第5章　数组和广义表

王　　勇
计算机/软件学院　　大数据分析与信息安全团队
21#533　　电　话 13604889411
课程网站：http://cstcsjjg.hrbeu.edu.cn
Email：　wangyongcs@hrbeu.edu.cn

哈尔滨工程大学
Harbin Engineering University

问题提出

- 数组存储与效率
- 矩阵运算
- 信息压缩存储
- 人工智能信息处理

本章说明

知识点

数组的类型定义
数组的存储表示
特殊矩阵的压缩存储表示方法
稀疏矩阵的压缩存储表示方法
广义表

重 点

数组类型的定义及其存储表示

理解

掌握

充分理解

领会

数组类型的特点

稀疏矩阵的两类存储压缩方法的特点及其适用范围

三元组表示稀疏矩阵的矩阵运算处理方法

标

本章内容

- 数组是线性表的推广
  - 数组可以看成是一种特殊的线性表，即线性表中数据元素本身也是一个线性表

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

列向量

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

行向量

定 义

ADT Array {

数据对象：$j_i=0,..., b_i-1, i=1,2,..,n$

$D=\{a_{j_1,j_2,\cdots j_n} | n(>0)$为数组的维数，$b_i$为数组第i维的长度，$j_i$为数组元素的第i维下标，$a_{j_1,j_2,\cdots j_n} \in ElemSet\}$

数据关系：$R=\{R1, R2, ..., Rn\}$

$Ri=\{< a_{j_1\cdots j_i\cdots j_n}, a_{j_1\cdots j_i+1,\cdots j_n} > |$

$0\le j_k\le b_k-1, 1\le k\le n$ 且 $k\ne i$,

$0\le j_i\le b_i-2$, $a_{j_1\cdots j_i\cdots j_n}, a_{j_1\cdots j_i+1,\cdots j_n} \in D$, $i=2,...,n\}$

基本操作：

InitArray(&A, n, bound1, ..., boundn)

操作结果：若维数 n 和各维长度合法，则构造相应的数组A

DestroyArray(&A)

初始条件：数组 A 已经存在。

操作结果：销毁数组 A。

Value(A, &e, index1, ..., indexn)

初始条件：A 是 n 维数组，e 为元素变量，随后是 n 个下标值

操作结果：若各下标不超界，则e赋值为所指定的A的元素值，并返回OK。

Assign(&A, e, index1, ..., indexn)

初始条件：A 是 n 维数组，e 为元素变量，随后是 n 个下标值

操作结果：若下标不超界，则将 e 的值赋给A中指定下标的元素

} ADT Array

连续的存储单元——数组

以行序

对二维数组进行“按行切分”，即将数组中的数据元素“按行依次排放”在存储器中

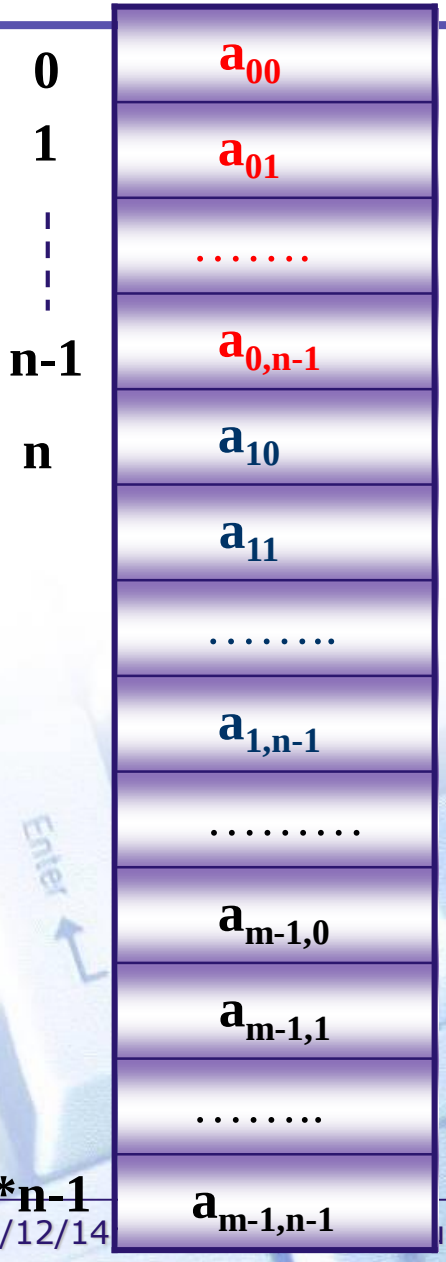
$$A_{m\times n}=\begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

以列序

对二维数组进行“按列切分”，即将数组中的数据元素“按列依次排放”在存储器中

- 以行序为主存储

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

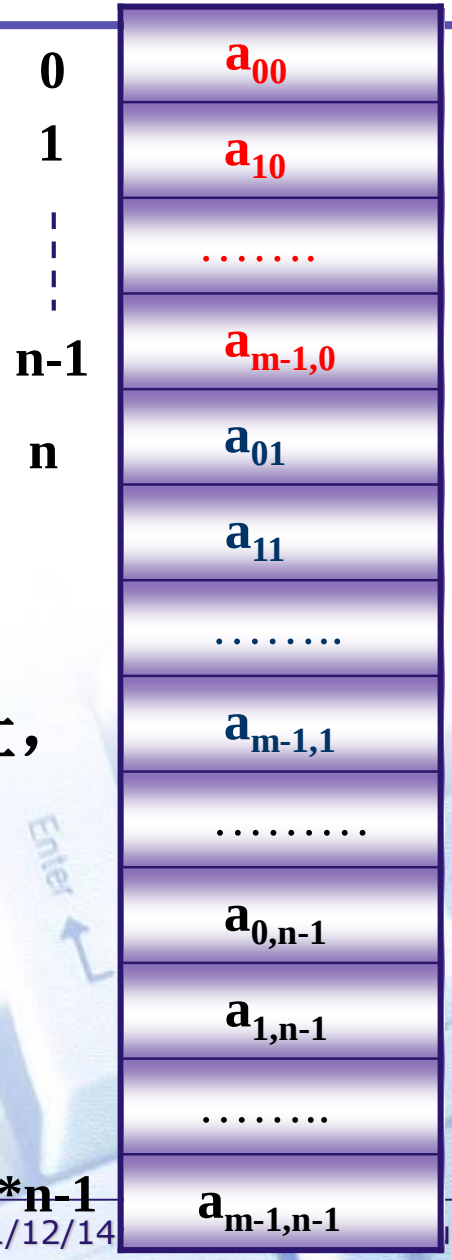


- 每个数据元素占L个存储单元;
- LOC(0,0)表示数据元素a_{00}的存储地址是数组的起始地址（基地址）;
- LOC(i,j)表示下标为(i,j)的数据元素a_{ij}的存储地址

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

- 以列序为主存储

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



- 每个数据元素占L个存储单元;
- LOC(0,0)表示数据元素a_{00}的存储地址，是数组的起始地址（基地址）
- LOC(i,j)表示下标为(i,j)的数据元素a_{ij}的存储地址

$$LOC(i,j)=LOC(0,0)+(j*m+i)*L$$

$$\begin{bmatrix} 7 & 2 & 3 & 4 \\ 2 & 7 & 1 & 9 \\ 3 & 1 & 7 & 8 \\ 4 & 9 & 8 & 6 \end{bmatrix} \quad \begin{bmatrix} 7 & 0 & 0 & 0 \\ 2 & 7 & 0 & 0 \\ 3 & 1 & 7 & 0 \\ 4 & 9 & 8 & 6 \end{bmatrix} \quad \begin{bmatrix} 4 & 6 & 0 & 0 & 0 \\ 8 & 5 & 16 & 0 & 0 \\ 0 & 7 & 8 & 2 & 0 \\ 0 & 0 & 1 & 9 & 3 \\ 0 & 0 & 0 & 2 & 12 \end{bmatrix} \quad \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 8 & 0 & 16 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

- 压缩存储
  - 为多个值相同的矩阵元只分配一个存储空间；对零元不分配空间

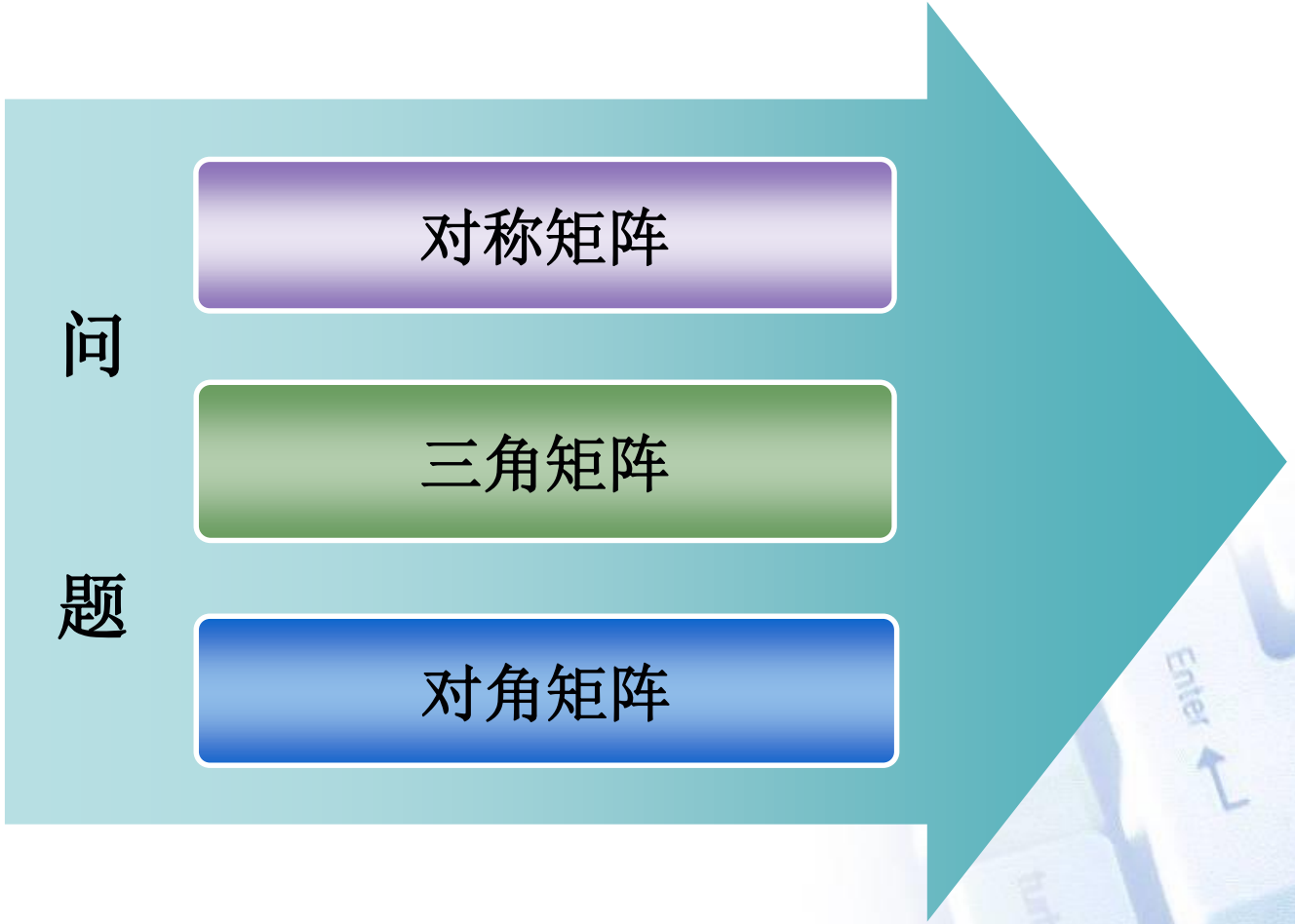
这帧图像
如何存储

问题

特殊矩阵

稀疏矩阵

压缩


问
题
对称矩阵
三角矩阵
对角矩阵
压缩

- 值相同的元素或者零元素在矩阵中的分布有一定规律

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

- n阶矩阵
- $a_{ij}=a_{ji}$　　$1 \le i,j \le n$

- 压缩存储——对称矩阵

仅存储下三角

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots\cdots & \cdots\cdots & a_{2n} \\ & & \cdots\cdots\cdots\cdots\cdots & & \\ a_{n1} & a_{n2} & \cdots\cdots & & a_{nn} \end{bmatrix}$$

按行序为主序：

a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	…	a_{n1}	…	a_{nn}
k=0	1	2	3	4				$\frac{n(n+1)}{2}-1$

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

- 值相同的元素或者零元素在矩阵中的分布有一定规律

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

  - n阶矩阵
  - 下（上）三角矩阵：矩阵的上（下）三角（不包括对角线）中的元均为常数c或零

矩阵的压缩存储　　特殊矩阵—三角矩阵

- 压缩存储——下三角矩阵

仅存储下三角

$$\begin{bmatrix} a_{11} & 1 & 1 \cdots\cdots & 1 \\ a_{21} & a_{22} & 1 \cdots\cdots & 1 \\ & & \cdots\cdots\cdots\cdots & 1 \\ a_{n1} & a_{n2} & a_{n3} \cdots\cdots & a_{nn} \end{bmatrix}$$

按行序为主序：

a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	…	a_{n1}	…	a_{nn}	1
k=0	1	2	3	4					$\frac{n(n+1)}{2}$

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

- 值相同的元素或者零元素在矩阵中的分布有一定规律

$$\begin{bmatrix} 4 & 6 & 0 & 0 & 0 \\ 8 & 5 & 16 & 0 & 0 \\ 0 & 7 & 8 & 2 & 0 \\ 0 & 0 & 1 & 9 & 3 \\ 0 & 0 & 0 & 2 & 12 \end{bmatrix}$$

  - n阶矩阵
  - 所有的非零元都集中在以主对角线为中心的带状区域中

- 压缩存储——对角矩阵

仅存储主对角线非0元素

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & & \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \cdots & 0 \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & 0 & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \cdots & \cdots & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

按行序为主序：

a_{11}	a_{12}	a_{21}	a_{22}	a_{23}	…	$a_{n,n-1}$	$a_{n,n}$
k=0	1	2	3	4			

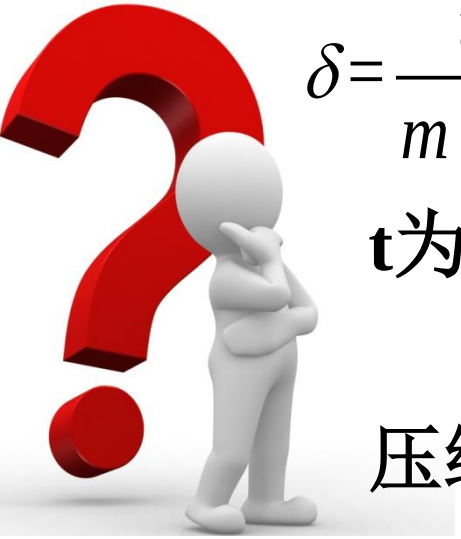
稀疏矩阵压缩存储——三元组顺序表

稀疏矩阵压缩存储——十字链表

矩阵的压缩存储　　稀疏矩阵—三

仅存储非0元素

- 稀疏矩阵



$$\delta = \frac{t}{m \times n} \leq 0.05$$

t为非0元个数

压缩存储？

$$M = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6\times 7}$$

- 非零元较零元少，且分布没有一定规律的矩阵
- 压缩存储原则：只存矩阵的行列维数和每个非零元的行列下标及其值

矩阵的压缩存储　　稀疏矩阵—三元组顺序表

- 稀疏矩阵—压缩存储

$$M=\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6\times 7}$$

- ❖ 矩阵维数
  - ✓（6,7,8）
- ❖非零元
  - ✓(1,2,12)
  - ✓(1,3,9)
  - ✓(3,1,-3)
  - ✓(3,6,14)
  - ✓(4,3,24)
  - ✓(5,2,18)
  - ✓(6,1,15)
  - ✓(6,4,-7)

用三元组
存储非0元

- 顺序存储结构表示

```
#define MAXSIZE 12500    //设非零元个数最大值为12500
typedef struct{
        int        i,j;              //该非零元的行下标和列下标
        ElemType e;
    }Triple;
typedef struct{
    Triple data[MAXSIZE+1];   //非零元三元组表，
                                          data[0]未用
    int mu, nu, tu;              //矩阵的行数、列数和非零元个数
    }TSMatrix ;
```

行序

矩阵的压缩存储　　稀疏矩阵—三元组顺序表

$$M=\begin{bmatrix}0&12&9&0&0&0&0\\0&0&0&0&0&0&0\\-3&0&0&0&0&14&0\\0&0&24&0&0&0&0\\0&18&0&0&0&0&0\\15&0&0&-7&0&0&0\end{bmatrix}_{6\times7}$$

行列下标　　非零元值

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

data

data[0].i,data[0].j,data[0].e分别存放矩阵行列维数和非零元个数

示　例　求转置矩阵

已知一个稀疏矩阵的三元组表，求该矩阵转置矩阵的三元组表

思　路

- 将矩阵行、列维数互换；
- 将每个三元组中的i和j相互调换；
- 重排三元组次序

$$M=\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6\times 7}$$

$$N=\begin{bmatrix} 0 & 0 & -3 & 0 & 0 & 15 \\ 12 & 0 & 0 & 0 & 18 & 0 \\ 9 & 0 & 0 & 24 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -7 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{7\times 6}$$

M.data

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

?

T.data

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

方法一　按矩阵的列序转置

按T.data中三元组次序依次在M..
应的三元组进行转置

Col从1~n
在M中找
每1列

M.data

	i	j	e
0	6	7	8
p→1	1	2	12
p→2	1	3	9
p→3	3	1	-3
p→4	3	6	14
p→5	4	3	24
p→6	5	2	18
p→7	6	1	15
p→8	6	4	-7

col=1

col=2

T.data

	i	j	e
0	7	6	8
q→1	1	3	-3
q→2	1	6	15
q→3	2	1	12
q→4	2	5	18
q→5	3	1	9
6	3	4	24
7	4	6	-7
8	6	3	14

方法一　按矩阵的列序转置的算法思想

(1) 令T.mu=M.nu, T.nu=M.mu, T.tu=M.tu, col=1，q=1
(转置矩阵T的三元组下标初始化，q指向转置后的第1个位置) p=1 (指向M的第1个位置)
(2) 如p所指元素的列下标=col，则送q所指位置，q++
(3) p++，若p<=M.tu(1~最后1个非0元)，则(2)，否(4)
(4) col++，如果col<=M.nu，p=1,转(2)，否(5)
(5) 结束

适用于
tu<<mu*tu

算法实现

Click

时间复杂度为O(nu×tu)

求稀疏矩阵的转置矩阵算法

```
Status TransposeSMatrix(TSMatrix M, TSMatrix &T)
{     //采用三元组表存储表示，求稀疏矩阵M的转置矩阵T
   T.mu=M.nu; T.nu=M.mu; T.tu=M.tu; //T的维数和非零元个数
   if(T.tu){       //M存在非零元
     q=1;          //T的序号
     for(col=1;col<=M.nu;++col)  //扫描M的所有列
         for(p=1;p<=M.tu;++p)    //扫描当前M中的每个三元组
            if(M.data[p].j == col)   //交换
                          {   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;
                          }
            }
   return OK;
}//TransposeMatrix
```

方法二　按矩阵的行序转置

- 按M.data中三元组次序转置，转置结果放入T.data中恰当位置。
- 此法关键是要预先确定M.data中每一列第一个非零元在T.data中位置。(cpot[col])
- 为确定这些位置，转置前应先求得M.data的每一列中非零元个数。(num[col])

设num[col]记录M中第col列中非0元个数，cpot[col]记录M中第col列的第1个非0元在T中的恰当位置，有：

cpot[1]=1;

cpot[col]=cpot[col-1]+num[col-1]　2≤col≤M.mu

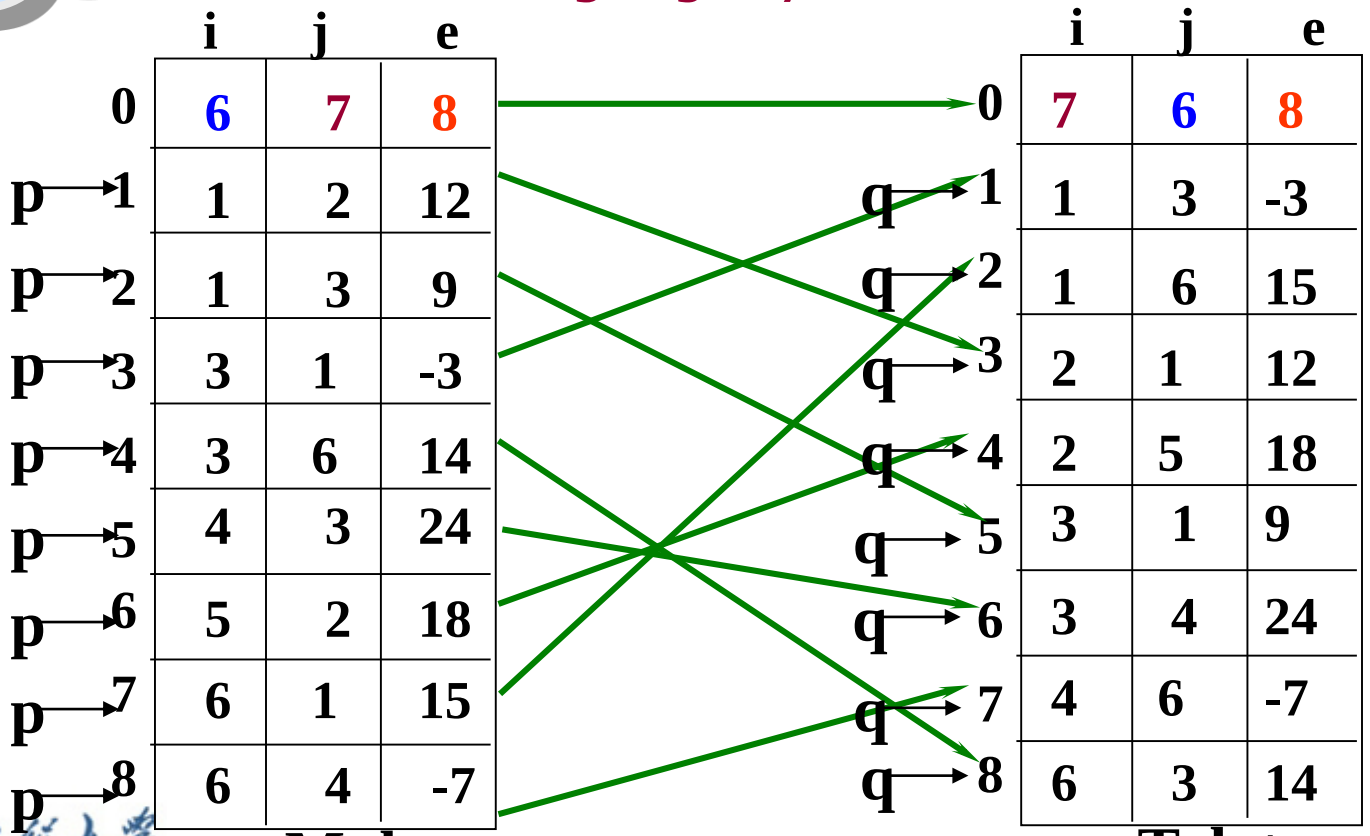
求稀疏矩阵的转置矩阵算法

M的列
即T的行

col	1	2	3	4	5	6	7
num[col]	2	2	2	1	0	1	0
cpot[col]	1	3	5	7	8	8	9

2 4 6 8 9

3 5 7



方法二　按矩阵的行序转置的算法思想

（1）由M求T的维数和非零元个数
（2）统计每一列的非零元个数num[]
（3）求每一列第一个非零元在T中的下标号cpot[]
（4）设p指向M的第1个非零元
（5）求该非零元的列号col，用q指向该列在T中的位置（q==cpot[col])
（6）从M到T转置（p指的
（7）p++，如p<=M.tu，
则（8）结束

用空间换取时间

算法实现

Click

时间复杂度为O(nu+tu)

求稀疏矩阵的转置矩阵算法

```
Status FastTransposeSMatrix(TSMatrix M, TSMatrix &T)
{    //采用三元组表存储表示，求稀疏矩阵M的转置矩阵T
   T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;     //T的维数和非零元个数
   if(T.tu)                             //M存在非零元
      { for(col=1;col<=M.nu;++col) num[col]=0;
        for(t=1;t<=M.tu;++t)
            ++num[M.data[t].j]; //M每一列非零元个数
            cpot[1]=1;    //第1列中第1个非0元素在T中的位置
        //求第col列中第一个非零元在T.data中的序号
        for(col=2;col<=M.nu;++col)
            cpot[col]=cpot[col-1]+num[col-1];
        for(p=1;p<=M.tu;++p)   //从M的第1个至最后1个转置
            { col=M.data[p].j;  q=cpot[col];  //拿j找其在T位号q
              T.data[q].i= M.data[p].j; T.data[q].j= M.data[p].i;
              T.data[q].e= M.data[p].e; ++cpot[col];
             }//for
      }//if
   return OK;
}//FastTransposeMatrix
```

三元组顺序表

优点

非零元在表中按行序有序存储，便于进行依行序处理的矩阵运算

缺点

若需按行号存取某一行的非零元，则需从头开始进行查找

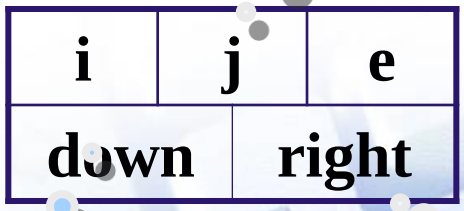
- 引入原因

  当矩阵的非零元的个数发生变化时，不宜采用三元组表。如A=A+B，非零元的插入或删除将会引起A.data中的数据移动，这是顺序结构三元组的弱势

非零元的行、列、值

- 十字链表结点形式

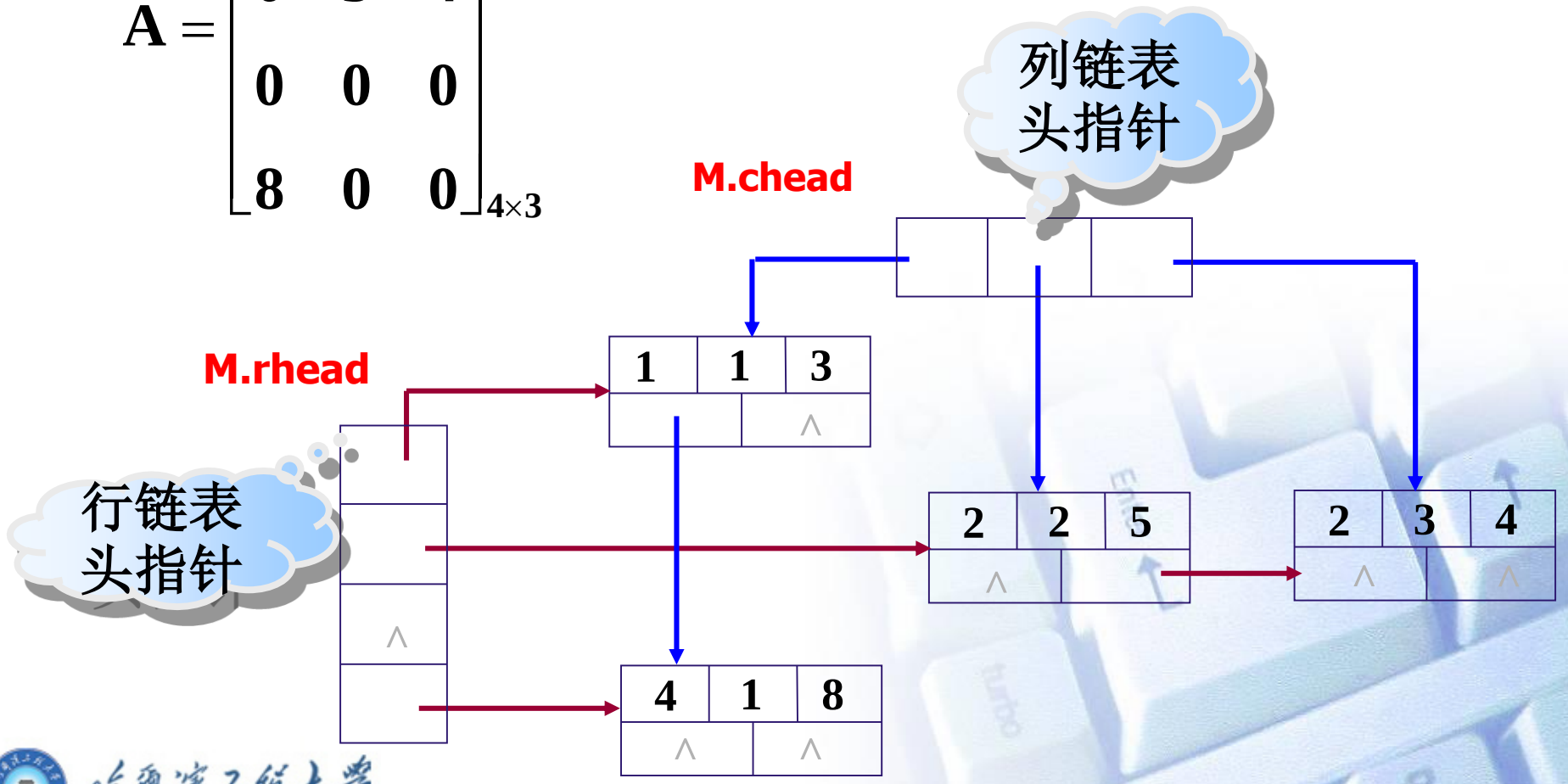
```
typedef struct OLNode{
    int                 i,j;
    ElemType            e;
    struct OLNode  *right,*down;
}OLNode; *OLink;
typedef struct
 { Olink  *rhead,*chead;
    int       mu, nu, tu;
}CrossList;
```



同一列下一个非零元

同一行下一个非零元

$$A=\begin{bmatrix}3&0&0\\0&5&4\\0&0&0\\8&0&0\end{bmatrix}_{4\times3}$$



❑建立十字链表的思想

m=4,n=3,t=5

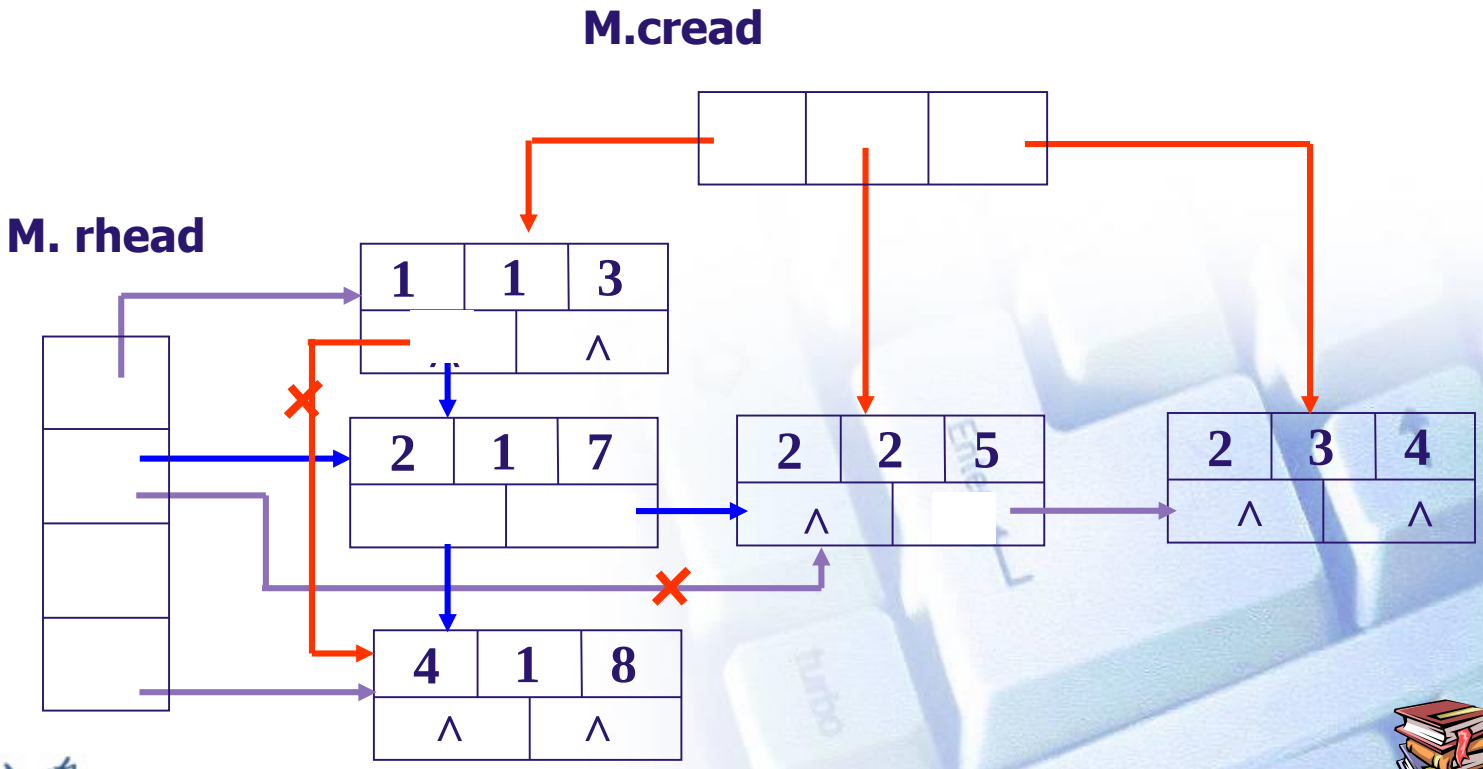
1,1,3

2,2,5

2,3,4

4,1,8

2,1,7



❑建立十字链表的算法思想

（1）初始化

（2）循环输入各非0元的(i,j,e)，直到最后一个，重复做: (3)-(5)

（3）申请结点，赋值

（4）寻找行i插入位置，插入

a)行首，M.rhead[i]==NULL || M.rhead[i]->j>j

b)行内，q=M.rhead[i] ;

(q->right)&&q->right.j<j ; q=q->right

（5）寻找列j插入位置，插入（同行）

建立十字链表的算法实现

Click

时间复杂度为O(t*s) ， s=max(m.n)

定 义

是线性表的推广，广泛地应用于人工智能等领域的表处理语言LISP语言。一般记作

$LS=(a_1, a_2, \ldots, a_n)$　　$(n\geq 0)$

术 语

名称：LS

长度：n

原子：a_i是单个元素，一般用小写字母a表示

子表：a_i是广义表，一般用大写字母A表示

表头(Head)：非空广义表 LS的第一个数据元素a_1

表尾(Tail)：非空广义表 LS除第一个数据元素外的其余数据元素构成的广义表 $(a_2, \ldots, a_n)$

特 性

数据元素有固定的相对次序

元素可以是子表，而子表的元素还可是子表

广义表可以为其它列表共享，可以是一个递归的表

A=()

F=(b,(c,d,e))

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

D=(E,A,F)

C=(A,D,F)

B=(a,B)=(a,(a,(a,v...)))

操 作 长度

A=()	A的长度为0
F=(b,(c,d,e))	F的长度为2
E=(b,(c),(d,e))	E的长度为3
D=(E,A,F)	D的长度为3
C=(A,D,F)	C的长度为3
B=(a,B)=(a,(a,(a,...)))	B的长度为2

广义表中元素的“长度”应由最外层括弧中的“逗号”来定

操 作

取表头、取表尾 F=(b,(c,d,e))

GetHead(F)=b

GetTail(F)=((c,d,e))=F1

GetHead(F1)=(c,d,e)=F2，GetTail(F1)=( )

GetHead(F2)=c，GetTail(F2)=(d,e)=F3

GetHead(F3)=d，GetTail(F3)=(e)=F4

GetHead(F4)=e，GetTail(F4)=( )

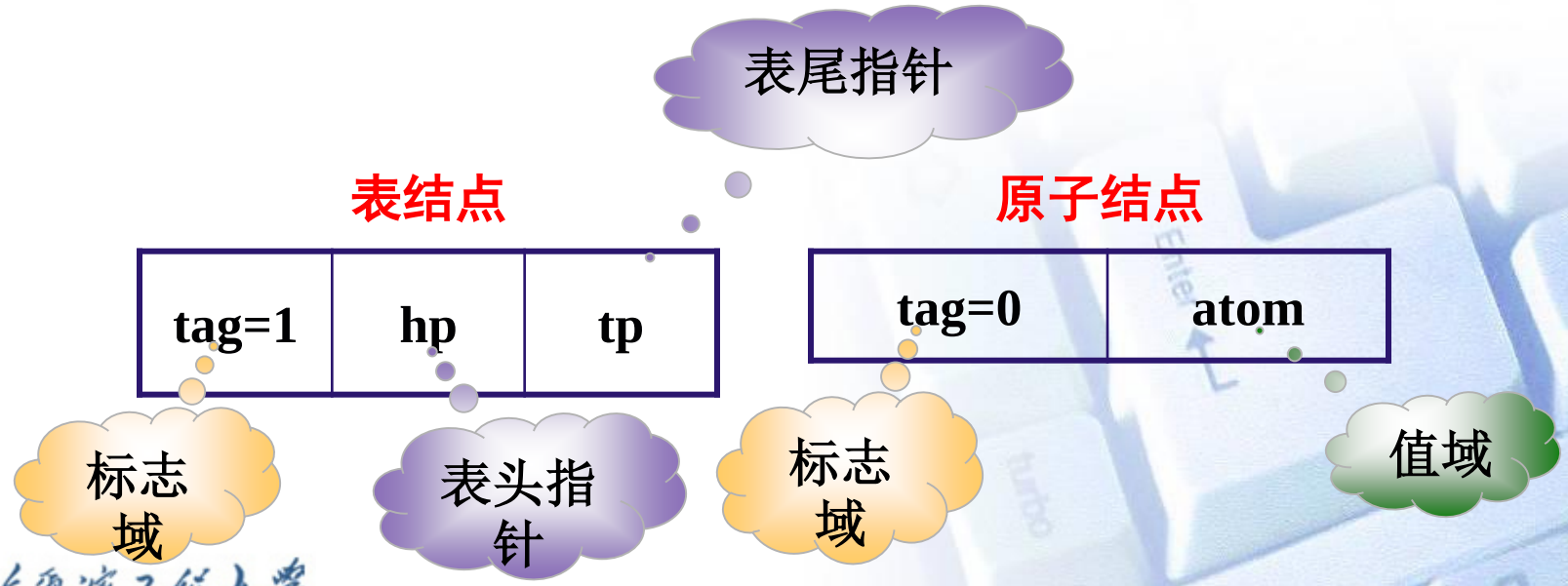
- 两个操作只对非空表有意义；
- 取表头的结果可能是原子，也可能是个广义表；
- 取表尾"必定"是个广义表，但可能是个空的广义表。

顺序存储？——不确定因素、缺乏灵活性、操作复杂

- 采用链式存储结构
- 两种结构的结点

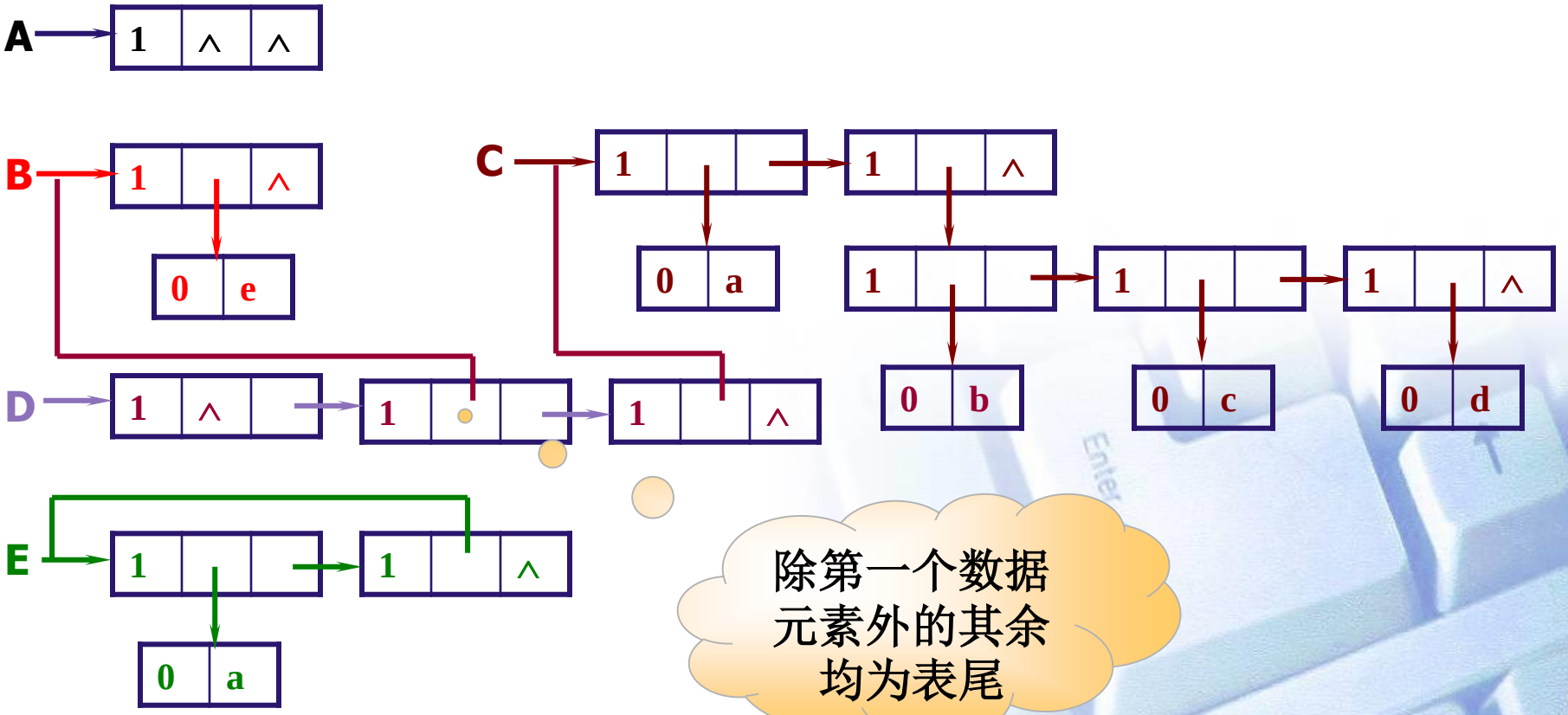
表结点：用来表示子表

原子结点：用来表示元素



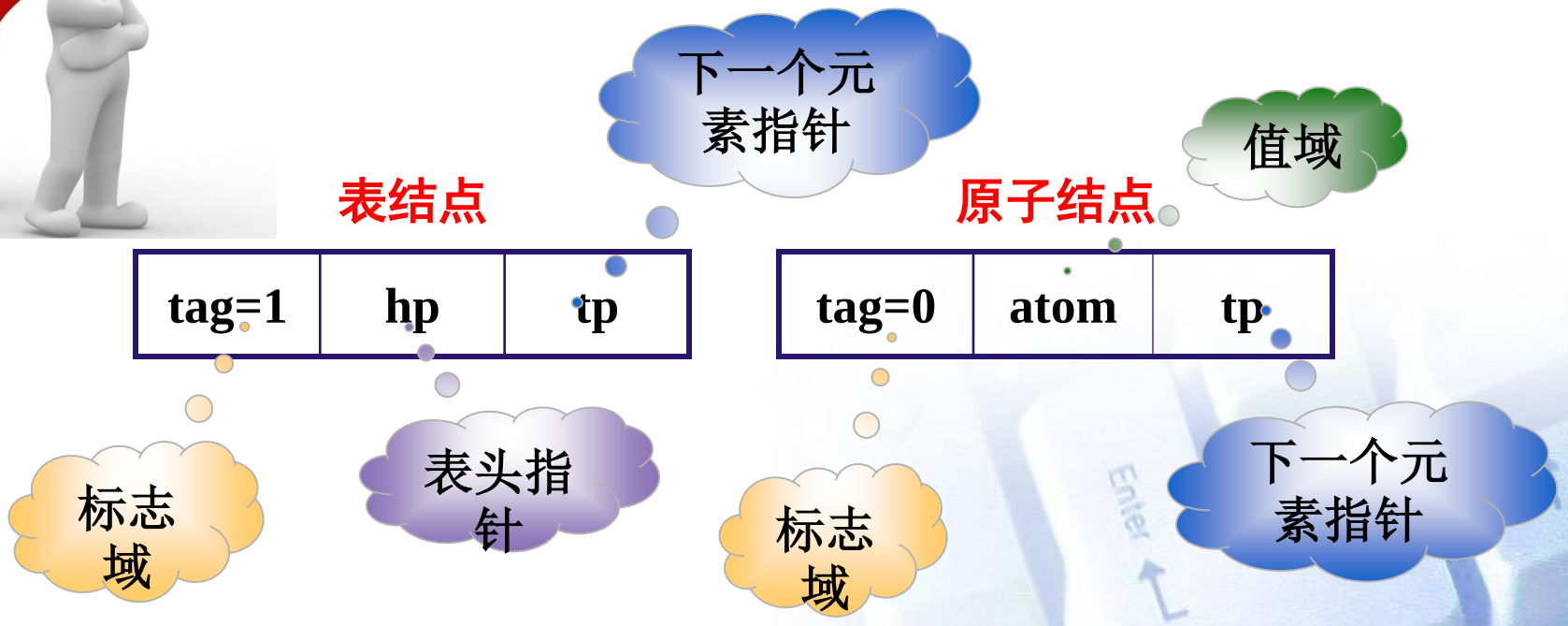
A=() B=(e) C=(a,(b,c,d))

D=(A,B,C) E=(a,E)



除第一个数据元素外的其余均为表尾

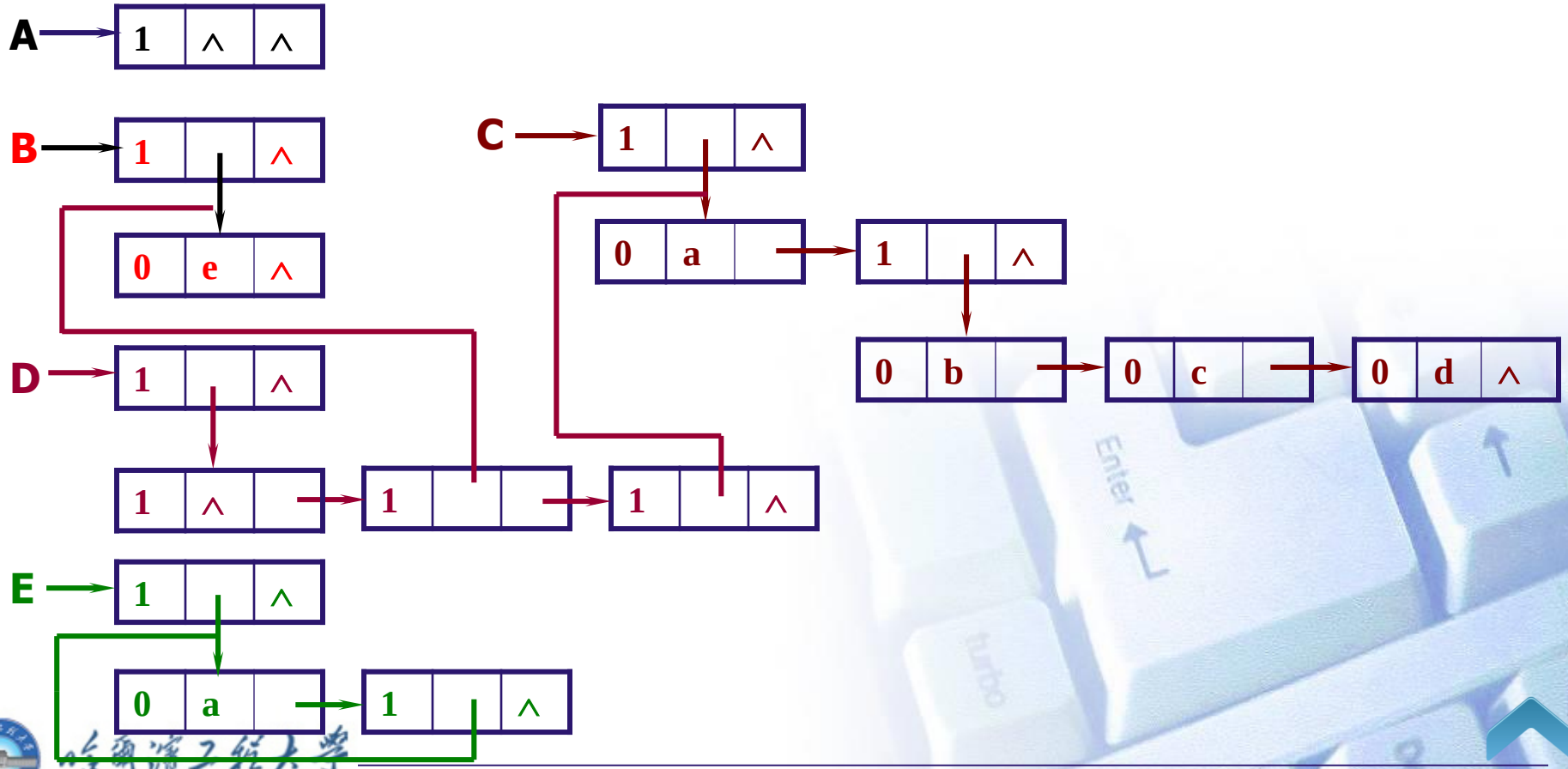
- 另一种结点结构


下一个元素指针
值域
表结点
原子结点
tag=1
hp
tp
tag=0
atom
tp
标志域
表头指针
标志域
下一个元素指针

广义表的存储结构

A=()　　B=(e)　　C=(a,(b,c,d))

D=(A,B,C)　　E=(a,E)



本章小结

- 了解数组的类型定义及其在高级语言中实现的方法
- 数组的特点是一种多维的线性结构，较多的操作是进行存取或修改某个元素的值，插入和删除操作较少，因此它主要采用顺序存储结构。
- 介绍了稀疏矩阵的两种表示方法。至于在具体应用问题中采用哪一种表示方法，取决于该矩阵主要进行什么样的运算
- 广义表是一种递归定义的线性结构，因此它兼有线性结构和层次结构的特点

下课休息一会！