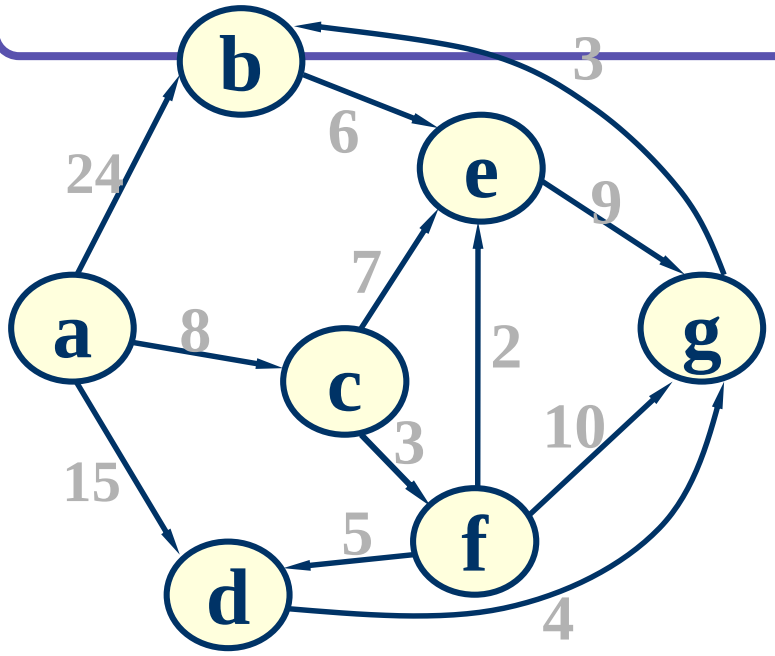
D[k]=Min{D[i] | vi ∈ V-S }

(3) 将vk加到S中，即final[vk]=true

(4) 更新从v0出发到集合V-S上任一顶点的最短路径长度，同时更改vi的前驱为vk

D[i]=D[k]+G.arcs[k][i]; Path[i]=k

(5) 重复(1)-(4)n-1次，即按路径长度递增顺序求得从v0到其余各顶点的最短路径



	0	1	2	3	4	5	6
	a	b	c	d	e	f	g
0 a	0	24	8	15	∝	∝	∝
1 b	∝	0	∝	∝	6	∝	∝
2 c	∝	∝	0	∝	7	3	∝
3 d	∝	∝	∝	0	∝	∝	4
4 e	∝	∝	∝	∝	0	∝	9
5 f	∝	∝	∝	5	2	0	10
6 g	∝	3	∝	∝	∝	∝	0

选出 dist 中的最小值在 i=1，求得第 6 条最短路径{adgb},顶点 b并入集合 S

i	a	b	c	d	e	f	g	S
dist[i]		22	8	15	13	11	19	a c f e
path[i]		adgb	ac	ad	acfe	acf	adg	d g b

```
void  ShortestPath_DIJ(MGraph G,int v0,PathMatrix &P,ShortPathTable &D)
{  //用Dijkstra算法求有向网G的v0顶点到其余顶点v的最短路径P[v]及带权   //长度D[v],若P[v][w]为真，则w是从v0到v当前求得最短路径上的顶点
   //final[v]为真当且仅当v属于S，即已经取得从v0到v的最短路径
    for (v=0; v<G.vexnum; ++v)   //对每个结点相关变量初始化
        {
          final[v]=FALSE;  D[v]=G.arcs[v0][v];     //v0到v有弧则赋权值
          for (w=0; w<G.vexnum; ++w)                //设空路径
              P[v][w]=FALSE;                         //v到w无路径
          if (D[v]<INFINITY)                         //判v到v0有无路径
             { p[v][v0]=TRUE; P[v][v]=TRUE;}
        }//for
    D[v0]=0;  final[v0]=TRUE;                       //初始化，v0顶点属于S集
```

```
//开始主循环，每次求得v0到某个v顶点的最短路径，并加v到S集
  for (i=0; i<G.vexnum; ++i)                //其余G.vexmun-1个顶点
     { min=INFINITY;                         //当前所知离v0最近的距离
       for (w=0; w<G.vexnum; ++w)    //找{min D[w]}
            if (!final[w])                      //如果w没并入S
               if (D[w]<min)              //w顶点离v0顶点更近
                  {
                     min=D[w];
                     v=w;
                  }
       final[v]=TRUE;                        //离v0最近的v加入S集
       for (w=0; w<G.vexnum;++w)     //更新当前最短路径及距离
            if (!final[w] &&(min+G.arcs[v][w]<D[w]))   //修改D[w]和P[w]
              { D[w]=min+G.arcs[v][w];
                P[w]=P[v];   P[w][w]=TRUE;
              }
     }
}
```

求每一对顶点之间的最短路径

- 每次以一个顶点为源点，重复执行Dijkstra算法n次——$T(n)=O(n^3)$

本章小结

1. 熟悉图的各种存储结构及其构造算法，了解实际问题的求解效率与采用何种存储结构和算法有密切联系。

2. 熟练掌握图的两种搜索路径的遍历：深度优先搜索（利用栈）和广度优先搜索（利用队列）算法。应注意图的遍历算法与树的遍历算法之间的类似和差异。

3. 熟练掌握最小生成树的构造方法，选点法和选边法。

4. 熟练掌握拓扑排序方法，有向图环检测方法

5. 理解求解关键路径和两顶点间最短路径问题及算法。

6. 理解教科书中讨论的各种图的算法。

下课休息！