



Data Structures

字符串和多维数组 String & Matrices

2024 年 10 月8日

学而不厌 诲人不倦

- ➡ 4.1 引言
- ➡ 4.2 字符串
- ➡ **4.3 多维数组**
- ➡ **4.4 矩阵的压缩存储**
- ➡ 4.5 扩展与提高
- ➡ 4.6 应用举例

4.3 多维数组

4-3-1 数组的逻辑结构



1. 数组的定义

➤ 数组是 n ($n \geq 0$) 个相同数据类型数据元素构成的有限序列。

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \mathbf{a_{22}} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

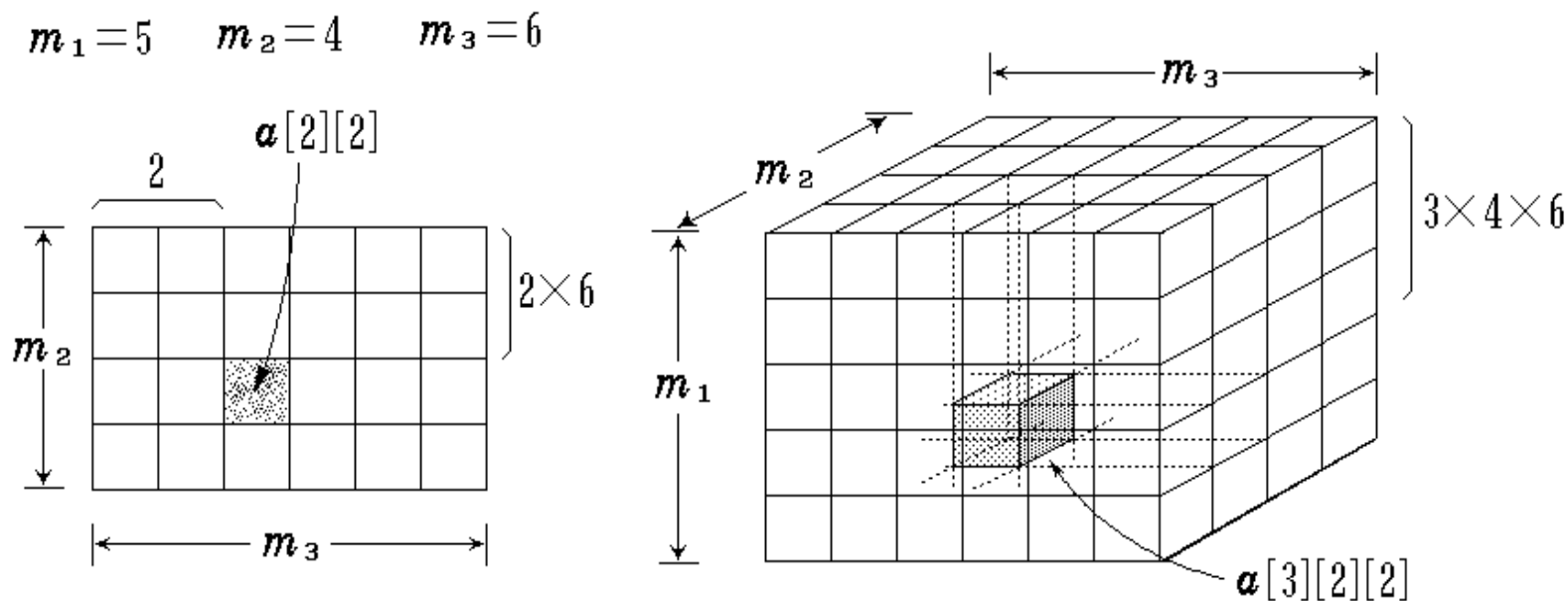
- 数组中的数据元素数目固定;
- 数组中的数据元素具有相同的数据类型;
- 数组中的每个数据元素都与一组唯一的下标值相对应;
- 数组是一种随机存储结构。



2. 数组的特点

- 一个**一维数组**，一旦第一个元素 a_i 的存储地址 $Loc(a_i)$ 确定，而每个元素所占用的存储空间大小为 k ，则第 i 个元素的地址可以由以下公式计算：

$$Loc(a_i) = Loc(a_0) + i \times k$$





2. 数组的特点

以**行序**为主序: C/C++

$$A=(A_1, A_2, \dots, A_m)$$

其中:

$$A_i=(a_{i1}, a_{i2}, \dots, a_{in}) (1 \leq i \leq m)$$

$$A_{m \times n} = \begin{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \end{bmatrix} \\ \begin{bmatrix} a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix} \end{bmatrix}$$

以**列序**为主序: Fortran

$$A=(A_1, A_2, \dots, A_n)$$

其中:

$$A_j=(a_{1j}, a_{2j}, \dots, a_{mj}) (1 \leq j \leq n)$$

$$A_{m \times n} = \begin{bmatrix} \begin{bmatrix} a_{00} \\ a_{10} \\ \vdots \\ a_{m-1,0} \end{bmatrix} & \begin{bmatrix} a_{01} \\ a_{11} \\ \vdots \\ a_{m-1,1} \end{bmatrix} & \begin{bmatrix} a_{02} \\ a_{12} \\ \vdots \\ a_{m-1,2} \end{bmatrix} & \cdots & \begin{bmatrix} a_{0,n-1} \\ a_{1,n-1} \\ \vdots \\ a_{m-1,n-1} \end{bmatrix} \end{bmatrix}$$



2. 数组的特点

 数组有什么特点呢？

- (1) 元素本身可以具有某种结构，属于同一数据类型；
- (2) 数组是一个具有固定格式和数量的数据集合。

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} & \cdots & \mathbf{a}_{1n} \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \cdots & \mathbf{a}_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{a}_{m1} & \mathbf{a}_{m2} & \cdots & \mathbf{a}_{mn} \end{pmatrix}$$

x
↓

在数组上一般不能执行插入或删除某个数组元素的操作



2. 数组的特点

 数组有什么基本操作呢？

- (1) **存取**：给定一组下标，读出对应的数组元素
 - (2) **修改**：给定一组下标，存储或修改与其相对应的数组元素
- } 寻址

ADT Matrix

DataModel

相同类型的数据元素的有序集合，每个元素受 n ($n \geq 1$) 个线性关系的约束

Operation

InitMatrix：数组的初始化

DestroyMatrix：数组的销毁

GetMatrix：读操作，读取这组下标对应的数组元素

SetMatrix：写操作，存储或修改这组下标对应的数组元素

endADT

4.3 多维数组

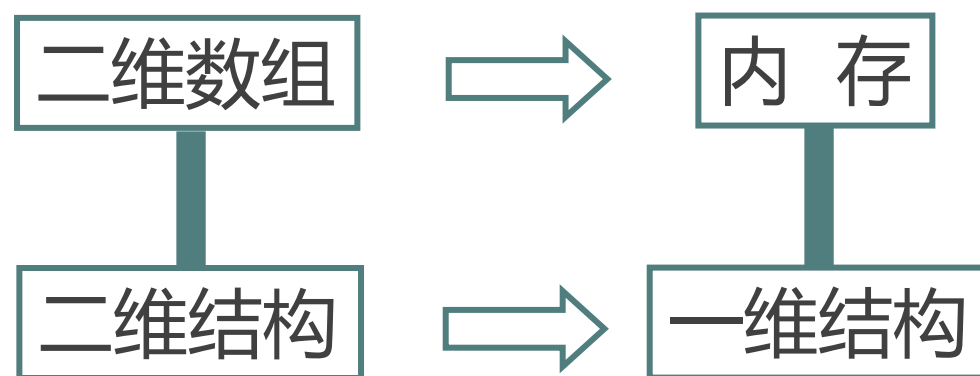
4-3-2 数组的存储结构



1. 数组的存储结

 如何存储（多维）数组呢？

数组没有插入和删除操作，所以，不用预留空间，适合采用顺序存储



 按行优先：先存储行号较小的元素，行号相同者先存储列号较小的元素

 按列优先：先存储列号较小的元素，列号相同者先存储行号较小的元素

4.3 多维数组

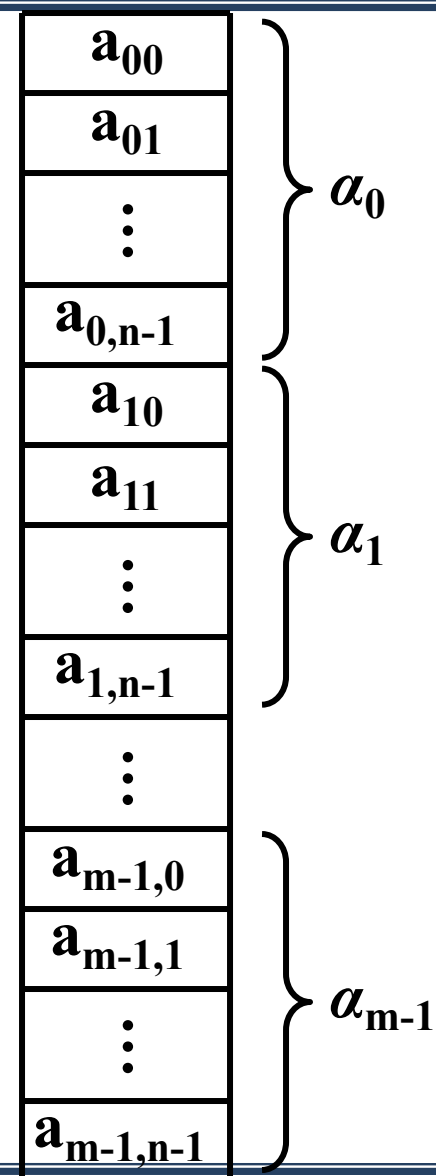
4-3-2 数组的存储结构



1. 数组的存储结构

以行序为主序: C/C++

$$\mathbf{A}_{m \times n}$$
$$\left[\begin{array}{c} \left[\begin{array}{ccccc} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \end{array} \right] \\ \left[\begin{array}{ccccc} a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \end{array} \right] \\ \vdots \\ \left[\begin{array}{ccccc} a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{array} \right] \end{array} \right]$$



4.3 多维数组



4-3-2 数组的存储结构

1. 数组的存储结构

设二维数组 $A(m \times n)$

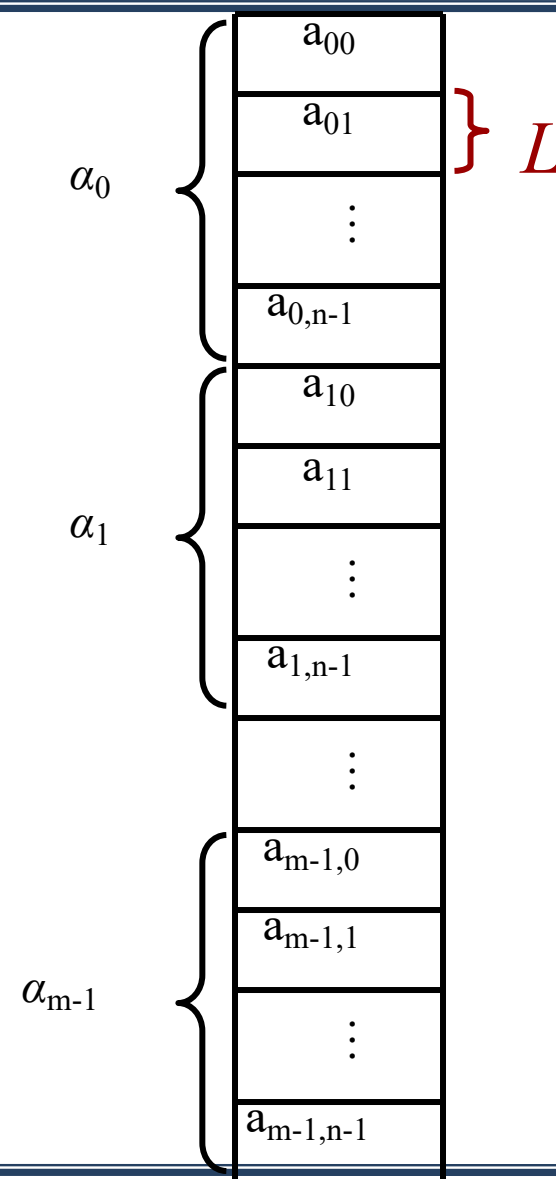
其数组元素 a_{ij} 的存储位置为

$$LOC(i, j) = LOC(0, 0) + (n \times i + j) L$$

其中, $LOC(0, 0)$ 是 a_{00} 的存储位置;

L 是每个数组元素占用的存储单元数;

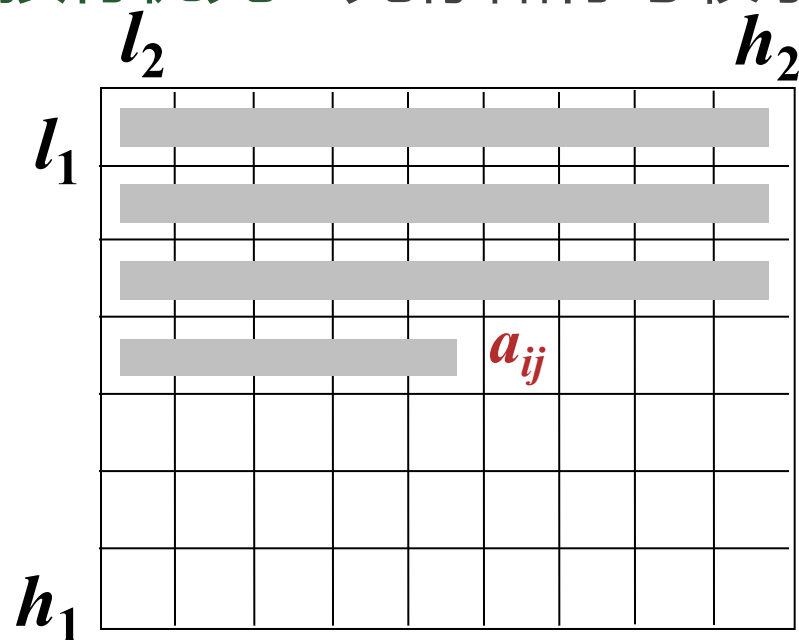
例, $LOC(1, 1) = LOC(0, 0) + (n \times 1 + 1) L$



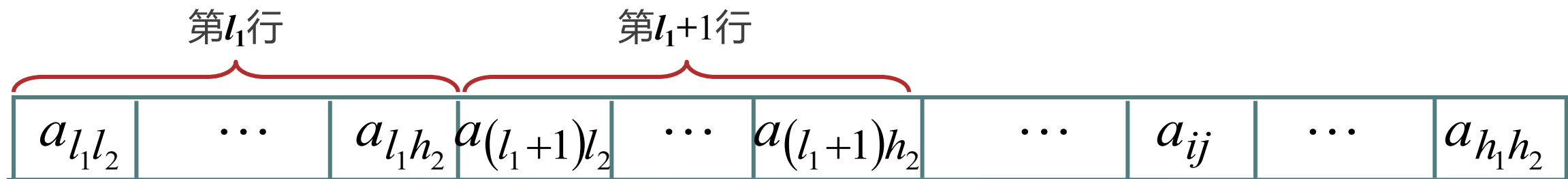


1. 数组的存储结

📌 按行优先：先存储行号较小的元素，行号相同者先存储列号较小的元素



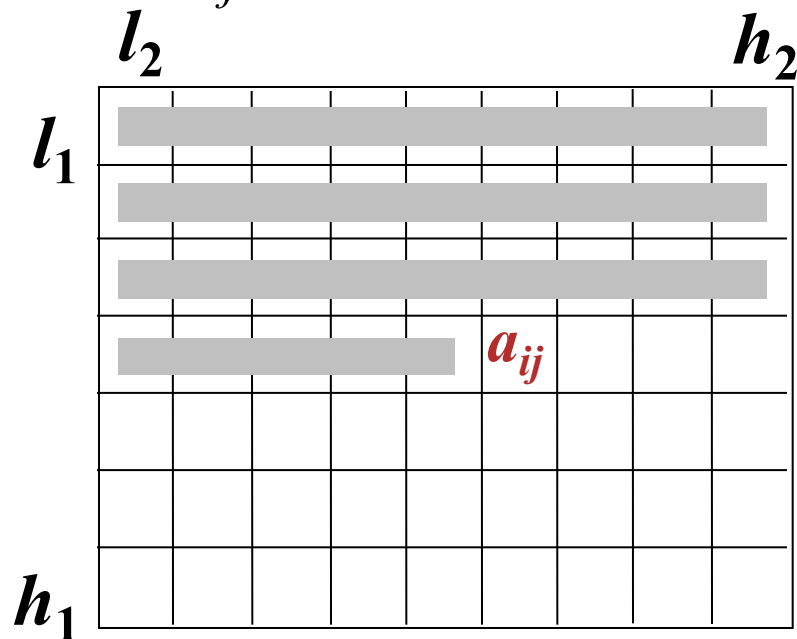
🕒 如何得到元素 a_{ij} 的存储地址？





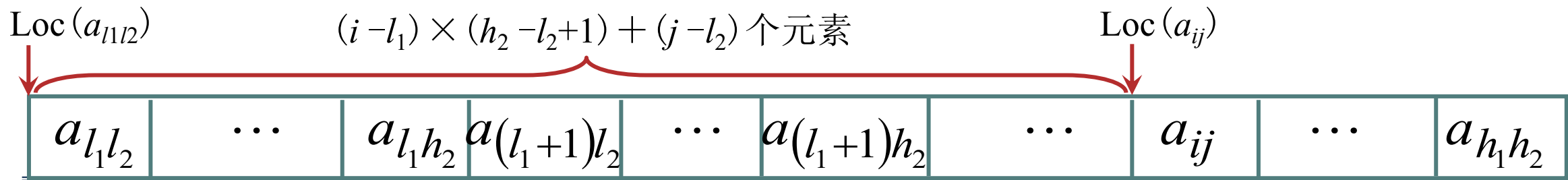
1. 数组的存储结构

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{l_1 l_2}) + ((i - l_1) \times (h_2 - l_2 + 1) + (j - l_2)) \times c$$



a_{ij} 前面的元素个数
 = 整行数 \times 每行元素个数 + 本行中
 a_{ij} 前面的元素个数
 = $(i - l_1) \times (h_2 - l_2 + 1) + (j - l_2)$

📌 按列优先与此类似，请自行给出



4.4 矩阵的压缩存储

4-4-1 特殊矩阵的压缩存储



特殊矩阵

✦ 特殊矩阵：矩阵中很多值相同的元素并且它们的分布有一定的规律
为值相同的元素分配一个存储空间
保证随机存取，即在 $O(1)$ 时间内寻址

压缩思路：用一维数组 SA 模拟存储 n 阶矩阵 A 。

关键问题：如何建立 数组元 $SA[k]$ 和 矩阵元 a_{ij} 之间的一一对应关系。

4.4 矩阵的压缩存储

4-4-1 特殊矩阵的压缩存储



1. 对称矩阵

对称矩阵特点: $a_{ij}=a_{ji}$

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

$$\begin{bmatrix} & & & & 0 \\ & & & & \\ & & & & \\ & & & & \\ 0 & & & & \end{bmatrix}$$



如何压缩存储对称矩阵呢? \Rightarrow 只存储下三角部分的元素

4.4 矩阵的压缩存储

4-4-1 特殊矩阵的压缩存储



1. 对称矩阵

a_{ij} 在一维数组中的序号 = $i \times (i-1)/2 + j$

∴ 一维数组下标从 0 开始

∴ a_{ij} 在一维数组中的下标

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

	1	...	j	...	n
1					
...					
i			a_{ij}		
n					

0	1	2	3	4	5	k		$n(n+1)/2-1$				
a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	...	a_{ij}	...	a_{n1}	a_{n2}	...	a_{nn}
第1行		第2行		第3行			第n行					



1. 对称矩阵

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

$$n = 5, \quad 1+2+3+4+5 = 5*(5+1)/2 = 15$$

一维数组 $SA[1..15]$ **作为数组** A **的存储结构:**

$$SA = (4 \ 5 \ 2 \ 3 \ 1 \ 3 \ 2 \ 5 \ 2 \ 8 \ 1 \ 6 \ 7 \ 9 \ 5)$$

$$\text{如: } a[5, 3] = a[3, 5] = 7$$

$$k = i(i-1)/2 + j-1 = 5(5-1)/2 + 3-1 = 12$$

$$\text{故: } sa[12] = 7$$



2. 三角矩阵

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

(a) 下三角矩阵

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

(b) 上三角矩阵



如何压缩存储三角矩阵呢？



下（上）三角部分的元素
相同的常数只存储一个

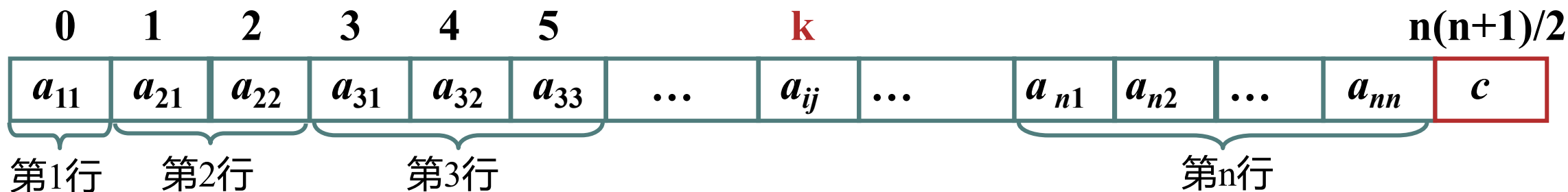
4.4 矩阵的压缩存储

4-4-1 特殊矩阵的压缩存储



2. 三角矩阵

📎 下三角矩阵的压缩存储



📎 下三角矩阵压缩存储后的寻址方法

对于下三角中的元素 a_{ij} ($i \geq j$) : $k = i \times (i - 1) / 2 + j - 1$

对于上三角中的元素 a_{ij} ($i < j$) : $k = n \times (n + 1) / 2$

📎 上三角矩阵的压缩存储请仿此给出



3. 对角矩阵

 **对角矩阵**：所有非零元素都集中在以**主对角线为中心**的带状区域中，所有其他元素都为零

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{pmatrix}$$

 如何压缩存储对角矩阵呢？


只存储非零元素



3. 对角矩阵

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{pmatrix}$$

元素 a_{ij} 在一维数组中的序号

$$= 2 + 3(i-2) + (j-i+2)$$

$$= 2i + j - 2$$

∵ 一维数组下标从 0 开始

∴ 元素 a_{ij} 在一维数组中的下标

$$= 2i + j - 3$$

0	1	2	3	4	5	6	7	8	9	10	11	12
a_{11}	a_{12}	a_{21}	a_{22}	a_{23}	a_{32}	a_{33}	a_{34}	a_{43}	a_{44}	a_{45}	a_{54}	a_{55}
第1行		第2行			第3行							

4.4 矩阵的压缩存储

4-4-2 稀疏矩阵的压缩存储



1. 稀疏矩阵定义及特点

✚ 稀疏矩阵：矩阵中有**很多**零元素，并且分布没有规律

🕒 稀疏矩阵如何压缩存储？

只存储非零元素，零元素不分配存储空间

🕒 如何只存储非零元素？

✚ 三元组：（行号，列号，非零元素值）

$$A = \begin{pmatrix} 3 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$



2. 三元组表

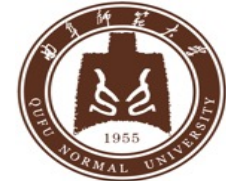
📌 **三元组表**：将稀疏矩阵的非零元素对应的三元组所构成的集合，按行优先的顺序排列成一个线性表

$((1, 1, 3), (1, 4, 7), (2, 3, 1), (3, 1, 2), (5, 4, 8))$

$$A = \begin{pmatrix} 3 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$

```
template <typename DataType>
struct element
{
    int row, col;
    DataType item;
};
```

📌 **三元组**：（行号，列号，非零元素值）



3. 三元组顺序表

✚ 三元组顺序表：采用顺序存储结构存储三元组表

$$A = \begin{pmatrix} 3 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix} \quad ((1, 1, 3), (1, 4, 7), (2, 3, 1), (3, 1, 2), (5, 4, 8))$$

稀疏矩阵的修改操作



三元组表的插入/删除操作



4.4 矩阵的压缩存储

4-4-2 稀疏矩阵的压缩存储

3. 三元组顺序表

 是否对应惟一的稀疏矩阵?

$$A = \begin{pmatrix} 3 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$

```
const int MaxTerm = 100;
struct SparseMatrix
{
    element data[MaxTerm];
    int mu, nu, tu; //行数、列数、非零元数目
};
```

MaxTerm-1

	row	col	item
0	1	1	3
1	1	4	7
2	2	3	1
3	3	1	2
4	5	4	8
	空	空	空
	闲	闲	闲
5 (矩阵的行数)			
4 (矩阵的列数)			
5 (非零元个数)			

缺点：稀疏矩阵的加法、乘法等操作，非零元素的**个数及位置**都会发生变化，则在三元组顺序表中就要进行**插入和删除**操作，顺序存储就十分不便



4. 十字链表

✚ 十字链表：采用链接存储结构存储三元组表

row	col	item
down		right

row : 非零元行号

col : 非零元列号

item : 非零元值

```
struct OrthNode
{
    Element data;
    OrthNode *right, *down;
};
```

right : 本行下一个非零元

down : 本列下一个非零元

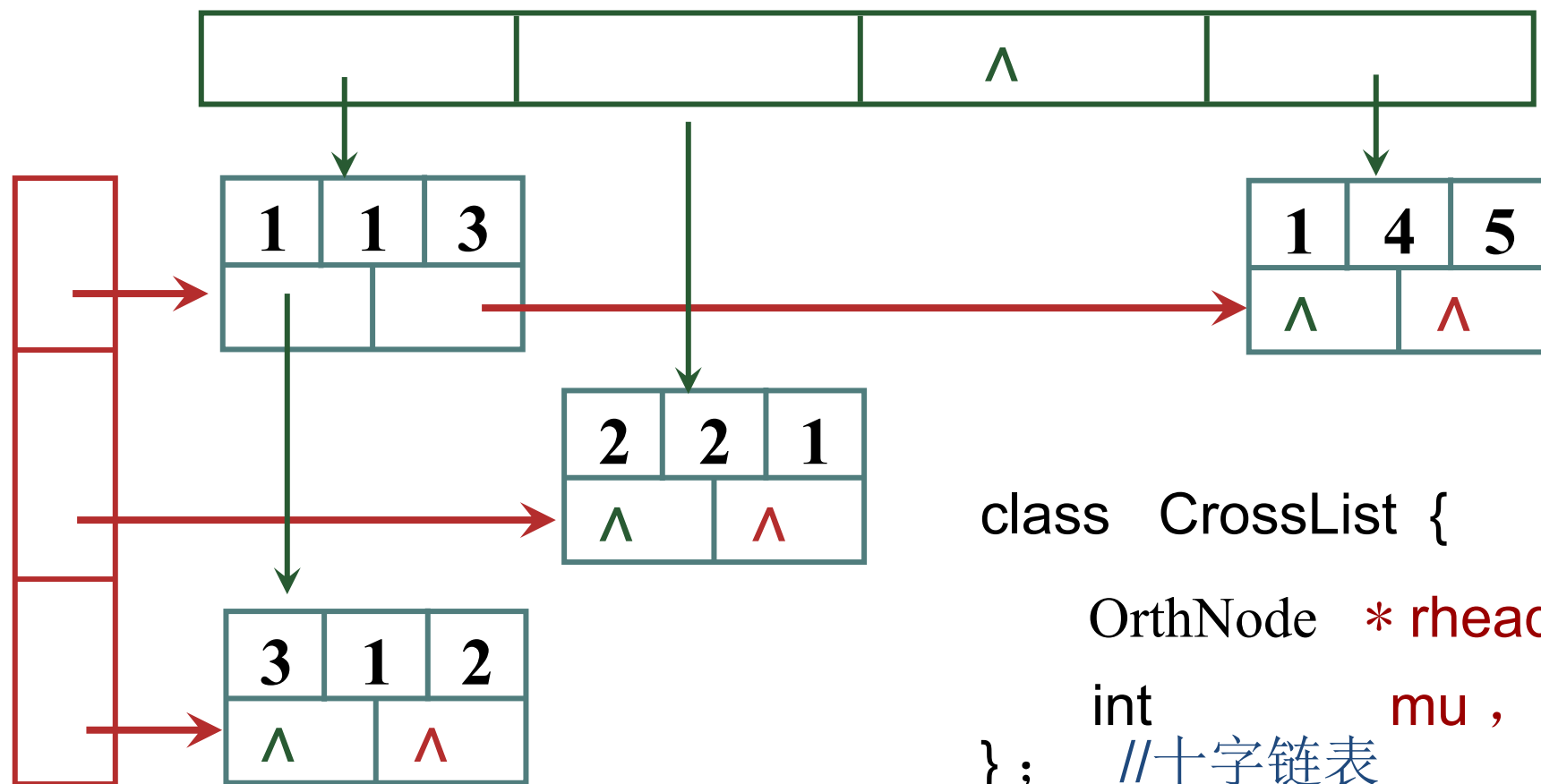
4.4 矩阵的压缩存储

4-4-2 稀疏矩阵的压缩存储



4. 十字链表

每个非零元既是某个行链表中的结点，又是某个列链表中的结点，整个矩阵构成了一个十字交叉的链表，故称为**十字链表**结构。



$$M = \begin{pmatrix} 3 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{pmatrix}$$

```
class CrossList {  
    OrthNode * rhead , * chead ; //一维数组  
    int mu , nu , tu ;  
}; //十字链表
```

4.5 扩展与提高

4.5 扩展与提高

4-5-1 稀疏矩阵的转置



矩阵乘法的最新进展


nature

Explore content ▾ About the journal ▾ Publish with us ▾

[nature](#) > [articles](#) > [article](#)

Article | [Open Access](#) | [Published: 05 October 2022](#)

Discovering faster matrix multiplication algorithms with reinforcement learning

[Alhussein Fawzi](#) , [Matej Balog](#), [Aja Huang](#), [Thomas Hubert](#), [Bernardino Romera-Paredes](#), [Mohammadamin Barekatin](#), [Alexander Novikov](#), [Francisco J. R. Ruiz](#), [Julian Schrittwieser](#), [Grzegorz Swirszcz](#), [David Silver](#), [Demis Hassabis](#) & [Pushmeet Kohli](#)

[Nature](#) **610**, 47–53 (2022) | [Cite this article](#)

[Metrics](#)

一个4X5和5X5矩阵乘，标准算法需要100次乘法，Strassen算法需要80次乘法，DeepMind的AlphaTensor找到了一种只需要76次乘法的解法。

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}$$

Standard algorithm

$$\begin{aligned} h_1 &= a_{1,1} b_{1,1} \\ h_2 &= a_{1,1} b_{1,2} \\ h_3 &= a_{1,2} b_{2,1} \\ h_4 &= a_{1,2} b_{2,2} \\ h_5 &= a_{2,1} b_{1,1} \\ h_6 &= a_{2,1} b_{1,2} \\ h_7 &= a_{2,2} b_{2,1} \\ h_8 &= a_{2,2} b_{2,2} \end{aligned}$$

$$\begin{aligned} c_{1,1} &= h_1 + h_3 \\ c_{1,2} &= h_2 + h_4 \\ c_{2,1} &= h_5 + h_7 \\ c_{2,2} &= h_6 + h_8 \end{aligned}$$

Strassen's algorithm

$$\begin{aligned} h_1 &= (a_{1,1} + a_{2,2})(b_{1,1} + b_{2,2}) \\ h_2 &= (a_{2,1} + a_{2,2})b_{1,1} \\ h_3 &= a_{1,1}(b_{1,2} - b_{2,2}) \\ h_4 &= a_{2,2}(-b_{1,1} + b_{2,1}) \\ h_5 &= (a_{1,1} + a_{1,2})b_{2,2} \\ h_6 &= (-a_{1,1} + a_{2,1})(b_{1,1} + b_{1,2}) \\ h_7 &= (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2}) \end{aligned}$$

$$\begin{aligned} c_{1,1} &= h_1 + h_4 - h_5 + h_7 \\ c_{1,2} &= h_3 + h_5 \\ c_{2,1} &= h_2 + h_4 \\ c_{2,2} &= h_1 - h_2 + h_3 + h_6 \end{aligned}$$

1. 稀疏矩阵的转置

实质: 行、列号交换

$(1, 2, 12) \longrightarrow (2, 1, 12)$

$(4, 3, 24) \longrightarrow (3, 4, 24)$

矩阵 $M_{6 \times 7}$

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

转置矩阵 $T_{7 \times 6}$

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



1. 稀疏矩阵的转置

矩阵 $M_{6 \times 7}$

i	j	a_{ij}
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

①



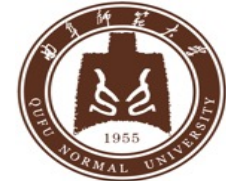
i	j	a_{ij}
2	1	12
3	1	9
1	3	-3
6	3	14
3	4	24
2	5	18
1	6	15
4	6	-7

②



转置矩阵 $T_{7 \times 6}$

i	j	a_{ij}
1	3	-3
1	6	15
2	1	12
2	5	18
3	1	9
3	4	24
4	6	-7
6	3	14



2. 稀疏矩阵的转置算法

按照 T 中三元组的次序依次在 M 中寻找相应的三元组进行转置。

T 按行序等价于 M 按列序

实质：按照 M 的**列序**进行转置。

思想：按照 M 的**列序**从头到尾对 M 三元组进行扫描，找到一个，就进行转置，并插入到 T 三元组中。



2. 稀疏矩阵的转置算法

矩阵 $M_{6 \times 7}$

i	j	a_{ij}
<u>1</u>	<u>2</u>	<u>12</u>
<u>1</u>	<u>3</u>	<u>9</u>
<u>3</u>	<u>1</u>	<u>-3</u>
<u>3</u>	<u>6</u>	<u>14</u>
<u>4</u>	<u>3</u>	<u>24</u>
<u>5</u>	<u>2</u>	<u>18</u>
<u>6</u>	<u>1</u>	<u>15</u>
<u>6</u>	<u>4</u>	<u>-7</u>

col = 6

转置矩阵 $T_{7 \times 6}$

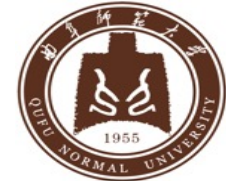
i	j	a_{ij}
1	3	-3
1	6	15
2	1	12
2	5	18
3	1	9
3	4	24
4	6	-7
6	3	14



3. 稀疏矩阵的转置实现

✚ 基本思想：设两个顺序表A、B，在A中依次查找第1列、第2列、...、最后一列，将找到后的元素行号列号交换并转储到B中。

```
void Transpose_1(SparseMatrix &A, SparseMatrix &B)
{
    int pa, pb, col;
    B.mu = A.nu; B.nu = A.mu; B.tu = A.tu;
    pb = 0;
    for (col = 1; col <= A.nu; col++)
        for (pa = 0; pa < A.tu; pa++)
            if (A.data[pa].col == col)
            {
                B.data[pb].row = A.data[pa].col;
                B.data[pb].col = A.data[pa].row;
                B.data[pb].item = A.data[pa].item;
                pb++;
            }
}
```



4. 算法复杂度分析

算法的时间复杂度： $O(nu \times tu)$

若 tu 和 $mu \times nu$ 同数量级，时间复杂度为 $O(mu \times nu^2)$

一般矩阵的转置算法为：

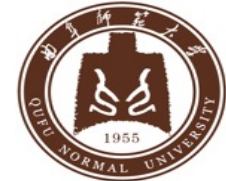
```
for ( col = 1 ; col <= nu ; ++ col )  
    for ( row = 1 ; row <= mu ; ++ row )  
        T[col][row] = M[row][col] ;
```

时间复杂度： $O(mu \times nu)$

由此可知，此算法仅适用于 $tu \ll mu \times nu$ 的情况。

优点： 非零元在表中按行序有序存储，便于进行依行顺序处理的矩阵运算。

缺点： 无法直接存取某行的某个非零元素。



1. 广义表的概念

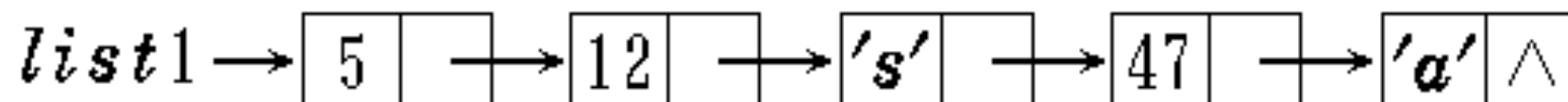
概念：广义表是 $n(n \geq 0)$ 个元素 a_1, a_2, \dots, a_n 的有限序列，其中 a_i 或者是原子或者是一个广义表。

- 表的深度：表展开后所含括号的层数。
- 递归表：允许递归的表。
- 再入表：允许结点共享的表。
- 纯表：与树对应的表。
- 线性表：

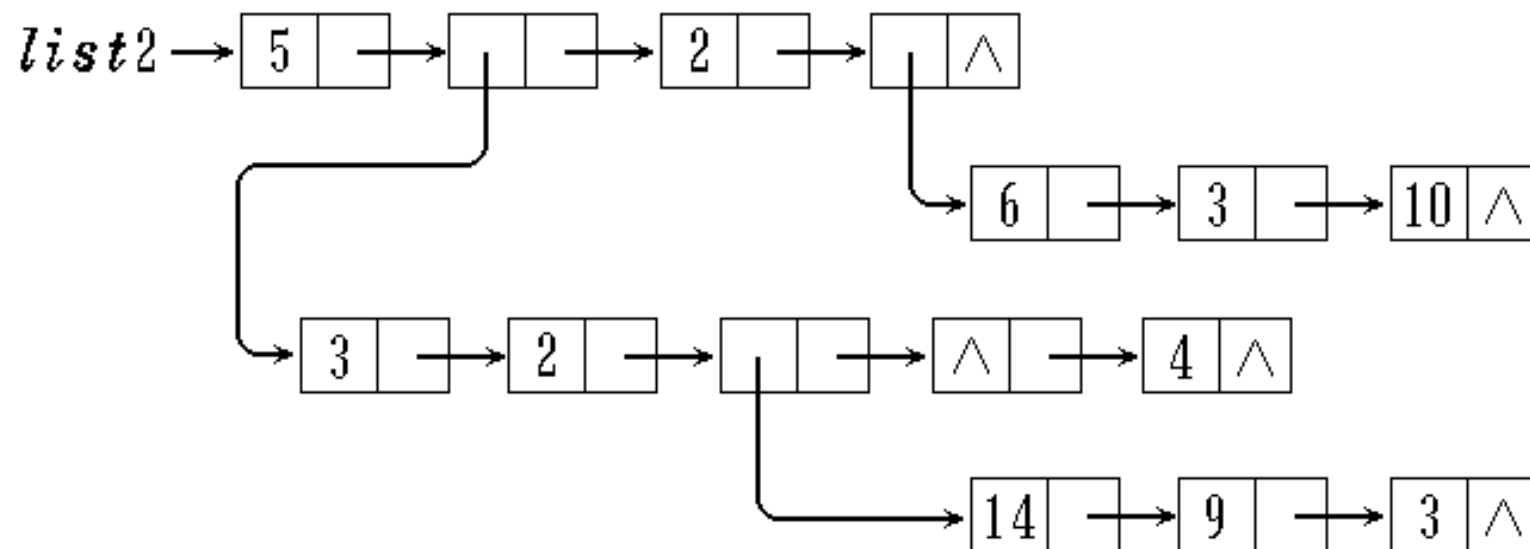
通常用圆括号将广义表括起来，用逗号分隔其中的元素。为了区分原子和广义表，书写时用大写字母表示广义表，用小写字母表示原子。

2. 广义表的表示

只包括整数和字符型数据的广义表链表表示

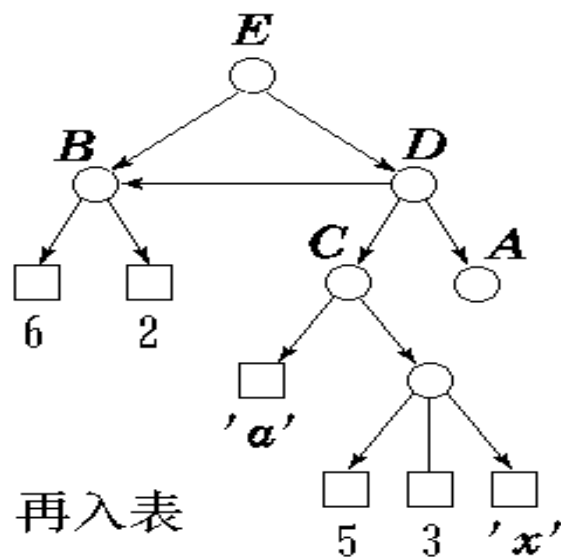
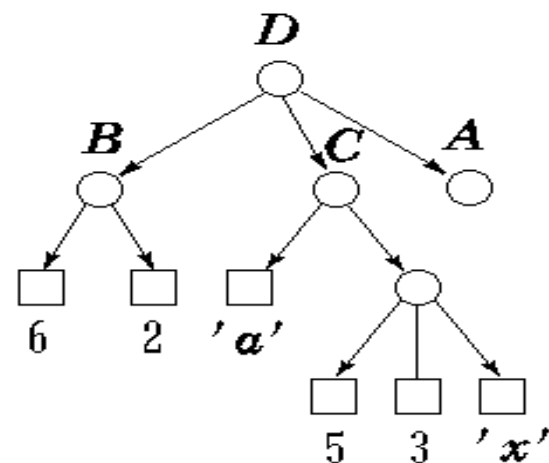
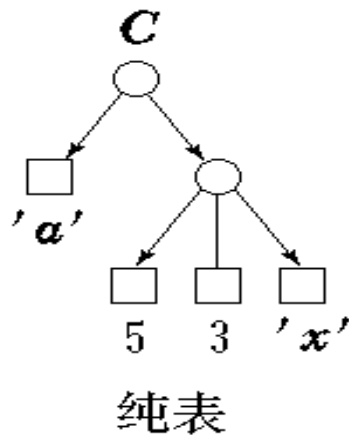
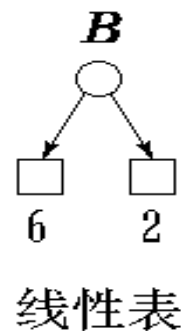


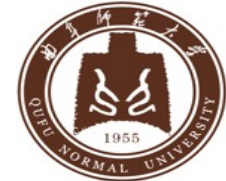
表中套表情形下的广义表链表表示



2. 广义表的表示

A
○
空表





3. 广义表的定义

广义表(列表) $LS = (a_1, a_2, \dots, a_n)$

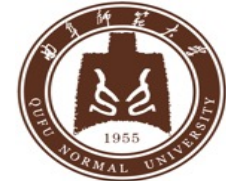
LS : 列表名称

n : 列表长度

a_i : 可以是单个元素, 也可以是子列表, 分别称为原子和子表。

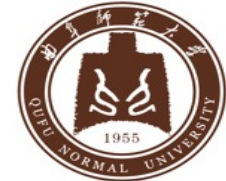
a_1 : LS 的表头 (第一个元素)。

(a_2, a_3, \dots, a_n) : LS 的表尾 (列表)。



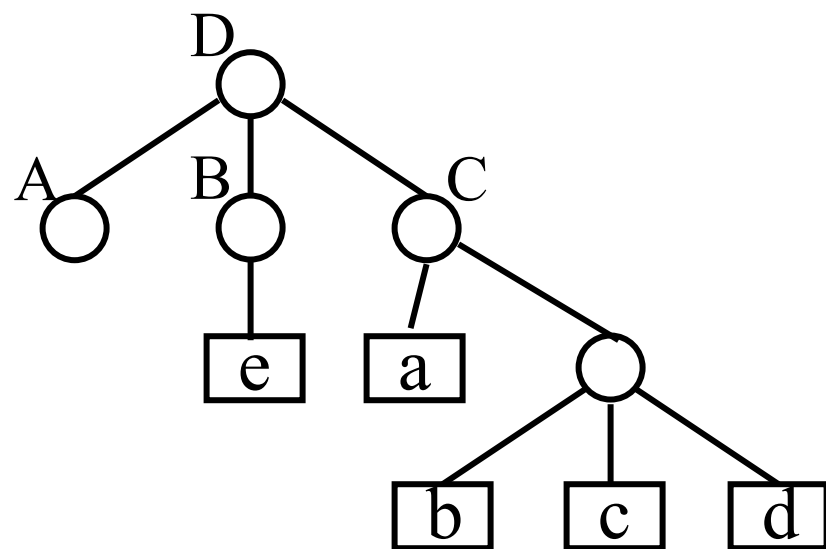
3. 广义表的定义

- (1) $A = ()$ —— A 是一个空表，长度为 0。
- (2) $B = (e)$ —— B 中只有一个原子 e ，长度为 1。
- (3) $C = (a, (b, c, d))$ —— C 有两个元素，原子 a 和子表 (b, c, d) ，长度为 2。
- (4) $D = (A, B, C)$ —— D 有三个元素，都是子表，长度为 3。且有
 $D = ((), (e), (a, (b, c, d)))$ 。
- (5) $E = (a, E)$ —— E 为一个递归的表，长度为 2。且有 $E = (a, (a, (a, \dots)))$ 。



4. 广义表的性质

(1) 列表是一个多层次的结构。



可见，列表可以描述为一棵树。

(2) 列表可为其它列表所共享。

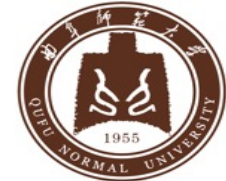
例， $D = (A, B, C)$

(3) 列表可以是一个递归的表。

例， $E = (a, E)$

(4) $() \neq (())$

$()$ 为空表，长度 0； $(())$ 长度 1，
可分解得到表头、表尾均为空表 $()$ 。



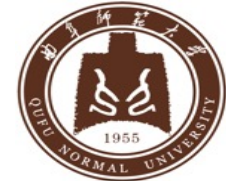
4. 广义表的性质

(5) 任何一个非空列表，其**表头**可能是原子或列表，而**表尾**一定是列表。

例， $B = (e)$ 表头为原子 e ；表尾为空表 $()$ 。

例， $B = ((a,b),c)$ 表头为列表 (a,b) ；表尾为列表 (c) 。

两个重要的函数: $\text{GetHead}()$ $\text{GetTail}()$



5. 广义表举例

1. $A=()$ $B=(e)$

2. $C=(a,(b,c,d))$

3. $D=(A,B,C)$

即 : $D=((),(e),(a,(b,c,d)))$

5. $E=(a,E)=(a,(a,(a,\dots)))$

6. (B,C)

$\text{GetHead}(B)=e$

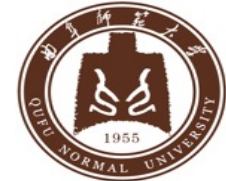
$\text{GetHead}(D)=A$

$\text{GetTail}(B)=()$

$\text{GetTail}(D)=(B,C)$

$\text{GetHead}((B,C))=B$

$\text{GetTail}((B,C))=(C)$



5. 广义表举例

$\text{GetHead}(\text{GetHead}(\text{GetTail}((a,(b,c,d))))))$

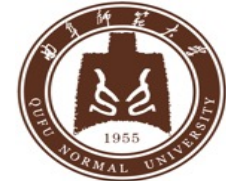
$\text{GetTail}(\text{GetHead}(\text{GetTail}((a,(b,c,d))))))$

$\text{GetTail}(a,(b,c,d))=((b,c,d))$

$(\text{GetHead}(\text{GetTail}(a,(b,c,d))))=(b,c,d)$

$\text{GetHead}(\text{GetHead}(\text{GetTail}(a,(b,c,d))))=? \quad b$

$\text{GetTail}(\text{GetHead}(\text{GetTail}(a,(b,c,d))))=? \quad (c,d)$



5. 广义表举例

$Ls=((a,b,c),(d,e,f,g))$ 中取出原子 e 的运算

$$L1 = \text{GetTail}(Ls) = ((d,e,f,g))$$

$$L2 = \text{GetHead}(L1) = (d,e,f,g)$$

$$L3 = \text{GetTail}(L2) = (e,f,g)$$

$$L4 = \text{GetHead}(L3) = e$$

本章小结

1. 理解字符串的定义和存储结构
2. 掌握字符串的模式匹配的BF算法和KMP算法
3. 掌握KMP算法中next、nextval数组的计算方法及匹配过程
4. 掌握数组的逻辑结构、存储结构及寻址方法
5. 掌握特殊矩阵的压缩存储方法
6. 理解稀疏矩阵的三元组顺序表和十字链表存储方法
7. 理解稀疏矩阵转置的计算方法
8. 了解广义表的概念和基本操作

数组总结

数组是 n ($n \geq 0$) 个相同数据类型数据元素构成的有限序列。

1. 数组/矩阵的定义、存储、定位

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

$$A_{m \times n} = \begin{bmatrix} [a_{00} & a_{01} & a_{02} & \dots & a_{0,n-1}] \\ [a_{10} & a_{11} & a_{12} & \dots & a_{1,n-1}] \\ \vdots & \vdots & \vdots & & \vdots \\ [a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \dots & a_{m-1,n-1}] \end{bmatrix}$$

以行序为主序: C/C++

$$LOC(i, j) = LOC(0, 0) + (n \times i + j) L$$

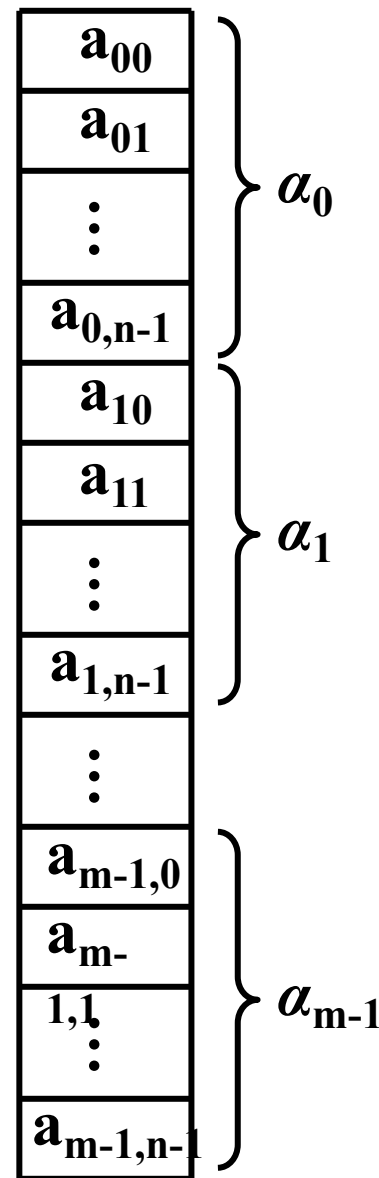
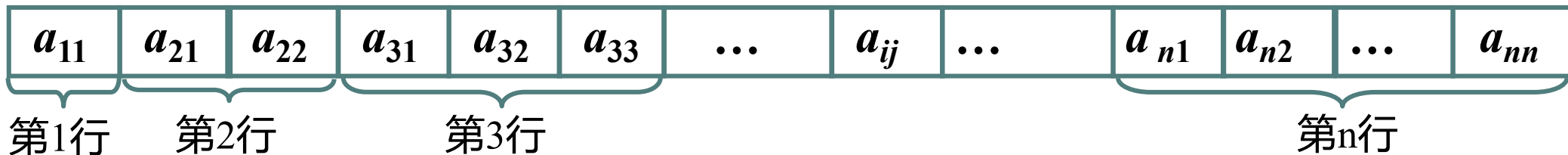
2. 特殊矩阵: 矩阵中很多值相同的元素并且它们的分布有一定的规律

对称矩阵、三角矩阵、对角矩阵等

关键问题: 如何建立数组元 $SA[k]$ 和 矩阵元 a_{ij} 之间的一一对应关系。

0 1 2 3 4 5 k

$n(n+1)/2-1$



3. 稀疏矩阵的压缩存储: 三元组顺序表 (行号, 列号, 非零元素值)

1. 若两个 $M \times N$ 的矩阵**A**和**B**都是利用三元组顺序表压缩存储，若要得到同样采用三元组顺序表压缩存储的矩阵**C**， $C=A+B$ （对应元素相加），请写出算法。

Status AddSMatrix(TSMatrix &C, TSMatrix A, TSMatrix B)

2. 假设按行序为主序的存储方式存储整数数组**A** $9 \times 3 \times 5 \times 8$ ，第一个元素的字节地址为**100**，每个元素占两个字节，问下列元素的地址是什么？

(1)**A0000** (2)**A1111** (3)**A3125** (4)**A8247**

3. 若将 $n \times n$ 的右上半三角矩阵**A_{ij}**压缩存储为一维数组**S_k**，请写出二维下标*i, j*与一维下标*k*的对应关系。

4. 求下列广义表操作的结果。

(1)**GetHead((p,h,w))** (2)**GetTail((p,h,w))**

(3)**GetHead(((a,b),(c,d)))** (4)**GetTail(((a,b),(c,d)))**

(5)**GetHead(GetTail(((a,b),(c,d))))**

(6)**GetTail(GetHead(GetTail(((a,b),(c,d)))))**

实验四、字符串和多维数组的实现与应用

一、实验目的

1. 掌握字符串模式匹配的BF和KMP方法。
2. 掌握特殊矩阵/稀疏矩阵的压缩存储方法。
3. 用C++语言实现相关算法，并上机调试。

二、实验内容

1. 实现BF算法和**KMP算法**，并用不同的主串和模式串进行测试。（**基本要求1**）
2. 设计算法实现特殊矩阵的压缩存储，并实现矩阵元素按下标检索的过程。（**基本要求2**）
3. 设计稀疏矩阵存储的三元顺序表数据结构，进一步实现稀疏矩阵的转置。（**扩展内容**）
4. 给出测试过程和测试结果。

实验时间： 第8周周四晚 19:00-21:00
实验地点： 格物楼A216



Thank You !

Q & A