

# 数据结构

# Data Structure

2017年秋季学期  
刘鹏远

# 数组与广义表

数组和广义表是线性表的扩展：宏观上可视为一种特殊的线性表

① 元素的值并非原子类型，可以再分解，表中元素也是一个线性表（即广义的线性表）。

② 所有数据元素仍属同一数据类型。

## 1 数组的定义

2 数组的顺序表示和实现

3 矩阵的压缩存储(即数组的应用)

4 广义表的定义与存储结构

5 广义表其他（\*\*，见教材）

# 数组的定义

**数组：** 由一组名字相同、下标不同的相邻变量构成。

- ① 数组中各元素具有统一的类型；
- ② 数组元素的下标一般具有固定的上界和下界；
- ③ 数组的基本操作比较简单，除了结构的初始化和销毁之外，一般只提供存取元素和修改元素值的操作。

本章所讨论的数组与高级语言中的数组有所区别：  
高级语言中的数组是顺序结构；  
本章的数组既可以是顺序的，也可以是链式结构；  
基本操作不同。

一维数组：线性结构， $n$ 个相同类型元素连续组成的有限序列，限定元素为原子类型（或集合）。

二维数组可以视为其每一个数组元素为一维数组的一维数组，因此为一种特殊的线性结构

（但从微观来看，每一个数组元素同时处于两个向量（行、列），它可能有两个直接前驱，有两个直接后继，表现为非线性结构）

N维数组的特点：

$n$ 个下标，每个元素受到 $n$ 个关系约束。

但同时，一个 $n$ 维数组可以看成是由若干个 $n-1$ 维数组组成的线性表。

根据定义与基本操作限制，数组元素的下标一般具有固定的下界和上界，因此它比其他复杂的非线性结构简单。

参见教材P90

# 抽象数据类型

ADT Array {

数据对象:  $D = \{a_{j_1 j_2 \dots j_n} \mid \text{其中: } a_{j_1 j_2 \dots j_n} \in \text{ElemSet},$

$j_i$  为数据元素第  $i$  维的下标,  $0 \leq j_i \leq b_i - 1$ ,

$b_i$  是数组第  $i$  维的长度, 即维界,

$i = 1, 2, \dots, n$ ,  $n$  为数组的维数,  $n > 0$

}

一维数组  $[1, 2, 3, 4, 5]$ , 记:  $A = (1, 2, 3, 4, 5)$ ;

二维数组  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ , 记:  $B = ((1, 2, 3), (4, 5, 6))$ ;

或:  $B = ((1, 4), (2, 5), (3, 6))$ 。

# 抽象数据类型

数据关系:  $R = \{R_1, R_2, \dots, R_n\}$

$$R_i = \{ \langle a_{j_1 \dots j_i \dots j_n}, a_{j_1 \dots j_i+1 \dots j_n} \rangle$$

$$| a_{j_1 \dots j_i \dots j_n}, a_{j_1 \dots j_i+1 \dots j_n} \in D,$$

$$0 \leq j_i \leq b_i - 2, \quad 0 \leq j_k \leq b_k - 1, \quad 1 \leq k \leq n, \quad k \neq i \}$$



# 抽象数据类型

基本操作（一般为以下四种）

$\text{InitArray}(\&A, n, \text{bound}_1, \dots, \text{bound}_n)$

操作结果：若给定的维数和各维长度值合法，则构造数组A

$\text{DestroyArray}(\&A)$

操作结果：销毁数组A

$\text{Value}(A, \&e, \text{index}_1, \dots, \text{index}_n)$

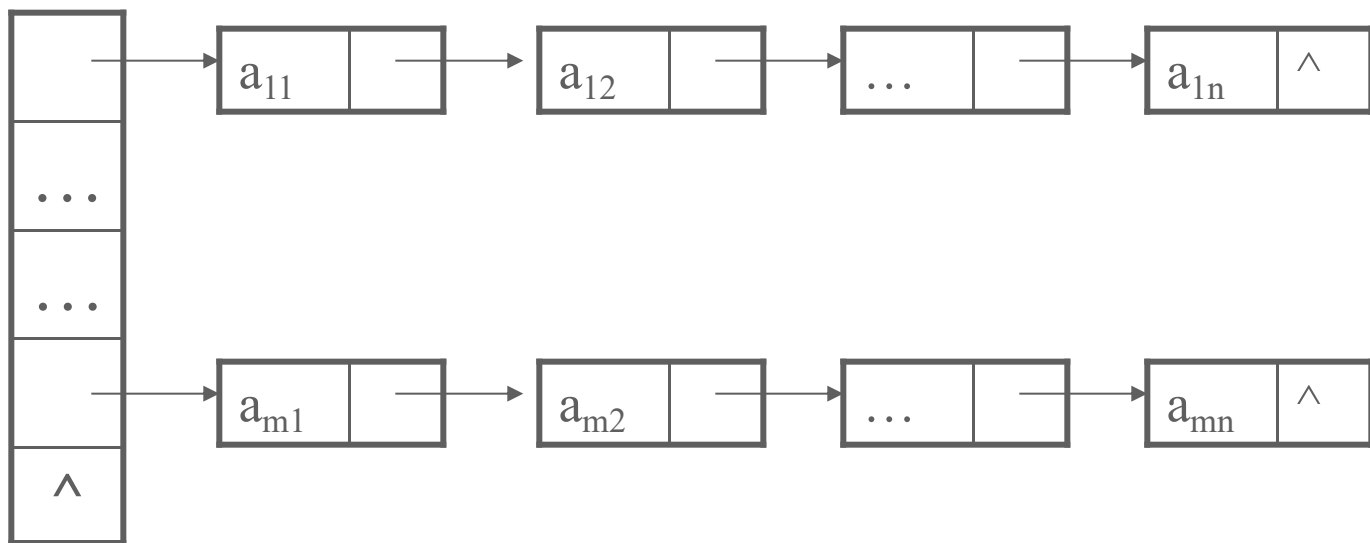
操作结果：若各给定下标值不越界，则由 e 返回相应元素值

$\text{Assign}(\&A, e, \text{index}_1, \dots, \text{index}_n)$

操作结果：若各给定下标值不越界，则将 e 值赋给指定元素

链式存储方式：

用带行指针向量的单链表来表示。  
(非随机存储)



## 特殊情况---二维数组的顺序结构

### 静态结构定义

```
typedef struct{
```

```
int m, n;
```

//实际行数和列数

```
elem_type A[Max_m][Max_n];
```

```
}array_static2d; //静态结构定义
```

或者直接: `int m=xxx,n=yyy; elem_type A[m][n];`

静态定义的二维数组不能扩充（其实一般也不扩充多维数组），在定义时要充分考虑各维的上、下界。

## 顺序存储动态结构定义

```
typedef struct{  
    int m, n;                //行数和列数  
    ElemType **A;           //动态指针定义  
}array_dyn2d;
```

初始化时候，给定m,n，然后用动态存储分配建立的二维数组。

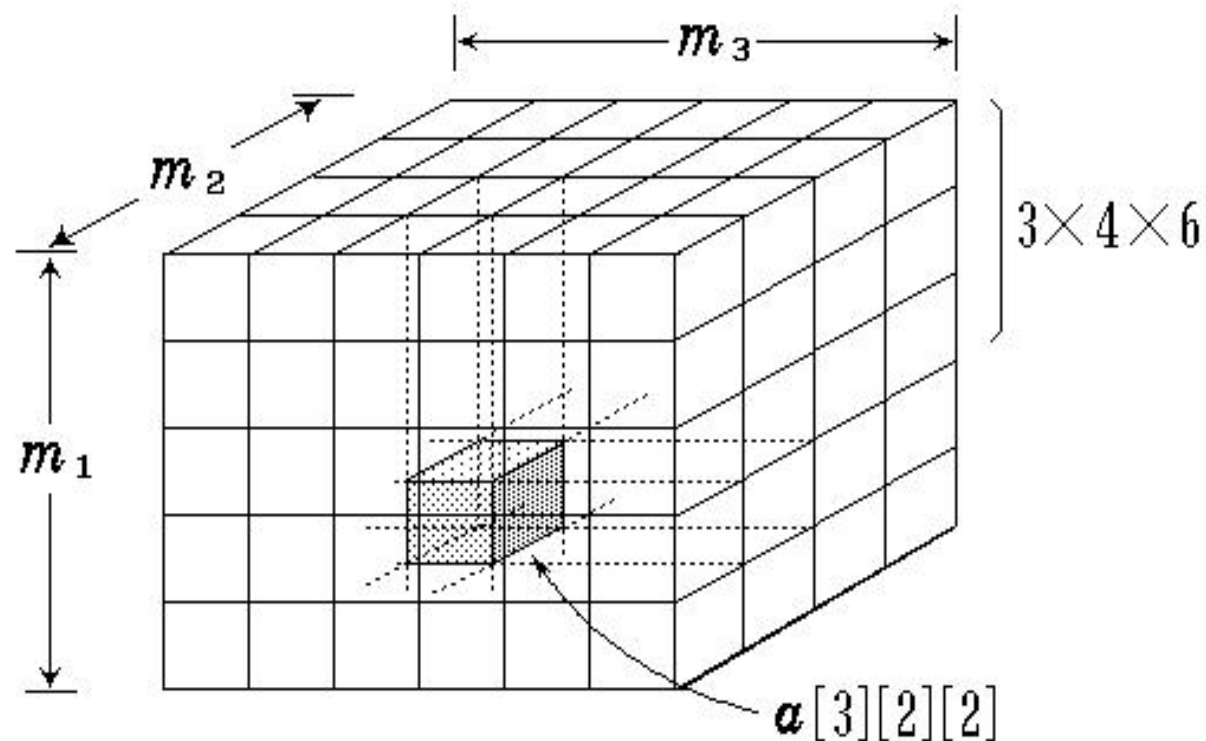
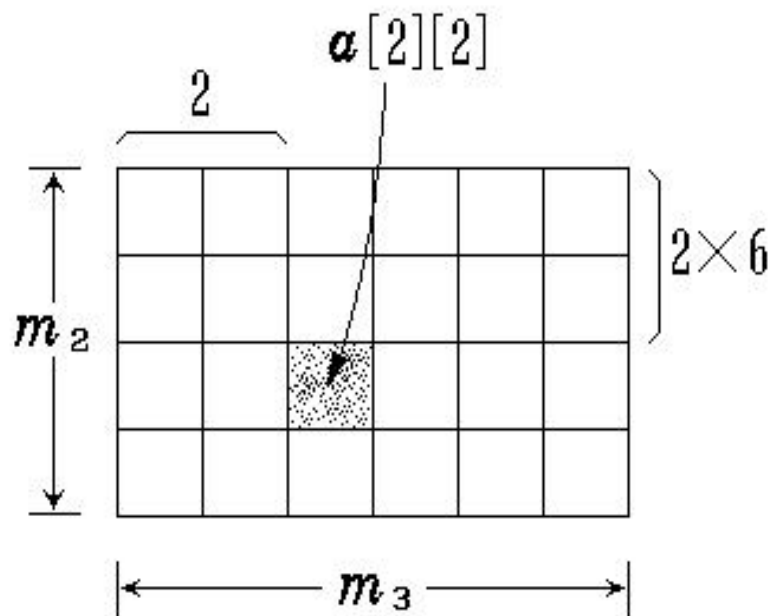
```
A = (int**)malloc(sizeof(int*)*m);  
for ( i = 0; i < m; i++) A[i]=(int*)malloc(sizeof(int)*n);
```

动态回收也需要分两步：

```
for ( i = 0; i < m; i++) free(A[i]); free(A);
```

# 一般情况

$$m_1=5 \quad m_2=4 \quad m_3=6$$



行向量 下标  $i$

页向量 下标  $i$

列向量 下标  $j$

行向量 下标  $j$  列向量 下标  $k$

思考：计算机中内存如何存放....."立方体？"

## 列/行主序

### 存储结构设计

用一组地址连续的存储单元依次存储数组中数据元素，存储时需考虑各维的顺序问题。

设数组A=

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

2000

2002

2004

2006

2008

2010

|   |   |          |
|---|---|----------|
|   |   |          |
|   |   |          |
| 1 | 1 | } 2 byte |
| 4 | 2 |          |
| 2 | 3 |          |
| 5 | 4 |          |
| 3 | 5 |          |
| 6 | 6 |          |
|   |   |          |
|   |   |          |

**问题：**如何按给定的下标，计算对应数组元素的存储地址。

|        |     |        |     |          |
|--------|-----|--------|-----|----------|
| 0, 0   | ... | 0, j   | ... | 0, n-1   |
| 1, 0   | ... | 1, j   | ... | 1, n-1   |
| ...    | ... | ...    | ... | ...      |
| i, 0   | ... | i, j   | ... | i, n-1   |
| ...    | ... | ...    | ... | ...      |
| m-1, 0 | ... | m-1, j | ... | m-1, n-1 |

行优先存放（以行为主序，C语用等采用）：

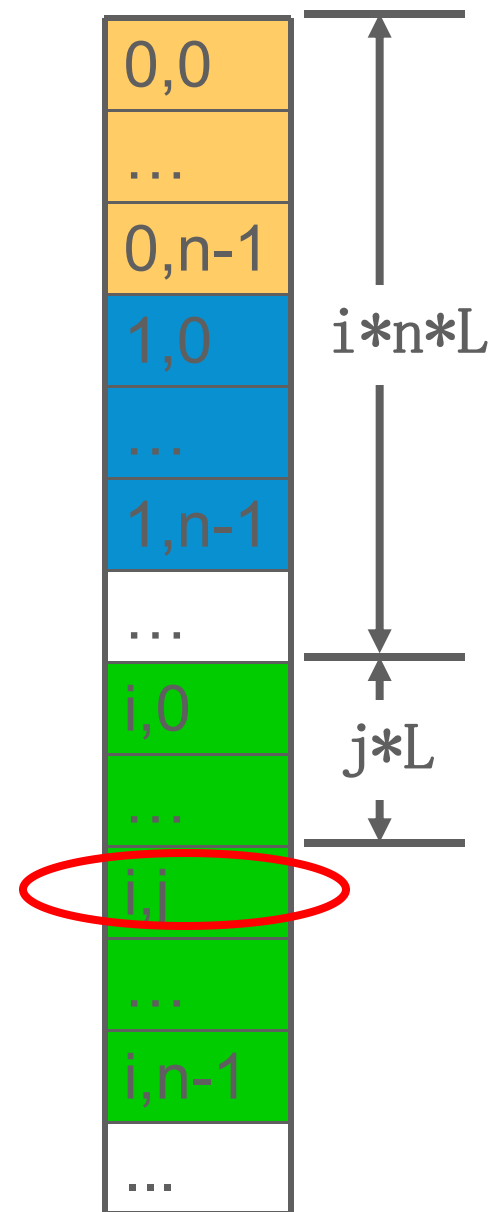
数组开始存放位置为 $\text{LOC}(a[0][0])$ ，每个元素占用  $l$  个存储单元，则 $a[i][j]$ 的存储位置为：

$$\text{LOC}(a[i][j]) = \text{LOC}(a[0][0]) + (i*n + j)*l$$

其中， $n$  是每行元素个数，即列数。

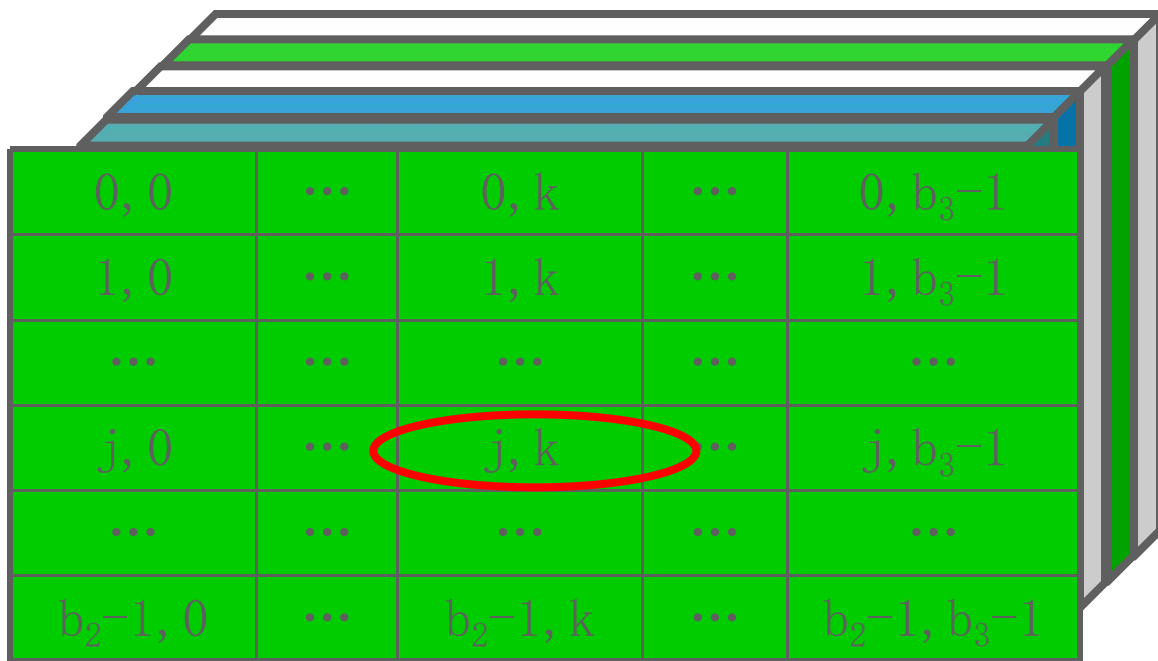
列优先（以列为主序，Fortran语言等）：

$$\text{LOC}(a[i][j]) = \text{LOC}(a[0][0]) + (j*m + i)*l$$

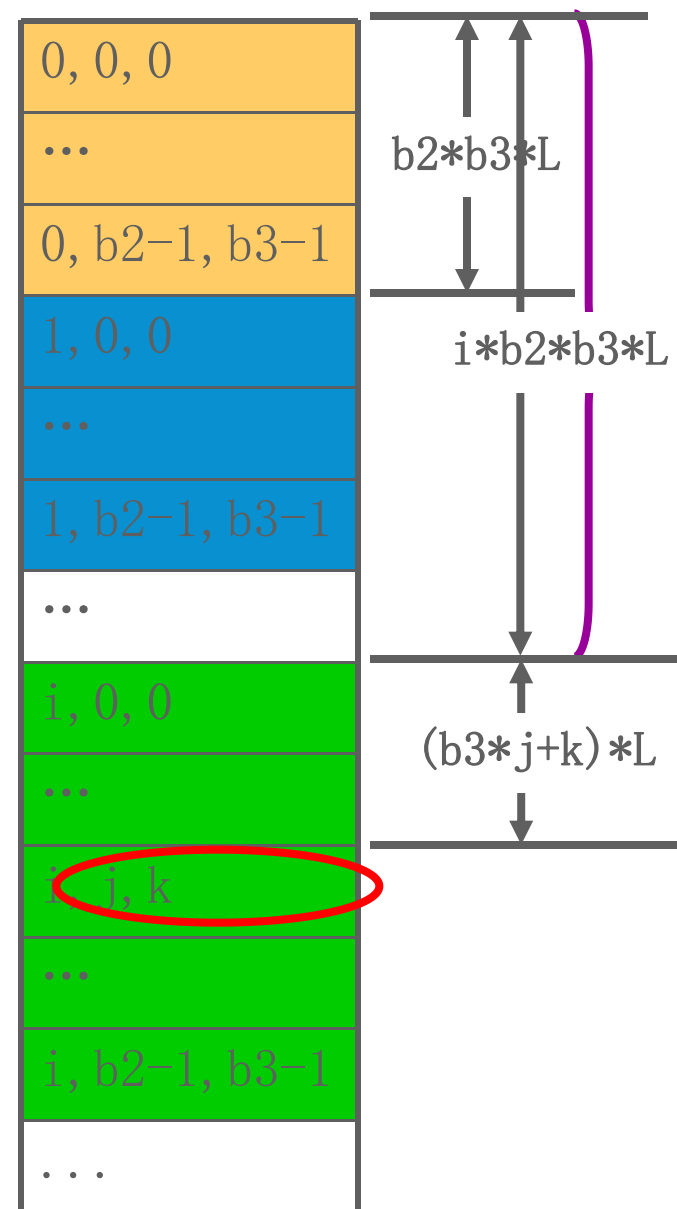


假设各维维度为  $b_1, b_2, b_3$ , 每个元素占据  $L$  个存储单元, 按页/行/列方式存放, 那么, 下标为  $i, j, k$  的数组元素的存储地址为:  $LOC(a[i][j][k]) = LOC(a[0][0][0]) + (i*b_2*b_3 + j*b_3 + k) * L$

更高维度可类似处理



$LOC(0, 0, 0)$





设  $n$  维数组  $A_{b_1 \times b_2 \times \dots \times b_n}$ ，则：

$$\begin{aligned} \text{LOC}(j_1, j_2, \dots, j_n) \\ &= \text{LOC}(0, 0, \dots, 0) + \\ &\quad j_1 * b_2 * \dots * b_n * L + j_2 * b_3 * \dots * b_n * L + \dots + j_{n-1} * b_n * L + j_n * L \\ &= \text{LOC}(0, 0, \dots, 0) + \sum_{i=1}^n (c_i * j_i) \end{aligned}$$

其中， $c_n = L$ ， $c_{i-1} = c_i * b_i$ ， $1 < i \leq n$ 。与  $j$  无关，一旦各维度确定，则  $c_i$  确定，因此称为求址常量。因此计算任意元素实际存储位置的时间相同，随机存取。

## 多维数组的顺序存储---一般情况

```
#define MAX_ARRAY_DIM 8
```

```
typedef struct{  
    elem_type *base;    //数组元素基址  
    int        dim;      //数组维数  
    int        *bound;   //数组各维长度信息保存区基址  
    int        *constants;  
    //常量Ci的基址, 提高存取效率。可设为1。  
}Array;
```

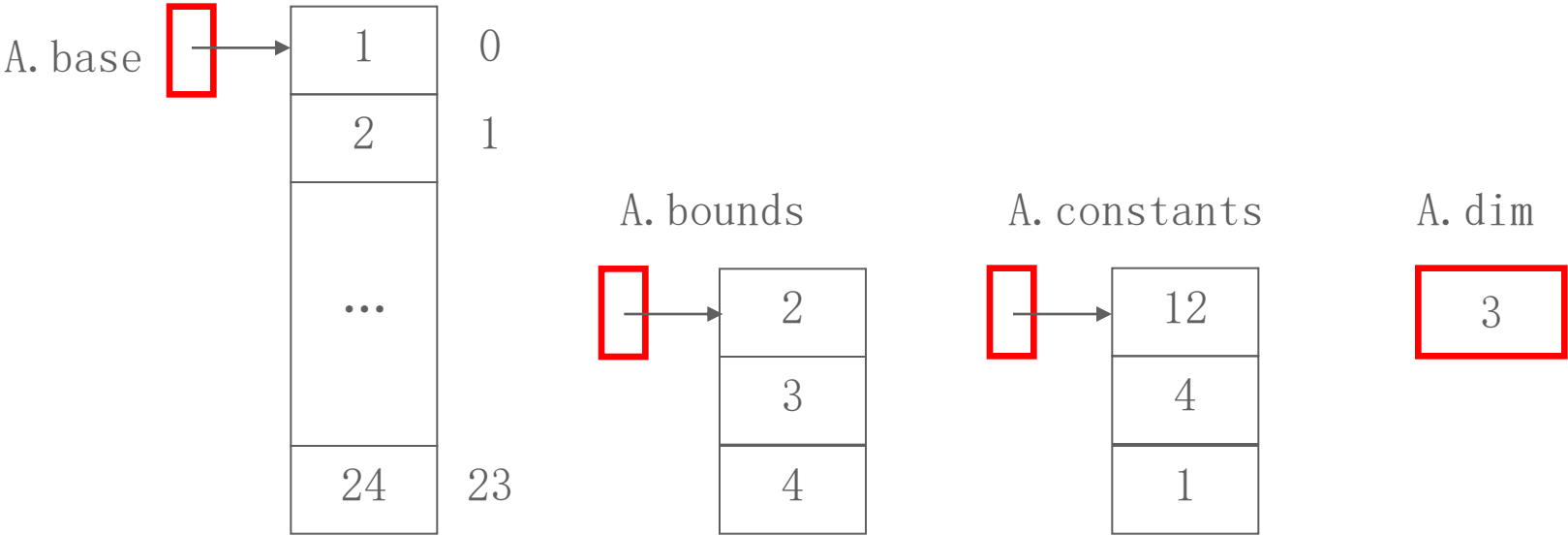
操作等可参考教材P93

# 多维数组的顺序存储---一般情况

送分题： 设三维数组

$A = (((1, 2, 3, 4), (5, 6, 7, 8), (9, 10, 11, 12)),$

$((13, 14, 15, 16), (17, 18, 19, 20), (21, 22, 23, 24)))$  按行序存放。画图。



# 矩阵的压缩存储

编程时通常用二维数组保存矩阵，但有时会浪费空间。

【例】  $A = \begin{bmatrix} 1 & 2 & 7 & 3 & 5 \\ 2 & 9 & 4 & 0 & 6 \\ 7 & 4 & 6 & 3 & 7 \\ 3 & 0 & 3 & 2 & 8 \\ 5 & 6 & 7 & 8 & 5 \end{bmatrix}$        $B = \begin{bmatrix} 0 & 0 & 0 & 9 & 0 \\ 2 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 3 & 0 \\ 3 & 0 & 8 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}$

【问题】对值相同的元素或零元素分布有规律的矩阵，如何存储才既能节省空间，又能保证矩阵运算的正常进行。

## 重点

多维空间的元素压缩到一维空间，元素前后的对应关系（有规律）

如果能找到这种对应关系，那么对任意元素，存取时间相同，其顺序表示就是随机存取结构

# 矩阵的压缩存储

## 1. 什么是压缩存储？

若多个数据元素的值都相同，则只分配一个元素值的存储空间，且一般零元素不占存储空间。

## 2. 什么样的矩阵具备压缩条件？

阶数较高的：

特殊矩阵(对称阵，对角阵，三角阵):元素分布有规律和稀疏矩阵。

## 3. 什么叫稀疏矩阵？

矩阵中非零元素的个数少且元素分布无规律（一般小于5%）

# 对称阵

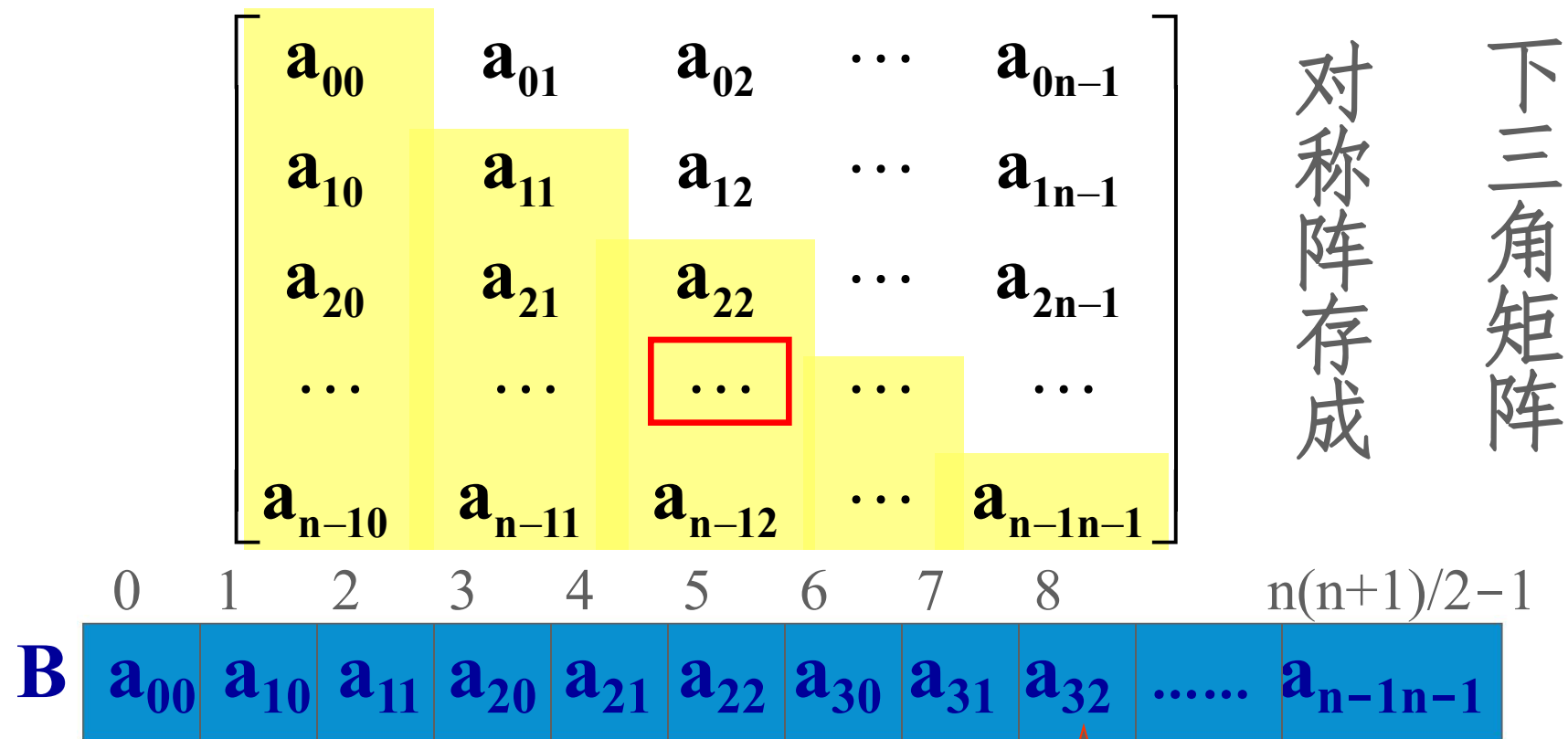
☞ 定义若 **n阶方阵**  $A$  有  $a_{ij}=a_{ji}$   $i, j \in [0, n-1]$ ，则称  $A$  为  $n$  阶对称阵。

【例】  $A=$

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 2 | 7 | 3 | 5 |
| 2 | 9 | 4 | 0 | 6 |
| 7 | 4 | 6 | 3 | 7 |
| 3 | 0 | 3 | 2 | 8 |
| 5 | 6 | 7 | 8 | 5 |

存储结构：用一组地址连续的存储单元按**行**或列序保存其上/下三角（含对角线）中的数据元素，共  $n(n+1)/2$  个

以后仅以行序为例，列序可自行推导



若  $i \geq j$ , 数组元素  $A[i][j]$  在数组  $B$  中的存放位置为

$$1 + 2 + \cdots + i + j = (i + 1) * i / 2 + j$$

前  $i$  行元素总数 第  $i$  行第  $j$  个元素前元素个数



矩阵元素在一维数组的**定位**:

若  $i \geq j$ , 数组元素  $A[i][j]$  在数组  $B$  中的存放位置为

$$A[i][j] \leftarrow (i + 1) * i / 2 + j$$

若  $i < j$ , 数组元素  $A[i][j]$  在矩阵的上三角部分, 定位在数组  $B$  中存放的对称元素

$$A[j][i] \leftarrow (j + 1) * j / 2 + i$$

一维数组元素在矩阵中的定位：

若已知某矩阵元素位于数组 B 的第 k 个位置，元素一定位于 i,i+1 两行之间，可寻找满足：

$i(i+1)/2 \leq k < (i+1)(i+2)/2$  的 i, 为该元素行号。

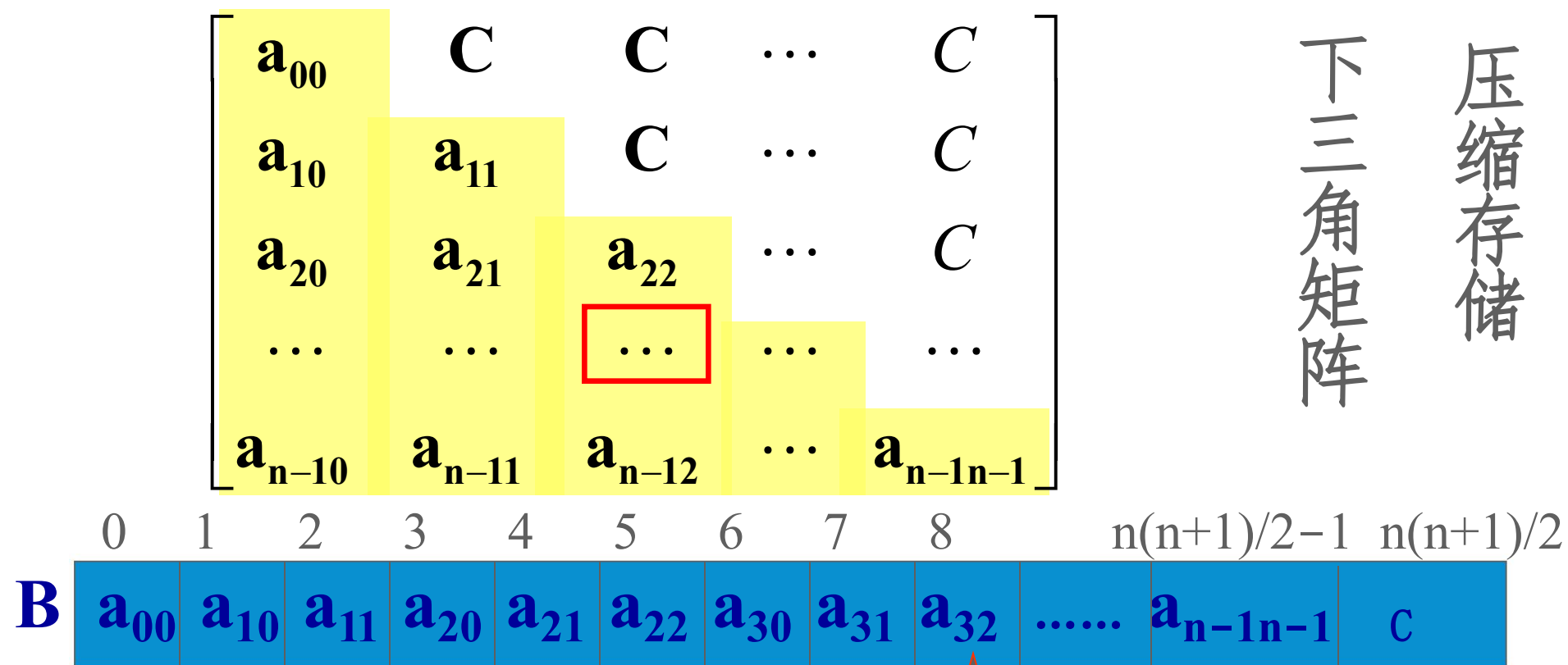
$j = k - i(i+1)/2$  即为该元素的列号。

例，在数组B中存放位置为  $k = 8$ , 求其在矩阵中的位置：

$3*4/2 = 6 \leq k < 4*5/2 = 10$ , 取  $i = 3$ 。

则  $j = 8 - 3*4/2 = 2$ 。

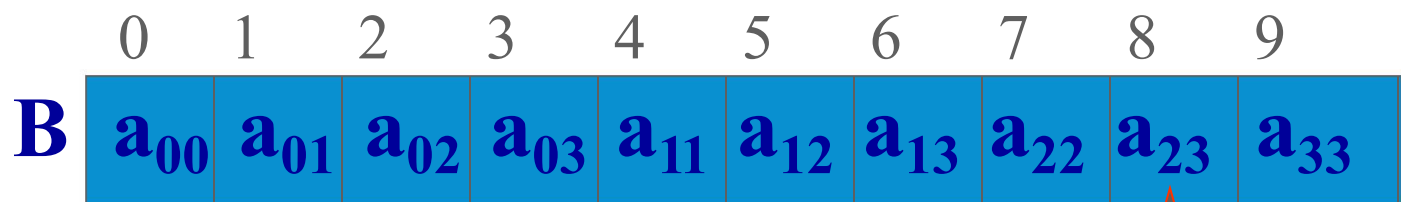
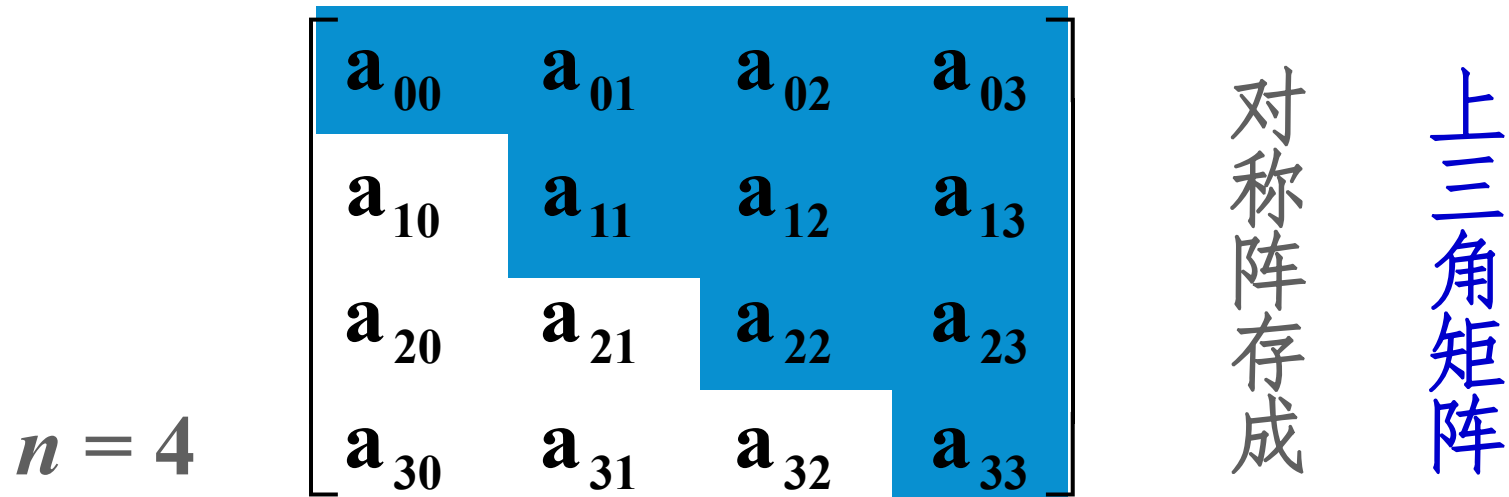
得到位置8的元素在矩阵中的位置为 (3,2)



若  $i \geq j$ , 数组元素  $A[i][j]$  在数组  $B$  中的存放位置为

$$1 + 2 + \dots + i + j = (i + 1) * i / 2 + j$$

对  $i < j$ , 所有元素为常数  $C$ , 存在  $n(n+1)/2$  位置



若  $i \leq j$ ，数组元素  $A[i][j]$  在数组  $B$  中的存放位置为

$$n + (n-1) + (n-2) + \cdots + (n-i+1) + j - i$$

前  $i$  行元素总数

第  $i$  行第  $j$  个元素前元素个数

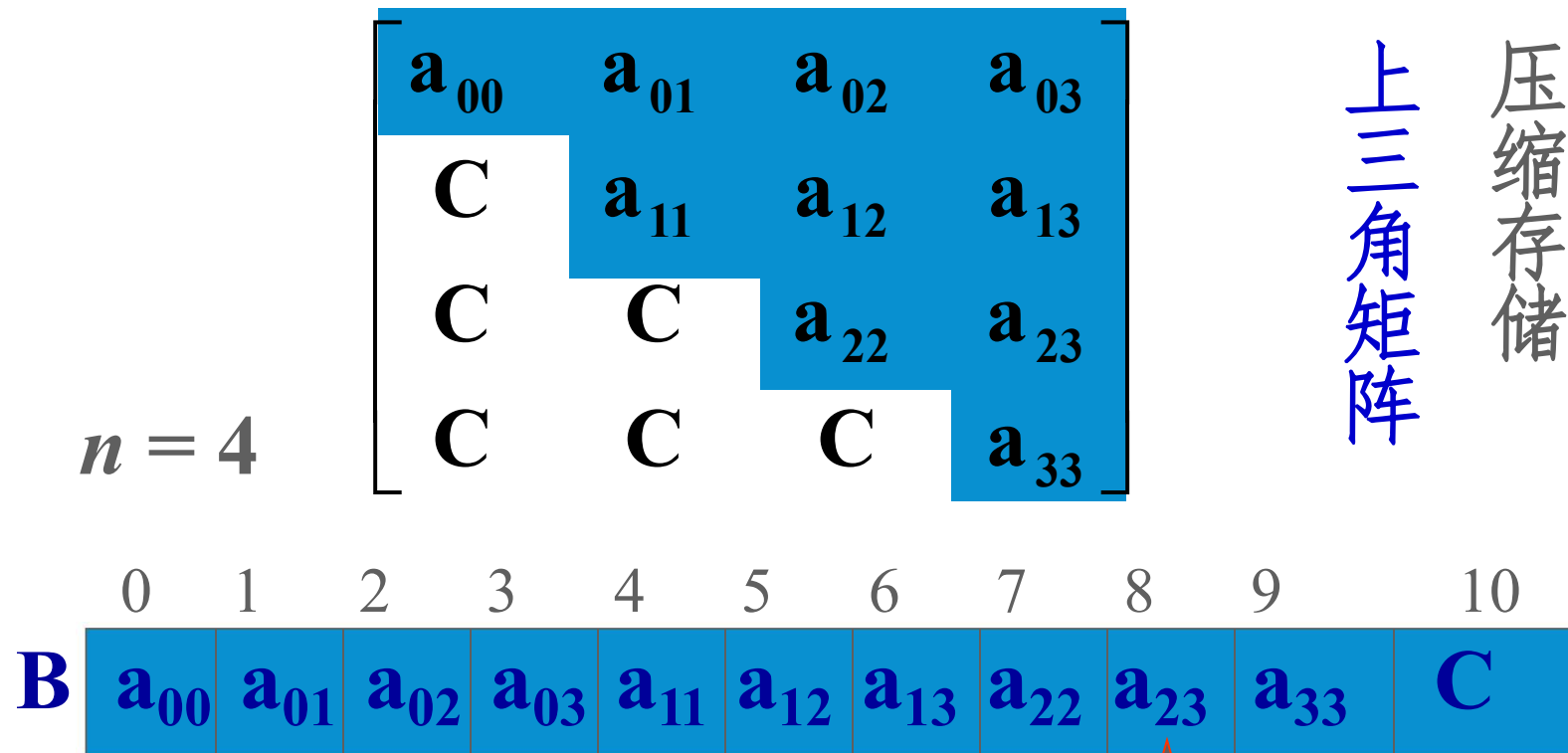
若  $i \leq j$ ，数组元素  $A[i][j]$  在数组  $B$  中的存放位置为

$$\begin{aligned} & n + (n-1) + (n-2) + \cdots + (n-i+1) + j - i = \\ & = (2*n-i+1) * i / 2 + j - i = \\ & = (2*n-i-1) * i / 2 + j \end{aligned}$$

若  $i > j$ ，数组元素  $A[i][j]$  在矩阵的下三角部分，找它的对称元素  $A[j][i]$ 。

$A[j][i]$  在数组  $B$  的第  $(2*n-j-1) * j / 2 + i$  的位置中找到

(一般不用记，推导上式，然后换  $i, j$  得到下式)



若  $i \leq j$ , 数组元素  $A[i][j]$  在数组 B 中的存放位置为

$$n + \underbrace{(n-1) + (n-2) + \cdots + (n-i+1)}_{\text{前 } i \text{ 行元素总数}} + \underbrace{j-i}_{\text{第 } i \text{ 行第 } j \text{ 个元素前元素个数}}$$

前  $i$  行元素总数

第  $i$  行第  $j$  个元素前元素个数

$i > j$ , 位置均在  $n(n+1)/2$ , 且值=C

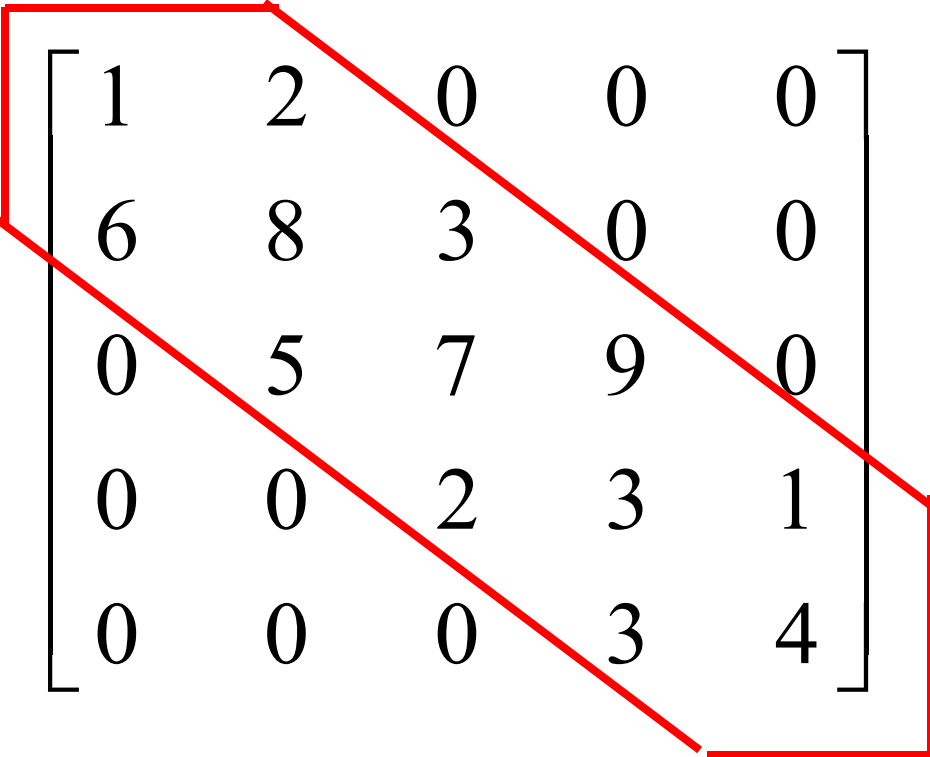
## 三对角矩阵的压缩存储

三对角矩阵中除主对角线及在主对角线上下最临近的两条对角线上的元素外，所有其它元素均为0。

总共有 $3n-2$ 个非零元素。在三条对角线上的元素 $a_{ij}$ 满足

$$0 \leq i \leq n-1 \quad (\text{此条必然})$$

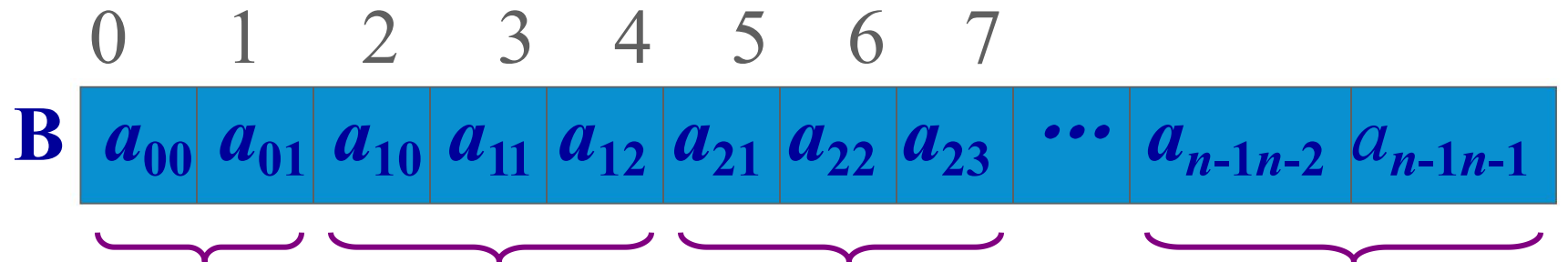
$$i-1 \leq j \leq i+1 \quad (\text{因中间是 } i=j)$$


$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 6 & 8 & 3 & 0 & 0 \\ 0 & 5 & 7 & 9 & 0 \\ 0 & 0 & 2 & 3 & 1 \\ 0 & 0 & 0 & 3 & 4 \end{bmatrix}$$

将三对角矩阵A中三条对角线上的元素按行存放在一维数组B中，且  $a_{00}$  存放于B[0]。

在一维数组B中， $A[i][j]$  在第  $i$  行，它前面有  $3*i-1$  个非零元素，在本行中第  $j$  列前面有  $j-i+1$  个，所以元素  $A[i][j]$  在B中位置为  $k = 2*i + j$ 。

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 6 & 8 & 3 & 0 & 0 \\ 0 & 5 & 7 & 9 & 0 \\ 0 & 0 & 2 & 3 & 1 \\ 0 & 0 & 0 & 3 & 4 \end{bmatrix}$$





若已知三对角矩阵中某元素  $A[i][j]$  在数组  $B[ ]$  存放于第  $k$  个位置，则有

$$i = \lfloor (k + 1) / 3 \rfloor$$

$$j = k - 2 * i$$

例如，当  $k = 8$  时，

$$i = \lfloor (8+1) / 3 \rfloor = 3, j = 8 - 2*3 = 2$$

当  $k = 10$  时，

$$i = \lfloor (10+1) / 3 \rfloor = 3, j = 10 - 2*3 = 4$$

# 稀疏矩阵的压缩存储

稀疏矩阵(**Sparse Matrix**): 对于稀疏矩阵, 目前还没有一个确切的定义。设矩阵A是一个 $n \times m$ 的矩阵, 其中有s个非零元素, 设  $\delta = s / (n \times m)$ , 称 $\delta$ 为稀疏因子, 如果某一矩阵的稀疏因子 $\delta$ 满足 $\delta \leq 0.05$ 时称为稀疏矩阵, 如图所示。

稀疏矩阵示例A

$$A = \begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 24 & 0 & 0 & 2 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 & 0 & 0 \end{pmatrix}$$

# 稀疏矩阵的压缩存储

## 问题:

如果只存储稀疏矩阵中的非零元素，那这些元素的位置信息该如何表示？

## 解决思路:

对每个非零元素增开若干存储单元，例如存放其所在的行号和列号，便可准确反映该元素所在位置。

## 实现方法:

将每个非零元素用一个三元组  $(i, j, a_{ij})$  来表示，则稀疏矩阵的每一个元素可用一个三元组表来表示。

三元素组表中的每个结点对应于稀疏矩阵的一个非零元素，它包含有三个数据项，分别表示该元素的\_\_\_\_行下标、\_\_\_\_列下标\_\_\_\_和\_\_\_\_元素值\_\_\_\_。

右图所示稀疏矩阵的压缩存储形式。

$$\begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 \end{pmatrix}$$

元素可用三元组表表示如下：

$((1, 2, 12), (1, 3, 9), (3, 1, -3), (3, 5, 14),$   
 $(4, 3, 24), (5, 2, 18), (6, 1, 15), (6, 4, -7))$

元素用三元组顺序表示：

|   | i | j | value |
|---|---|---|-------|
| 1 | 1 | 2 | 12    |
| 2 | 1 | 3 | 9     |
| 3 | 3 | 1 | -3    |
| 4 | 3 | 5 | 14    |
| 5 | 4 | 3 | 24    |
| 6 | 5 | 2 | 18    |
| 7 | 6 | 1 | 15    |
| 8 | 6 | 4 | -7    |

|    |    |    |    |    |   |
|----|----|----|----|----|---|
| 0  | 12 | 9  | 0  | 0  | 0 |
| 0  | 0  | 0  | 0  | 0  | 0 |
| -3 | 0  | 0  | 0  | 14 | 0 |
| 0  | 0  | 24 | 0  | 0  | 0 |
| 0  | 18 | 0  | 0  | 0  | 0 |
| 15 | 0  | 0  | -7 | 0  | 0 |

稀疏矩阵压缩存储的缺点：

失去随机存取功能 :- (

# 三元组顺序表示

若以行序为主序，稀疏矩阵中所有非0元素的三元组，就可以得构成该稀疏矩阵的一个**三元组顺序表**。相应的数据结构定义如下：

```
#define MAX_SIZE 4096
```

```
typedef struct
```

```
{
```

```
    int row ;    /* 行下标 */
```

```
    int col ;    /* 列下标 */
```

```
    elem_type value; /* 元素值 */
```

```
}Triple ;
```

```
Triple data[MAX_SIZE] ; //这样就可以存储稀疏矩阵了嘛？
```

# 三元组顺序表

- 1、无法知道矩阵究竟多大
- 2、不知道有几个非零元素

因此：为可靠描述，再加一行“**总体**”信息：即**总行数**、**总列数**，再加上**非零元素总个数**

|   | i | j | value |
|---|---|---|-------|
| 0 | 6 | 6 | 8     |
| 1 | 1 | 2 | 12    |
| 2 | 1 | 3 | 9     |
| 3 | 3 | 1 | -3    |
| 4 | 3 | 5 | 14    |
| 5 | 4 | 3 | 24    |
| 6 | 5 | 2 | 18    |
| 7 | 6 | 1 | 15    |
| 8 | 6 | 4 | -7    |

# 三元组顺序表

在前面定义的基础上，

```
typedef struct
{
    int rn;      /* 行数 */
    int cn;      /* 列数 */
    int tn;      /* 非0元素个数 */
    Triple data[MAX_SIZE];
}triple_matrix;
```

```
typedef struct
{
    int row;
    int col;
    elem_type value;
}Triple;
```



# 期中考试

时间： 下午4： 00

地点： 主北253

方式： 现场编程

题目： 应用、 改错、 操作

下次课程进度：

稀疏矩阵转置  
广义表表示