教学内容---第七章

DS/SS

1. 绪论
2. 线性表
3. 栈、队列和串
4. 数组
5. 广义表
6. 树和二叉树
7. 图
8. 动态存储管理
9. 查找
10. 内部排序
11. 外部排序
12. 文件

7.1图的逻辑结构（续）

基本概念和术语

- 顶点(Vertex)：图中数据元素。
- 弧(Arc)： v,w属于V , <v,w> 属于VR，则<v,w> 是从v到w的一条弧。v为弧尾(Tail)或初始点(Initial node), w为弧头(Head)或终端点(Terminal node)
- 有向图(Digraph)：由顶点与弧构成的图。
- 边(Edge)： v,w属于V ,有 <v,w> 属于VR，则必有<w,v> 属于VR,称<v,w>以及<w,v>是一条边，记作(v,w)。
- 无向图(Undigraph)：由顶点与边构成的图。
- 完全图 (Completed graph)：设图的顶点数为n，有 n(n-1)/2条边的无向图为完全图。
- 有向完全图：设图的顶点数为n，有n(n-1)条弧的有向图为有向完全图。

7.1图的逻辑结构（续）

DS/SS

基本概念和术语

- 稀疏图(Sparse graph)：有很少条边或弧($e<n*\log n$)的图。反之，称为稠密图(Dense graph)。
- 权(Weight)：与图的边或弧相关的数叫做权。
- 网(Network)：带权的图称为网。
- 子图(Subgraph)：设有两个图G=(V,{E})和G'=(V',{E'})，如果V'包含在V中且E'包含在E中，则称G'为G的子图。
- 邻接点(Adjacent)：对于无向图G=(V,{E})，如果边(v,v')属于E，则称顶点v与v'互为邻接点；或边(v,v')依附(Incident)于顶点v和v';或者边(v,v')和顶点v和v'相关联。对于有向图G=(V,{A})，如果弧<v,v'>属于A，则称顶点v邻接到顶点v'；顶点v'邻接自顶点v；弧(v,v')和顶点v和v'相关联。
- (顶点的)度(Degree)：和该顶点v相关联的顶点的数目。记为：TD(v)。

7.1图的逻辑结构（续）

DS/SS

基本概念和术语

- （顶点的）入度(InDegree)：以顶点v为头的弧的数目称为顶点v的入度。记为：ID(v)。
- （顶点的）出度(OutDegree)：以顶点v为尾的弧的数目称为顶点v的出度。记为：OD(v)。
- （顶点之间）路径(Path)：对无向图G=(V,{E})，从顶点v到顶点v'的路径是一个顶点序列($v=v_{i,0}, v_{i,1}, \ldots, v_{i,m}=v'$),其中（ $v_{i,j-1}, v_{i,j}$ ）属于E，$1<=j<=m$；对有向图G，则路径也是有向的，顶点序列应该满足<$v_{i,j-1}, v_{i,j}$>属于A。
- （顶点之间）路径长度：路径上边或弧的数目。
- 回路或环(Cycle)：第一个顶点与最后一个顶点相同的路径。
- 简单路径：序列中不重复出现的路径。
- 简单回路（简单环）：除第一个和最后一个顶点外，其余顶点不重复出现的回路。

7.1图的逻辑结构（续）

DS/SS

基本概念和术语

- （顶点之间）连通：无向图中，如果顶点v到顶点v'有路径，则称v和v'是连通的。
- 连通图(Connected Graph)：如果对于无向图G中任意两个顶点v_i、v_j属于V，v_i和v_j都是连通的，则称G是连通图。
- 连通分量(Connected Component)：无向图中的极大连通子图。
- 强连通图：如果对于有向图G中任意两个顶点v_i、v_j属于V，v_i!=v_j,从v_i到v_j和从v_j到v_i都存在路径，则称G是强连通图。
- 强连通分量：有向图中的极大强连通子图。
- （连通图的）生成树：一个极小连通子图，它含有图中全部顶点，但只有足以构成一棵树的n-1条边。（对无向图）
- （有向图的）生成森林：由若干棵有向树（恰有一个顶点入度为0，其余顶点入度为1的有向图）组成，含有图中全部顶点，但只有足以构成若干棵不相交的有向树的弧。

7.1图的逻辑结构

图的抽象数据类型

ADT Graph {

数据对象：V={具有相同性质的数据元素}

数据关系：R:

R={VR}

VR = {<v,w> | v,w属于V且P(v,w), <v,w> 是从v到w的一条弧，谓词P(v,w)定义了弧<v,w> 的意义或信息 }

基本操作:

CreateGraph(&G,V,VR); DFSTraverseGraph(G,v,Visit());

…; BFSTraverseGraph(G,v,Visit())

}ADT Graph

7.2图的存储结构

图的多重链表


```
//----------用结构指针描述------------
typedef  struct  MultiLNode{
        ElemType          data      //数据域
        struct MultiLNode          *link1    //第1个指针域
        struct MultiLNode          *link2    //第2个指针域
        …….
        struct MultiLNode          *linkk    //第k个指针域
} MultiLNode, *MultiLinkGraph
```

7.2树的存储结构（续）

数组表示法

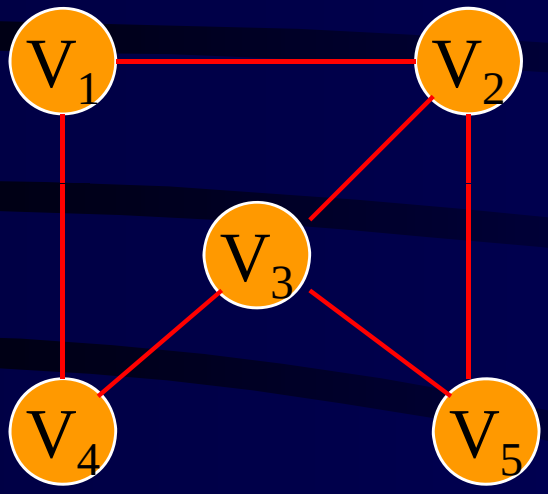
用两个数组分别存储数据元素（顶点）的信息和数据元素之间关系（边或弧）的信息。

```
//----------图的数组（邻接矩阵）存储表示-----------
#define MAX_VERTEX_NUM      20 //最大顶点数
#define enum {DG,DN,AG,AN} GraphKind;  //{有向图,有向网,无向图,无向网}
typedef  struct  ArcCell{
        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 {
        VertexType        vexs [MAX_VERTEX_NUM];  //顶点向量
        AdjMatrix         arcs;     //邻接矩阵
        GraphKind         kind;     //图的种类标志    }MGraph;
```

7.2树的存储结构（续）

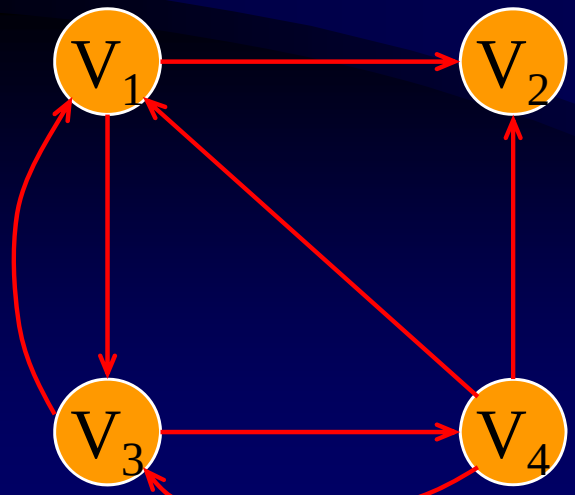
数组表示法

G1



$$G1.arcs = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

G2



$$G2.arcs = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

7.2树的存储结构（续）

DS/SS

数组表示法

邻接矩阵的优点：

- 容易判定任意两个顶点之间是否有边（或弧）相连；
- 容易求得各个顶点的度；
  - *对无向图，$TD(v_i) = A[i][0]+A[i][1]+\ldots+A[i][n-1]$
  - *对有向图，$OD(v_i) = A[i][0]+A[i][1]+\ldots+A[i][n-1]$

    $ID(v_i) = A[0][i]+A[1][i]+\ldots+A[n-1][i]$
- 容易求得当前顶点的第一个邻接点、下一个邻接点。

7.2图的存储结构（续）

邻接表---图的链式存储

基本思路：

对图中每个顶点建立一个单链表，第i个单链表中的结点表示依附于顶点v_i的边（对有向图是以顶点v_i为尾的弧）。每个**表结点**由三个域组成：邻接点域(adjvex)指示与顶点v_i邻接的点在图中的位置；链域(nextarc)指示下一条边或弧的结点；数据域(info)存储和边或弧相关的信息。**头结点**由两个域组成：链域(firstarc)指向链表中第一个结点；数据域(data)存储顶点v_i信息。

头结点以顺序结构形式存取，以便随机访问任一顶点的链表。

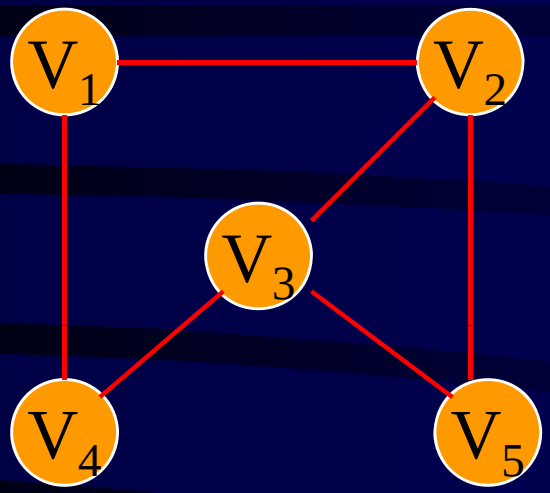
表结点　　头结点

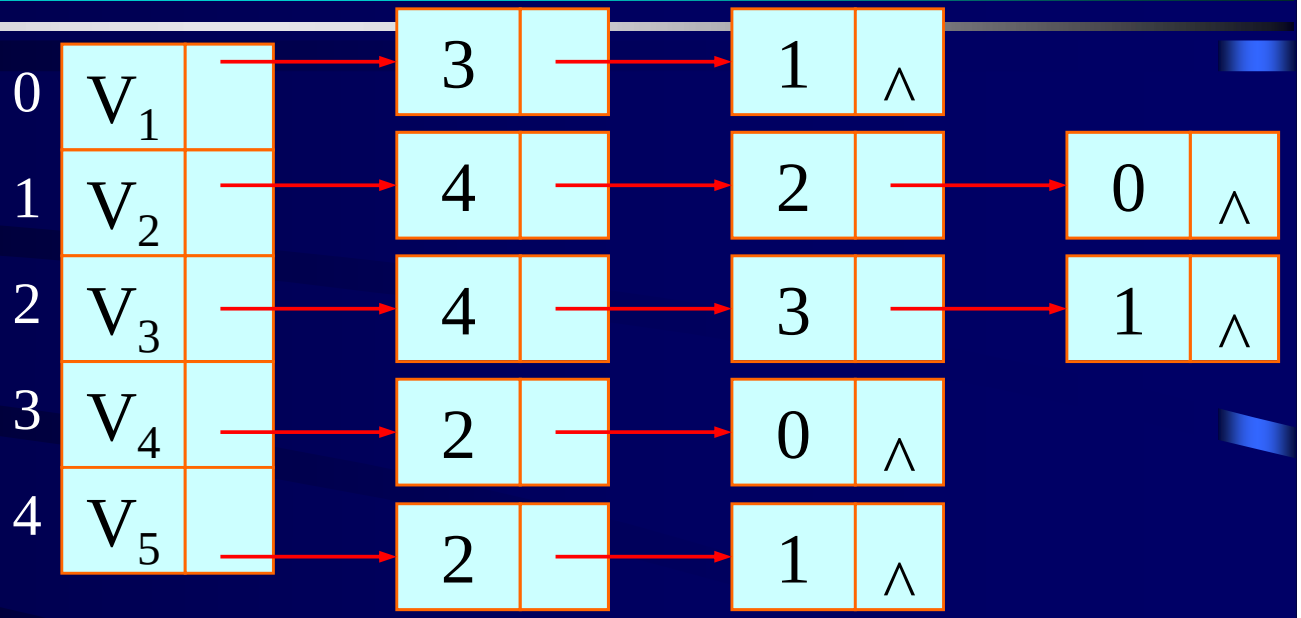
7.2图的存储结构（续）

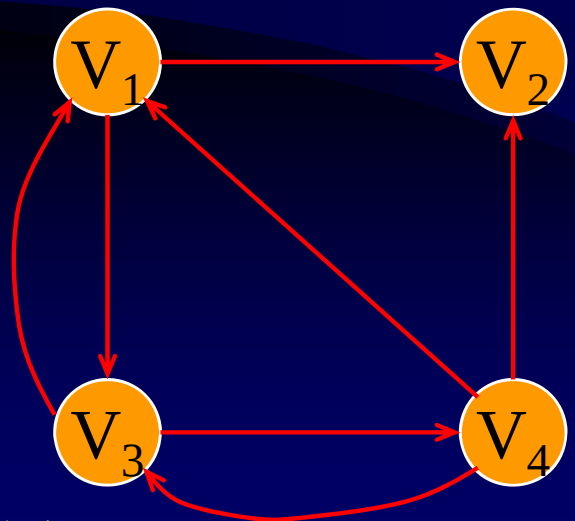
邻接表---图的链式存储

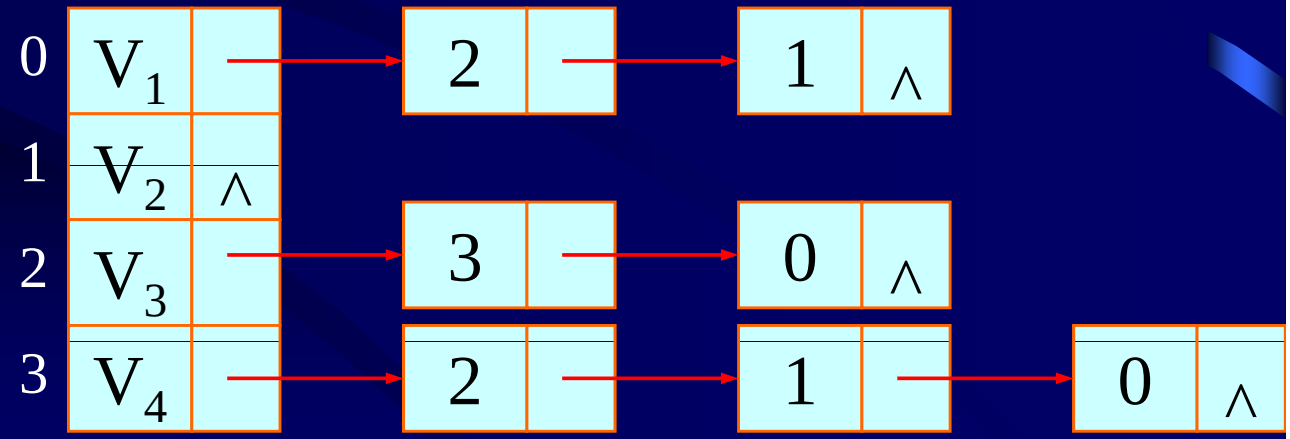
```
#define MAX_VERTEX_NUM      20 //最大顶点数
typedef  struct  ArcNode{
        int         adjvex;             //该弧所指向的顶点的位置
        struct  ArcNode   *nextarc;  //指向下一条弧的指针
        InfoType          *info;    //该弧相关信息的指针;  }ArcNode;
typedef  struct  VNode{
        VertexType        data;      //顶点信息
         ArcNode          *firstarc; //指向第一条依附该顶点的弧的指针
}VNode,  AdjList[MAX_VERTEX_NUM];
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;
```

7.2图的存储结构（续）


G1
V1
V2
V3
V4
V5


0
V1
3
1
∧
1
V2
4
2
0
∧
2
V3
4
3
1
∧
3
V4
2
0
∧
4
V5
2
1
∧


G2
V1
V2
V3
V4


0
V1
2
1
∧
1
V2
∧
2
V3
3
0
∧
3
V4
2
1
0
∧

7.2图的存储结构（续）

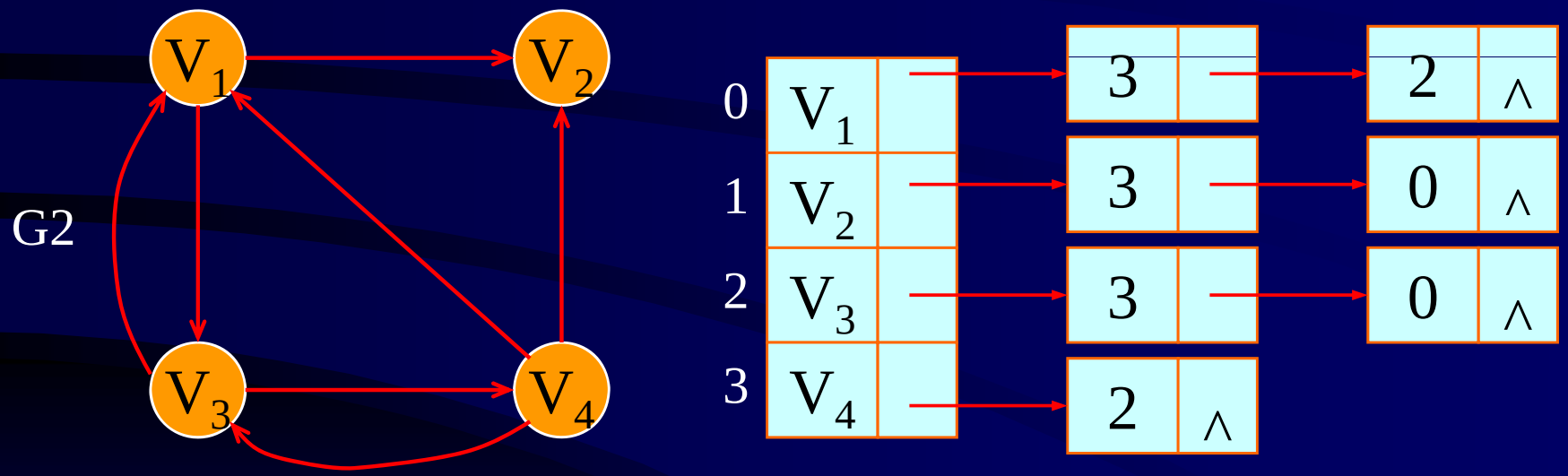
邻接表---图的链式存储

邻接表的优点：

- 边（或弧）稀疏时，节省空间；
- 和边（或弧）相关的信息较多时，节省空间；
- 容易求得当前顶点的第一个邻接点、下一个邻接点。

对有向图，也可建立逆向邻接表，即对每个顶点建立一个链接以该顶点为头的弧的表。

7.2图的存储结构（续）



7.2图的存储结构（续）

（有向图）十字链表

基本思路：

将有向图的邻接表和逆邻接表结合在一起得到的链表。在十字链表中，对应于有向图每一条弧的结点称为弧结点；对应于每个顶点的结点为顶点结点。每个弧结点由五个域组成：尾域(tailvex)和头域(headvex)分别指示弧尾和弧头两个顶点在图中的位置；链域hlink 指向弧头相同的下一条弧；链域tlink 指向弧尾相同的下一条弧；数据域(info)存储和弧相关的信息。顶点结点由三个域组成：链域firstin与firstout分别指向以该顶点为弧头或弧尾的第一个弧结点，指向链表中第一个结点；数据域(data)存储顶点v_i信息。

顶点结点以顺序结构形式存取，以便随机访问任一顶点的链表。

将有向图邻接矩阵看作稀疏矩阵时，则可将有向图十字链表看作邻接矩阵十字链表存储。

tailvex	headvex	hlink	tlink	info

弧结点

data	firstin	firstout

顶点结点

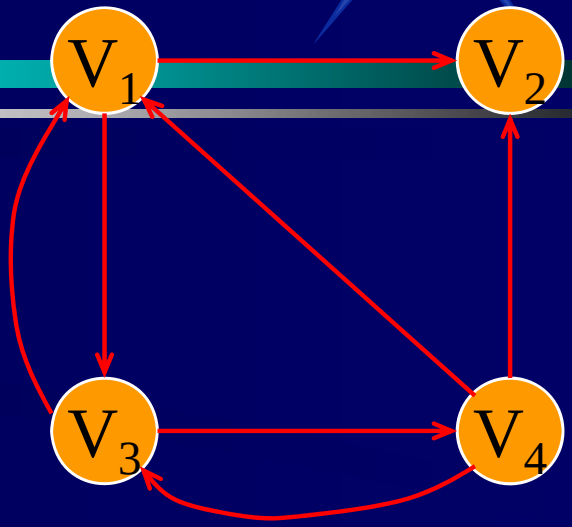
7.2图的存储结构（续）

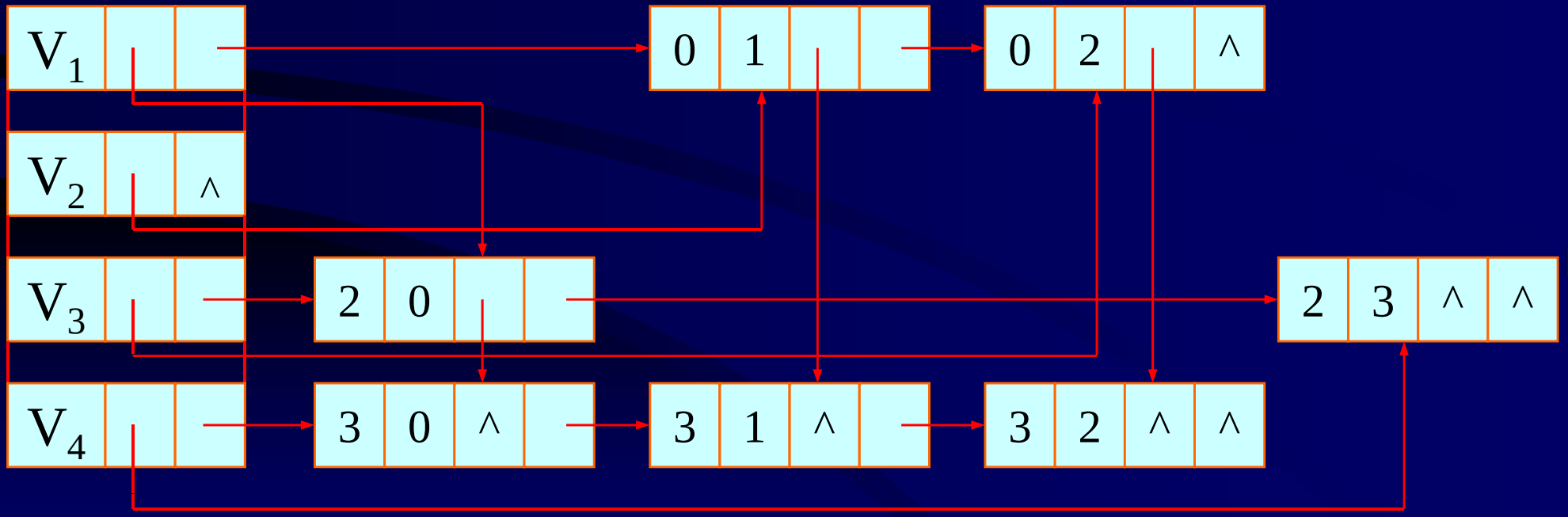
（有向图）十字链表

```
#define MAX_VERTEX_NUM      20 //最大顶点数
typedef  struct  ArcBox{
            int         tailvex, headvex;          //该弧的尾和头顶点的位置
            struct  ArcBox     *hlink,  *tlink
            InfoType           *info;    //该弧相关信息的指针;   }ArcBox;
typedef  struct  VexNode{
            VertexType         data;      //顶点信息
             ArcBox            *firstin,  firstout; //分别指向第一条入弧和出弧
}VexNode;
typedef  struct {
            VexNode             xList[MAX_VERTEX_NUM];  //表头向量
            int         vexnum,arcnum;  //图的当前顶点数和弧数
}OLGraph;
```

7.2图的存储结构（续）

（有向图）十字链表





7.2图的存储结构（续）

（无向图）邻接多重表

基本思路：

无向图的邻接表中任何一条边在两个链表中，不便于图的某些操作。在邻接多重表中，对应于无向图每一条边的结点称为边结点；对应于每个顶点的结点为顶点结点。每个**边结点**由六个域组成：mark是标志域，标记该条边是否被搜索过；ivex和jvex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jvex的边；数据域(info)存储和边相关的信息。**顶点结点**由两个域组成：数据域(data)存储顶点v_i信息；firstedge指向第一条依附于该顶点的边。

顶点结点以顺序结构形式存取，以便随机访问任一顶点的链表。

所需存储量与邻接表相同。

mark	ivex	ilink	jvex	jlink	info

边结点

data	firstedge

顶点结点

7.2图的存储结构（续）

（无向图）邻接多重表

```
#define MAX_VERTEX_NUM      20 //最大顶点数
Typedef enum {unvisited,visited} VisitIf;
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
          InfoType           *info;    //该弧相关信息的指针;  }EBox;
typedef  struct  VexBox{
          VertexType         data;     //顶点信息
           EBox              *firstedge;   }VexBox;
typedef  struct {
          VexBox   adjmuList[MAX_VERTEX_NUM];  //表头向量
          int        vexnum,arcnum;  //图的当前顶点数和弧数 }AMLGraph;
```

7.2图的存储结构（续）

（无向图）邻接多重表