

7.2 图的存储结构

7.2.1 数组表示方法

7.2.2 邻接表

7.2.3 十字链表

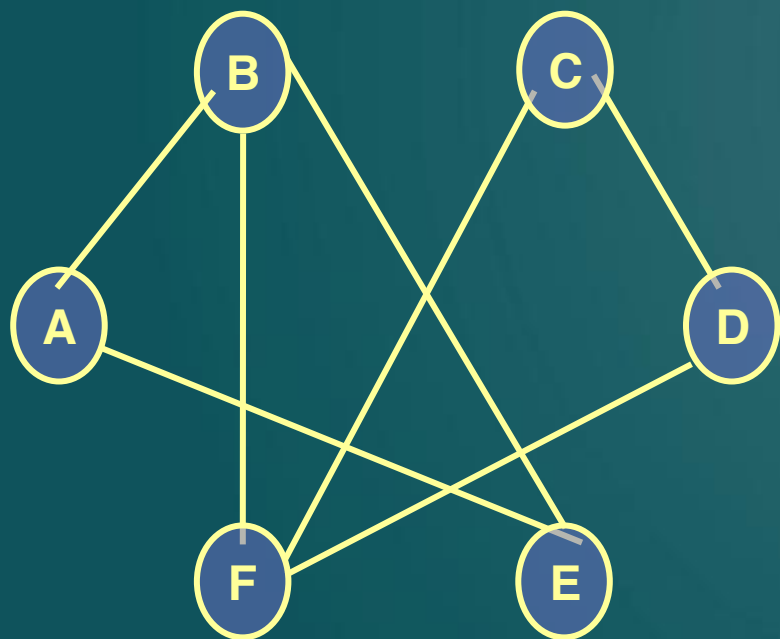
7.2.4 邻接多重表

7.2.1 数组表示方法

用两个数组分别存储数据元素（顶点）和数据元素之间的关系（边或弧）的信息。

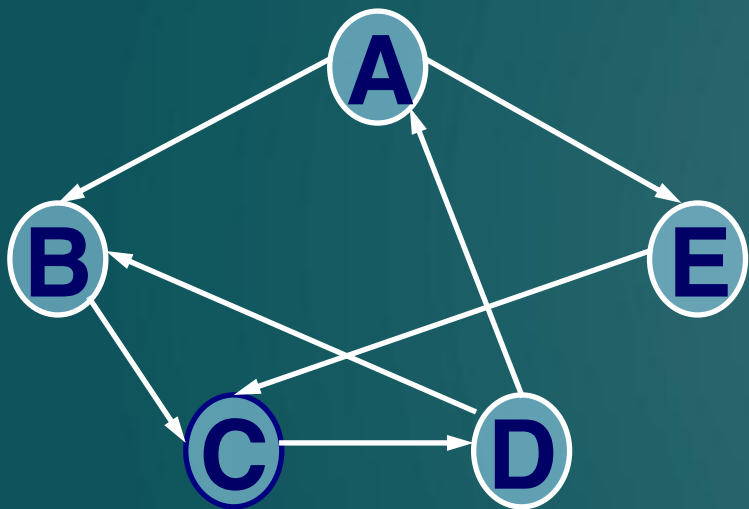
定义:邻接矩阵的元素为

$$A_{ij} = \begin{cases} 0 & (i,j) \text{ 或 } \langle i,j \rangle \notin VR \\ 1 & (i,j) \text{ 或 } \langle i,j \rangle \in VR \end{cases}$$



0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
1	1	0	0	0	0
0	1	1	1	0	0

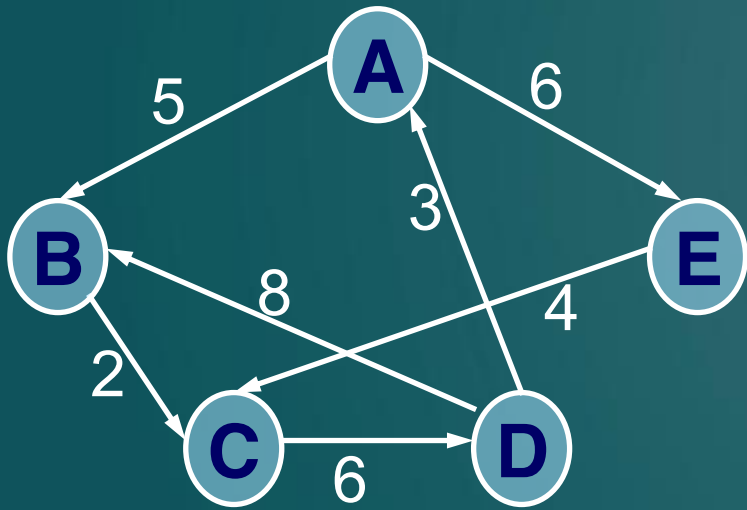
无向图的邻接矩阵一定是对称矩阵，而有向图的邻接矩阵则不一定为非对称矩阵。



0	1	0	0	1
0	0	1	0	0
0	0	0	1	0
1	1	0	0	0
0	0	1	0	0

网的邻接矩阵可定义为：

$$A_{ij} = \begin{cases} W_{ij} & \text{若 } (i,j) \text{ 或 } \langle i,j \rangle \in VR \\ \infty & \text{否则} \end{cases}$$



0	5	∞	∞	6
∞	0	2	∞	∞
∞	∞	0	6	∞
3	8	∞	0	∞
∞	∞	4	∞	0

//-----图的数组（邻接矩阵）存储表示-----

```
#define INFINITY  INF_MAX  //最大值 $\infty$ 
```

```
#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表示相邻否；对带权图，则为权值类型。
```

```
    InfoType *info; // 该弧相关信息的指针
```

```
} ArcCell ,  AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
```

```
typedef struct { // 图的定义
    VertexType vexs[MAX_VERTEX_NUM]; // 顶点向量
    AdjMatrix arcs; // 邻接矩阵
    int vexnum, arcnum; // 图的当前顶点数和弧数
    GraphKind kind; // 图的种类标志
} MGraph;
```

```
Status CreateGraph( MGraph &G ) {  
    //采用数组（邻接矩阵）表示法，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
```

算法 7.1

```

Status CreateUND( MGraph &G ) {
    //采用数组（邻接矩阵）表示法，构造无向网G.
    scanf(&G.vexnum,&G.arcnum,&IncInfo);
    // IncInfo为0则各弧不含其他信息
    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};

    //{adj,info}
    for(k=0; k<G.vexnum; ++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) Input(*G.arcs[i][j].info); //若弧含有相关信息，则输入
        G.arcs[j][i]=G.arcs[i][j];    //置<v1,v2>的对称弧
        <v2,v1>
    }
    return OK;
}
} // CreateUDN

```

算法 7.2

7.2.2 邻接表

弧的结点结构

adjvex	nextarc	info
--------	---------	------

```
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; // 该弧相关信息的指针  
} ArcNode;
```

顶点的结点结构


data	firstarc
------	----------

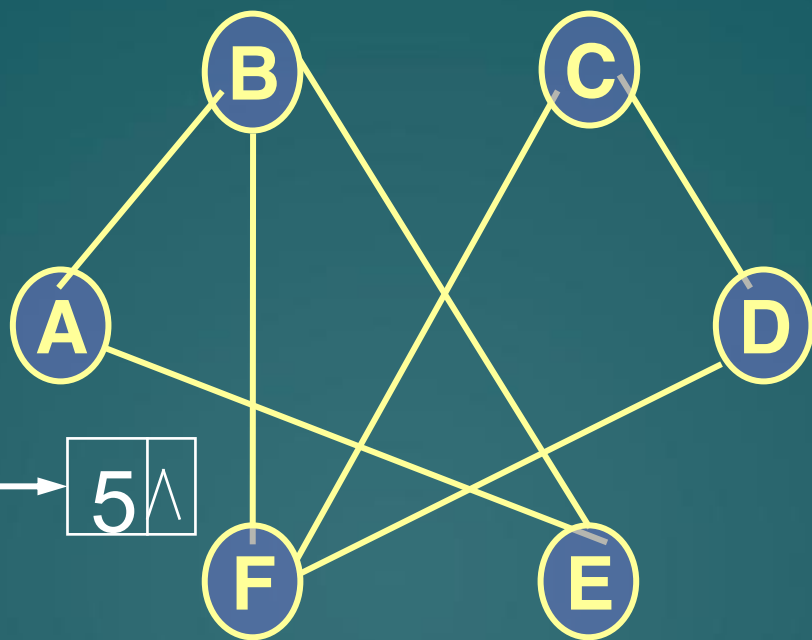
```
typedef struct VNode {  
    VertexType data; // 顶点信息  
    ArcNode *firstarc;  
                // 指向第一条依附该顶点的弧  
} VNode, AdjList[MAX_VERTEX_NUM];
```

图的结构定义

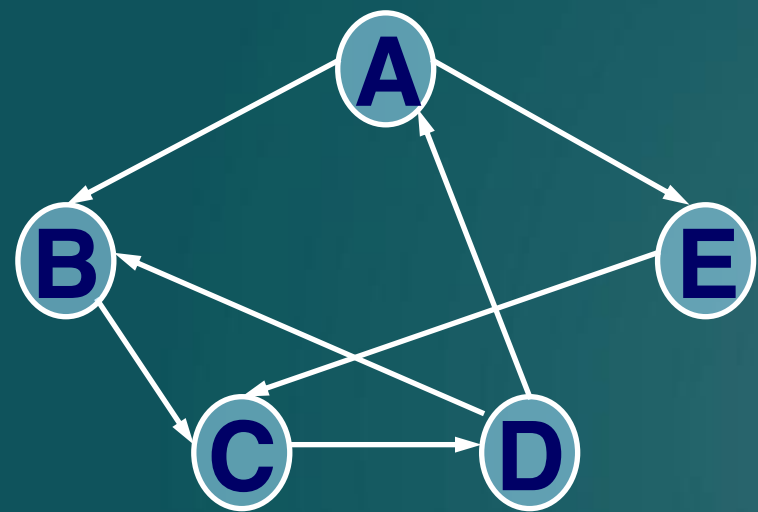
```
typedef struct {  
    AdjList vertices;  
  
    int vexnum, arcnum; //图的当前顶点数和弧数  
  
    int kind;          // 图的种类标志  
  
} ALGraph;
```

无向图的邻接表

0	A	→	1	→	4	∧		
1	B	→	0	→	4	→	5	∧
2	C	→	3	→	5	∧		
3	D	→	2	→	5	∧		
4	E	→	0	→	1	∧		
5	F	→	1	→	2	→	3	∧



有向图的邻接表



0

1

2

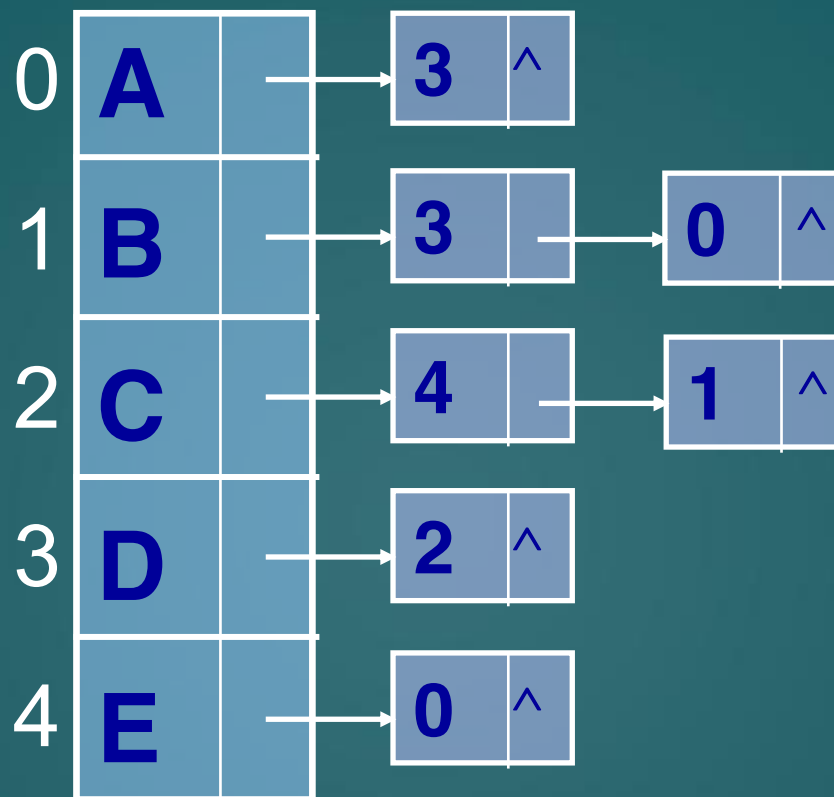
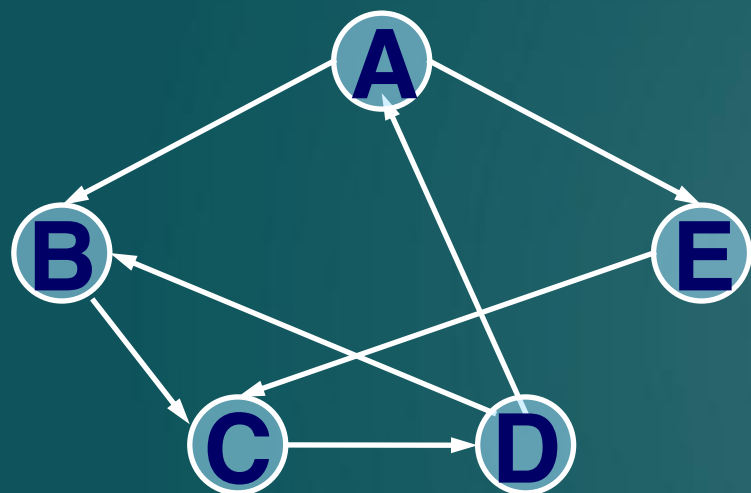
3

4

A	→	1	→	4	^
B	→	2	^		
C	→	3	^		
D	→	0	→	1	^
E	→	2	^		

可见，在有向图的邻接表中不易找到指向该顶点的弧。

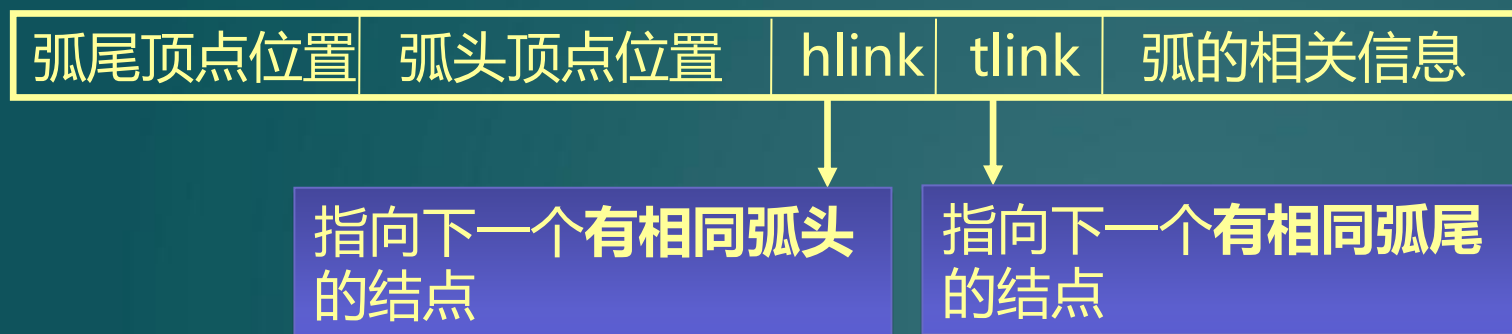
有向图的逆邻接表



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。

7.2.3 十字链表

弧的节点结构



typedef struct ArcBox { // 弧的结构表示

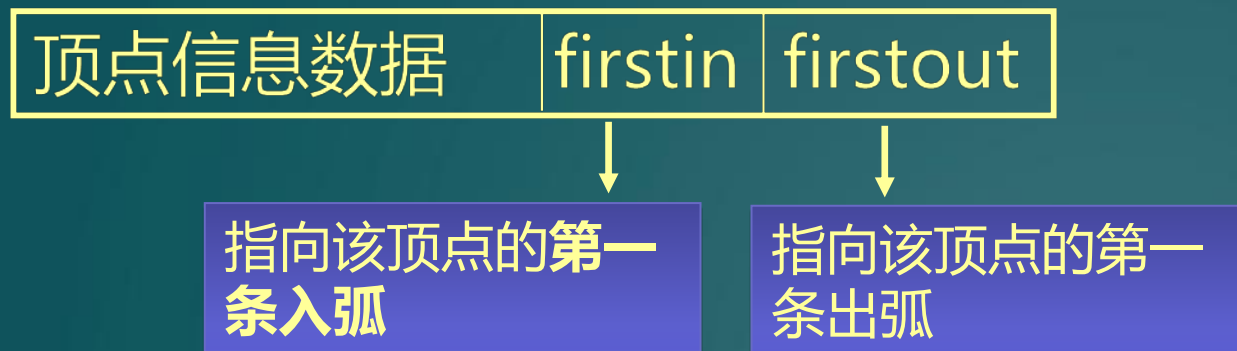
int tailvex, headvex; //该弧尾和头顶点的位置

struct ArcBox *hlink, *tlink; //含义见上面框内

InfoType *info; //该弧相关信息的指针

} VexNode;

顶点的节点结构



```
typedef struct VexNode { // 顶点的结构表示
    VertexType data;

    ArcBox *firstin, *firstout; //分别指向该顶点
                                //第一条入弧和出弧
} VexNode;
```


有向图的结构表示（十字链表）

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

例：有向图如右
图所示：

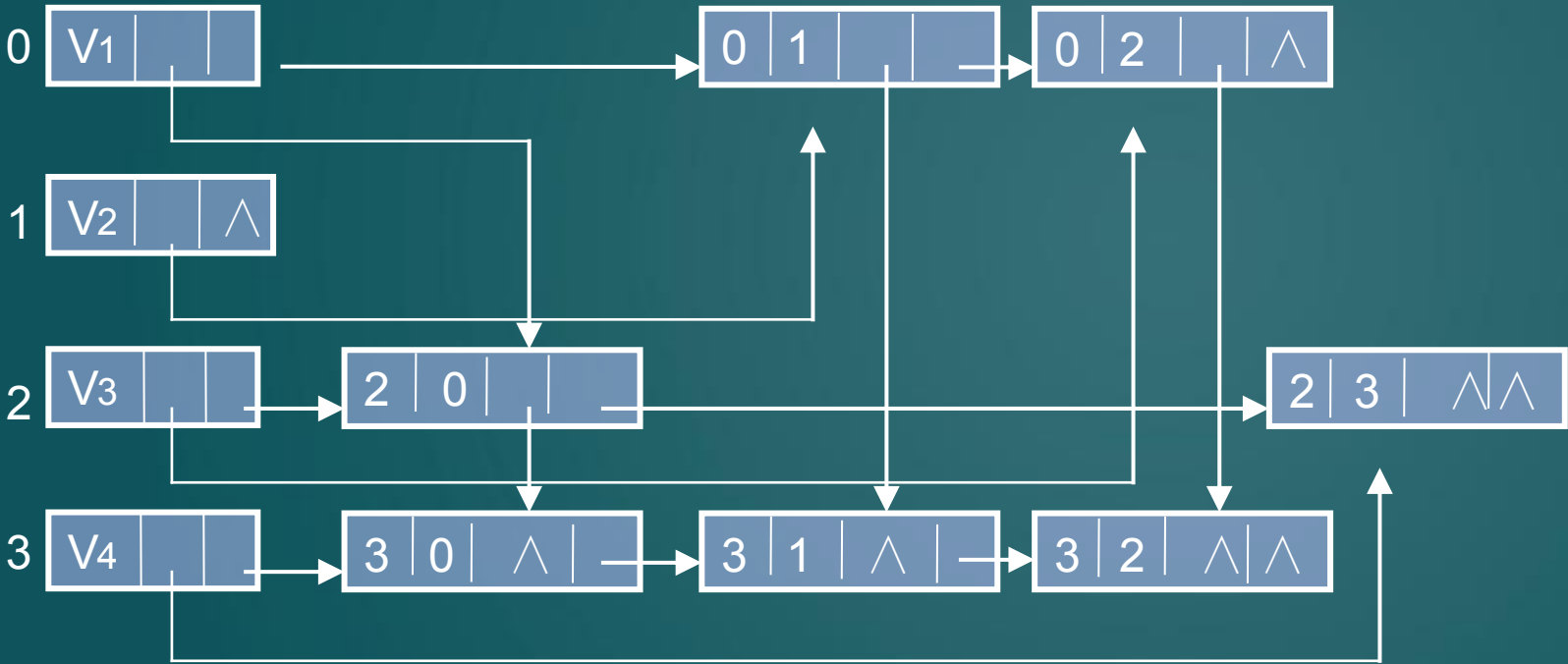
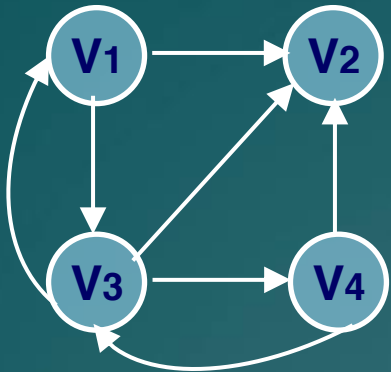


图7.11 有向图的十字链表

7.2.4 邻接多重表

邻接多重表是无向图的另一种链式存储结构。

边的结点结构

mark	ivex	ilink	lvex	jlink	info
------	------	-------	------	-------	------

其中： Mark为标志域；ivex和lvex为该边依附的两个顶点在图中的位置；
ilink指向下一条依附于顶点ivex 的边；jlink指向下一条依附于顶点
lvex 的边；info为指向和边相关的各种信息的指针域。

其边的结构表示如下：

```
typedef struct Ebox {  
    VisitIf    mark;    // 访问标记  
  
    int    ivex, jvex;    //该边依附的两个顶点的位置  
    struct EBox *ilink, *jlink; //分别指向依附这两个  
                                //顶点的下一条边  
  
    InfoType    *info;    // 该边信息指针  
} EBox;
```

顶点的结点结构



其中： data域存储和该顶点相关的信息；firstedge指向第一条依附于该顶点的边。

```
typedef struct VexBox {  
    VertexType data;  
    EBox *firstedge; // 指向第一条依附该顶点的边  
} VexBox;
```

无向图的邻接多重表结构表示

```
typedef struct { // 邻接多重表
    VexBox adjmulist[MAX_VERTEX_NUM];

    int vexnum, edgenum; //无向图的当前顶点数
                        //和边数
} AMLGraph;
```

例：无向图如右
图所示：

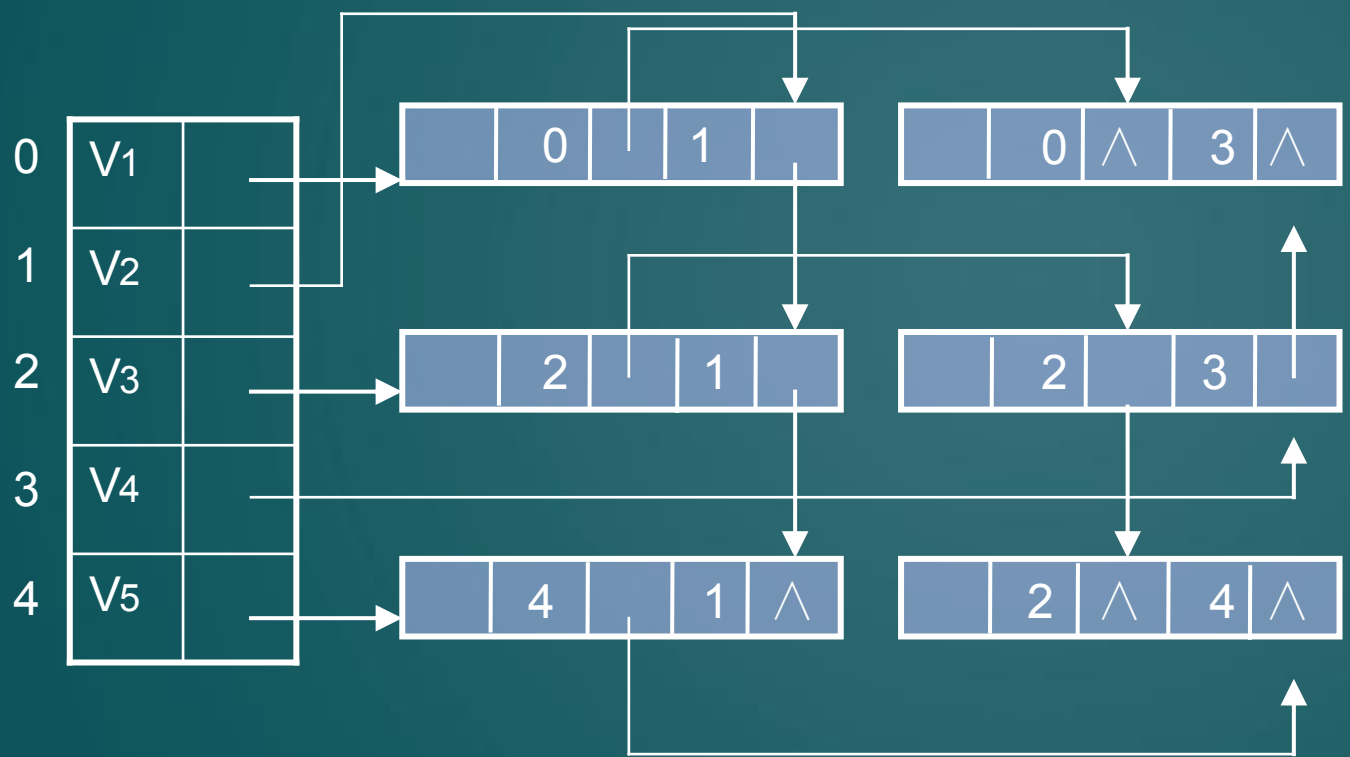
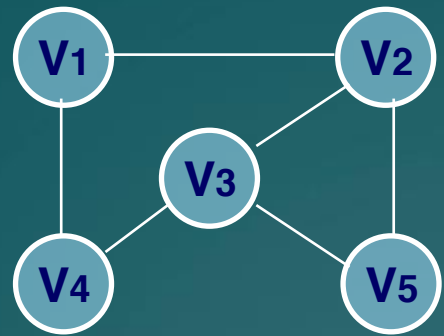


图7.12 无向图G2的邻接多重表