第七章 图

- 图的定义和基本术语
- 图的存储构造
- 图的遍历
- 生成树
- 最短途径

图的定义和基本术语

本章简介另一种非线性数据构造—图，主要学习图的存储构造以及若干图的操作的实现。

图是一种多对多的构造关系，每个元素能够有零个或多种直接前趋；零个或多种直接后继。

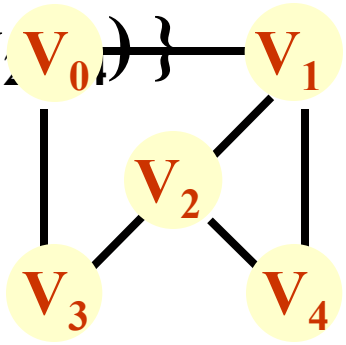
一、图的定义

图G由两个集合构成，记作G= (V, {A})
其中V是顶点的非空有限集合， A 是边的有限集合，其中边是顶点的无序对或有序对(此时的图称为无向图或有向图)。

无向图$G_1=(V_1, \{A_1\})$，$V_1=\{ v_0 ,v_1 ,v_2 ,v_3 ,v_4 \}$，

$A_1=\{ (v_0,v_1) , (v_0,v_3) , (v_1,v_2) , (v_1,v_4) , (v_2,v_3)$

$, (v_2,v_4) \}$



G1图示

无序对(v_i ,v_j)：

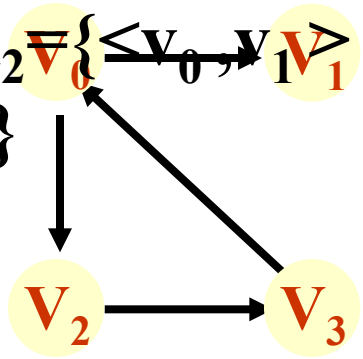
用连接顶点v_i、v_j的线段表达，称为无向边；

有向图$G_2=(V2, \{A2\})$，$V_2=\{v_0\ v_1,v_2,v_3\}$

，

$A_2=\{<v_0 ,v_1>, <v_0 ,v_2 > , <v_2 ,v_3 > , <v_3 ,v_0$

$>\}$



G2 图示

有序对$<v_i,v_j>$：

用觉得v_i起点(弧尾)、以v_j为终点(弧头)的有向线段表达，称为有向边或弧；

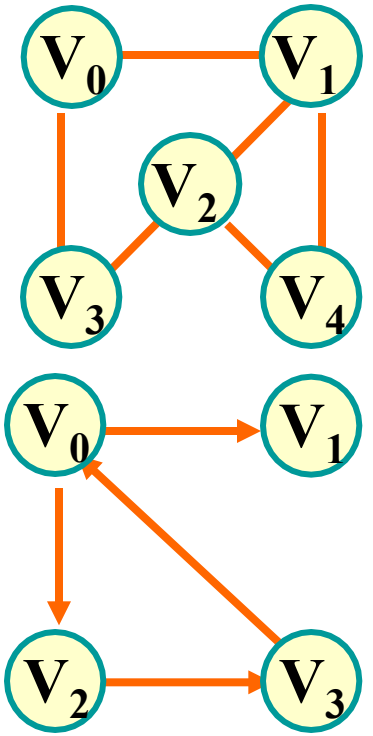
图的实例

例1 交通图（公路、铁路）　　顶点：地点
边：连接地点的路
交通图中的有单行道、双行道，分别用有向边、无向边表达；

例2　电路图　　顶点：元件
边：连接元件之间的线路

例3 通讯线路图　顶点：地点
边：地点间的连线

例4 多种流程图(如生产流程图)
顶点：工序
边：各道工序之间的顺序关系



ADT 图的定义

ADT Graph{

数据对象V：V是具有相同特征的数据元素的集合，称为顶点集。

数据关系R：

R={VR}

VR ={<v ,w > | v ,w ∈V ， 且p(v ,w) , <v ,w > 表示从v到w 的弧，谓词p(v ,w)定义了弧<v ,w >的意义或信息。}

基本操作：

CreateGraph(&G , V, V R) // 按定义构造图

初始条件：V是图的顶点集, V R 是图中弧的集合。

操作成果：按V和V R的定义构造图G 。

DestroyGraph (&G) // 销毁

初始条件:图G存在。

操作成果:销毁图G 。

LocateVex(G, u) // 定位

初始条件:图G存在，u 和G中顶点有相同特征 。

操作成果: 若G中存在顶点u ，则返回该顶点在图中位置 ；不然返回其他信息。

GetVex(G, v)// 求值

初始条件:图G存在，v 是G中某个顶点。

操作成果:返回v的值。

PutVex(&G, v, value) // 赋值

初始条件:图G存在，v 是G中某个顶点。

操作成果:对 v 赋值value。

FirstAdjVex(G, v) // 求第一种邻接点

初始条件:图G存在，v 是G中某个顶点。

操作成果:返回 v 的第一种邻接点。若顶点v 在 G 中没有邻接顶点，则返回“空” 。

NextAdjVex(G, v, w) // 求下一种邻接点

初始条件:图G存在，v 是G中某个顶点， w 是 v 的邻接点 。

操作成果: 返回 v 的（相对于 w 的）下一种邻接点。若 w 是 v 的最终一种邻接点，则返回“空”

InsertVex(&G, v); // 插入顶点

初始条件:图G存在，v和G中顶点有相同特征 。

操作成果: 在图G中增添新顶点v。

DeleteVex(&G, v) //删除顶点

初始条件: 图G存在， v和G中顶点有相同特征 。

操作成果:删除G中顶点v及其有关的弧。

InsertArc(&G, v, w) //插入弧

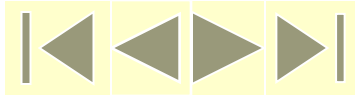
初始条件:图G存在，v 和w是G中两个顶点。

操作成果:在G中增添弧<v,w>，若G是无向的，则还增添对称弧<w,v>。

DeleteArc(&G, v, w) //删除弧

初始条件:图G存在，v 和w是G中两个顶点。

操作成果:在G中删除弧<v,w>，若G是无向的，则还删除对称弧<w,v>。



DFSTraverse(G, v, Visit()) //深度优先遍历

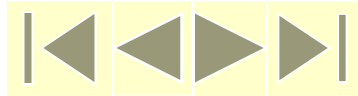
初始条件:图G存在，v 是G中某个顶点，
Visit是顶点的应用函数 。

操作成果: 从顶点v起深度优先遍历图G，并对每个顶点调用函数Visit一次且仅一次。
一旦Visit（）失败，则操作失败。

BFSTraverse(G, v, Visit()) //广度优先遍历

初始条件:图G存在，v 是G中某个顶点，
Visit是顶点的应用函数 。

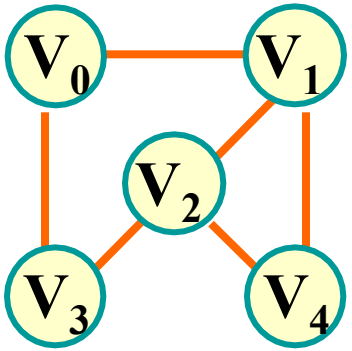
操作成果: 从顶点v起广度优先遍历图G，并对每个顶点调用函数Visit一次且仅一次。
一旦Visit（）失败，则操作失败。



二、图的基本术语

1 邻接点及关联边　邻接点：边的两个顶点；

关联边：若边e= (v, u), 则称边e与顶点v和u 有关联；



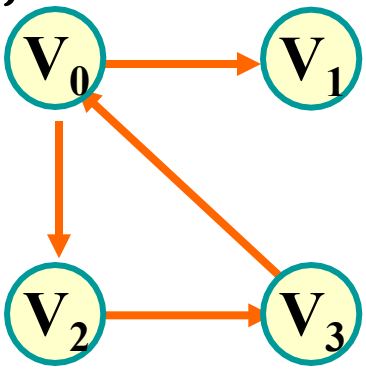
2 顶点的度、入度、出度

顶点v的度 TD(v)= 与v有关联的边的数目。

在有向图中：顶点v的出度OD(v) =以v为起点有向边数；

顶点v的入度ID(v) =以v为终点有向边数；

$$TD(v) = OD(v) + ID(v)$$



设图G的顶点数为n，边数为e

则图的全部顶点的度数之和 = 2*e

（每条边对图的全部顶点的度数之和 “贡献” 2度）

3 途径、回路(环)

4 无向图G_1=（V_1，E_1）中的顶点序列$v_1,v_2,\ldots,v_k$若$(v_i,v_{i+1})\in E_1$ (i=1,2,…k-1), v =v_1, u =v_k, 则称该序列是从顶点v到顶点u的途径；若v=u，则称该序列为回路；在G_1中，v_0,v_1,v_2,v_3是v_0到v_3的途径；

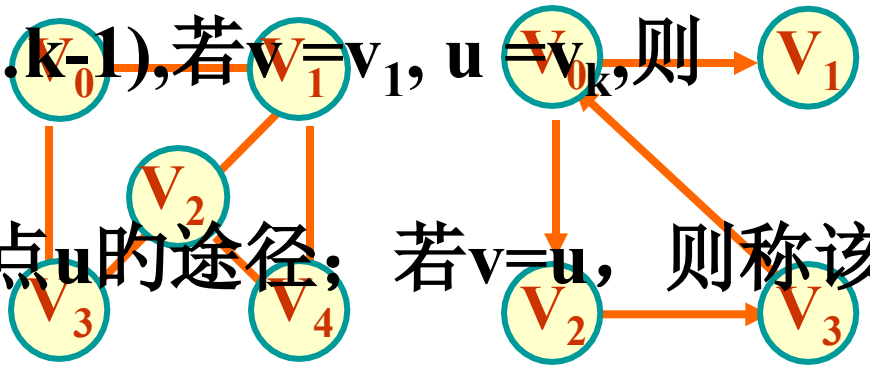
5 v_0,v_1,v_2,v_3,v_0是回路；

6 有向图G_2=（V_2，E_2）中的顶点序列$v_1,v_2,\ldots,v_k$,

7 $<v_i,v_{i+1}>\in E_2$ (i=1,2,…k-1),若v_1=v_1, u =v_k,则称该

8 序列是从顶点v到顶点u的途径；若v=u，则称该序

9 列为回路；在G_2中，



无向图G1　　有向图G2

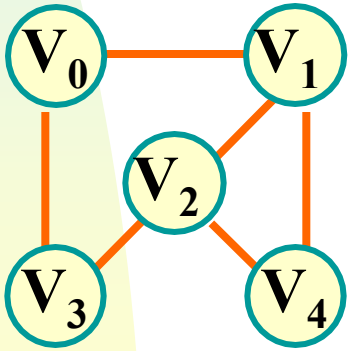
3 简朴途径和简朴回路

4 在一条途径中，若全部顶点各不相同，则称该途径为简朴途径；若除起点和终点外，其他顶点各不相同的回路称为简朴回路。

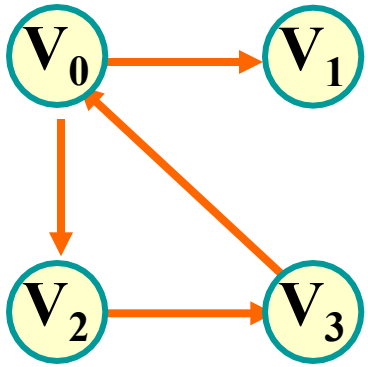
例 在图G1中，V_0,V_1,V_2,V_3是简朴途径；

V_0,V_1,V_2,V_4,V_1不是简朴途径；

在图G2中，V_0,V_2,V_3,V_0是简朴回路；



无向图G1



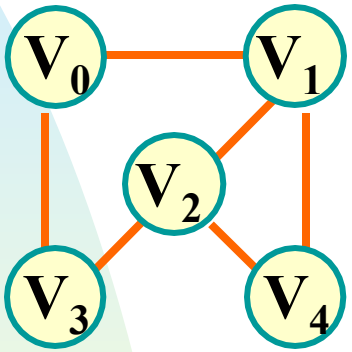
有向图G2

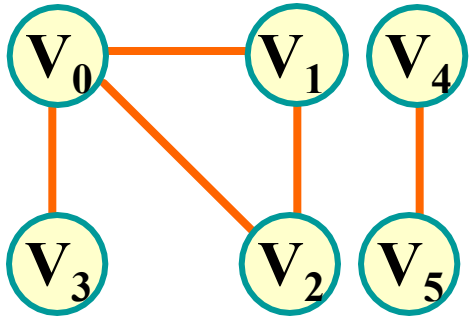
4 连通图（强连通图）

5

在无（有）向图G=(V, E)中，若对任何两个顶点 v、u 都存在从v 到 u 的途径，则称G是连通图（强连通图）

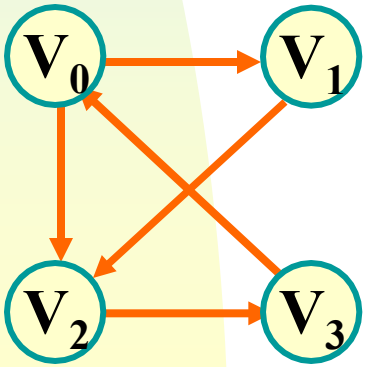
连通图

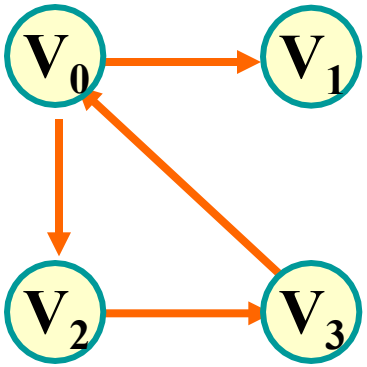




非连通图

强连通图



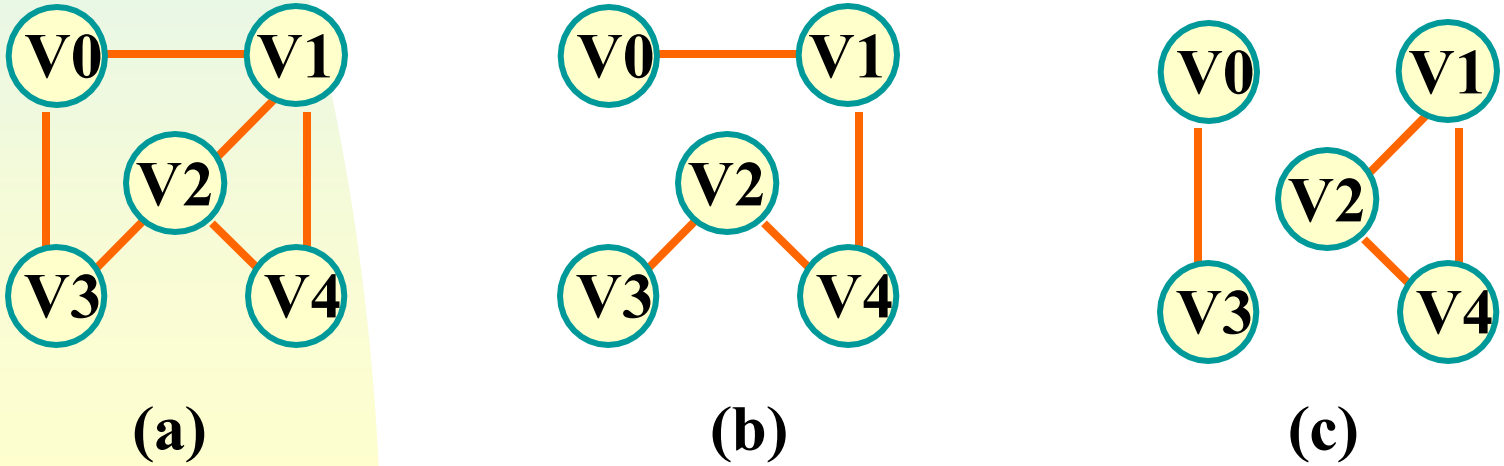


非强连通图

5 子图

设有两个图G=（V，E）、G1= （V1，E1），若V1⊆ V，E1 ⊆ E，E1关联的顶点都在V1中，则称 G1是G的子图；

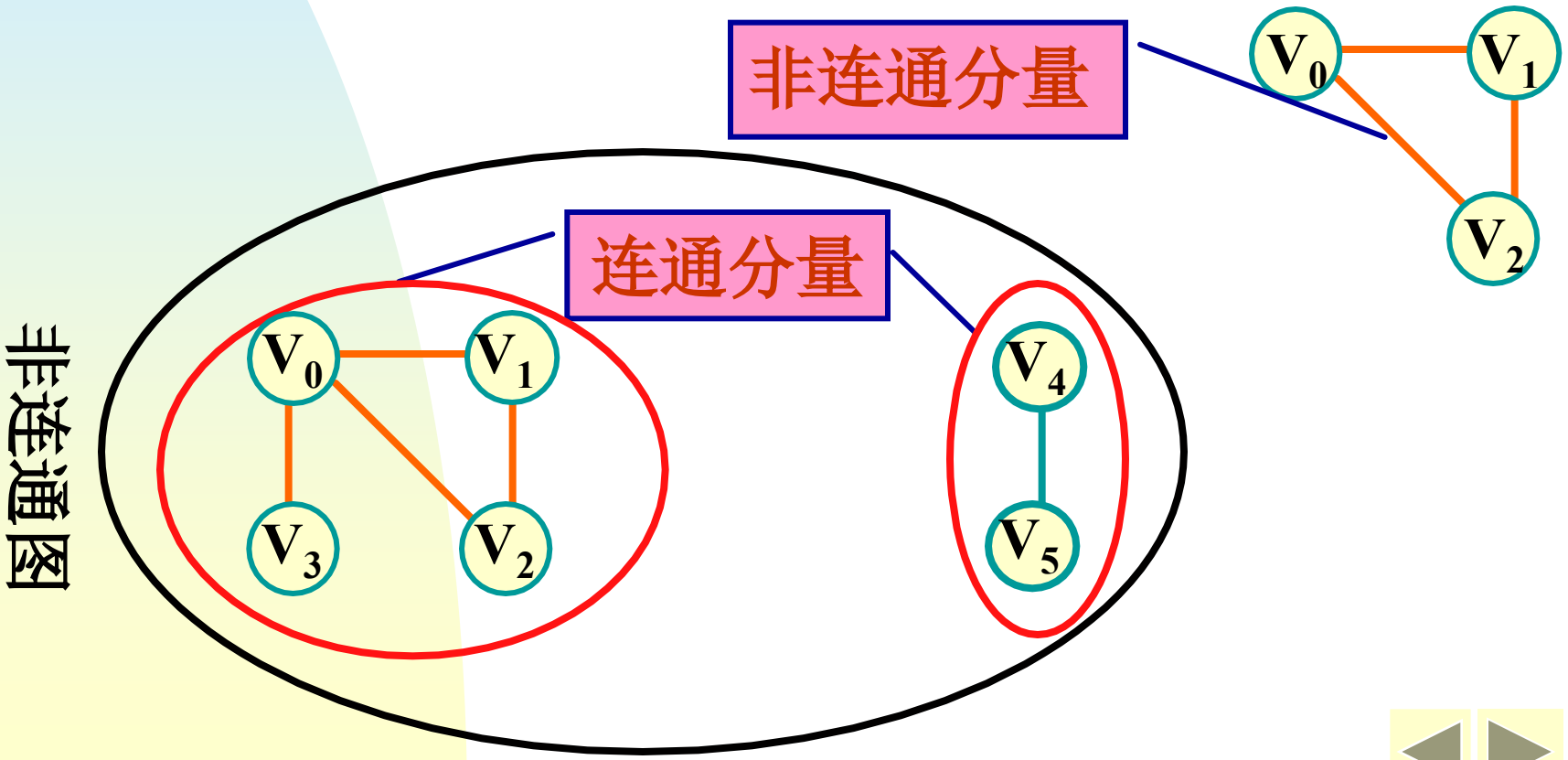
例　下列 (b)、(c) 是 (a) 的子图



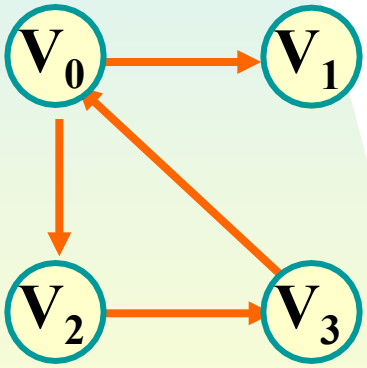
6 (强)连通分量

7 无向图G的极大连通子图称为G的连通分量。

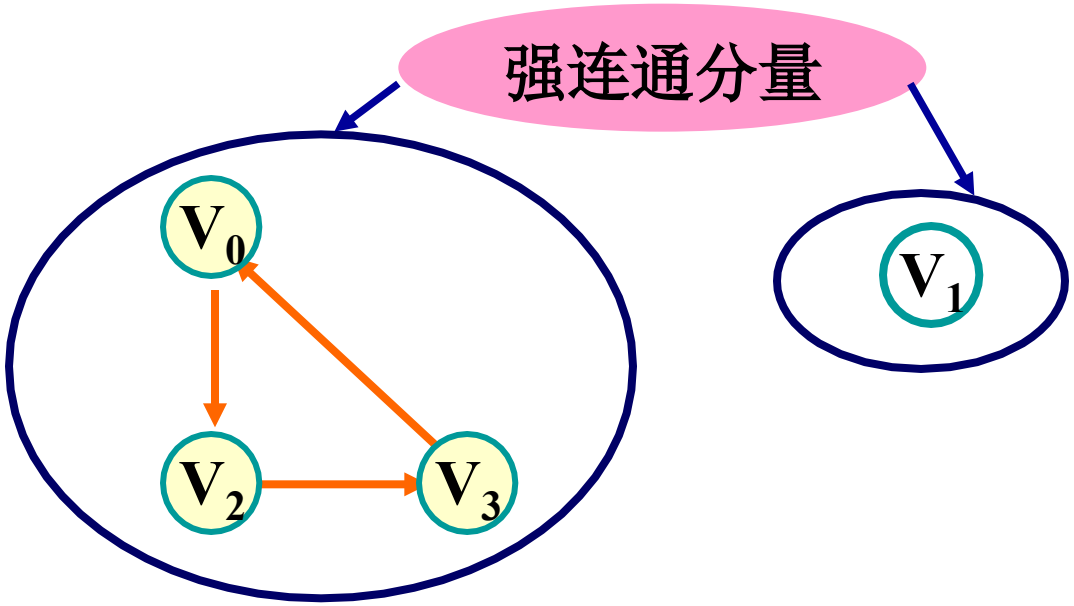
8 极大连通子图意思是：该子图是 G 连通子图，将 G 的任何不在该子图中的顶点加入，子图不再连通；



有向图D 的极大强连通子图称为D 的强连通分量。极大强连通子图意思是：该子图是D强连通子图，将D的任何不在该子图中的顶点加入，子图不再是强连通的。



非强连通图

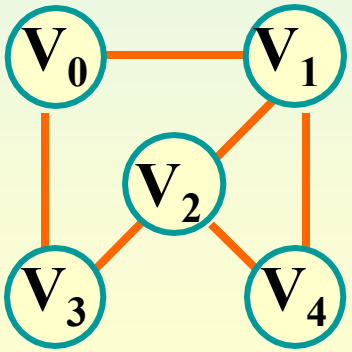


7 生成树

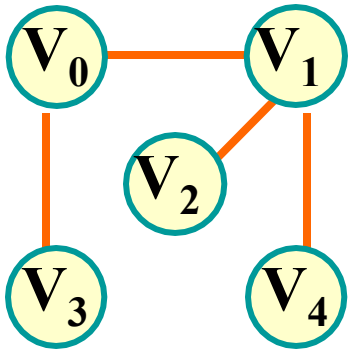
包括无向图G 全部顶点的极小连通子图称为G 的生成树。极小连通子图意思是：该子图是G 的连通子图，在该子图中删除任何一条边，子图不再连通。

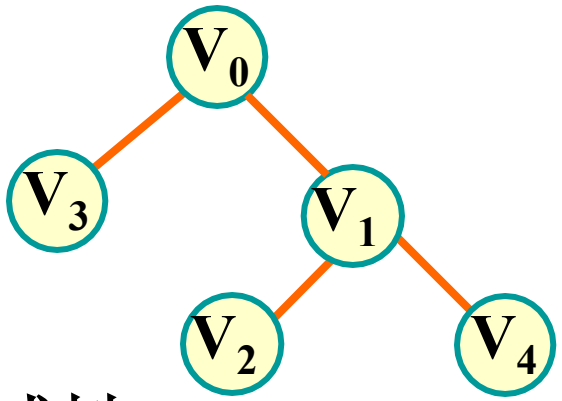
若T是G 的生成树当且仅当T 满足如下条件：

- T是G 的连通子图；
- T包括G 的全部顶点；
- T中无回路。



连通图 G

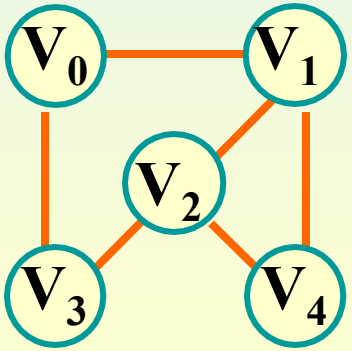




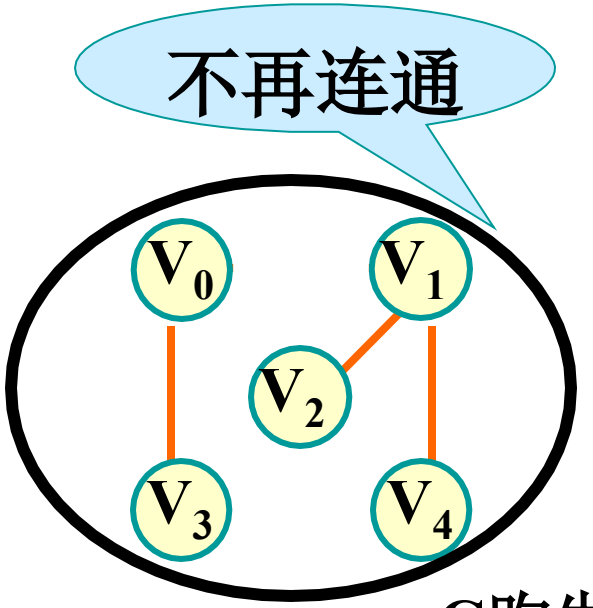
G的生成树

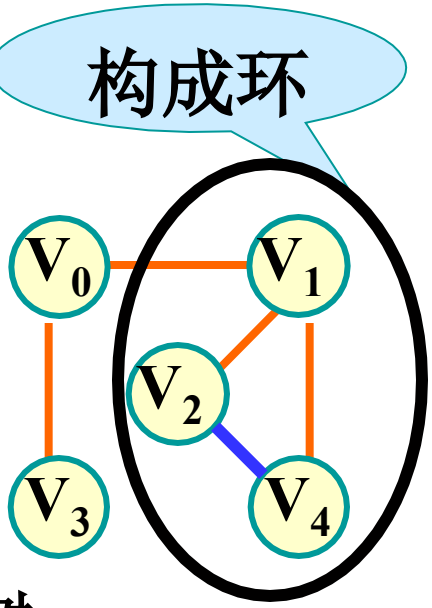
阐明：

- 一棵n个顶点的生成树有且仅有足以构成树的n-1条边。
- 若在一棵生成树上删除一条边，就不再连通
- 若在一棵生成树上添加一条边，肯定构成一种环。



连通图 G





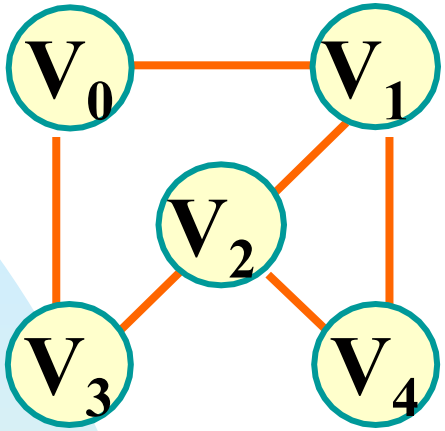
G的生成树

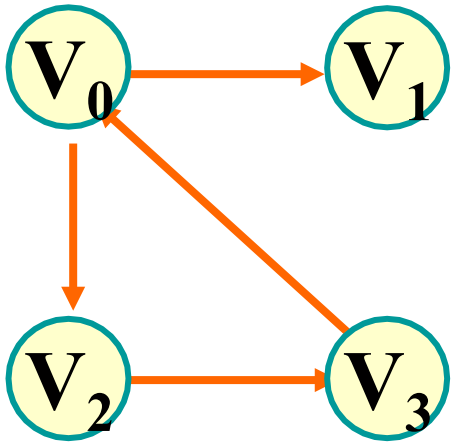
图的存储构造

因为图中任意两个顶点之间都可能存在联络，所以难以以数据元素在存储区中物理位置表达它们间的关系，仍能够借助数组表达之。

另一方面，用也能够多重链表达图。但因为图中顶点的度可能相差悬殊，会所以造成空间的挥霍；反之，若按每个顶点的度设计不同的结点构造，又会造成操作上的不便。

应根据详细的图和需要，设计恰当的结点构造和表构造。





图的存储构造至少要保存两类信息：

1）顶点的数据；

2）顶点间的关系。

怎样表达顶点间的关系？

常用图的存储表达

一、图的数组(邻接矩阵)存储表达

二、图的邻接表存储表达

三、有向图的十字链表存储表达

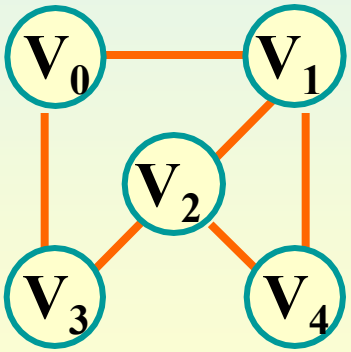
四、无向图的邻接多重表存储表达

一、数组表达法(邻接矩阵表达法)

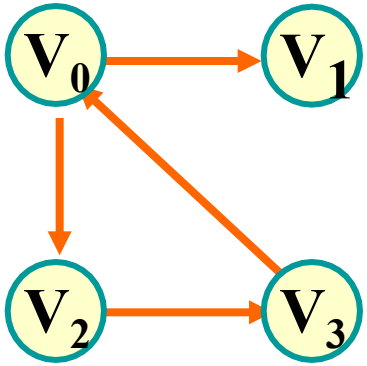
在数组表达法中，
用邻接矩阵表达顶点间的关系

邻接矩阵：G的邻接矩阵是满足如下条件的n阶矩阵：

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



$$\begin{pmatrix}0&1&0&1&0\\1&0&1&0&1\\2&0&1&0&1\\3&0&1&0&0\\4&0&1&1&0\end{pmatrix}$$



$$\begin{pmatrix}0&1&1&0\\0&0&0&0\\0&0&0&1\\1&0&0&0\end{pmatrix}$$

//— —图的数组(邻接矩阵)存储表达——

```
#define INFINITY  INT_MAX //最大值∞
#define MAX_VERTEX_NUM  20//最大顶点个数
typedef enum{DG,DN,UDG, UDN }graphkind;
                //{有向图，有向网，无向图，无向网}
typedef struct ArcCell {// 弧的定义
  VRType  adj;//VRType是顶点关系类型。对无权图，
      //用1或0表达相邻否；对带权图，则为权值类型。
  InfoType  *info;  // 该弧有关信息的指针
} ArcCell ,
   AdjMatrix[MAX_VERTEX_NUM]
[MAX_VERTEX_NUM];
```

```
typedef struct { //图的定义
    VertexType
vexs[MAX_VERTEX_NUM];
                    //顶点向量—保存顶点数据
    AdjMatrix    arcs;
                  //邻接矩阵—保存顶点间关系
    int    vexnum, arcnum; //顶点数，弧数
    GraphKind   kind;     //图的种类标志
 } MGraph;
```

无向图数组表达法特点：

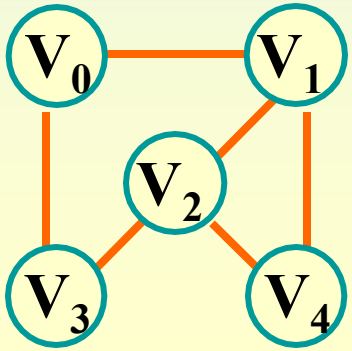
1）无向图邻接矩阵是对称矩阵，同一条边表达了两次；

2）顶点v的度：等于二维数组相应行（或列）中1的个数；

3）判断两顶点v、u是否为邻接点：只需判二维数组相应分量是否非零；

4）顶点不变，在图中增长、删除边：只需对二维数组相应分量赋非零值或清零；

5）用二维数组存储顶点数为 n的图, 占用存储空间只与它的顶点数n有关，与边数e无关。合用于边稠密的图。



$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 0 & 1 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} \\ \\ 1 \\ \\ 0 \end{matrix}$$

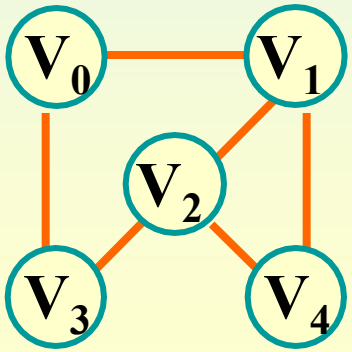
对有向图的数组表达法可做类似的讨论

图的基本操作的实现(采用数组表达法)：

1）求无向图某顶点v_i的度：(或有向图v_i的出度)
A[i][0] 到A[i][n-1]中的非零个数，即数组A第i行的非零元素的个数；

2）求有向图某顶点v_i 的 入度：A[0][i]到A[n-1][i] 中的非零个数，即数组A中第i列的非零元素的个数；

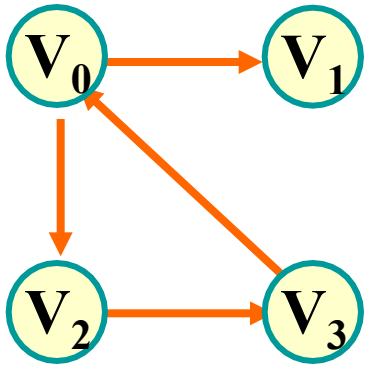
3）求图中的总边数：扫描整个数组A，统计出数组中非零元素的个数。无向图的总边数为非零元素个数的二分之一，而有向图的总弧数为非零元素个数。



$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 0 & 1 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{pmatrix}$$

1

0



$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

图的构造操作的实现(采用数组表达法)

```
Status CreateGraph(Mgraph &G) {
      //采用数组表达法，构造图G
  scanf(&G.kind);
  switch(G.kind) {
    case   DG: return CreateDG(G);   //构造有向图G
    case   DN: return CreateDN(G);   //构造有向网G
    case UDG: return CreateUDG(G);//构造无向图G
    case UDN: return CreateUDN(G);//构造无向网G
    default    : return ERROR;
  }
}// CreateGraph
```

无向网的构造算法

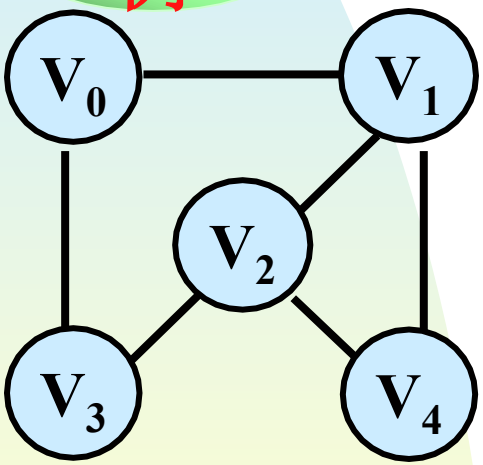
```
Status CreateUDN(Mgraph &G) {
      //采用数组表达法，构造无向网G
  scanf(&G.vexnum, &G.arcnum , &IncInfo);
      //IncInfo为0则各不含其他信息
  for(i=0;i< G.vexnum;++i) scanf(G.vexs[i]); //构造顶点向量
  for(i=0;i< G.vexnum;++i) //初始化邻接矩阵
    for(j=0;j< G.vexnum;++j)
G.arcs[i][j]={INFINITY,NULL};
  for(k=0;k< G.arcnum;++k) { //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);
                    //拟定v1和v2在G中位置
     G.arcs[i][j].adj=w; //弧< v1, v2 >的权值
     if (IncInfo) input(* G.arcs[i][j].info)
                    //若弧具有有关信息,则输入
```

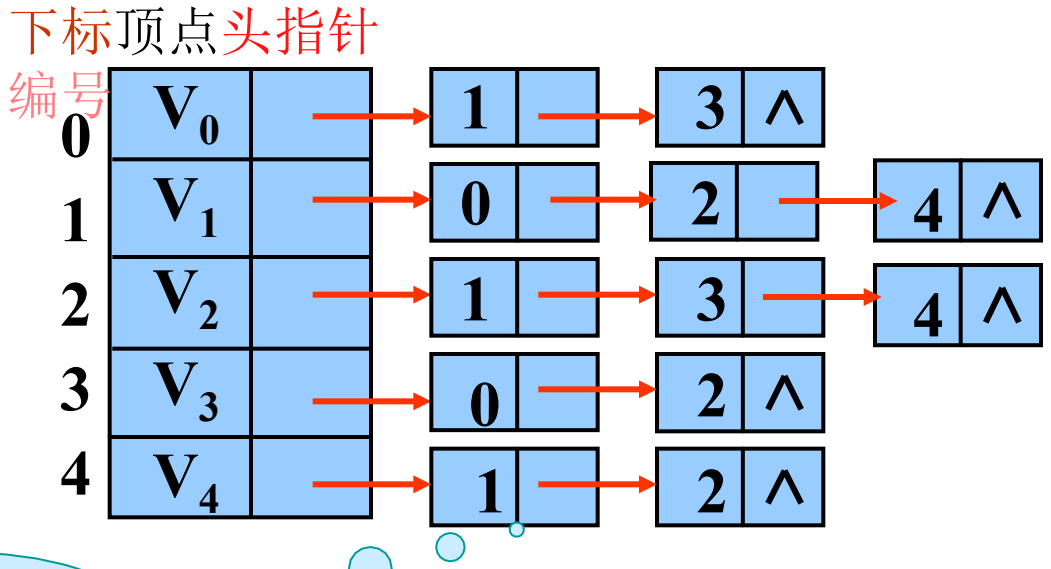
二、邻接表

1 无向图的邻接表

顶点一般按编号顺序将顶点数据存储在一维数组中；关联同一顶点的边用线性链表存储。

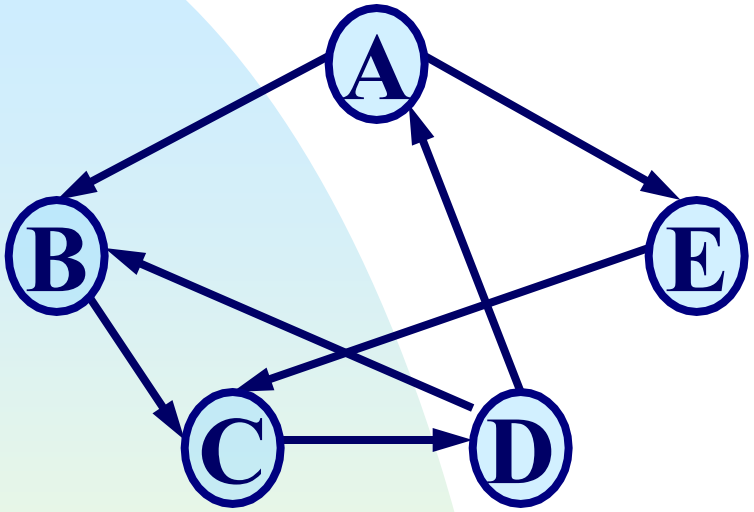
例

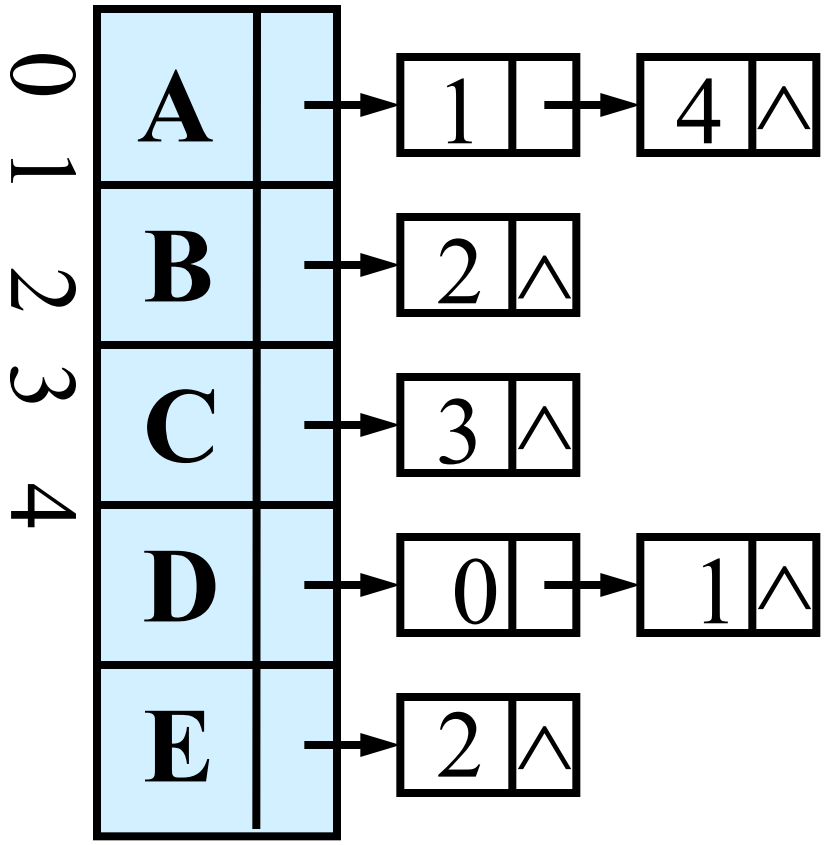




该结点表达边（V_4, V_j），其中的j是V_j在一维数组中的位置

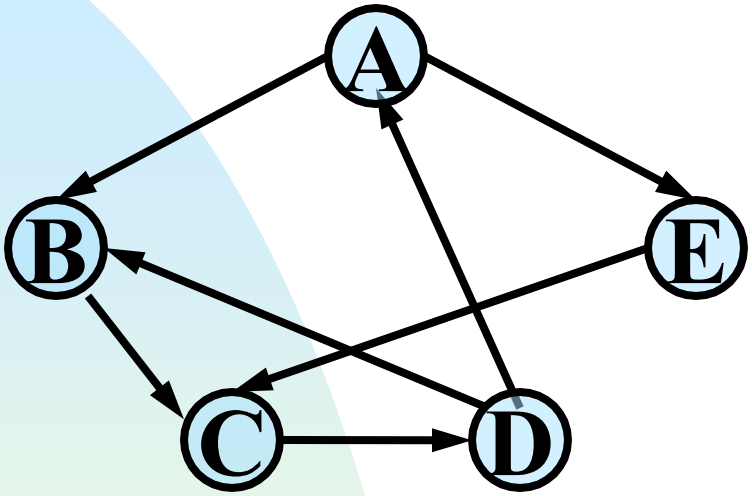
2 有向图的邻接表

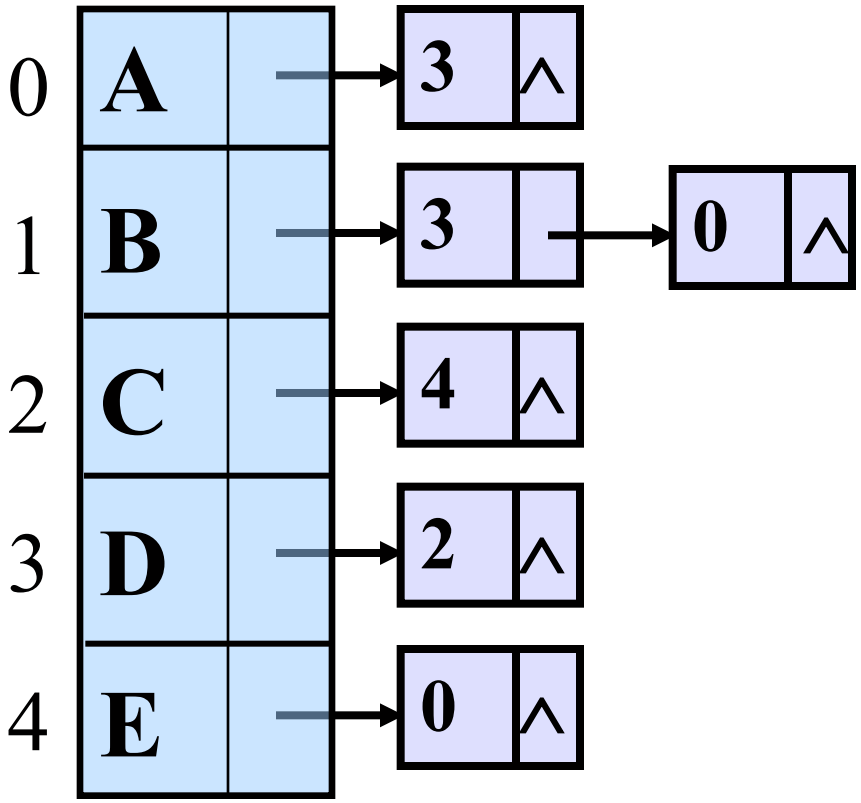




可见，在有向图的邻接表中不易找到指向该顶点的弧。

3 有向图的逆邻接表





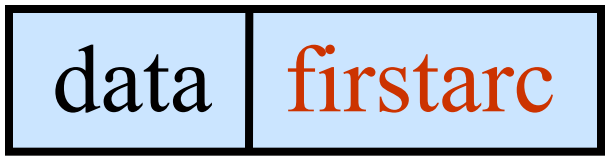
在有向图的逆邻接表中，对每个顶点，链接的是指向该顶点的弧。

//— —图的邻接表存储表达——

弧的结点构造


```
typedef struct ArcNode {
  int      adjvex; //  该弧所指向的顶点的位置
  struct ArcNode  *nextarc;
                   //  指向下一条弧的指针
  InfoType    *info;   //  该弧有关信息的指
针
} ArcNode; //  表结点
```

顶点的结点构造




```
typedef struct VNode {
  VertexType  data;   // 顶点信息
  ArcNode  *firstarc;
        // 指向第一条依附自该顶点的弧
} VNode,
AdjList[MAX_VERTEX_NUM];
                // 头结点、头结点数组类型
```

图的构造定义(邻接表)

```
typedef struct {
    AdjList  vertices;
    int      vexnum, arcnum;
    int      kind;   //  图的种类标志
} ALGraph;
```

无向图的邻接表的特点

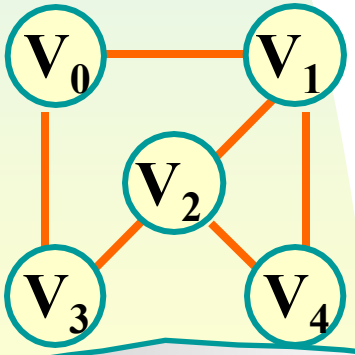
1）在G邻接表中，同一条边相应两个表结点；

2）顶点v的度：等于v相应线性链表的长度；

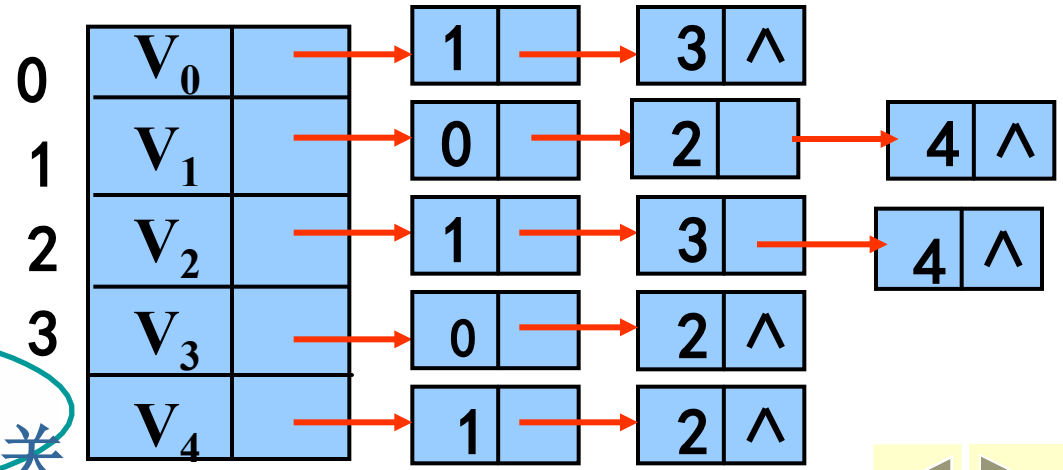
3）鉴定两顶点v ，u是否邻接：要看v相应线性链表中有无相应的结点u ；

4）在G中增减边：在两个单链表插入、删除结点；

5）设存储顶点的一维数组大小为m(m≥图的顶点数n)，图的边数为e，G占用存储空间为：m+2×e，与 G 的顶点数、边数都有关，合用于边稀疏的图。

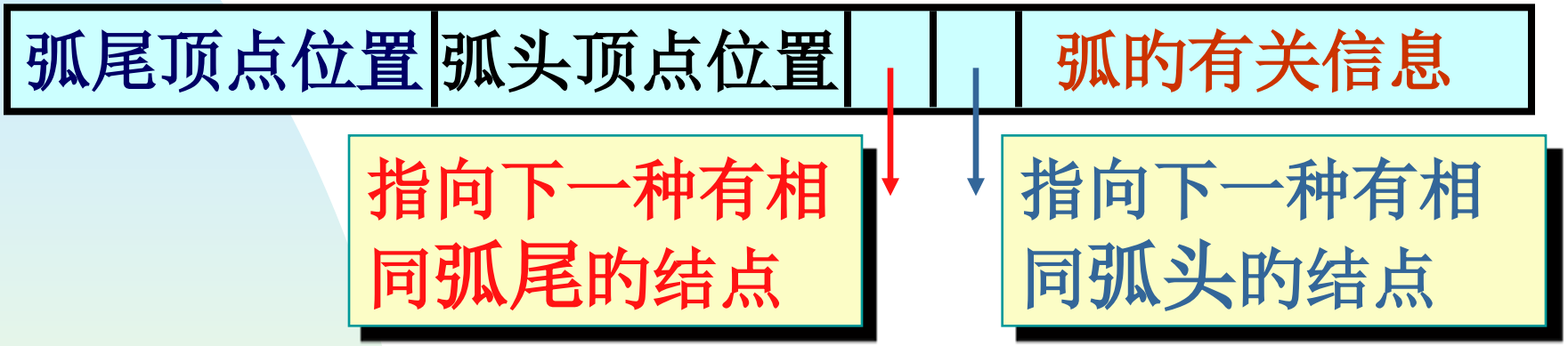


邻接表的空间代价
与图的边及顶点数都有关



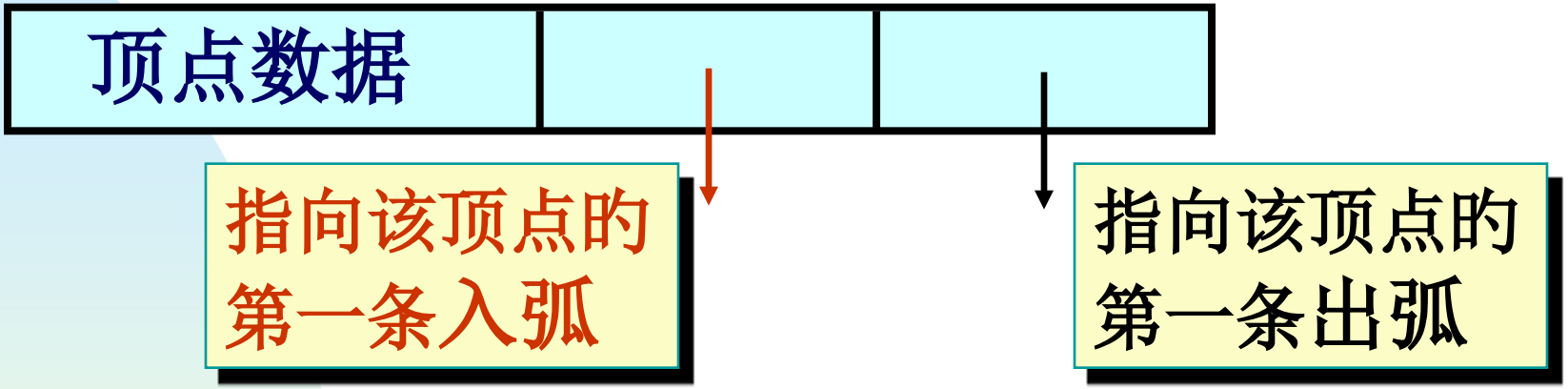
三、有向图的十字链表存储表达

弧的结点构造




```
typedef struct ArcBox { // 弧的构造表达
    int tailvex, headvex;   InfoType  *info;
    struct ArcBox  *tlink, *hlink;
} ArcBox;
```

顶点的结点构造




```
typedef struct VexNode { //  顶点的构造表达
    VertexType  data;
    ArcBox  *firstin, *firstout;
} VexNode;
```

有向图的构造表达(十字链表)

```
typedef struct {
   VexNode
xlist[MAX_VERTEX_NUM];
          //  顶点结点表头向量
   int   vexnum, arcnum;
          //有向图的目前顶点数和弧数
} OLGraph;
```