数据结构

（C语言版）（第2版）

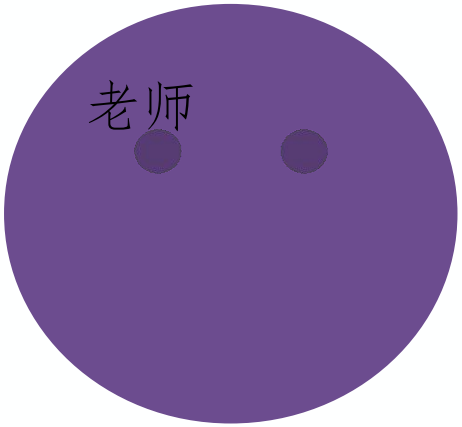
图

图的存储结构

主讲教师：汪红松

教 学 内 容 Contents

一、邻接矩阵
二、邻接表
三、十字链表——用于有向图
四、邻接多重表——用于无向图



图的存储结构

顺序存储结构

数组表示法
（邻接矩阵）

链式存储结构

多重链表

邻接表
邻接多重表
十字链表

重点介绍

邻接矩阵(数组)表示法

邻接表(链式)表示法

一、邻接矩阵

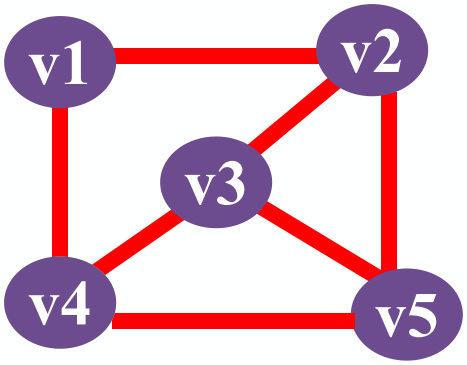
1.数组（邻接矩阵）表示法

- 邻接矩阵是表示顶点之间相邻关系的矩阵。
- 设图 A = (*V*, *E*) 有 n 个顶点，则图的邻接矩阵是一个二维数组 A.Edge[n][n]，定义为：

$$\mathbf{A}.Edge[i][j]=\begin{cases}\mathbf{1}, & \text{如果} <i,j>\in E \text{ 或者 } (i,j)\in E \\ \mathbf{0}, & \text{否则}\end{cases}$$

一、邻接矩阵

2.无向图的邻接矩阵表示法



顶点表：（ v1 v2 v3 v4 v5 ）

邻接矩阵：

$$A.Edge = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} v1 \\ v2 \\ v3 \\ v4 \\ v5 \end{matrix}$$

分析1：无向图的邻接矩阵是对称的；

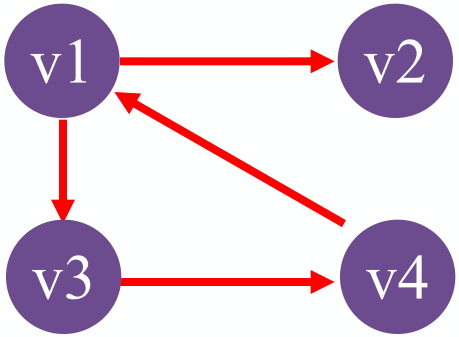
分析2：顶点i 的度 = 第 i 行 (列) 中1 的个数；

特别：完全图的邻接矩阵中，对角元素为0，其余1。

一、邻接矩阵

3.有向图的邻接矩阵表示法

A



顶点表： (v1 v2 v3 v4)

邻接矩阵：

$$A.Edge = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

v1
v2
v3
v4

注：在有向图的邻接矩阵中，

第i行含义：以结点v_i为尾的弧(即出度边）；

第i列含义：以结点v_i为头的弧(即入度边）。

分析1：有向图的邻接矩阵可能是不对称的。

分析2：顶点的出度=第i行元素之和

顶点的入度=第i列元素之和

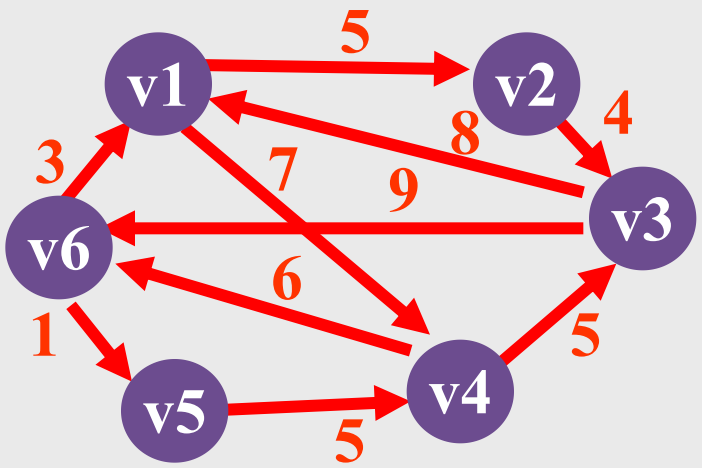
顶点的度=第i行元素之和+第i列元素之和

一、邻接矩阵

4.网（即有权图）的邻接矩阵表示法

定义为：$A.Edge[i][j]=\begin{cases} W_{ij} & <vi, vj> \text{或} (vi, vj) \in VR \\ \infty & \text{无边（弧）} \end{cases}$

N



顶点表：（ v1 v2 v3 v4 v5 v6 ）

邻接矩阵：

$$N.Edge = \begin{pmatrix} \infty & 5 & \infty & 7 & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & 5 & \infty & \infty & 6 \\ \infty & \infty & \infty & 5 & \infty & \infty \\ 3 & \infty & \infty & \infty & 1 & \infty \end{pmatrix}$$

一、邻接矩阵

5.邻接矩阵表示法的特点

优点：

容易实现图的操作，如：求某顶点的度、判断顶点之间是否有边、找顶点的邻接点等等。

缺点：

n个顶点需要$n*n$个单元存储边;空间效率为$O(n^2)$。 对稀疏图而言尤其浪费空间。

一、邻接矩阵

6.邻接矩阵的存储表示

```
//用两个数组分别存储顶点表和邻接矩阵
#define MaxInt 32767                    //表示极大值，即∞
#define MVNum 100                       //最大顶点数
typedef char VerTexType;                //假设顶点的数据类型为字符型
typedef int ArcType;                    //假设边的权值类型为整型
typedef struct{
  VerTexType vexs[MVNum];               //顶点表
  ArcType arcs[MVNum][MVNum];           //邻接矩阵
  int vexnum,arcnum;                    //图的当前点数和边数
}AMGraph;
```

一、邻接矩阵

7.采用邻接矩阵表示法创建无向网

【算法思想】

输入总顶点数和总边数。

依次输入点的信息存入顶点表中。

初始化邻接矩阵，使每个权值初始化为极大值。

构造邻接矩阵。

```
4    5
A B C D
A B 500
A C 200
A D 150
B C 400
C D 600
```

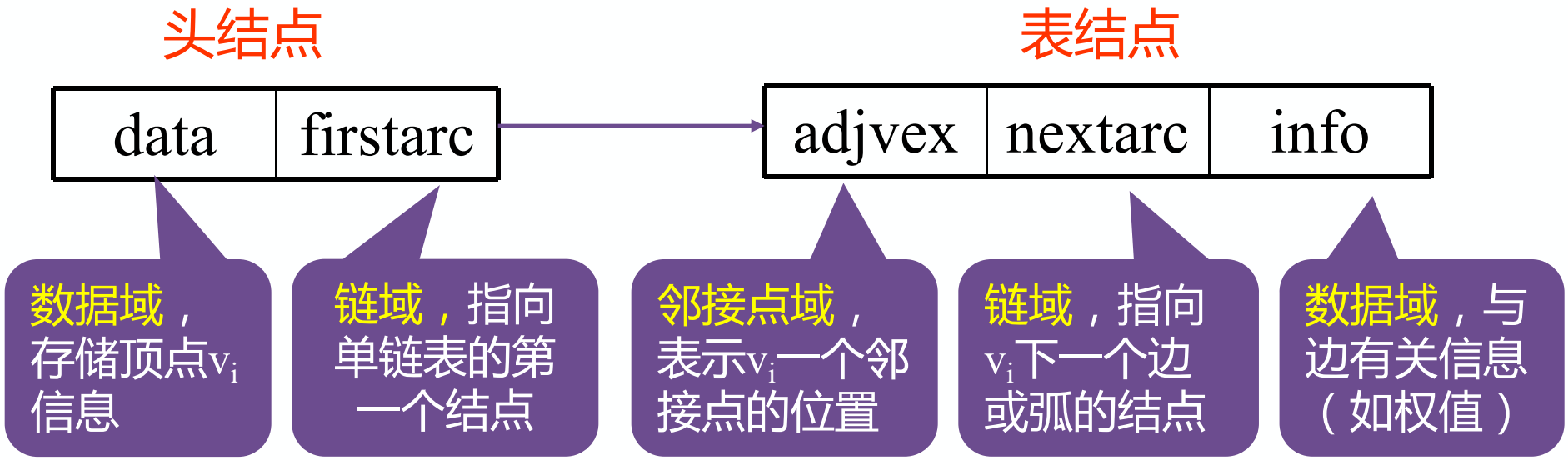
【算法描述】

```
Status CreateUDN(AMGraph &G){
  //采用邻接矩阵表示法，创建无向网G
  cin>>G.vexnum>>G.arcnum;          //输入总顶点数，总边数
  for(i = 0; i<G.vexnum; ++i)
    cin>>G.vexs[i];                //依次输入点的信息
  for(i = 0; i<G.vexnum;++i)       //初始化邻接矩阵，边的权值均置为极大值
    for(j = 0; j<G.vexnum;++j)
      G.arcs[i][j] = MaxInt;
  for(k = 0; k<G.arcnum;++k){              //构造邻接矩阵
    cin>>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中的位置
    G.arcs[i][j] = w;            //边<v1, v2>的权值置为w
    G.arcs[j][i] = G.arcs[i][j]; //置<v1, v2>的对称边<v2, v1>的权值为w
  }//for
  return OK;
}//CreateUDN
```

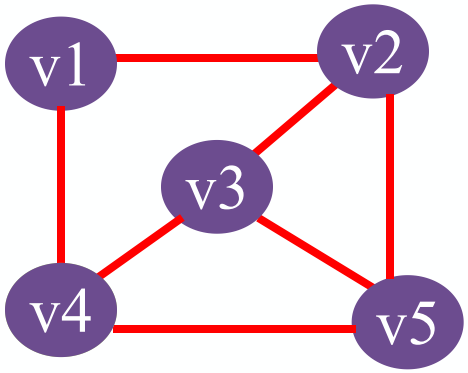
```
4   5
A B C D
A B 500
A C 200
A D 150
B C 400
C D 600
```

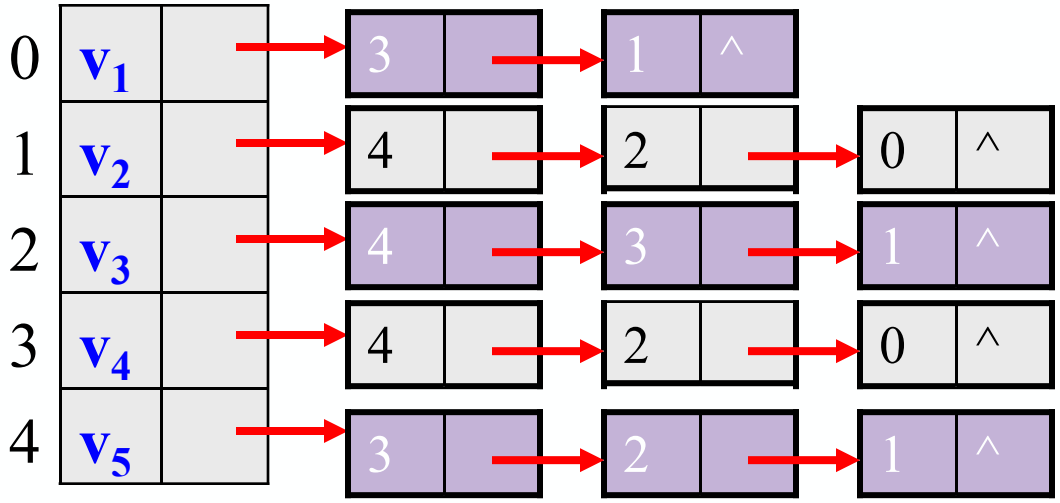
二、邻接表　　1.邻接表（链式）表示法

- 对每个顶点v_i建立一个单链表，把与v_i有关联的边的信息链接起来，每个结点设为3个域；



- 每个单链表有一个头结点（设为2个域），存v_i信息；
- 每个单链表的头结点另外用顺序存储结构存储。





注：邻接表不唯一，因各个边结点的链入顺序是任意的。

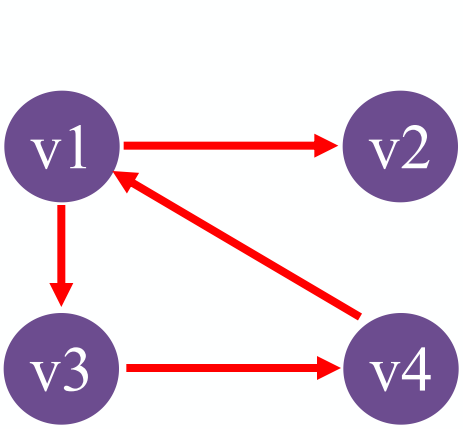
空间效率为O(n+2e)。

若是稀疏图($e<<n^2$)，比邻接矩阵表示法$O(n^2)$省空间。

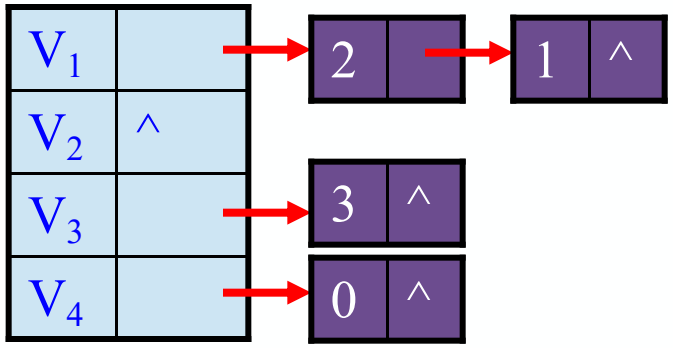
TD(Vi)=单链表中链接的结点个数

二、邻接表

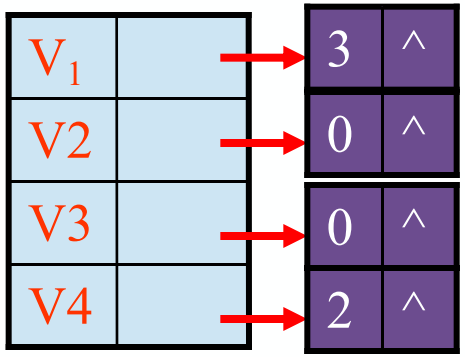
3.有向图的邻接表表示



邻接表(出边)



逆邻接表(入边)



空间效率为O(n+e)

出度　OD(Vi) = 单链出边表中链接的结点数

入度　ID(Vi) = 邻接点域为Vi的弧个数

度：　TD(Vi) = OD(Vi) + I D(Vi)

二、邻接表

4.邻接表的存储表示

```
#define MVNum 100                          //最大顶点数
typedef struct ArcNode{                    //边结点
    int adjvex;                            //该边所指向的顶点的位置
    struct ArcNode * nextarc;              //指向下一条边的指针
    OtherInfo info;                        //和边相关的信息
}ArcNode;
typedef struct VNode{
    VerTexType data;                       //顶点信息
    ArcNode * firstarc;                    //指向第一条依附该顶点的边的指针
}VNode, AdjList[MVNum];                    //AdjList表示邻接表类型
typedef struct{
    AdjList vertices;                      //邻接表
    int vexnum, arcnum;                    //图的当前顶点数和边数
}ALGraph;
```

【算法思想】

输入总顶点数和总边数。

依次输入点的信息存入顶点表中，使每个表头结点的指针域初始化为NULL。

创建邻接表。

【算法描述】

```
Status CreateUDG(ALGraph &G){
  //采用邻接表表示法，创建无向图G
  cin>>G.vexnum>>G.arcnum;       //输入总顶点数，总边数
  for(i = 0; i<G.vexnum; ++i){        //输入各点，构造表头结点表
    cin>> G.vertices[i].data;          //输入顶点值
    G.vertices[i].firstarc=NULL; //初始化表头结点的指针域为NULL
  }//for
```

6.采用邻接表表示法创建无向网

```
for(k = 0; k<G.arcnum;++k){                    //输入各边，构造邻接表
    cin>>v1>>v2;                               //输入一条边依附的两个顶点
    i = LocateVex(G, v1);  j = LocateVex(G, v2);
    p1=new ArcNode;                                  //生成一个新的边结点*p1
    p1->adjvex=j;                                    //邻接点序号为j
    p1->nextarc= G.vertices[i].firstarc;  G.vertices[i].firstarc=p1;
    //将新结点*p1插入顶点vi的边表头部
    p2=new ArcNode; //生成另一个对称的新的边结点*p2
    p2->adjvex=i;                                    //邻接点序号为i
    p2->nextarc= G.vertices[j].firstarc;  G.vertices[j].firstarc=p2;
    //将新结点*p2插入顶点vj的边表头部
  }//for
  return OK;
}//CreateUDG
```

7.邻接表表示法的特点

优点：

空间效率高，容易寻找顶点的邻接点；

缺点：

判断两顶点间是否有边或弧，需搜索两结点对应的单链表，没有邻接矩阵方便。

三、十字链表——用于有向图

结点表中的结点的表示：

data	firstin	firstout

data：结点的数据域，保存结点的数据值。

firstin: 结点的指针域，给出自该结点出发的的第一条边的边结点的地址。

firstout：结点的指针场，给出进入该结点的第一条边的 边结点的地址。

三、十字链表——用于有向图

边结点表中的结点的表示：

info	tailvex	headvex	hlink	tlink

info:边结点的数据域，保存边的权值等。

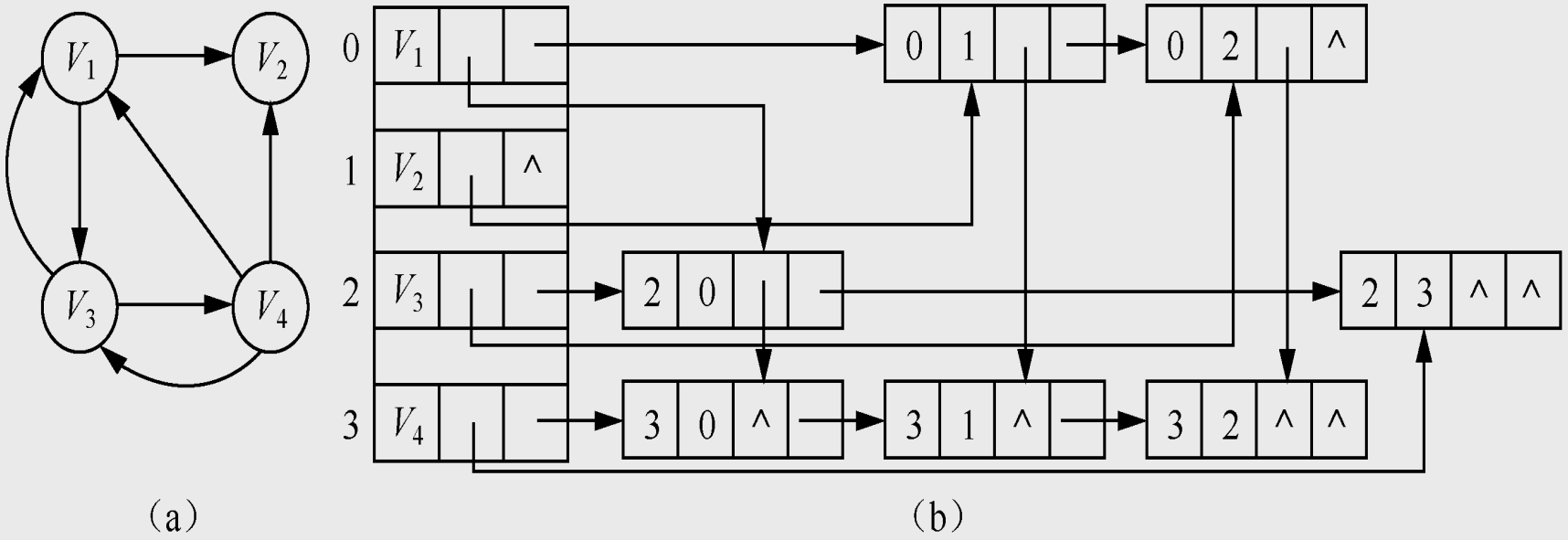
tailvex:　本条边的出发结点的地址。

headvex:本条边的终止结点的地址。

hlink:终止结点相同的边中的下一条边的地址。

tlink:出发结点相同的边 中的下一条边的地址。

三、十字链表——用于有向图



四、邻接多重表——用于无向图

边结点表中的结点的表示：

mark	ivex	ilink	jvex	jlink	info

ivex: 本条边依附的一个结点的地址。

ilink: 依附于该结点（地址由ivex给出）的边中的下一条边的的地址。

jvex: 本条边依附的另一个结点的地址。

jlink: 依附于该结点（地址由jvex给出）的边中的下一条边的的地址。

info: 边结点的数据域，保存边的权值等。

mark:边结点的标志域，用于标识该条边是否被访问过。

四、邻接多重表——用于无向图

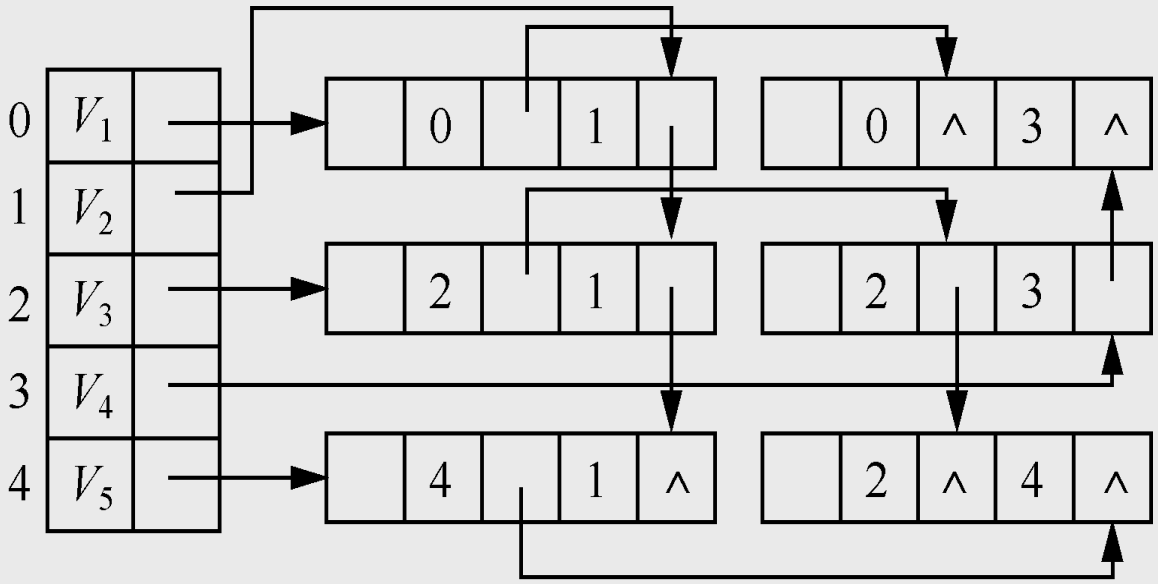
结点表中的结点的表示：

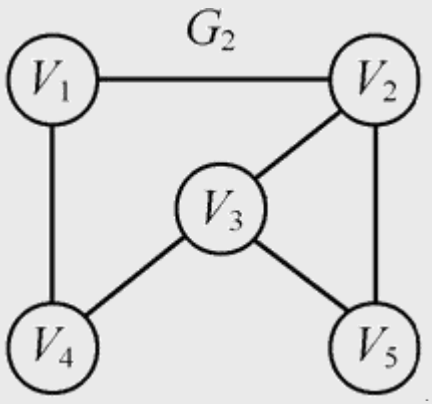
data	firstedge

data:结点的数据域，保存结点的数据值。

firstedge: 结点的指针域，给出自该结点出发的的第一条边的边结点的地址。

四、邻接多重表——用于无向图





小结

1. 用二维数组存储图顶点之间相邻关系的邻接矩阵。
2. 用单链表表示顶点之间邻接关系的邻接表。
3. 有向图的链式存储结构十字链表。
4. 无向图的链式存储结构邻接多重表。