第5章 数组和广义表

5.1 数组的定义及其操作

5.2 数组的存储结构

5.3 矩阵的压缩存储

5.4 广义表的定义及其操作

5.5 广义表的存储结构

5.6 小结

5.1 数组的定义及其操作

1. 什么是数组（Array）？

n（n >=1）维同类型元素组成的序列，且连续存储

二维数组：

$$A^{(2)} = A[m][n] = \begin{pmatrix} a_{00} & a_{01} \cdots\cdots & a_{0j} & \cdots\cdots & a_{0,n-1} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ a_{i0} & a_{i1} \cdots\cdots & a_{ij} & \cdots\cdots & a_{i,n-1} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ a_{m-1,0} & a_{m-1,1} \cdots & a_{m-1,j} & \cdots & a_{m-1,n-1} \end{pmatrix}$$

m、**n**为行列数，a_{ij}为第**i**行、第**j**列的元素，**0≤i≤m-1，0≤j≤n-1**；
元素个数为**m×n**。

5.1 数组的定义及其操作

二维数组的形式化描述：

$A^{(2)} = (D, R)$

其中：$D=\{a_{ij} | a_{ij} \in datatype,\ 0 \le i \le m-1,\ 0 \le j \le n-1\};$

$R=\{Row, Col\}$

行关系：$Row=\{<a_{ij}, a_{i\ j+1}> | a_{ij}, a_{i\ j+1} \in D,\ 0 \le i \le m-1,\ 0 \le j \le n-2\}$

列关系：$Col=\{<a_{ij}, a_{i+1\ j}> | a_{ij}, a_{i+1\ j} \in D,\ 0 \le i \le m-2,\ 0 \le j \le n-1\}$

二维数组可以看作其元素是一维数组的一维数组

n维数组可以看作其元素是**n-1**维数组的一维数组

5.1 数组的定义及其操作

2. 数组的抽象数据类型表示

ADT Array {

数据元素集： $D=\{a_{j1j2\ldots jn}|a_{j1j2\ldots jn}\in datatype，ji=0，\ldots，b_i-1，i=1，2，\ldots，n\}$ n(n>0)为数组维数，b_i是第i维长度， ji是数组元素第i维下标。

数据关系集：$R=\{R_1，R_2，\ldots， R_n\}$

其中：$R_i=\{<a_{j1\ldots ji\ldots jn}, a_{j1\ldots ji+1\ldots jn}>|0\le jk\le b_k-1，1\le k\le n$且$k\ne i，0\le ji\le b_i-2，a_{j1\ldots ji\ldots jn}，a_{j1\ldots ji+1\ldots jn}\in D，i=1，2，\ldots，n\}$

基本操作集：P

ArrayInit(&A, n, d_1, d_2, ..., d_n)

ArrayDestroy(&A)

ArrayGet(A, i1, ..., in, &e)

ArrayAssign(&A, i_1, ..., i_n, e)

}ADT Array；

数组的基本操作一般不包括插入和删除

5.2 数组的存储结构

1. 数组的静态存储

存储空间是在程序执行前分配的，且大小固定

2种不同的存储方式，如二维数组：

- ✓ 行主次序（row major）：按行优先，C语言如此
- ✓ 列主次序（column major）：按列优先

5.2 数组的存储结构

数组元素地址的计算：

以C语言为例

设数组的起始地址为b，每个元素占L个字节，存储空间以字节编址，元素a的地址用Loc(a)表示。

1）一维数组a[n]

Loc(a[i]) = b + i*L;

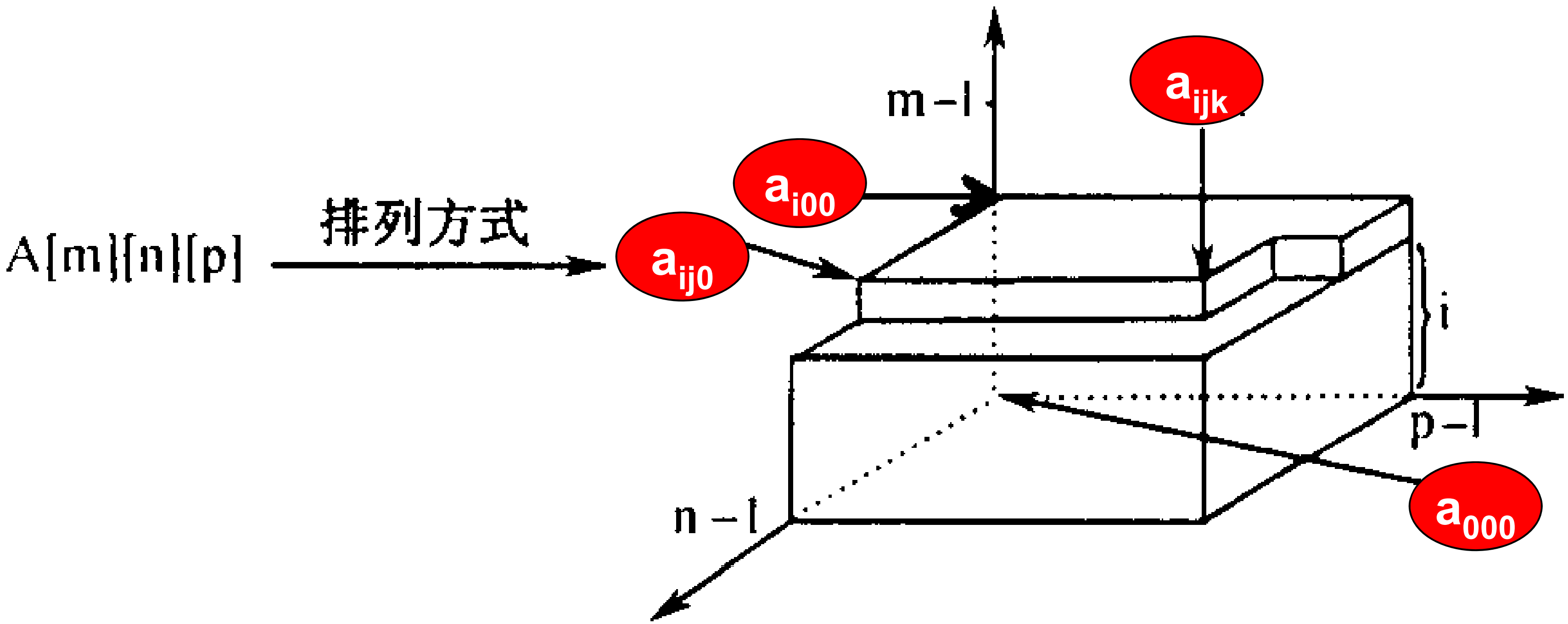
2）二维数组a[m][n]

Loc(a[i][j]) = b+(a[i][j]前的元素个数)*L

= b + (i*n + j)*L

5.2 数组的存储结构

3）三维数组a[m][n][p]



Loc(a[i][j][k]) = b + (i*n*p + j*p + k)*L

5.2 数组的存储结构

4）n维数组$a[u_1][u_2]\ldots[u_n]$

$$
\begin{aligned}
Loc(a[i_1][i_2]\ldots[i_n]) = b + (& i_1*u_2*u_3*\ldots*u_n \\
&+ i_2*u_3*u_4*\ldots*u_n \\
&+ \ldots \\
&+ i_{n-1}*u_n + i_n)*L
\end{aligned}
$$

$$
=b+\left(\sum_{j=1}^{n-1} i_j \prod_{k=j+1}^{n} u_k + i_n\right)*L
$$

5.2 数组的存储结构

2. 数组的动态存储

存储空间是在程序执行时动态分配的

n维数组$A[u_1][u_2]\ldots[u_n]$ $\xrightarrow{\text{映射}}$ 一维数组B[k]

如何实现2维数组int $a[u_1][u_2]$空间的动态分配和元素$a[i_1][i_2]$的访问？

1) 数组空间的分配

total = $u_1*u_2*u_3*\ldots*u_n$; //总的元素个数

B = (datatype*)malloc(total * sizeof(datatype)); //一维存储空间

2) A中元素在**B**中位置的映射

$$A[i_1][i_2]\ldots[i_n] \Rightarrow B[i_1*u_2*u_3*\ldots*u_n + i_2*u_3*u_4*\ldots*u_n + \ldots + i_{n-1}*u_n + i_n]$$

$$= B[i_1*C_1 + i_2*C_2 + \ldots + i_{n-1}*C_{n-1} + i_n*C_n]$$

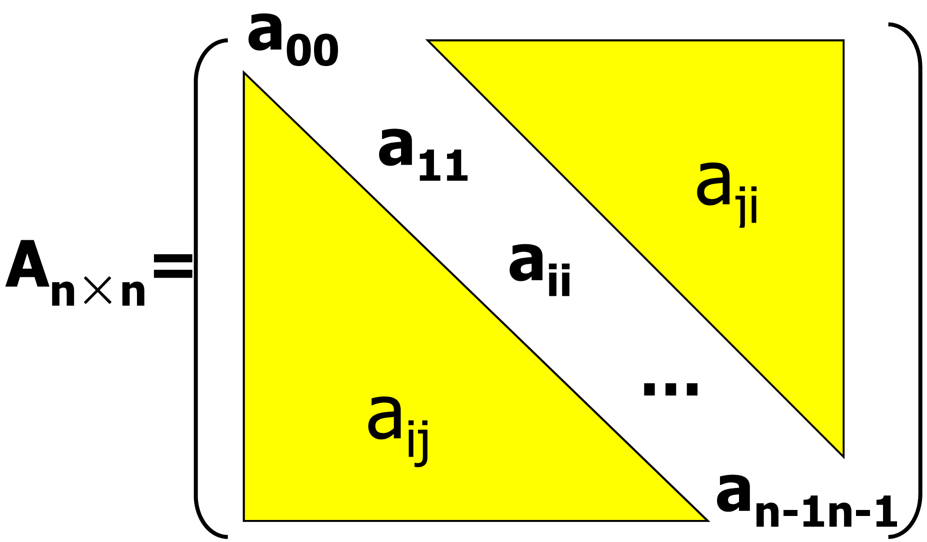
$$C_1=u_2*C_2,\ C_i=u_{i+1}*C_{i+1},\ \ldots,\ C_{n-1}=u_n,\ C_n=1$$

5.3 矩阵的压缩存储

1. 特殊矩阵的压缩存储

(1) 对称矩阵

满足$a_{ij} = a_{ji}$，(0<=i, j<n)



5.3 矩阵的压缩存储

只要存储包括主对角线的下三角元素即可。

$$A_{n\times n} = \begin{pmatrix} a_{00} \\ a_{10}\ a_{11} \\ \cdots\cdots\cdots \\ a_{i0}\ \ a_{i1}\ \ldots a_{ii} \\ \cdots\cdots\cdots\cdots\cdots \\ a_{n-1,0}\ \ a_{n-1,1}\ \ldots a_{n-1,n-1} \end{pmatrix}$$

以一维数组按行主次序存放

$A_{n\times n}$需存储的元素个数：

$1+2+3+\ldots+n = n(n+1)/2$

即S[n(n+1)/2]

S[n(n+1)/2]:

a_{00}	a_{10}	a_{11}	…	a_{ij}	…	a_{n-1n-1}
0	1	2	……	k	……	n(n+1)/2-1

确定元素的位置：设i>=j

a[i][j] = a[j][i] = S[1+2+…+（i+1） + j] = S[i(i+1)/2 +j]

特殊矩阵的压缩存储

(2) 下三角矩阵

$$
\mathbf{A}_{n\times n} = \begin{pmatrix}
a_{00} & & & C \\
a_{10}\ a_{11} & & & \\
\cdots\cdots\cdots & & & \\
a_{i0}\ \ a_{i1}\ \cdots a_{ii} & & & \\
\cdots\cdots\cdots\cdots\cdots & & & \\
a_{n-1,0}\ \ a_{n-1,1}\ \cdots a_{n-1,n-1} & & &
\end{pmatrix}
$$

C是常数

压缩存储方法类似于对称矩阵

$$
a[i][j] = \begin{cases} S[i(i+1)/2 + j] & \text{当} i>=j \text{时} \\ C & \text{当} i<j \text{时} \end{cases}
$$

特殊矩阵的压缩存储

(3) 上三角矩阵

$$A_{n\times n} = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ & a_{11} & \cdots & a_{1,n-1} \\ & & \cdots\cdots\cdots & \\ & C & a_{ii} \; \cdots & a_{i,n-1} \\ & & \cdots\cdots & \\ & & & a_{n-1,n-1} \end{pmatrix}$$

C是常数

按行主次序存储上三角元素

$$a[i][j] = \begin{cases} S[in-i(i+1)/2 +j] & \text{当}i<=j\text{时} \\ C & \text{当}i>j\text{时} \end{cases}$$

特殊矩阵的压缩存储

(4) 对角线矩阵

如包括主对角线的三对角线矩阵

$$A_{n\times n}=\begin{pmatrix} a_{00} & a_{01} & & & & \\ a_{10} & a_{11} & a_{12} & & C & \\ & & \cdots\cdots & & & \\ & C & a_{i,i-1} & a_{i,i} & a_{i,i+1} & \\ & & & \cdots\cdots & & \\ & & & & a_{n-1,n-2} & a_{n-1,n-1} \end{pmatrix}$$

C是常数

按行主次序存储于一维数组S

元素个数：**(n-1)+n + (n+1)**

= 3n-2

$$a[i][j]=\begin{cases} S[2i+j] & \begin{cases} 当i=j+1，S[3i-1] \\ 当i=j，S[3i] \\ 当i=j-1，S[3i+1] \end{cases} \\ C & 否则 \end{cases}$$

5.3 矩阵的压缩存储

2. 稀疏矩阵的压缩存储

稀疏矩阵：矩阵中有大量的**0**元素

压缩存储方法：只存储非0元素。

(1) 三元组表

存储每个非**0**元素的行号、列号、值

三元组：(row, col, val)，其中row、col、val分别为非0元素的行号、列号、值。

以行主次序将稀疏矩阵中非0元素以三元组形式存入一个数组，即三元组表。

三元组表的存储结构如何表示？

三元组表

三元组表的存储结构描述：

```
#define MAXSIZE 64   //最大非0元素个数
typedef struct {        //三元组类型
     int row, col;
     datatype val;
}tritype;
typedef struct { //三元组表
    tritype data[MAXSIZE];   //三元组表存储空间
    int mu, nu, tu;              //原矩阵的行数、列数、非0元素个数
}Tsmtype, *Tsmlink;         //三元组表说明符
```

三元组表

例 5.3 矩阵的转置。

求矩阵A的转置矩阵B，算法很简单：

```
for (col = 0; col < nu; col++)
    for (row = 0; row < mu; row++)
        B[col][row] = A[row][col];
```

三元组表

三元组表所表示的矩阵的转置：

$$A_{4\times5}=\begin{pmatrix}0&2&0&0&4\\6&0&8&0&0\\0&9&0&0&0\\3&0&0&0&0\end{pmatrix}$$

$$B_{5\times4}=\begin{pmatrix}0&6&0&3\\2&0&9&0\\0&8&0&0\\0&0&0&0\\4&0&0&0\end{pmatrix}$$

现在，要通过**A**的三元组表求其转置矩阵**B**的三元组表。

row	col	val
0	1	2
0	4	4
1	0	6
1	2	8
2	1	9
3	0	3

row	col	val
0	1	6
0	3	3
1	0	2
1	2	9
2	1	8
4	0	4

怎么求？

行号与列号互换？

三元组表的转置

方法1：设矩阵A是m行、n列、t个非0元素

从头到尾扫描A，将第0列的元素依次放入B（行号列号互换）；

从头到尾扫描A，将第1列的元素依次放入B（行号列号互换）；

. . . . . .

从头到尾扫描A，将第n-1列的元素依次放入B（行号列号互换）；

扫描A几趟？ 矩阵A的列数n

三元组表的转置

算法描述：

```
void Transm(Tsmtype *A, Tsmtype *B)
{
    int p, q, col;
    B->mu = A->nu; B->nu = A->mu; B->tu = A->tu;
    if (A->tu == 0) return;  //无非0元素
    q =0;    //目标表的序号
    for (col = 0; col < A->nu; col++)  //扫描A的所有列
        for (p = 0; p < A->tu; p++)    //扫描所有非0元素
            if (A->data[p].col == col) {
                B->data[q].row = A->data[p].col; //行列号互换
                B->data[q].col = A->data[p].row;
                B->data[q].val = A->data[p].val;
                q++;
            }
}
```

三元组表的转置

$$A_{4\times5}=\begin{array}{c}\begin{pmatrix}0&2&0&0&4\\6&0&8&0&0\\0&9&0&0&0\\3&0&0&0&0\end{pmatrix}\\ \begin{matrix}0&1&2&3&4\end{matrix}\end{array}$$

col

（矩阵A 的三元组表）

p → 0

	row	col	val
0	0	1	2
1	0	4	4
2	1	0	6
3	1	2	8
4	2	1	9
5	3	0	3

（转置后的三元组表）

q → 0

	row	col	val
0	0	1	6
1	0	3	3
2	1	0	2
3	1	2	9
4	2	1	8
5	4	0	4

$$B_{5\times4}=\begin{pmatrix}0&6&0&3\\2&0&9&0\\0&8&0&0\\0&0&0&0\\4&0&0&0\end{pmatrix}$$

算法的时间复杂度：

O(t*n)

n=列数, t=非0元素个数

效率低！

原因：反复扫描A

如何改进？

三元组表的转置

最理想的算法：从头开始扫描A的三元组**1趟**，在扫描过程中，将A的每个三元组放入B的正确位置。

（矩阵A 的三元组表）

	row	col	val
p → 0	0	1	2
1	0	4	4
2	1	0	6
3	1	2	8
4	2	1	9
5	3	0	3

第1列放到B中的什么位置?

依赖于A中第0列非0元素个数

第i列在B中的位置依赖于A中第0~i-1列非0元素的个数

第0列放到B中的什么位置?
可以确定!

（转置后B的三元组表）

	row	col	val
q → 0	0	1	6
1	0	3	3
2	1	0	2
3	1	2	9
4	2	1	8
5	4	0	4

需要一些预处理工作。

只要预先计算出A中每列的非0元素个数就容易解决了。

三元组表的转置

处理方法： 设原矩阵A为m行、n列、t个非0元素。

（1）扫描1趟A，得到每列非0元素个数。

引入S[n]，存放原矩阵每列非0元素个数，即

S[i]：A第i列非0元素个数

如：S：2，2，1，0，1

时间复杂度为O(t)。

（2）计算原矩阵A每列在B中的起始位置。

引入T[n]，存放原矩阵每列首个非0元素在B中的位置，即

$$\mathbf{T[i]} = \begin{cases} \mathbf{0} & \mathbf{i = 0} \\ \mathbf{T[i-1] + S[i-1]} & \mathbf{i > 0} \end{cases}$$

如：T：0，2，4，5，5

时间复杂度：O(n)。

（3）扫描1趟A，把每个三元组行号列号交换后放入B中正确位置。

时间复杂度：O(t)。

整个算法的时间复杂度：O(n + t)

三元组表的转置

改进的算法描述：

```
void Transm1(Tsmtype *A, Tsmtype *B)
{
    int j, j, k;
    int S[MAXSIZE], T[MAXSIZE];
    B->mu = A->nu; B->nu = A->mu; B->tu = A->tu;
    if (A->tu == 0) return;
    for (i = 0; i < A->nu; i++)  //初始化S，O(n)
        S[i] = 0;
    for (i = 0; i < A->tu; i++)  //求S，O(t)
        S[A->data[i].col] += 1;
    T[0] = 0;
    for (i = 1; i < A->nu; i++)  //求T，O(n)
        T[i] = T[i - 1] + S[i - 1];
```

三元组表的转置

```
for (k = 0; k < A->tu; k++) { // O(t)
    j  = A->data[k].col);  //列号
    i = T[j]; //当前元素在B中的序号
    B->data[i].row = j;
    B->data[i].col = A->data[k].row;
    B->data[i].val = A->data[k].val;
    T[j]++;  //指向第j列下一个非0元素在B中的序号
  }
}
```

稀疏矩阵的压缩存储

(2) 十字链表

- ✓ 将同一行的非**0**元素构成一个单循环链表；
- ✓ 将同一列的非**0**元素构成一个单循环链表；
- ✓ 每个非**0**元素同时出现在所在行和列的链表中。

十字链表

3种节点：

row	col	val
down	right	

row: 行号

col: 列号

val: 取值

down: 指向同一列下一个非0元素节点

right: 指向同一行下一个非0元素节点

1）非0元素节点

0	0	next
down	right	

next: 指向下一个表头节点

down: 指向对应列的首个非0元素节点

right: 指向对应行的首个非0元素节点

第i行与第i列共用1个表头节点

2）行(列)的表头节点

m	n	next
t	不用	

m: 矩阵行数

n: 矩阵列数

next: 指向第0行（列）的表头节点

t: 非0元素个数，或不用

3）总表头节点

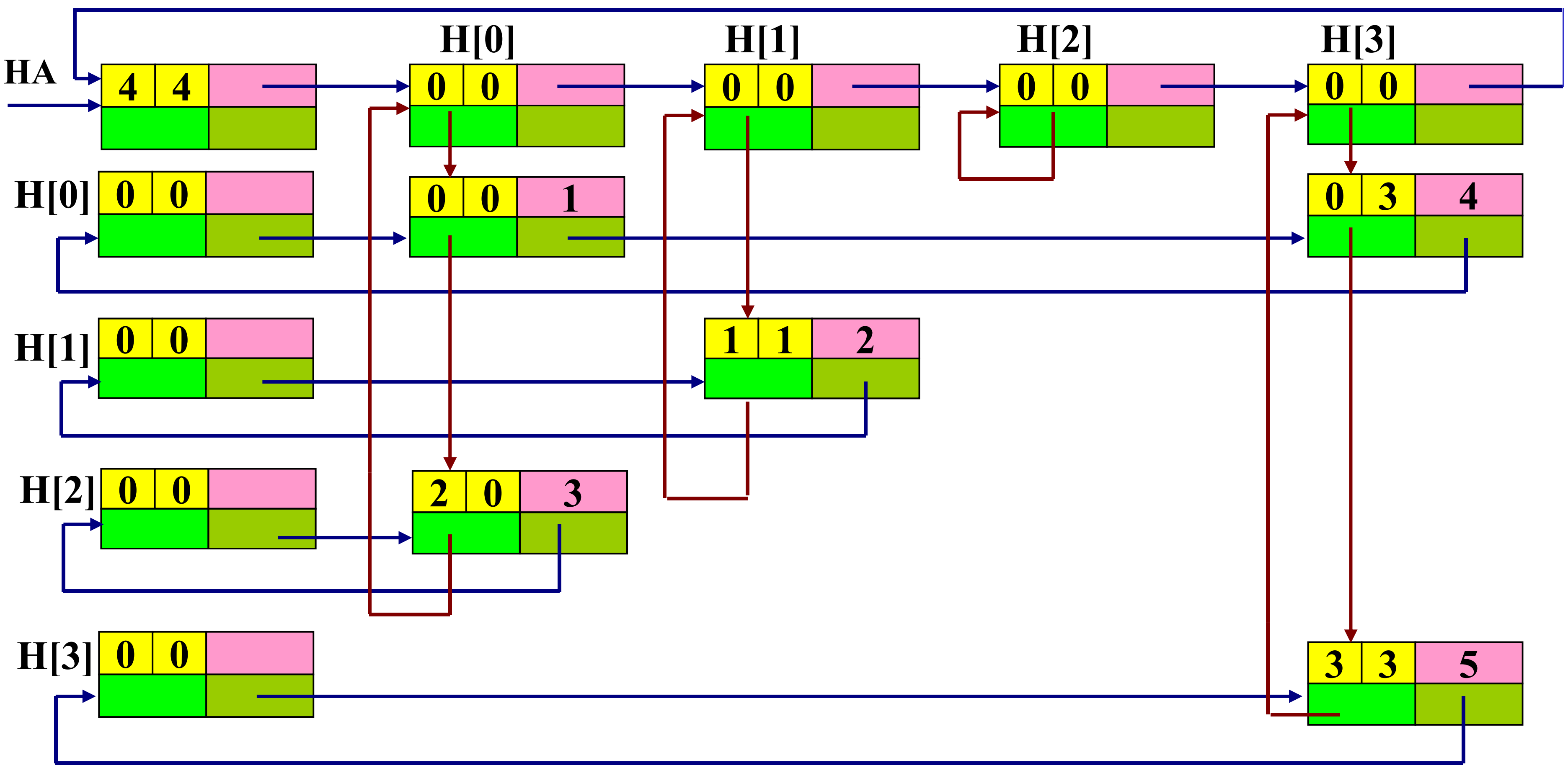
十字链表

节点类型描述：

```
typedef struct node {
    int row, col;
    union {
        struct node *next;
        datatype val;
    } vdata;
    struct node *down,*right;
} nodetype, *tlink;
```

十 字 链 表

例 **5.4** 稀疏矩阵：$A_{4\times4}=\begin{bmatrix}1&0&0&4\\0&2&0&0\\3&0&0&0\\0&0&0&5\end{bmatrix}$ A的十字链表如下图：



十字链表

说明：

- ✓ 第i行与第i列共用1个表头节点，表头节点个数为

max(m, n)，其中，m, n分别为矩阵的行数、列数；

- ✓ 所有表头节点构成1个单循环链表，节点间以next链接；
- ✓ 同一行的非0元素节点（与相应表头节点一起）构成1个单循环链表，节点间以right域链接；
- ✓ 同一列的非0元素节点（与相应表头节点一起）构成1个单循环链表，节点间以down域链接；
- ✓ 总表头节点的next域指向第0行（列）的表头节点。

5.4 广义表的定义及其操作

1. 广义表的定义

广义表的非形式化定义：

广义表或为空表，或者其元素为：

1）原子。或称单元素。

2）广义表。

线性表是同类型元素组成的序列

广义表(multilist)可以表中套表，元素之间的关系体现次序关系和层次关系

线性表是广义表的特例。

5.4 广义表的定义及其操作

广义表的形式化定义：

广义表LS **= (d_0，d_1，…，d_i，…，d_{n-1})**的形式化描述为

LS=(D, R)

$D=\{ d_i \mid d_i \in datatype$ or $d_i \in LS$ (递归定义)，$i=0,1,\ldots,n-1, n \geq 0\}$

$R=\{<d_i, d_{i+1}> \mid d_i, d_{i+1} \in D, 0 \leq i \leq n-2\}$

n为表长（n=0 时为空表），若d_i为原子，则称d_i为LS的单元素，否则d_i称为LS的子表。

当广义表非空时，定义两个函数

$head(LS) = d_0$

$tail(LS) = (d_1, \ldots, d_{n-1})$

5.4 广义表的定义及其操作

2. 广义表的抽象数据类型

ADT Lists {

数据元素集：$D=\{d_i|d_i \in datatype \text{ or } d_i \in Lists, i=0,1,2, \ldots\ldots n-1, n\geq 0\}$

数据关系集：$R=\{<d_i, d_{i+1}>|d_i, d_{i+1} \in D, 0\leq i\leq n-2\}$

基本操作集：P

GetHead(L)

初始条件：广义表L存在。

操作结果：取广义表L的头元素d_0。

GetTail(L)

初始条件：广义表L存在。

操作结果：取广义表L的表尾$(d_1,\ldots, d_{n-1})$。

ListsDepth(L)

初始条件：广义表L存在。

操作结果：返回广义表L的深度(最大层数)。

. . . . . .

} ADT Lists;

5.4 广义表的定义及其操作

约定：大写字母A～Z为表名，小写字母 a～z为单元素。

A=() —— 空表，表长=0，无表头、表尾

B=(a, b) —— 线性表 (广义表特例), 表长=2, head(B)=a, tail(B)=(b)

C=(e, B) —— 表长=2，head(C)=e, tail(C)=(B)=((a,b)), 即 C = (e, (a, b))

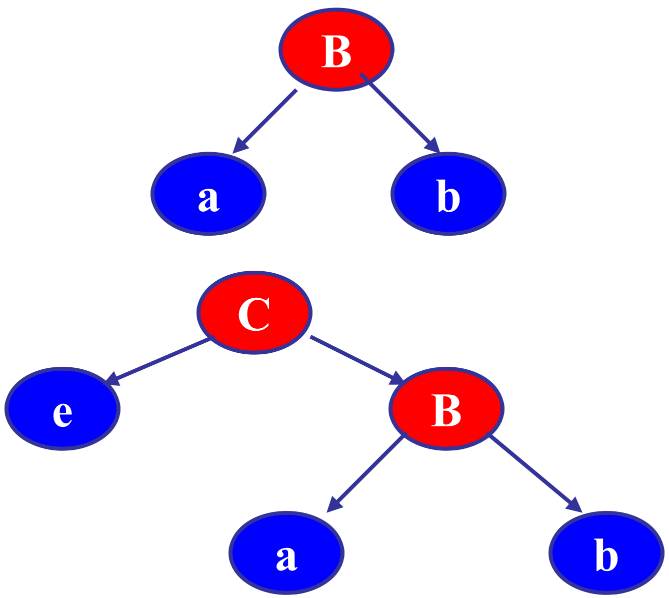
D=(A, B, C) —— 表长=3，head(D)=A=(), tail(D)=(B,C)=((a, b), (e, (a, b)))

E=(a, E) ——表长=2，head(E)=a, tail(E)=(E), 实际上表E为 (a, (a, (a, ...)))，是一个特殊的广义表，称为递归表，或无限表。

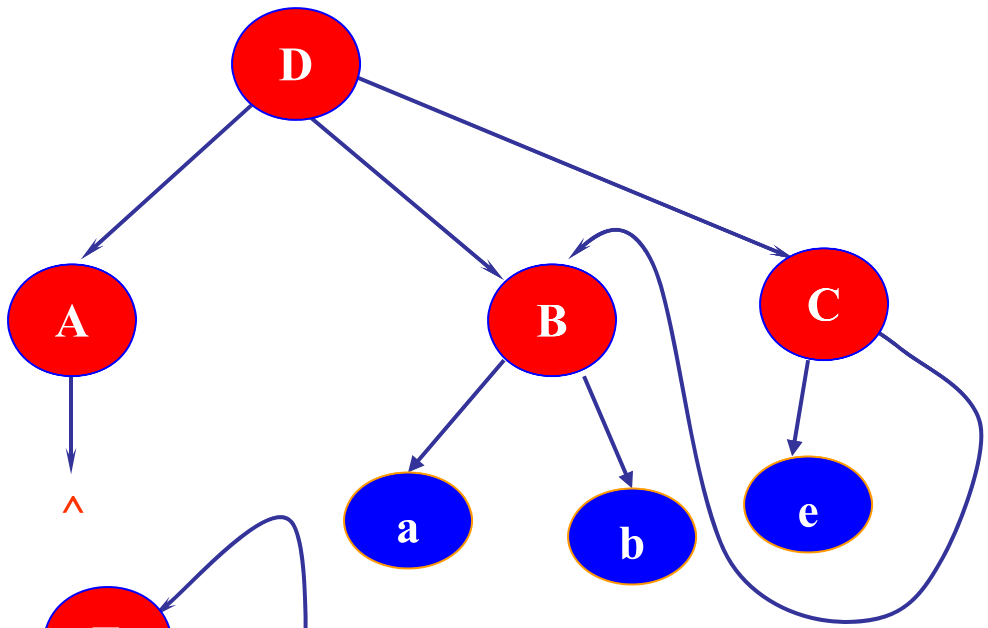
5.4 广义表的定义及其操作

广义表的特点：

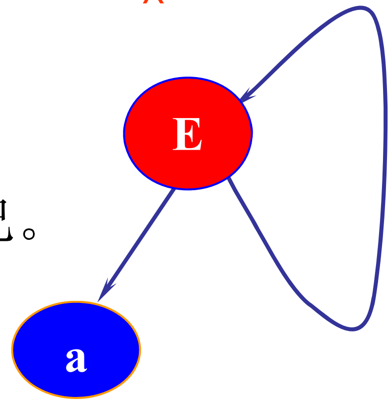
1）嵌套：表中套表。



2）共享：表可以是其它表的子表，
或表的元素取自其他的表。



3）递归：表元素可以是表自己。



5.4 广义表的定义及其操作

表中任一单元素可由**Gethead(Ls)**和**Gettail(Ls)**函数导出

如：取表**A=(a,(b,d,e))**中单元素**d**的运算为：

Gethead(Gettail(Gethead(Gettail(A))))

5.5 广义表的存储结构

一般采用链表，称为广义链表

1. 单链表示法

元素d_i的节点形式：

atom	data/link	next

$$atom = \begin{cases} 0 & 当d_i为原子时 \\ 1 & 当d_i为子表时 \end{cases} \qquad data/link = \begin{cases} data & 当atom=0时 \\ link & 当atom=1时 \end{cases}$$

next指向下一个元素节点d_{i+1}

为便于处理，引入广义表的头节点：

1	↓	^

atom=1；

data/link取link，指向首个元素节点d_0；

next=NULL

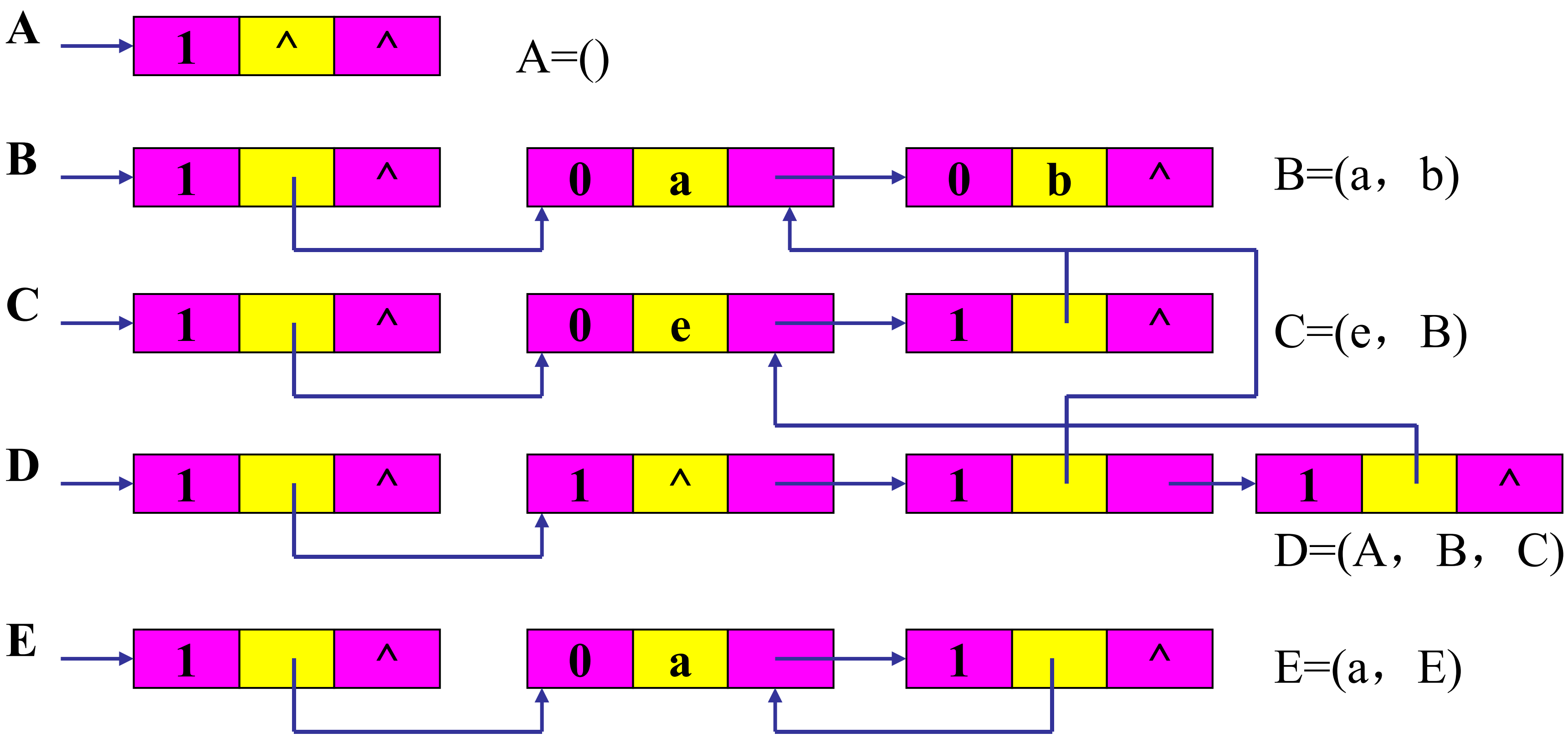
5.5 广义表的存储结构

节点描述：

```
typedef struct node {
    int atom;
    union {
        datatype data;
        struct node *link;
    } dtype;
    struct node *next;
}Lsnode, *Lslink;
```

5.5 广义表的存储结构

例：



5.5 广义表的存储结构

2. 双链表示法

元素d_i的节点形式：

link1	data	link2

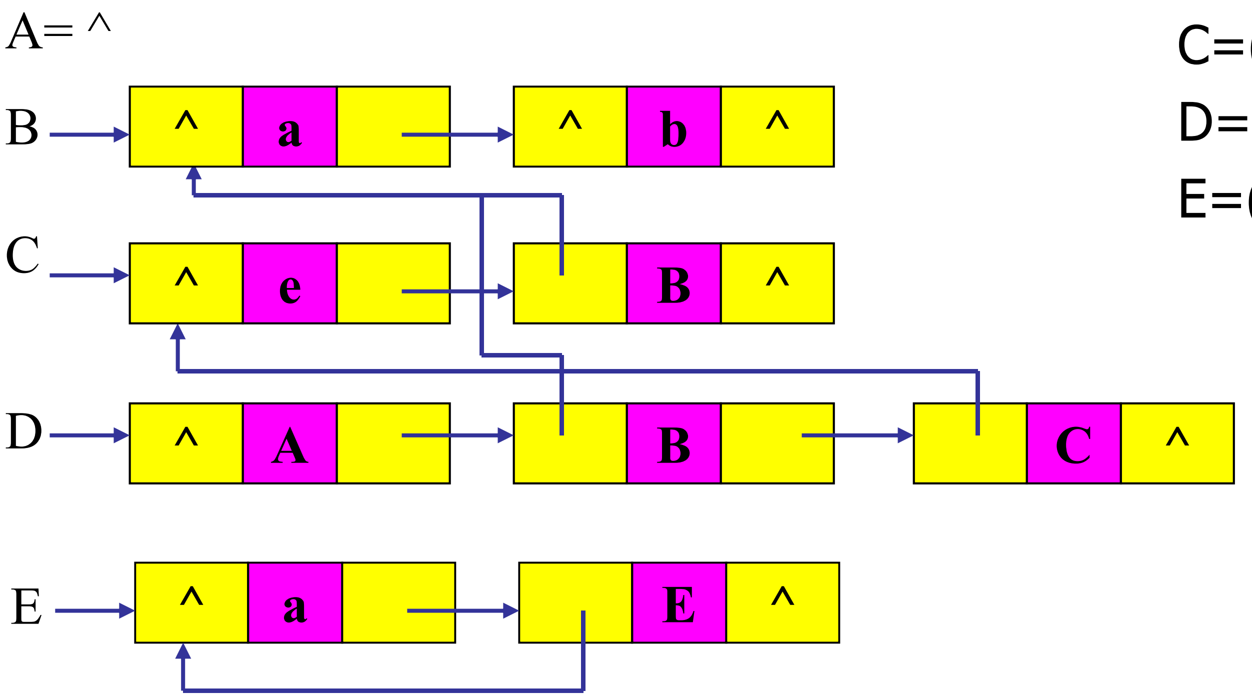
$$\text{link1} = \begin{cases} \text{指向子表的指针} & \text{当}d_i\text{为子表时} \\ \wedge & \text{当}d_i\text{为原子时} \end{cases}$$

$$\text{data} = \begin{cases} \text{表名} & \text{当}d_i\text{为子表时} \\ \text{原子值} & \text{当}d_i\text{为原子时} \end{cases}$$

link2指向下一个元素节点d_{i+1}

5.5 广义表的存储结构

例：



A=()

B=(a, b)

C=(e, B)

D=(A, B, C)

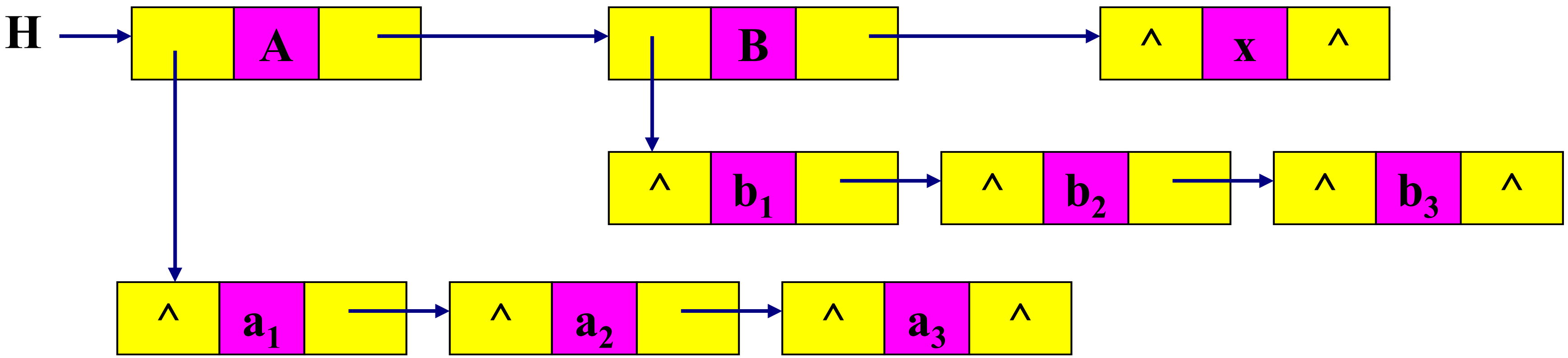
E=(a, E)

5.5 广义表的存储结构

例 **5.6** 设职工工资的表头H：

工资收入A			扣除B			实发x
基本工资 a_1	岗位津贴 a_2	福利 a_3	房租 b_1	水电 b_2	其他 b_3	

即H=(A, B, x)，A=(a1，a2，a3)，B=(b1，b2，b3)。H的双链结构：



5.5 广义表的存储结构

3. 求广义表深度的算法

深度：广义表括号的层数

例如：

() —— 空表，深度=1

(a, b, c) —— 深度=1

(a, (b, c)) —— 深度=2

递归定义：

原子深度=0；

空表深度=1；

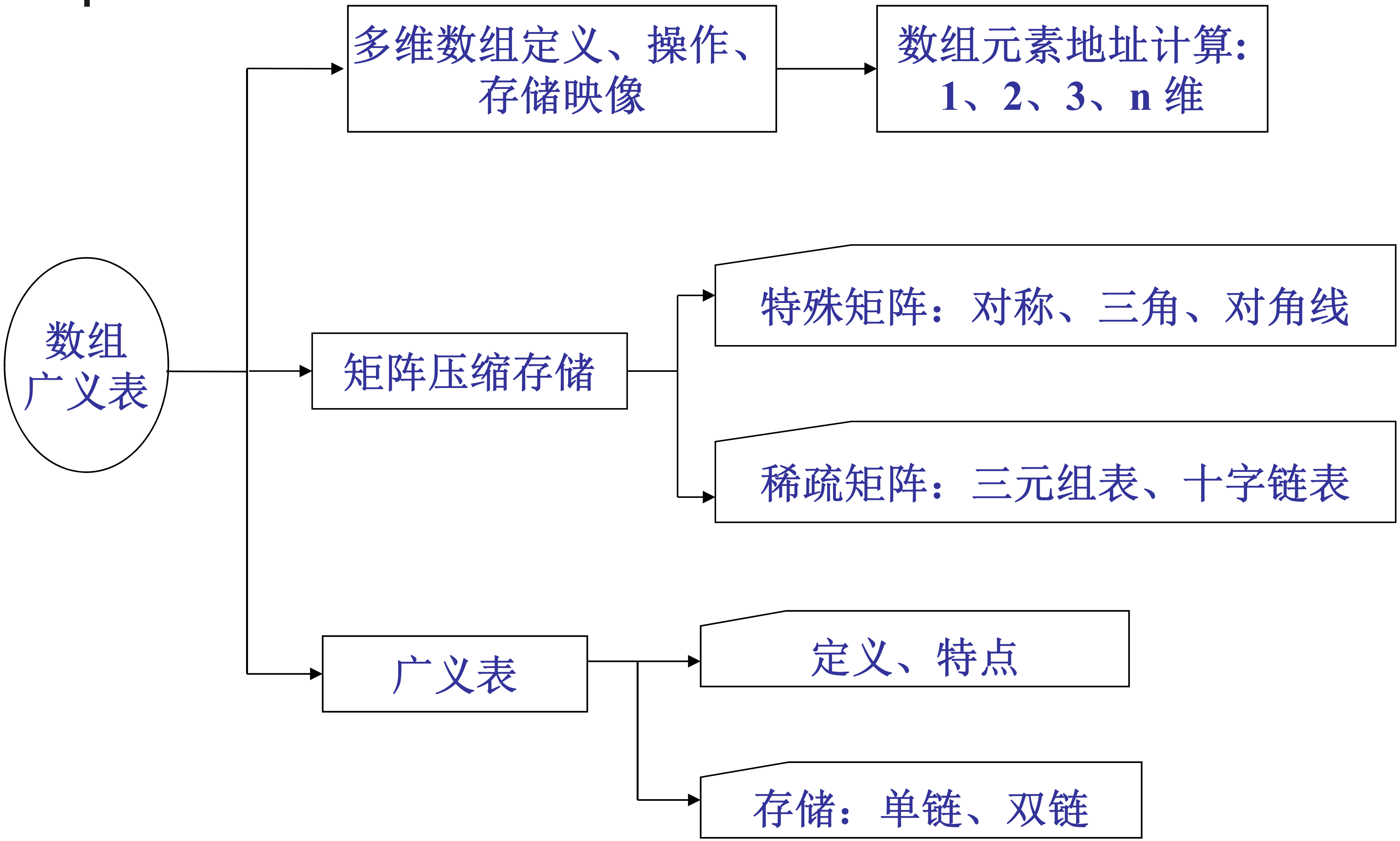
广义表深度 = max(各元素深度) + 1

5.5 广义表的存储结构

求广义表深度的递归算法：

```
//广义表采用单链结构，返回LS指向的广义表深度
int ListsDepth(Lslink LS)
{
    int d, maxd;
    if (LS->atom == 0) return 0; //原子深度=0
    Lslink p = LS->dtype.link;
    if (p == NULL) return 1; //空表深度=1，该行代码可去掉
    maxd = 0;
    while (p) {  //求每个元素的深度
        d = ListsDepth(p);
        if (d > maxd) maxd = d;
        p = p->next;
    }
    return (maxd + 1)
}
```

5.6 小结



数组
广义表
多维数组定义、操作、
存储映像
数组元素地址计算：
1、2、3、n 维
矩阵压缩存储
特殊矩阵：对称、三角、对角线
稀疏矩阵：三元组表、十字链表
广义表
定义、特点
存储：单链、双链

第5章 作业

1. 设矩阵:

$$A_{5\times5}=\begin{pmatrix}1&0&0&0&2\\0&3&0&0&4\\0&0&0&5&0\\0&0&0&6&0\\0&0&0&0&7\end{pmatrix}$$

(行列下标i、j满足：$1\leq i$，$j\leq5$)

1) 若将A视为一个上三角矩阵时，请画出A的“按行优先存储”的压缩存储表S，并写出A中元素之下标[i,j]与S中元素之下标k之间的关系；

2) 若将A视为一个稀疏矩阵时，请画出A的三元组表和十字链表结构。

第5章 作 业

2. 设银行一天营业业务表头H：

<table>
<tr><th colspan="5">存款A</th><th colspan="3">取款B</th><th rowspan="3">进款X</th></tr>
<tr><th rowspan="2">活期
a_1</th><th colspan="3">定期D</th><th rowspan="2">总存款
a_3</th><th rowspan="2">支取
b_1</th><th rowspan="2">利息
b_2</th><th rowspan="2">总支付
b_3</th></tr>
<tr><th>1年
d_1</th><th>2年
d_2</th><th>3年
d_3</th></tr>
</table>

1) 试用广义表形式表示H，并用head(H)和tail(H)函数提取d2；

2) 画出H 的单链及双链结构。