数据结构

第5章　数组和广义表

5.1 数组的定义

5.1.1 数组的递归定义

1. 一维数组的定义：是一个定长线性表$(a_1, a_2, ..., a_n)$。

记为：A= $(a_1, a_2, ..., a_n)$

其中：a_i为数据元素，i为下标/序号，$1 \leqslant i \leqslant n$

上面定义的一维数组的元素，下标范围是1～n，一般情况下，可以给下标一个取值范围：

$$A = (a_{low}, a_{low+1}, ..., a_{high})$$

这里：$low \leqslant high$，数据元素的个数为high-low+1

在C语言中，low取值0。

2. 二维数组是一个定长线性表$(\alpha_1,\alpha_2,\ldots,\alpha_m)$

其中: $\alpha_i=(a_{i1},a_{i2},\ldots,a_{in})$为行向量，$1\le i\le m$， 由m个行向量组成，记作:

$$A_{m*n}=\begin{pmatrix} (\ a_{11} & a_{12} & \ldots & a_{1n}\ ) \\ (\ a_{21} & a_{22} & \ldots & a_{2n}\ ) \\ & \ldots\ldots\ldots\ldots & & \\ (\ a_{m1} & a_{m2} & \ldots & a_{mn}\ ) \end{pmatrix}$$

即 $A_{m*n}=((a_{11}\ a_{12}\ \ldots a_{1n}),(a_{21}\ a_{22}\ \ldots a_{2n}),\ldots,(a_{m1}\ a_{m2}\ \ldots a_{mn}))$

或由n个列向量组成，记作:

$$A_{mxn}=\begin{pmatrix} \overbrace{a_{11}} & \overbrace{a_{12}} & & \overbrace{a_{1n}} \\ a_{21} & a_{22} & \vdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ \underbrace{a_{m1}} & \underbrace{a_{m2}} & & \underbrace{a_{mn}} \end{pmatrix}$$

3. 三维数组是一个定长线性表($\beta_1, \beta_2, ..., \beta_p$)。

其中：β_k=($\alpha_1, \alpha_2, ..., \alpha_m$)为定长二维数组，$1 \leqslant k \leqslant p$

例　三维数组A[1..3,1..4,1..2]， p=3， m=4， n=2

$$A_{3*4*2} = \begin{pmatrix} a_{111} & a_{112} \\ a_{121} & a_{122} \\ a_{131} & a_{132} \\ a_{141} & a_{142} \end{pmatrix} \quad \begin{pmatrix} a_{211} & a_{212} \\ a_{221} & a_{222} \\ a_{231} & a_{232} \\ a_{241} & a_{242} \end{pmatrix} \quad \begin{pmatrix} a_{311} & a_{312} \\ a_{321} & a_{322} \\ a_{331} & a_{332} \\ a_{341} & a_{342} \end{pmatrix}$$

第1页　　第2页　　第3页

5.1.2 数组的操作

（1）**InitArray(&A，n，bound1，…，boundn)**
创建一个**n**维数组，各维的长度通过$\mathbf{bound_i}$表示

（2）**DestroyArray(&A)**
销毁数组**A**

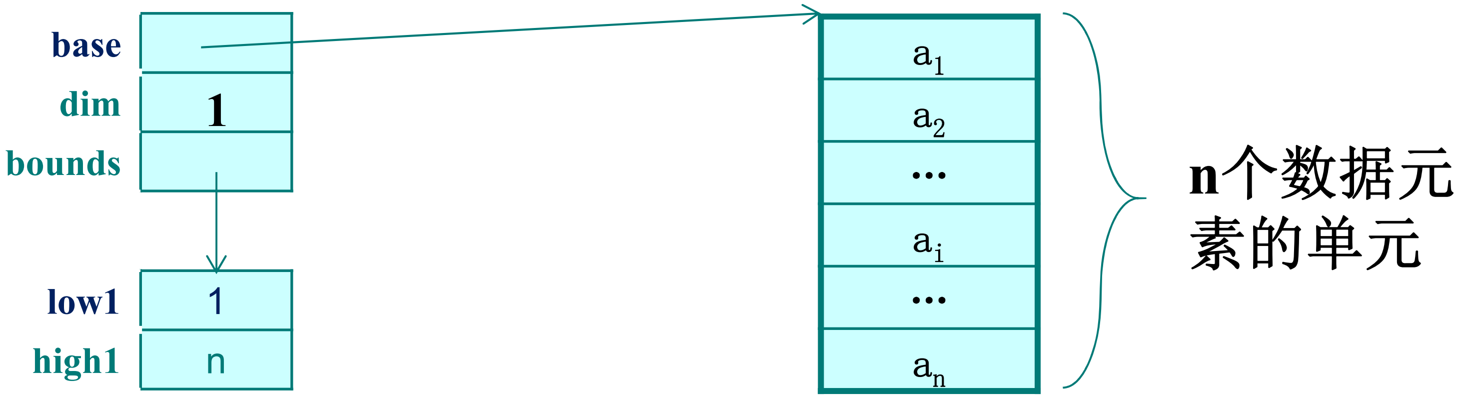
（3）**Value(A，&e，index1，…，indexn)**
读取数组指定下标的元素值到**e**中

（4）**Assign(A，e ，index1，…，indexn)**
将**e**的值赋值给指定下标的数组元素中

5.2数组的顺序表示和实现

5.2.1一维数组的顺序表示

例1 二维数组A= ($a_1, a_2, ..., a_n$), b是分配的连续存储单元首地址, L是1个数据元素所占的单元数 。



一维数组数据元素a_i的地址计算公式

$$LOC(i)=b+(i-1)L=LOC(1)+(i-1)L$$

其中：LOC(i)为元素a_i的存储位置，b是连续存储单元首地址,L是一个数据元素所占的单元数 。

5.2.2 二维数组的顺序表示

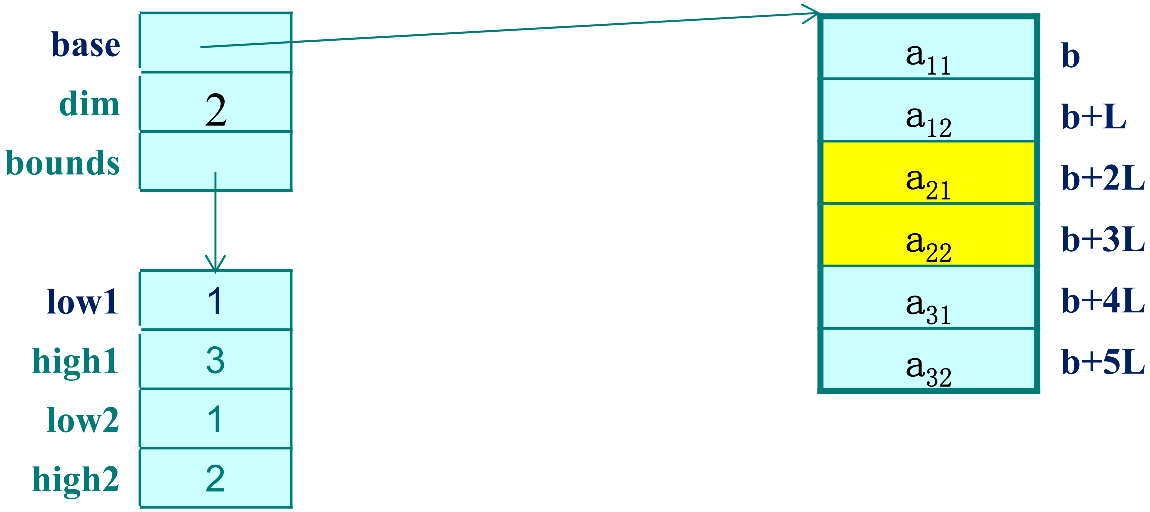
例2 二维数组a[1..3,1..2]

1. 以行序为主序的顺序存储方式

逐行将数组元素放到数据的连续空间中，体现在右边的下标先变化，左边的下标后变化。

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$

逻辑结构



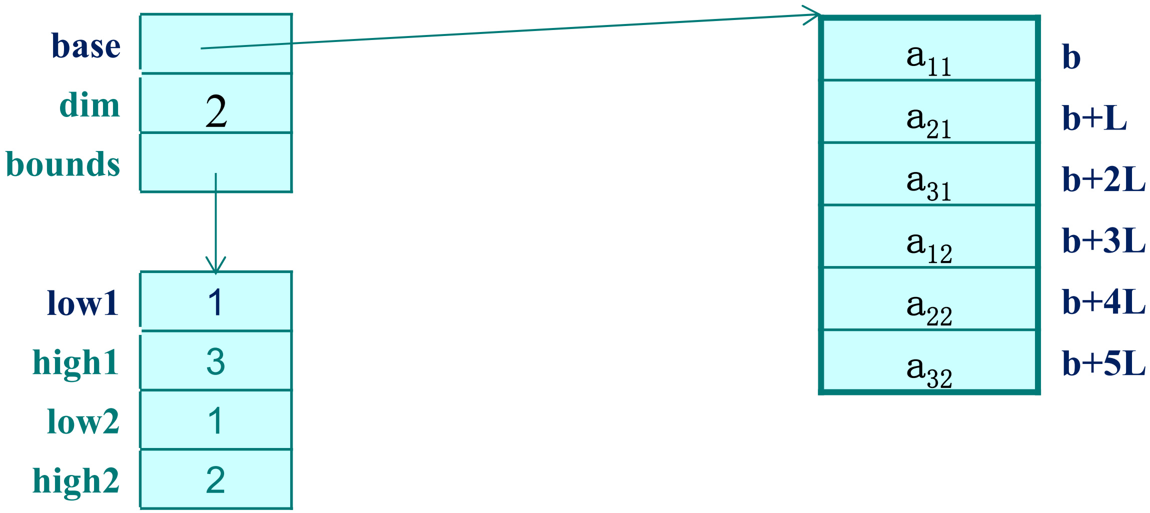
例2（续） 二维数组a[1..3,1..2]

2. 以列序为主序的顺序存储方式

逐列将数组元素放到数据的连续空间中，体现在左边的下标先变化，右边的下标后变化。

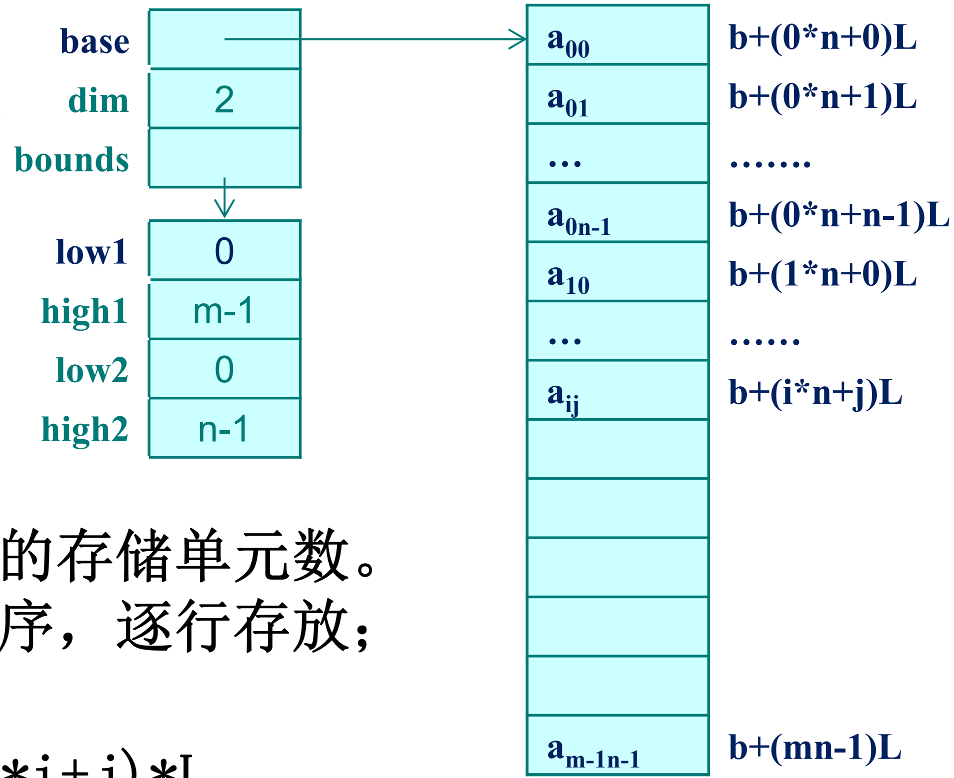
$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$

逻辑结构



例3 二维数组a[0..m-1,0..n-1],行序优先

$$A_{mxn}=\begin{pmatrix} a_{00} & \cdots & a_{0j} & \cdots & a_{0n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i0} & \cdots & a_{ij} & \cdots & a_{in-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \underbrace{a_{m-10} \;\cdot\cdot}_{\text{共j列}} & & a_{m-1j}\cdot\cdot & & a_{m-1n-1} \end{pmatrix} \left.\begin{matrix} \\ \\ \end{matrix}\right\} \text{共i行}$$



(1)开空间：明确开多大的空间；

b为首地址，L为每个元素所占的存储单元数。

(2) 确定摆放方式：以行序为主序，逐行存放；

(3)分析寻址公式,a_{ij}的地址为:

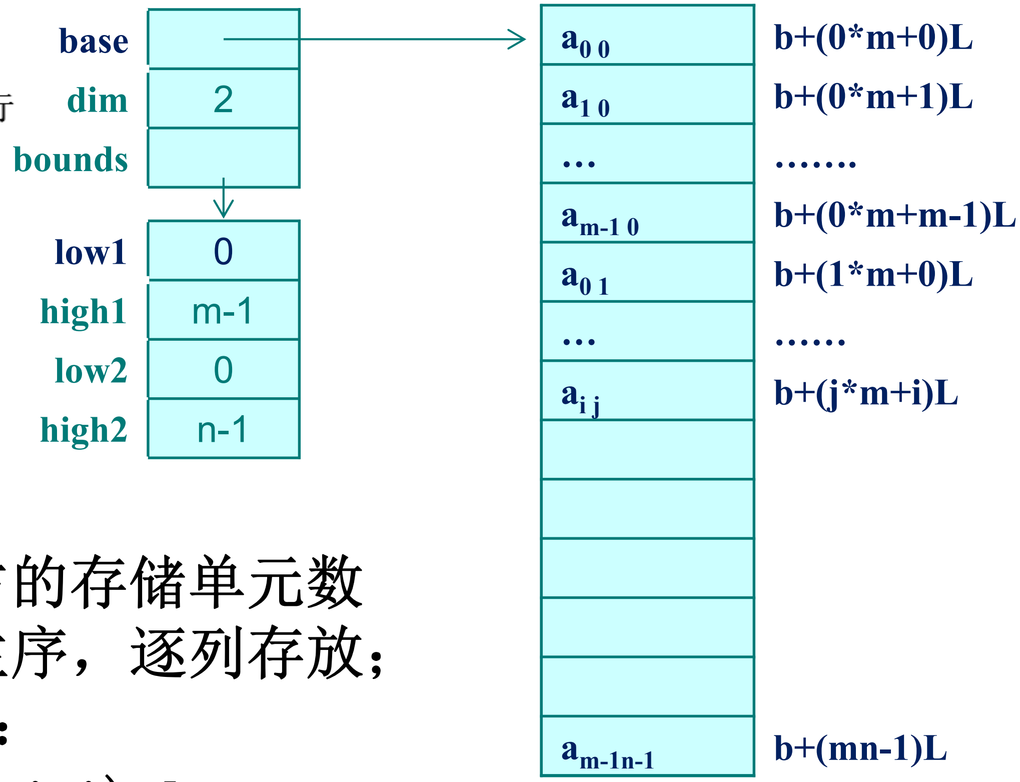
$$Loc(i,j)=Loc(0,0)+(n*i+j)*L$$
$$=b+(n*(i-1)+j)*L \qquad 0\leqslant i\leqslant m-1,\ 0\leqslant j\leqslant n-1$$

例3（续） 二维数组a[0..m-1,0..n-1],列序优先

$$A_{m\times n}=\begin{pmatrix} a_{00} & \cdots & a_{0j} & \cdots & a_{0\,n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i0} & \cdots & a_{ij} & \cdots & a_{i\,n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m-1\,0} & \cdots & a_{m-1\,j} & \cdots & a_{m-1\,n-1} \end{pmatrix}$$

共i行

共j列



(1)开空间：明确开多大的空间；

b为首地址，L为每个元素所占的存储单元数

(2) 确定摆放方式：以列序为主序，逐列存放；

(3)分析寻址公式，a_{ij}的地址为：

$$Loc(i,j)=Loc(0,0)+(m*j+i)*L$$
$$=b+(m*j+i)*L \qquad 0\leqslant i\leqslant m-1,\ 0\leqslant j\leqslant n-1$$

n维数组按行序关系的分解

$A[k_1]\ [k_2]\ [k_3]\cdots\cdots\ [k_n]$

分解成了k_1个n-1维数组

$A[0][k_2][k_3]\cdots\ [k_n]$ $\cdots$ $A[i][k_2][k_3]\cdots\ [k_n]$ $\cdots$ $A[k_1-1][k_2][k_3]\cdots\ [k_n]$

分解成了k_2个n-2维数组

$A[i][0][k_3]\cdots\ [k_n]$ $\cdots$ $A[i][j][k_3]\cdots\ [k_n]$ $\cdots$ $A[i\ ][k_2-1][k_3]\cdots\ [k_n]$

$\cdots$ $\cdots$

行序优先存放，元素$A[i_1,i_2,\ldots,i_n]$的寻址公式：

$$LOC(i_1,i_2,\ldots,i_n)=LOC(0,0,\ldots,0)+(i_1*k_2*k_3*\ldots*k_n+i_2*k_3*\ldots*k_n+\ldots\ldots+i_{n-1}*k_n+i_n)*L$$

n维数组按列序关系的分解

$A[k_1]\ [k_2]\ [k_3]\cdots\cdots\ [k_n]$

分解成了k_n个n-1维数组

$A\ [k_1][k_2]\cdots[k_{n-1}][0]$ … $A\ [k_1][k_2]\cdots[k_{n-1}][i]$ … $A\ [k_1][k_2]\cdots[k_{n-1}][k_n-1]$

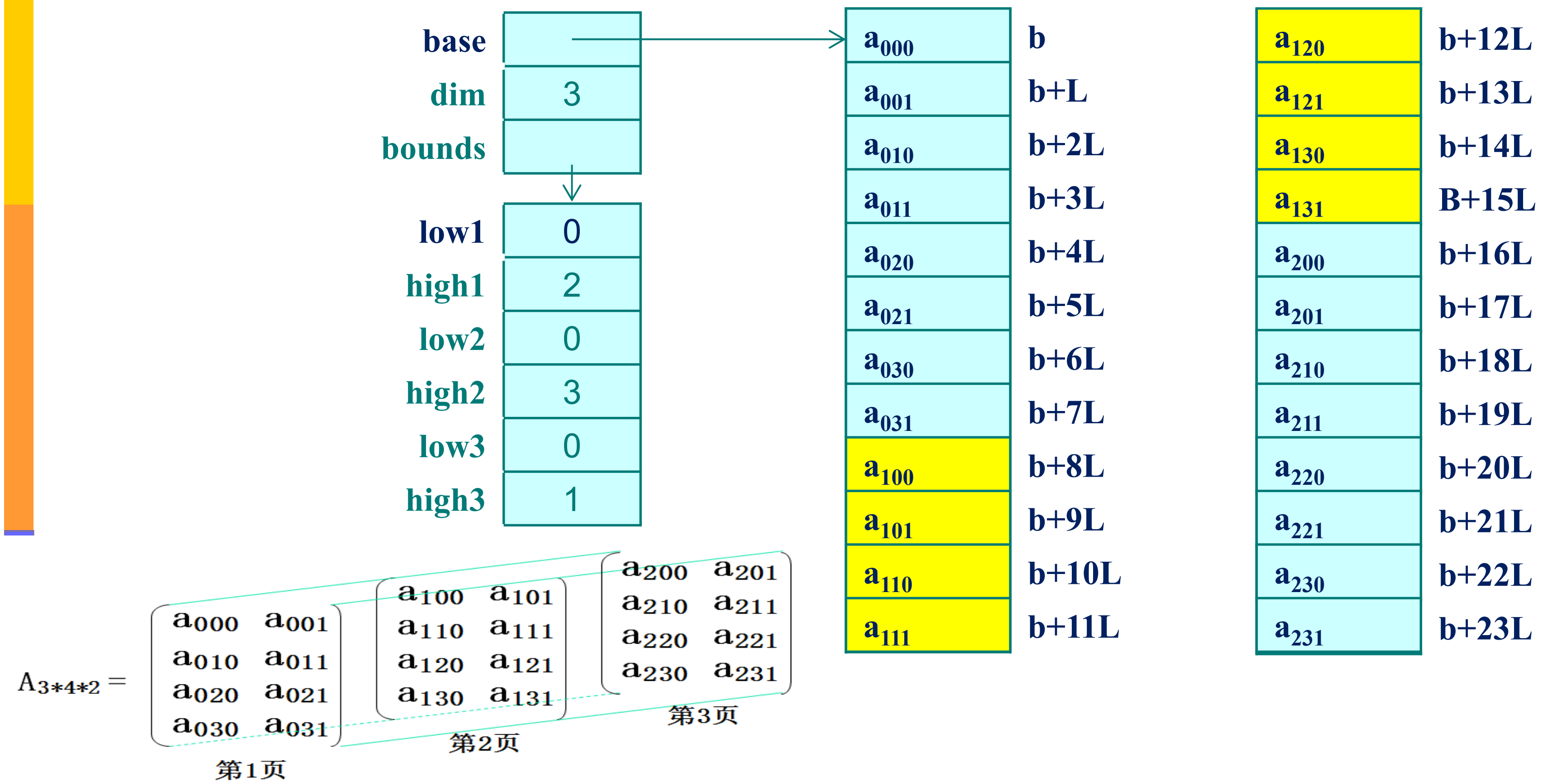
分解成了k_{n-1}个n-2维数组

$A[k_1]\cdots[k_{n-2}][0][i]$ … $A[k_1]\cdots[k_{n-2}][j][i]$ … $A[k_1]\cdots[k_{n-2}][k_{n-1}-1][i]$

列序优先存放，元素$A[i_1,i_2,\ldots,i_n]$的寻址公式：

$$LOC(i_1,i_2,\ldots,i_n)=LOC(0,0,\ldots,0)+(i_n*k_1*k_2*\ldots*k_{n-1}+i_{n-1}*k_1*\ldots*k_{n-2}+\ldots\ldots+i_2*k_1+i_1)*L$$

例4 三维数组A[1..,0..n-1],行序优先



4.3 矩阵的压缩存储

4.3.1特殊矩阵的压缩存储

1. n阶对称矩阵

$$A_{nxn}=\overbrace{\underbrace{\begin{pmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & & \\ \cdots & & a_{ji} & \cdots & \\ \cdots & \cdots & \cdots & & \\ a_{i1} & \cdots\ a_{ij} & \cdots & & a_{in} \\ \cdots & \cdots & \cdots & & \\ a_{n1} & \cdots & a_{ni} & \cdots & a_{nn} \end{pmatrix}}_{下三角}}^{上三角}$$

$a_{ij}==a_{ji}$

$1 \leqslant i, j \leqslant n$

下三角元素a_{ij}满足$i \geqslant j$　　数据元素的个数$=1+2+\cdots+n=n(n+1)/2$

假定以行序为主，顺序存储下三角元素到SA[1..n(n+1)/2]

	a_{11}	a_{21}	a_{22}	a_{31}	...	a_{i1}	...	a_{ij}	...	a_{ii}	...	a_{n1}	...	a_{nn}
SA[0]	SA[1]	SA[2]	SA[3]	SA[4]		...		SA[k]		...				SA[n(n+1)/2]

如何求a_{ij}在SA中的位置，即序号k？

(1) 设aij在下三角，i≥j

∵ 第1~i-1行共有元素

1+2+3+…(i-1)=i(i-1)/2 （个）

ai1~aij共有j个元素

∴ aij的序号为:

k=i(i-1)/2+j

(2) 设a_{ij}在上三角，i<j

∵ 上三角的a_{ij}等于下三角的a_{ji}

下三角的a_{ji}的序号为

$k=j(j-1)/2+i \qquad i<j$

∴ 上三角的a_{ij}的序号为

$k=j(j-1)/2+i \qquad i<j$

由(1)和(2)，任意a_{ij}在SA中的序号，为

$$k=\begin{cases} i(i-1)/2+j & i\geqslant j \\ j(j-1)/2+i & i<j \end{cases}$$

或：

$$A[i,j]=\begin{cases} SA[i(i-1)/2+j] & i\geqslant j \\ SA[j(j-1)/2+i] & i<j \end{cases}$$

该公式称为在SA中的映象函数，下标转换公式。

2. 三对角矩阵

$$A_{nxn}=\begin{pmatrix} a_{11} & a_{12} & & & & \text{全}0 \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & \cdots & a_{ij} & \cdots & \\ \text{全}0 & & & a_{n-1n-2} & a_{n-1n-1} & a_{n-1n} \\ & & & & a_{nn-1} & a_{nn} \end{pmatrix}$$

（1）元素a_{ij}，在三对角的条件：$|i-j| \leqslant 1$；

（2）三对角的元素个数： 3n-2

2. 三对角矩阵(续)

假定以行序优先，将三对角元素顺序存储非0元素到SA[1..3n-2]中

	a_{11}	a_{12}	a_{21}	a_{22}	a_{23}	a_{32}	...	a_{ij}	...	a_{nn-1}	a_{nn}
SA[0]	SA[1]	SA[2]	SA[3]	SA[4]	SA[5]	SA[6]	···	SA[k]		...	SA[3n-2]

$$A_{nxn}=\begin{pmatrix} a_{11} & a_{12} & & & & \text{全}0 \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & \cdots & a_{ij} \cdots\cdots & & \\ \text{全}0 & & & a_{n-1n-2} & a_{n-1n-1} & a_{n-1n} \\ & & & & a_{nn-1} & a_{nn} \end{pmatrix}$$

任意三对角元素a_{ij}，在SA中的序号：

$k=(3*(i-1)-1)+(j-i+2)=2i+j-2$

$$A[i,j]=\begin{cases} SA[k] & |i-j|\leqslant 1 \\ 0 & \text{其它} \end{cases}$$

5.3.2 稀疏矩阵的压缩存储

1.三元组表

例　稀疏矩阵M及其转置矩阵T

$$M=\begin{pmatrix} 0 & 33 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 & 36 & 0 \\ 0 & 0 & 16 & 0 & 0 & 0 & 0 \\ 0 & 28 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 37 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{三元组表}}$$

（i，　j，e　）
（1，2，33）
（1，3，　9）
（3，1，10）
（3，6，36）
（4，3，16）
（5，2，28）
（6，4，37）

（6，　　7）行列数

三元组顺序表

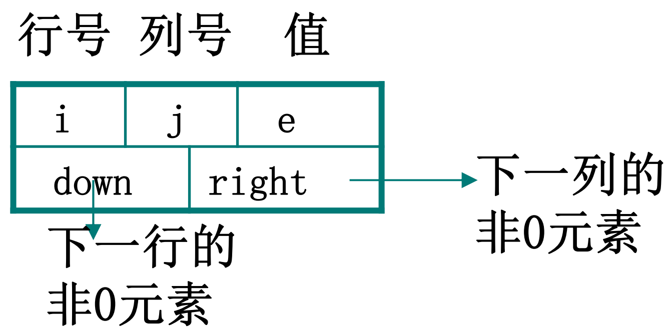
1	2	33
1	3	9
3	1	10
3	6	36
4	3	16
5	2	28
6	4	37
///	////	///
行数(mu)：6		
列数(nu)：7		
非零元(tu)：7		

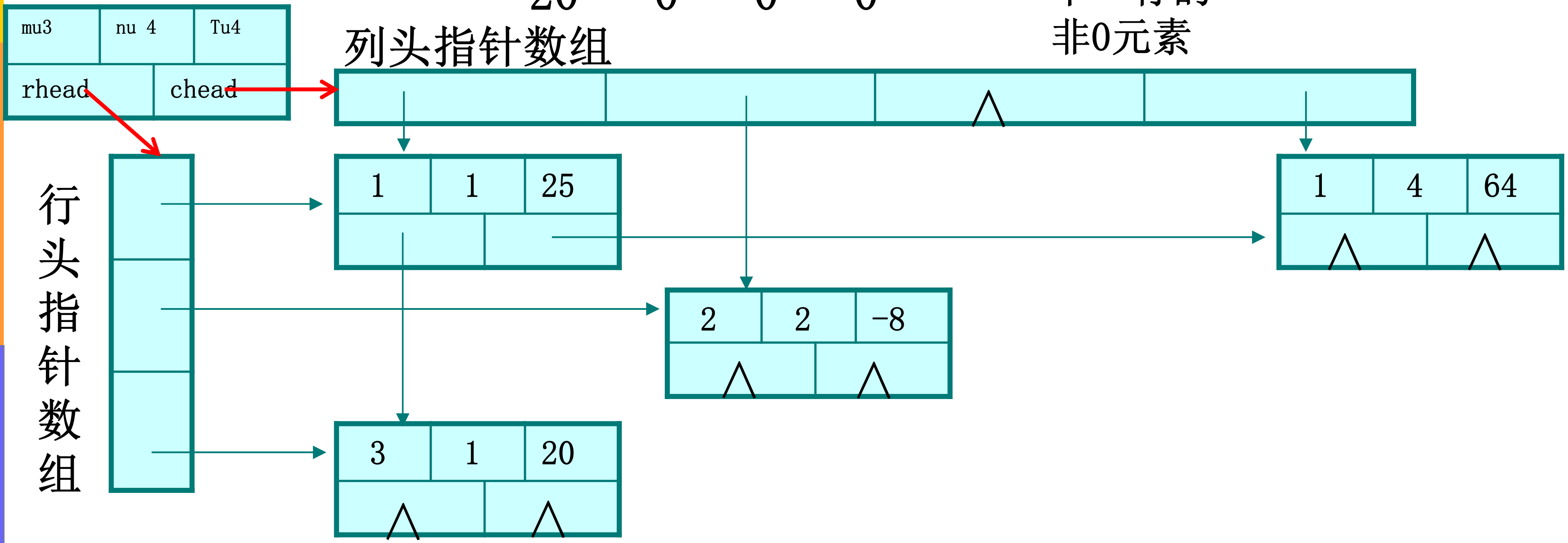
用C语言定义三元组顺序表类型

```
#define  MAXSIZE 10000
typedef struct  {
   int       i,j;//非零元行、列下标
   ElemType  e;
 } Triple;  //定义三元组

typedef struct  {
   Triple  data[MAXSIZE+1];
   int  mu,nu,tu;
 } TSMatrix; //定义三元组顺序表

TSMatrix M;
```

2.十字链接表

$$M=\begin{pmatrix} 25 & 0 & 0 & 64 \\ 0 & -8 & 0 & 0 \\ 20 & 0 & 0 & 0 \end{pmatrix}$$





用C语言定义十字链表表类型：

```
typedef struct OLNode{      //三元组结点结点定义
   int        i, j;          //非零元素行列位置
   ElemType  e;
   struct OLNode *right, *down;
} OLNode, *OLink;

typedef struct   {          //十字链表类型定义
   OLink *rhead, *chead;   //行列头指针数组
   int    mu, nu, tu;      //稀疏矩阵的行数、列数和非零元个数
} CrossList;
```

3 求转置矩阵

$$
M=\begin{pmatrix}
0 & 33 & 9 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & 0 & 36 & 0 \\
0 & 0 & 16 & 0 & 0 & 0 & 0 \\
0 & 28 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 37 & 0 & 0 & 0
\end{pmatrix}
\Rightarrow
T=\begin{pmatrix}
0 & 0 & 10 & 0 & 0 & 0 \\
33 & 0 & 0 & 0 & 28 & 0 \\
9 & 0 & 0 & 16 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 37 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 36 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

M的三元表存储结构

1	2	33
1	3	9
3	1	10
3	6	36
4	3	16
5	2	28
6	4	37
///	///	///

行数(mu)：6
列数(nu)：7
非零元(tu)：7

T的三元表存储结构

2	1	33
3	1	9
1	3	10
6	3	36
3	4	16
2	5	28
4	6	37
///	///	///

行数(mu)：7
列数(nu)：6
非零元(tu)：7

不符合以行为主的存放

求转置矩阵算法1

M的三元表存储结构

1	2	33
1	3	9
3	1	10
3	6	36
4	3	16
5	2	28
6	4	37
///	///	///

行数(mu)：6
列数(nu)：7
非零元(tu)：7

T的三元表存储结构

1	3	10
2	1	33
2	5	28
3	1	9
3	4	16
4	6	37
6	3	36
///	///	///

转置算法1：根据M的三元组顺序表得到T的三元组顺序表

```
void TransMatrix1(TSMatrix M, TSMatrix &T)
T.mu=M.nu;T.nu=M.mu; T.tu=M.tu;
if (T.tu)  {
  q=1;                                    //指示向T写时的位置
  for(col=1;col<=M.nu; ++col)  //扫描M的三元组表M.nu次
   for(p=1;p<=M.tu;++p)          //扫描M的长度为M.tu的三元表
    if(M.data[p].j==col)           //找到一个符合条件的三元组
       { T.data[q].i=M.data[p].j;
         T.data[q].j=M.data[p].i;
         T.data[q].e=M.data[p].e;
         q++; }
  }
```

时间复杂度： O(M. nu*M. tu)

求转置矩阵算法2：　改进算法

col	1	2	3	4	5	6	7
num[col]	1	2	2	1	0	1	0
cpot[col]	1	2	4	6	7	7	8

cpot[1]=1;

cpot[col]=cpot[col-1]+num[col-1]

2≤col≤M.nu

M的三元表存储结构

1	2	33
1	3	9
3	1	10
3	6	36
4	3	16
5	2	28
6	4	37
///	////	///

行数(mu)：6

列数(nu)：7

非零元(tu)：7

M的三元表存储结构

1	2	33
1	3	9
3	1	10
3	6	36
4	3	16
5	2	28
6	4	37
///	///	///

行数(mu)：6

列数(nu)：7

非零元(tu)：7

T的三元表存储结构

1	3	10
2	1	33
2	5	28
3	1	9
3	4	16
4	6	37
6	3	36
///	///	///

位置	T行号
	1
	2
	3
	4
	5
	6
	7

转置算法2：根据M的三元组顺序表得到T的三元组顺序表

```
void TransMatrix2(TSMatrix M, TSMatrix &T) {
T.mu=M.nu;T.nu=M.mu; T.tu=M.tu;
if (T.tu) {
    for(col=1;col<=M.nu;col++)   num[col]=0;
    for(t=1;t<=M.tu;t++)  ++num[M.data[t].j];
    cpot[1]=1;                                  //计算数组cpot
    for(col=2;col<=M.nu;col++) cpot[col]=cpot[col-1]+num[col-1];
  for(p=1;p<=M.tu;++p) {         //扫描M三元组表
   col=M.data[p].j;               //确定M当前元素列号
   q=cpot[col];                   //确定在T的存放位置
   T.data[q].j=M.data[p].i;
   T.data[q].i=M.data[p].j;
   T.data[q].e=M.data[p].e;
   ++cpot[col];      //修改T的当前行下一元素存放位置
   }
}
```

时间复杂度：O(M.nu+M.tu)　　空间复杂度：O(M.nu)

5.4 广义表

5.4.1广义表的定义

广义表(也称为列表)是n(n≥0)个元素的有限序列。

记作：LS=(a_1, a_2, . . . , a_n)。

其中：LS：广义表名　　　n：LS的长度

a_i(1≤i≤n)或者是数据元素，或者是广义表

通常，用大写字母表示广义表的名称，小写字母表示数据元素。

当广义表LS中的元素是一个数据元素时，称其为原子。否则称为广义表的子表。

当广义表非空时，称第一个元素a_1为LS的表头(head)，称其余的部分(a_2, . . . , a_n)为LS的表尾(tail)。

广义表举例：

(1) A=( ) A是一个空表，长度n=0。

(2) B=(e)

Head(B)=e　　Tail(B)=( )

(3) C=(a,b,c)

Head(C)=a　　Tail(C)=(b,c)

Head(Tail(C))=b　　Tail(Tail(C))=(c)

(4) D1=(a,(b,c))

Head(D1)=a　　Tail(D1)=((b,c))

D2=((a,b),c)

Head(D2)=(a,b)　　Tail(D2)=(c)

(5) E=((a,b),c,(d,e))

Head(E)=(a,b)　　Tail(E)=(c,(d,e))

Head(Tail(E))=c　　Tail(Tail(E))=((d,e))

任何一个非空的广义表的表头可能是原子、也可能是广义表；表尾一定是广义表。

(6) F1=(A, B, C, d)=(( ), (e), (a, b, c), d)

Head(F1)=( )

Tail(F1)=((e), (a, b, c), d)

广义表允许共享子表

广义表允许递归

(7) G=(a, G)=(a, (a, (a, ……)))

Head(G)=a

Tail(G)=(G)=((a, G))

(8) H=(( ), (( ), ( )))

Head(H)=( )

Tail(H)=((( ), ( )))

5.4.2 广义表的图型表示

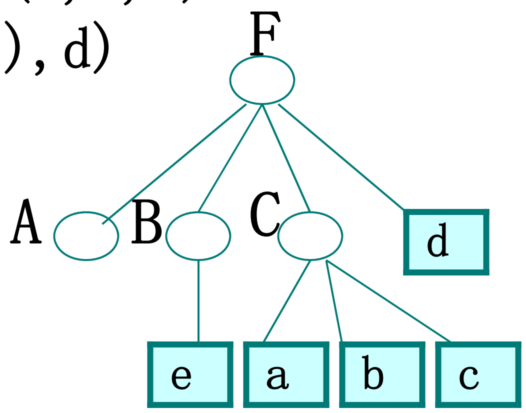
约定　□-----原子

　　　○-----列表,若有表名,附表名

例　(1) A=( )　　(2) B=(e)　　(3) C=(a,b,c)

　　(4) F=(A,B,C,d)=(( ),(e),(a,b,c),d)

　　F可以表示成多层次的图形结构



5.4.3 广义表的基本操作

1. InitGLsit(&L)　　　　创建一个空的广义表
2. CreateGLsit(&L，S)　　根据S的定义创建一个广义表
3. DestroyGlist(&L)　　　销毁一个广义表
4. GListLength(L)

　　求广义表的长度

　　a=( )　　　　　　　GListLength(A)=0

　　G=(a,G)　　　　　　GListLength(G)=2

　　H=(( ),(( ),( )))　GListLength(H)=2

　　F=(A,B,C,d)　GListLength(F)=4

5.GetHead(L)　　　求广义表的头

G=(a,G)　　　　GetHead(G)=a

E=((a,b),c,(d,e))　　GetHead(E)=(a,b)

6.GetTail(L) 求广义表的尾

G=(a,G)　　　　Tail(G)=(G)=((a,G))

E=((a,b),c,(d,e))　　Tail(E)=(c,(d,e))

7.GListDepth(L)　　求广义表深度

A=( )　　　　GListDepth(A)=1

E=((a,b),c,(d,e))　　GListDepth(E)=2

H=(( ),(( ),( )))　　GListDepth(H)=3

……

5.5 广义表的存储结构

广义表的的元素具有不同结构，一般用链式存储结构。

由于广义表中的元素既可以是原子，也可以是广义表，所以会有原子结点和列表结点。

原子结点:

tag=0	atom(元素)

列表结点:

tag=1	hp(表头)	tp(表尾)

原子结点：只有2个域，标志域与值域

列表结点：有3个域，标志域、表头指针域与表尾指针域

由于一个非空广义表被分成表头和表尾，所以结点的表头和表尾指针可唯一的确定一个广义表

原子结点:

tag=0	atom(元素)

列表结点:

tag=1	hp(表头)	tp(表尾)

这种链式存储结构结构中，为了统一管理这2类结点，采用了共用体（联合）来定义广义表的结点类型。

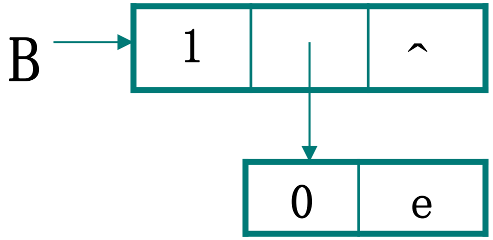
```
typedef  struct GLNode{
      ElemTag  tag;   //标志域，用以区分原子结点和表结点
      union { AtomType   atom;        //原子结点
              struct { struct GLNode    *hp,*tp;
                     }  ptr;          //表结点
            }
      } *GList;
```

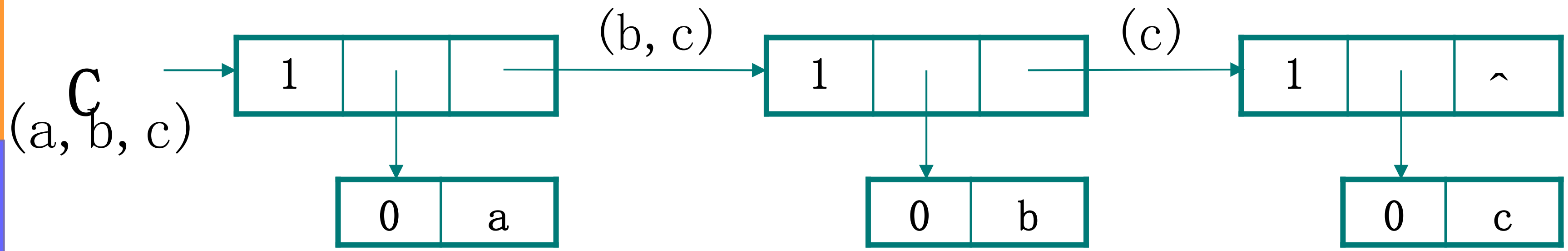
(1) A=( )

(2) B=(e)

(3) C=(a, b, c)

A=NULL





广义表的存储结构示例

5.6 广义表的递归算法

(1) 求广义表的长度： int GLisitLength(Glist L)

递归定义为
$$
\begin{cases}
0 & L==NULL \\
1+GLisitLength(L\text{->}ptr.tp) & L!=NULL
\end{cases}
$$

```
int GLisitLength(Glist L)
  {
  if (!L) return 0;
  return 1+GLisitLength(L->ptr.tp);
  }
```

(2) 求广义表的深度： int GLisitDepth(Glist L)

广义表： LS=(a_1, a_2, . . . , a_n)

$$
\text{递归定义为} \begin{cases} 1 & \text{L==NULL （空表）} \\ 0 & \text{L->tag==0（原子结点）} \\ 1+\mathrm{Max}(a_i) & \text{L!=NULL （非空表）} \end{cases}
$$

```
int GLisitDepth(Glist L){
  if (!L) return 1;
  if (L->tag==0) return 0;
  for (max=0,p=L; p; p=p->ptr.tp) {
       dep=GLisitDepth(p->ptr.hp);
       if (dep>max)  max=dep;
       }
  return max+1;
  }
```