第四章
数组
Array

主讲：顾为兵

第4章 数组（Array）

目录

§5.1 数组的定义

§5.2 数组的表示与实现

5.2.1存储方式: 行主序和列主序

5.2.2 数组元素的地址计算

5.2.3 数组基本运算的实现

§5.3 矩阵的压缩存储

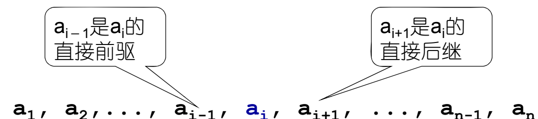
5.3.1 特殊矩阵

5.2.2 稀疏矩阵

§5.1 数组的定义

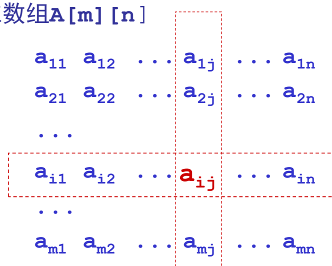
可以从两个角度来理解数组元素之间的逻辑关系。

对于一维向量：

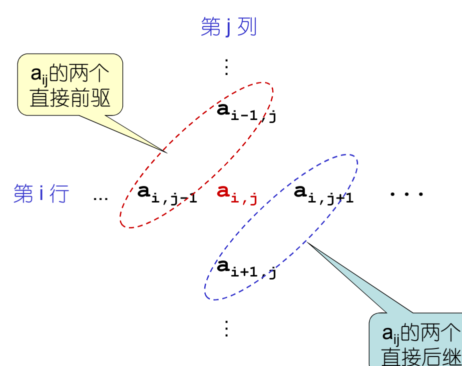


每个结点至多有1个直接前驱，至多有1个直接后继

对于二维数组A[m][n]



结点a_{ij}处在两个向量的交汇点上，至多有2个直接前驱，2个直接后继。



数组的定义

位于边界上的结点，可能只有1个前驱或1个后继：

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \cdots & \cdots \\ a_{i1}, & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{matrix}$$

a_{11}只有两个后继，没有前驱

a_{m1}有一个前驱和一个后继

a_{m2}有两个前驱和一个后继

数组的定义

二维数组A[m][n]的定义：

数据对象： D={ a_{ij} | i=1,2,...,m; j=1,2,...,n; a_{ij}∈ElemSet }

数据关系： R={ Row, Col }

$Row=\{ \langle a_{i,j}, a_{i,j+1}\rangle \mid 1\le i\le m, 1\le j\le n-1 \}$

$Col =\{ \langle a_{i,j}, a_{i+1,j}\rangle \mid 1\le i\le m-1, 1\le j\le n \}$

数组的定义

依此类推，在三维数组中结点a_{ijk}处在3个向量的交汇点上。除了边界上的结点外，每个结点有三个直接前驱和三个直接后继。

i　k　j

$a_{i\,j-1\,k}$

$a_{i+1\,j\,k}$

$a_{i\,j\,k-1}$　$a_{i\,j\,k}$　$a_{i\,j\,k+1}$

$a_{i-1\,j\,k}$

$a_{i\,j+1\,k}$

数组的定义

一般在n维数组中，每个结点处在n个向量的交汇点上，除了边界上的结点外，每个结点有且仅有n个直接前驱，n个直接后继：

$A[b_1][b_2]...[b_n]$　　----b_1,..., b每维的长度

数据关系：

$R=\{R_1, R_2, \ldots, R_n\}$

$R_i=\{\langle a_{j1,j2\ldots ji\ldots jn}, a_{j1,j2\ldots ji+1\ldots jn}\rangle \mid$
$0\le j_k\le b_k-1,\ 1\le k\le n,\ i\ne k,$
$0\le j_i\le b_i-2\ \}$

数组的定义

我们也可以从另一个角度来定义数组：

数组是线性表的扩展，其中每个数据元素不是原子数据，而是另外一个表。

数组的定义

如，两维数组

$$A_{m\times n}=\begin{pmatrix} a_{11}, & a_{12}, & \ldots, & a_{1j}, & \ldots, & a_{1n} \\ a_{21}, & a_{22}, & \ldots, & a_{2j}, & \ldots, & a_{2n} \\ \ldots & \ldots \\ a_{i1}, & a_{i2}, & \ldots, & a_{ij}, & \ldots, & a_{in} \\ \ldots & \ldots \\ a_{m1}, & a_{m2}, & \ldots, & a_{mj}, & \ldots, & a_{mn} \end{pmatrix}$$

可以看成为一个具有m个元素的线性表：

$A=(\alpha_1, \alpha_2, \ldots, \alpha_i, \ldots, \alpha_m)$

其中每个元素α_i不是原子元素，而是另外一个线性表（行向量）：

$\alpha_i=(a_{i1}, a_{i2}, \ldots, a_{in})$

同样，二维数组也可以看成为有n个元素的线性表：

$$A = (\beta_1, \beta_2, \ldots, \beta_j, \ldots, \beta_n)$$

其中每个元素β $_j$不是原子元素，而是一个列向量：

$$\beta_j = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \ldots \\ a_{mj} \end{pmatrix}$$

由此可见，二维数组是线性表的线性表

类似的，三维数组也可以看成是一个线性表，其中每个元素是一个两维数组。

推广到n维数组---- n维数组是一个线性表，该线性表中的每个元素是一个n-1维数组。

数组的基本运算：

初始化：InitArray(&A, n, b_1, b_2, ..., b_n)

销毁： DestroyArray(&A)

读元素： Value(A, &e, j1, j2, ..., jn)

写元素： Assign(&A, e, j1, j2, ..., jn)

注意：没有插入和删除操作

第3章 数组（Array）

目录

§5.1 数组的定义
§5.2 数组的表示与实现
5.2.1存储方式: 行主序和列主序
5.2.2 数组元素的地址计算
5.2.3 数组基本运算的实现
§5.3 矩阵的压缩存储
5.3.1 特殊矩阵
5.2.2 稀疏矩阵

§5.2 数组的表示和实现

为了操作方便起见，数组只能采用顺序存储结构存储。

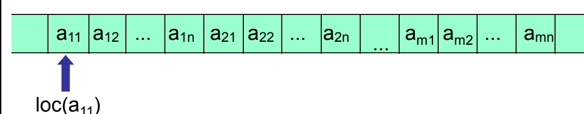
5.2.1 存放次序

数组是多维的，内存地址空间是一维的。要将数组存放到内存空间需要对存放次序进行约定：

1. 按行主序（行优先）次序存放
2. 按列主序（列优先）次序存放

以两维数组为例, 行主序是按行的次序一行一行地存放：

$$A_{m\times n} = \begin{matrix} a_{11}, a_{12}, \ldots, a_{1j}, \ldots, a_{1n} \\ a_{21}, a_{22}, \ldots, a_{2j}, \ldots, a_{2n} \\ \ldots \ldots \\ a_{i1}, a_{i2}, \ldots, a_{ij}, \ldots, a_{in} \\ \ldots \ldots \\ a_{m1}, a_{m2}, \ldots, a_{mj}, \ldots, a_{mn} \end{matrix}$$



loc(a_{11})

列主序是按列的次序一列一列地存放：

$$A_{m\times n}=\begin{matrix} a_{11}, a_{12}, \ldots, a_{1j}, \ldots, a_{1n} \\ a_{21}, a_{22}, \ldots, a_{2j}, \ldots, a_{2n} \\ \ldots \ldots \\ a_{i1}, a_{i2}, \ldots, a_{ij}, \ldots, a_{in} \\ \ldots \ldots \\ a_{m1}, a_{m2}, \ldots, a_{mj}, \ldots, a_{mn} \end{matrix}$$

a_{11}	a_{21}	…	a_{m1}	a_{12}	a_{22}	…	a_{m2}	…	a_{1n}	a_{2n}	…	a_{mn}

$loc(a_{11})$

对于三维数组，行主序的存放次序为：

	a_{211}	a_{212}	a_{213}	a_{214}
a_{111}	a_{112}	a_{113}	a_{114}	a_{223}	a_{224}
a_{121}	a_{122}	a_{123}	a_{124}	a_{233}	a_{234}
a_{131}	a_{132}	a_{133}	a_{134}		

a_{111}	a_{112}	a_{113}	a_{114}	a_{121}	a_{122}	…	a_{134}	…	a_{211}	a_{212}	…	a_{234}

$loc(a_{111})$

对于三维数组，列主序的存放次序为

	a_{211}	a_{212}	a_{213}	a_{214}
a_{111}	a_{112}	a_{113}	a_{114}	a_{223}	a_{224}
a_{121}	a_{122}	a_{123}	a_{124}	a_{233}	a_{234}
a_{131}	a_{132}	a_{133}	a_{134}		

a_{111}	a_{211}	a_{121}	a_{221}	a_{131}	a_{231}	a_{112}	a_{212}	a_{122}	a_{222}	a_{132}	…	a_{234}

$loc(a_{111})$

n维数组的一般规律：

行主序：最先排最右下标，依次向左，最后排最左下标；

列主序：最先排最左下标，依次向右，最后排最右下标

C、Pascal、Basic等语言采用行主序的方式存储数组；Fortran语言采用的是列主序方式存储数组。

第3章 数组（Array）

目录

§ 5.1 数组的定义
§ 5.2 数组的表示与实现
5.2.1存储方式: 行主序和列主序
5.2.2 数组元素的地址计算
5.2.3 数组基本运算的实现
§ 5.3 矩阵的压缩存储
5.3.1 特殊矩阵
5.2.2 稀疏矩阵

5.2.2 数组元素地址的计算

已知条件：

1. 数组的首地址
2. 数组的维数及每一维的长度
3. 数组元素的长度L
4. 存放方式（行/列主序）

地址＝首地址＋偏移量
＝首地址＋（元素的排列序号－1）×L

二维数组A[b₁][b₂]在行主序方式下的地址计算：

aij的前面有i行

a_{00}, a_{01}, …, a_{0j}, …, $a_{1\ b2-1}$
a_{10}, a_{11}, …, a_{1j}, …, $a_{1\ b2-1}$
… …
a_{i0}, a_{i1}, …, a_{ij}, …, $a_{i\ b2-1}$
… …
$a_{b1-1\ 0}$, $a_{b1-1\ 1}$, …, $a_{b1-1\ j}$, …, $a_{b1-1\ b2-1}$

在第i行上，a_{ij}是第j＋1个元素

$LOC(a_{ij}) = LOC(a_{00}) + (b_2 \times i + j + 1 - 1) \times L$
$= LOC(a_{00}) + (b_2 \times i + j) \times L$

二维数组A[b₁][b₂]列主序在方式下的地址计算：

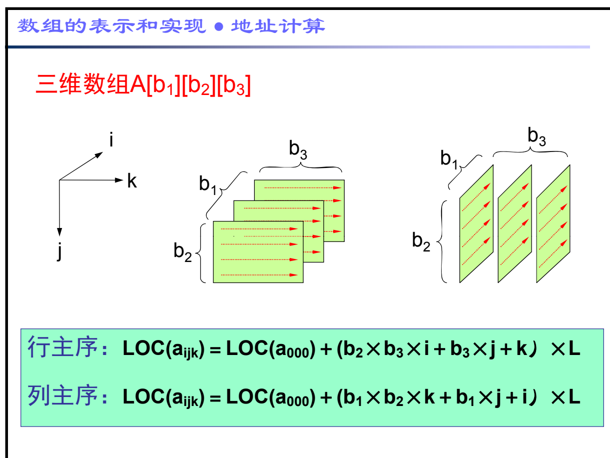
aij的前面有j列

在第j行上，a_{ij}是第i＋1个元素

a_{00}, a_{01}, …, a_{0j}, …, $a_{1\ b2-1}$
a_{10}, a_{11}, …, a_{1j}, …, $a_{1\ b2-1}$
… …
a_{i0}, a_{i1}, …, a_{ij}, …, $a_{i\ b2-1}$
… …
$a_{b1-1\ 0}$, $a_{b1-1\ 1}$, …, $a_{b1-1\ j}$, …, $a_{b1-1\ b2-1}$

$LOC(a_{ij}) = LOC(a_{00}) + (b_1 \times j + i + 1 - 1) \times L$
$= LOC(a_{00}) + (b_1 \times j + i) \times L$

三维数组A[b₁][b₂][b₃]



行主序：$LOC(a_{ijk}) = LOC(a_{000}) + (b_2 \times b_3 \times i + b_3 \times j + k) \times L$

列主序：$LOC(a_{ijk}) = LOC(a_{000}) + (b_1 \times b_2 \times k + b_1 \times j + i) \times L$

n维数组A[b₁][b₂][b₃]...[bₙ]

行主序：

$LOC(j_1, j_2, ..., j_n) = LOC(0,0,...,0) + (b_2 * b_3 * b_4 * ... * b_n * j_1 + b_3 * b_4 * ... * b_n * j_2 + ... b_n * j_{n-1} + j_n) * L$

列主序：

$LOC(j_1, j_2, ..., j_n) = LOC(0,0,...,0) + (b_1 * b_2 * ... * b_{n-1} * j_n + b_1 * b_2 * ... * b_{n-2} * j_{n-1} + ... b_1 * j_2 + j_1) * L$

第3章 数组（Array）

目录

§5.1 数组的定义
§5.2 数组的表示与实现
5.2.1存储方式: 行主序和列主序
5.2.2 数组元素的地址计算
5.2.3 数组基本运算的实现
§5.3 矩阵的压缩存储
5.3.1 特殊矩阵
5.2.2 稀疏矩阵

§5.3 矩阵的压缩存储

- 压缩存储的前提：当矩阵中含有大量零元素或值相同的元素。
- 压缩存储的方法：零元素不分配空间；同值元素共享一个存储空间。
- 根据零元素或值相同的元素在矩阵中分布是否有规律，可将矩阵分为两类：

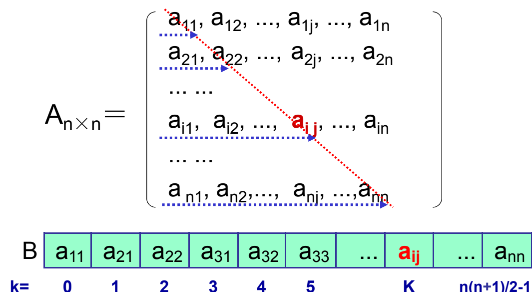
特殊矩阵　稀疏矩阵

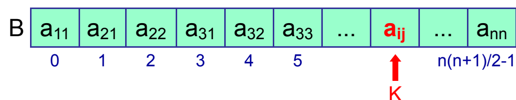
5.3.1特殊矩阵的压缩存储

一、对称矩阵

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

将矩阵的下三角部分按行主序的次序压缩存储到一维数组B[n(n+1)/2]中

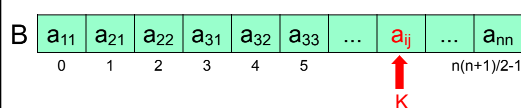




- 设元素a_{ij}在B数组中的下标是K，

  $B[K]=a_{ij}$

- 关键问题是如何通过下标(i,j)来计算K
- K=行主序下a_{ij}的排列序号 - 1

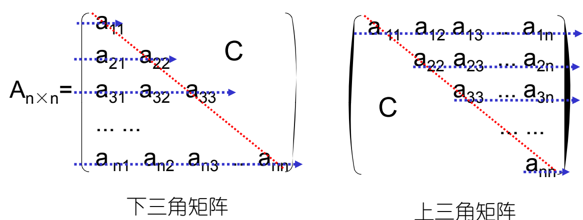


a_{ij}的前面有i-1行，共有i(i-1)/2个元素；

在第i行上，a_{ij}是第j个元素，所以有：

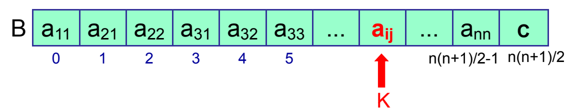
$$K=\begin{cases} i(i-1)/2+j-1 & \text{当}i\ge j\ (a_{ij}\text{位于下三角})\text{时} \\ j(j-1)/2+i-1 & \text{当}i<j\ (a_{ij}\text{位于上三角})\text{时} \end{cases}$$

二、三角矩阵



下三角矩阵　　上三角矩阵

下三角矩阵：



$$K=\begin{cases} i(i-1)/2+j-1 & \text{当}i\ge j\ (a_{ij}\text{位于下三角})\text{时} \\ n(n+1)/2 & \text{当}i<j\ (a_{ij}\text{位于上三角})\text{时} \end{cases}$$

上三角矩阵:

B	a_{11}	a_{12}	…	a_{1n}	a_{22}	a_{23}	…	a_{2n}	…	a_{nn}	c
	0	1								n(n+1)/2-1	n(n+1)/2

➩ 第t行有n-t+1 个元素，前i-1行共有

$$\sum_{t=1}^{i-1}(n-t+1)=(i-1)(2n-i+2)/2$$

➩ 在第i行上，a_{ij}是第 j-i+1个元素，所以

$$K=\begin{cases}(i-1)(2n-i+2)/2+j-i & \text{当}i\le j\text{(上三角)时}\\ n(n+1)/2 & \text{当}i>j\text{(下三角)时}\end{cases}$$

三、对角矩阵

$$A_{n\times n}=\begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{32} & & \\ & & a_{43} & a_{44} & a_{45} & \\ \cdots\cdots & & & & & \\ & & & & a_{n,n-1} & a_{nn}\end{bmatrix}$$

a_{ij}的前i-1行有3×(i-1)-1个元素;

在第i行上a_{ij}是第j-i+2个元素，所以

K=3×(i-1)-1+j-i+2-1=2i+j-3

第3章 数组 (Array)

目 录

§5.1 数组的定义
§5.2 数组的表示与实现
5.2.1存储方式：行主序和列主序
5.2.2 数组元素的地址计算
5.2.3 数组基本运算的实现
§5.3 矩阵的压缩存储
5.3.1 特殊矩阵
5.2.2 稀疏矩阵

5.3.2 稀疏矩阵(Sparse Matrix)

稀疏矩阵:

在矩阵中 ①非零元很少

②非零元的分布没有规律

稀疏矩阵的基本运算:

创建:	CreateSMatrix(&M)	
销毁:	DestroySMatrix(&M)	
打印:	PrintSMatrix(M)	
复制:	CopySMatrix(M, &T)	//T=M
求和:	AddSMatrix(M, N, &Q)	//Q=M+N
求差:	SubSMatrix(M, N, &Q)	//Q=M−N
求积:	MultSMatrix(M, N, &Q)	//Q=M×N
转置:	TransposeSMatrix(M, &T)	//T=M^T

5.3.2 稀疏矩阵

稀疏矩阵的几种压缩存储方法:

1. 三元组表
2. 行向量链表
3. 十字链表

1. 三元组表

(1，2，3)

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

三元组：（行号i，列号j，值e）

----用来描述一个非0元素

三元组表：矩阵中所有非0元素按行主序排成的表格

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

A	i	j	e
0			
1	1	2	3
2	1	3	1
3	2	1	6
4	2	2	4
5	2	5	7
6	3	2	2
7	3	5	1

此外还要加上三个参数：

行数mu、列数nu、非0元个数tu

mu=3,nu=5,tu=7

三元组表的类型定义：

```
#define maxsize 1000
typedef struct {
   int        i, j;
   ElemType   e;
}Triple;        //三元组

typedef struct {
   Triple  data[maxsize+1]; //三元组数组
   int     mu, nu, tu;      //行数、列数非零元个数
}TSMatrix //三元组表类型
```


稀疏矩阵运算在三元组表上的实现：**转置**

$$B=A^T$$

$$A_{3\times5}=\begin{pmatrix}0&3&1&0&0\\6&4&0&0&7\\0&2&0&0&1\end{pmatrix}\quad B_{5\times3}=\begin{pmatrix}0&6&0\\3&4&2\\1&0&0\\0&0&0\\0&7&1\end{pmatrix}$$

朴素的转置方法(算法5. 1)：

对三元组表做**nu趟**扫描，分别将第1列,第2列, ..., 第nu列元素依次抄入B表

第 1 趟

A	i	j	e
0	3	5	7
1	1	2	3
2	1	3	1
3	2	1	6
4	2	2	4
5	2	5	7
6	3	2	2
7	3	5	1

$B=A^T$	i	j	e
0	5	3	7
1	1	2	6
2			
3			
4			
5			
6			
7			

朴素的转置方法(算法5. 1)：

对三元组表做**nu趟**扫描，分别将第1列,第2列, ..., 第nu列元素依次抄入B表

第 2 趟

A	i	j	e
0	3	5	7
1	1	2	3
2	1	3	1
3	2	1	6
4	2	2	4
5	2	5	7
6	3	2	2
7	3	5	1

$B=A^T$	i	j	e
0	5	3	7
1	1	2	6
2	2	1	3
3	2	2	4
4	2	3	2
5			
6			
7			

朴素的转置方法(算法5. 1):

对三元组表做nu趟扫描，分别将第1列,第2列, ..., 第nu列元素依次抄入B表

第 3 趟

A

	i	j	e
0	3	5	7
1	1	2	3
2	1	3	1
3	2	1	6
4	2	2	4
5	2	5	7
6	3	2	2
7	3	5	1

$B=A^T$

	i	j	e
0	5	3	7
1	1	2	6
2	2	1	3
3	2	2	4
4	2	3	2
5	3	1	1
6			
7			

朴素的转置方法(算法5. 1):

对三元组表做nu趟扫描，分别将第1列,第2列, ..., 第nu列元素依次抄入B表

第 5 趟

A

	i	j	e
0	3	5	7
1	1	2	3
2	1	3	1
3	2	1	6
4	2	2	4
5	2	5	7
6	3	2	2
7	3	5	1

$B=A^T$

	i	j	e
0	5	3	7
1	1	2	6
2	2	1	3
3	2	2	4
4	2	3	2
5	3	1	1
6	5	2	7
7	5	3	1

```
void TransposeSMatrix(TSMatrix A, TSMatrix &B){
  // 朴素的转置算法B=AT
  B.mu=A.nu; B.nu=A.mu; B.tu=A.tu;
  if(!A.tu)return;
  q=1;
  for(col=1; col<=A.nu; col++)   //对列数的循环
      for(p=1; p<A.tu; p++)      //对非0元个数的循环
         if(A.data[p].j == col){
            B.data[q].i=A.data[p].j;
            B.data[q].j=A.data[p].i;
            B.data[q].e=A.data[p].e;  q++;
         }
}//end TransposeSMatrix
```


朴素转置算法的时间复杂度:

O(A.nu×A.tu)

时间性能不高。

特别当矩阵稀疏程度不高时，A.tu≈A.mu×A.nu

时间复杂度为**O(A.mu×A.nu²)**

快速转置方法(算法5.2):

- 希望只对三元组表A做一趟扫描。前提：每当扫描到一个元素时，能够知道它在转置以后的的位置。
- 利用两个辅助数组： num[A.nu+1] 和 col[A.nu+1]

  num[i]＝矩阵A第i列非0元素的个数

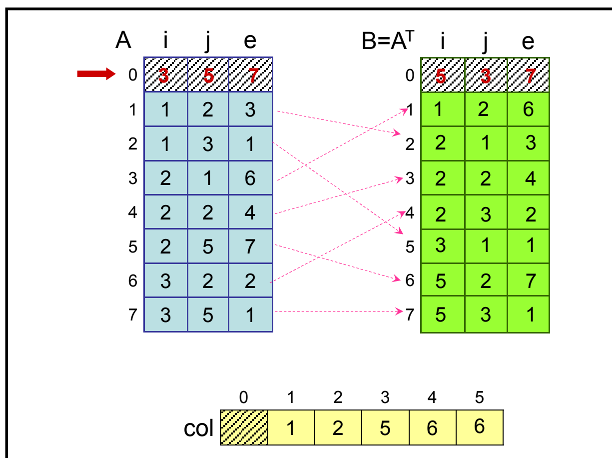
  col[i]＝矩阵A第i列第一个非0元素在A^T中的位置
- 这是一种用空间换时间的策略。

$$\begin{cases} col[1]=1 \\ col[i]=col[i-1]+num[i-1] \end{cases}$$

	0	1	2	3	4	5
num		1	3	1	0	2
col		1	2	5	6	6

A

	i	j	e
0			
1	1	2	3
2	1	3	1
3	2	1	6
4	2	2	4
5	2	5	7
6	3	2	2
7	3	5	1



	i	j	e
0	3	5	7
1	1	2	3
2	1	3	1
3	2	1	6
4	2	2	4
5	2	5	7
6	3	2	2
7	3	5	1

	i	j	e
0	5	3	7
1	1	2	6
2	2	1	3
3	2	2	4
4	2	3	2
5	3	1	1
6	5	2	7
7	5	3	1

	0	1	2	3	4	5
col		1	2	5	6	6

矩阵的压缩存储 • 稀疏矩阵

5.3.2 稀疏矩阵

稀疏矩阵的几种压缩存储方法：

1. 三元组表
2. 行向量链表
3. 十字链表

矩阵的压缩存储 • 稀疏矩阵 • 行向量链表

二、行向量链表

- 将稀疏矩阵中同一行非0元串成一个单链表
- 链表结点的结构为：

j	e	next
列号	元素的值	行后继

- 将mu个头指针放在一个指针型的数组内

矩阵的压缩存储 • 稀疏矩阵 • 行向量链表

行向量链表：

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

采用这种存储方式，找同一行元素容易，找同一列元素难.

头指针数组

A.rhead

j | e | next

按列号值有序的单链表

0
1 → 2 3 → 3 1 ∧
2 → 1 6 → 2 4 → 5 7 ∧
3 ∧
4 → 2 2 → 5 1 ∧

矩阵的压缩存储 • 稀疏矩阵 • 行向量链表

类型定义:

```
typedef struct  RNode{          //表结点类型
    int           j;            //列号
    ElemType      e;
    struct LNode  *next;
}RNode, *RLink;

typedef struct {             //矩阵类型
    RLink  *rhead;           //行链头指针数组的基地址
    int  mu, nu, tu;         //行数、列数、非零元个数
}RowList;
```

矩阵的压缩存储 • 稀疏矩阵 • 行向量链表

思考：稀疏矩阵的基本运算在行向量链表上的实现：

- 创建行向量链表
- 转置
- 求和
- 求积

0
1 → 2 3 → 3 1 ∧
2 → 1 6 → 2 4 → 5 7 ∧
3 ∧
4 → 2 2 → 5 1 ∧

5.3.2 稀疏矩阵

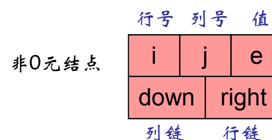
稀疏矩阵的几种压缩存储方法：

1. 三元组表
2. 行向量链表
3. 十字链表

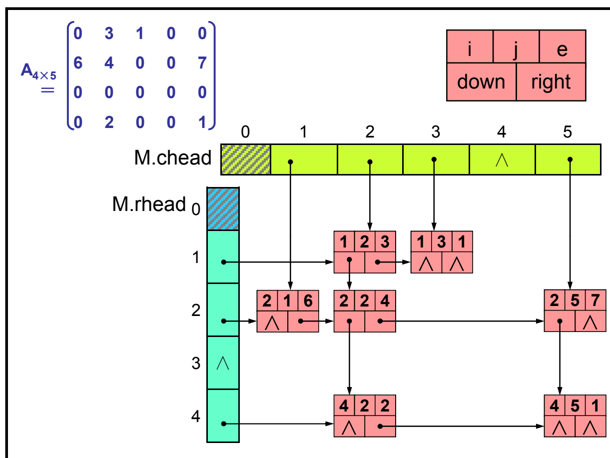
三、十字链表

将同一行非0元串成一个单链表，同一列非0元也串成一个单链表。每个结点都处在行链和列链的交汇点上。

非0元结点



矩阵结点



想一想：

在十字链表上如何实现矩阵的运算？

- 创建十字链表
- 求和
- 转置
- 求积