第五章 数组和广义表

- 数组
- 稀疏矩阵
- 广义表

数　组

一维数组

- **定义**

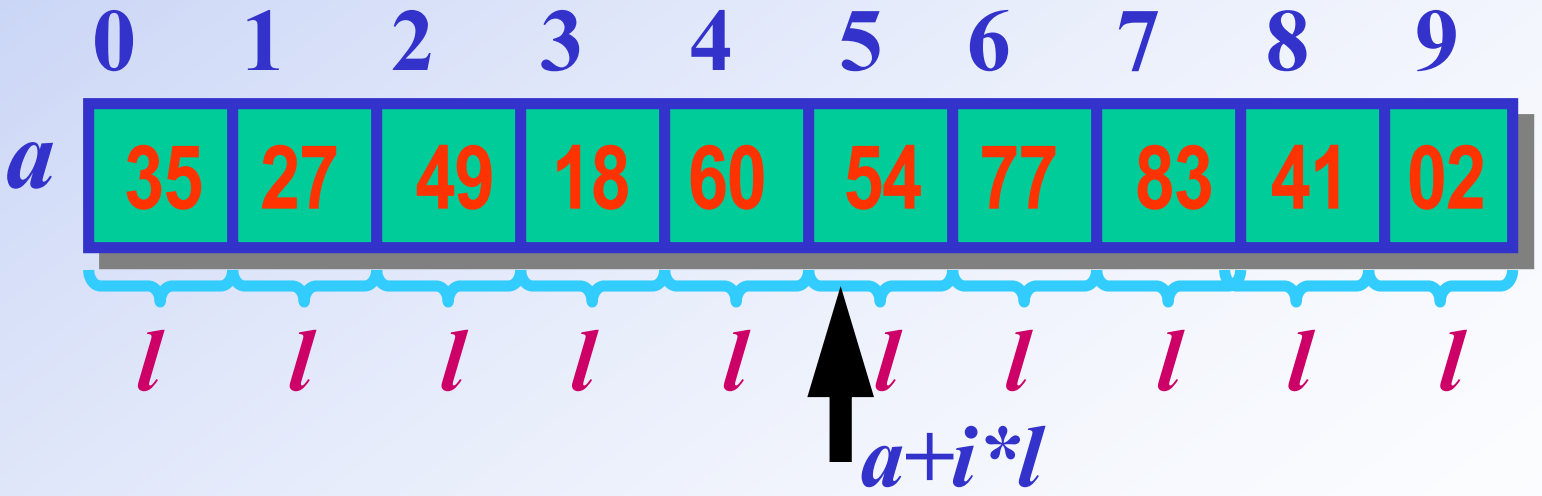
  相同类型的数据元素的集合。
- 一维数组的**示例**

数组的定义和初始化

```
main ( ) {
    int a1[3] = { 3, 5, 7 }, *elem;
    for ( int i = 0; i < 3; i++ )
        printf ( “%d”, a1[i], “\n” );     //静态数组
    elem = a1;
    for ( int i = 0; i < 3; i++ ) {
        printf ( “%d”, *elem, “\n” );   //动态数组
        elem++;
    }
}
```

一维数组存储方式

$$LOC(i)=\begin{cases} a, & i=0 \\ LOC(i-1)+l=a+i*l, & i>0 \end{cases}$$



$$LOC(i)=LOC(i-1)+l=a+i*l$$

二维数组

$$a=\begin{pmatrix} a[0][0] & a[0][1] & \cdots & a[0][m-1] \\ a[1][0] & a[1][1] & \cdots & a[1][m-1] \\ a[2][0] & a[2][1] & \cdots & a[2][m-1] \\ \vdots & \vdots & \ddots & \vdots \\ a[n-1][0] & a[n-1][1] & \cdots & a[n-1][m-1] \end{pmatrix}$$

行优先存放：

设数组开始存放位置 LOC(0, 0) = a, 每个元素占用 *l* 个存储单元

$$LOC(i,j)=a+(i*m+j)*l$$

三维数组

- 各维元素个数为 m_1, m_2, m_3
- 下标为 i_1, i_2, i_3的数组元素的存储地址：
- （按页/行/列存放）

$$LOC(i_1, i_2, i_3) = a + (i_1 * m_2 * m_3 + i_2 * m_3 + i_3) * l$$

$i_1 * m_2 * m_3$：前i_1页总元素个数

$i_2 * m_3$：第i_1页的前i_2行总元素个数

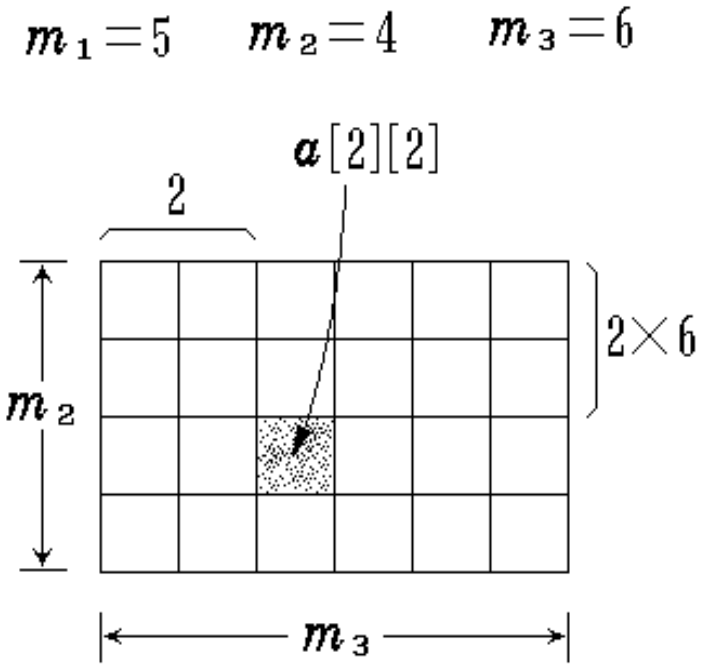
n 维数组

- 各维元素个数为 $m_1, m_2, m_3, \ldots, m_n$
- 下标为 $i_1, i_2, i_3, \ldots, i_n$ 的数组元素的存储地址：

$$LOC(i_1, i_2, \ldots, i_n) = a + (i_1*m_2*m_3*\ldots*m_n + i_2*m_3*m_4*\ldots*m_n + \ldots\ldots + i_{n-1}*m_n + i_n) * l$$

$$= a + \left(\sum_{j=1}^{n-1} i_j * \prod_{k=j+1}^{n} m_k + i_n \right) * l$$

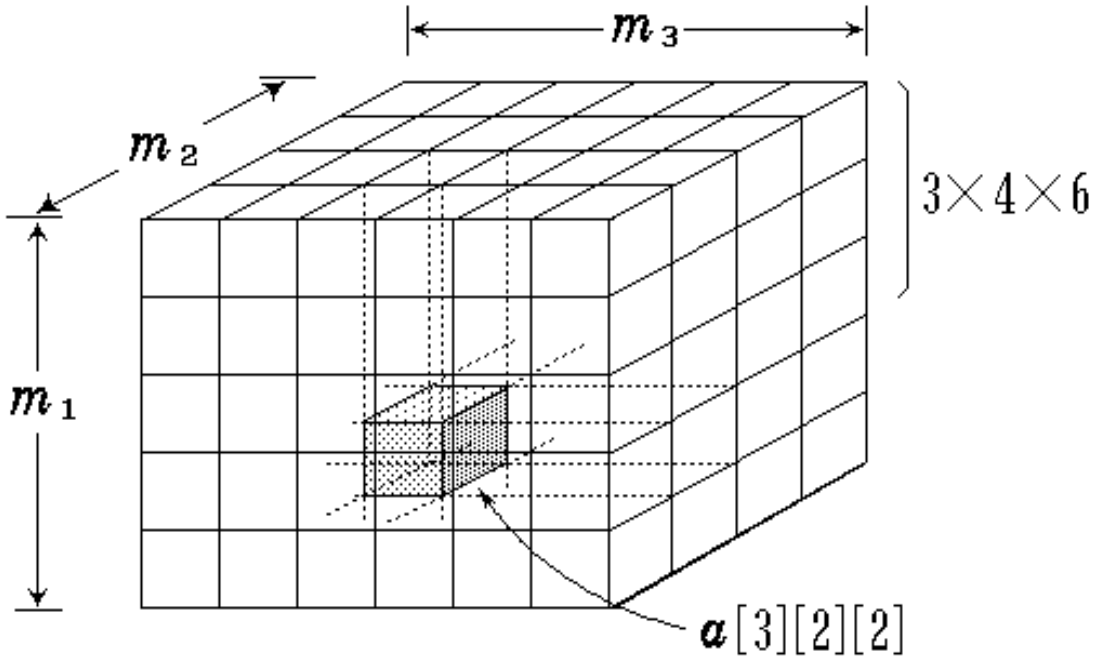
二维数组



行向量 下标 i

列向量 下标 j

三维数组



页向量 下标 i

行向量 下标 j

列向量 下标 k

特殊矩阵的压缩存储

- 特殊矩阵是指非零元素或零元素的分布有一定规律的矩阵。
- 特殊矩阵的压缩存储主要是针对阶数很高的特殊矩阵。为节省存储空间，对可以不存储的元素，如零元素或对称元素，不再存储。
  - 对称矩阵
  - 三对角矩阵

对称矩阵的压缩存储

设有一个 n×n 的对称矩阵 A。

$$A=\begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1n-1} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-10} & a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} \end{bmatrix}$$

在矩阵中，$a_{ij} = a_{ji}$

- 为节约存储空间，只存对角线及对角线以上的元素，或者只存对角线及对角线以下的元素。前者称为上三角矩阵，后者称为下三角矩阵。
- 把它们按行存放于一个一维数组 B 中，称之为对称矩阵 A 的压缩存储方式。
- 数组 B 共有 $n + (n - 1) + \cdots + 1 = n*(n+1)/2$ 个元素。

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

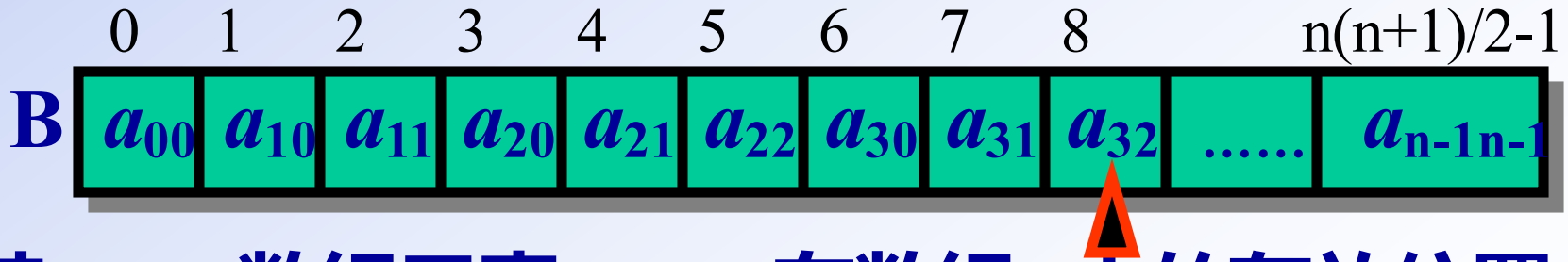
上三角矩阵

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

下三角矩阵

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1n-1} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-10} & a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} \end{bmatrix}$$

下三角矩阵



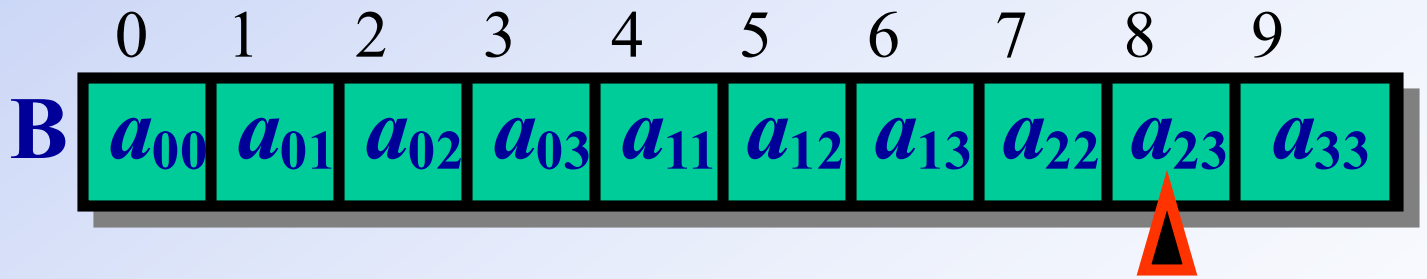
若 i ≥ j, 数组元素A[i][j]在数组B中的存放位置为 $1 + 2 + \cdots + i + j = (i + 1)* i / 2 + j$

前i行元素总数　第i行第j个元素前元素个数

若 i < j，数组元素 A[i][j] 在矩阵的上三角部分，在数组 B 中没有存放，可以找它的对称元素A[j][i]：= j *(j +1) / 2 + i

上三角矩阵

$$n=4 \quad \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$



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

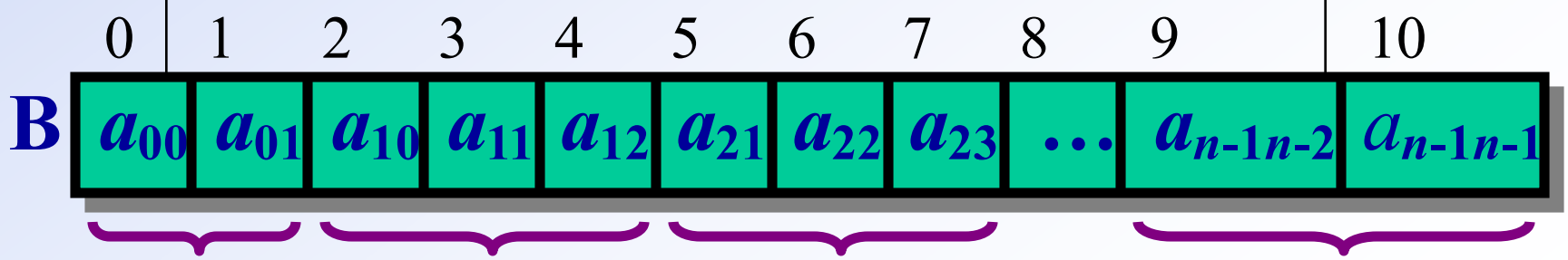
$$n + (n-1) + (n-2) + \cdots + (n-i+1) + j-i =$$
$$= (2*n-i+1) * i / 2 + j-i =$$
$$= (2*n-i-1) * i / 2 + j$$

若$i > j$，数组元素A[i][j]在矩阵的下三角部分，在数组 B 中没有存放。因此，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。

三对角矩阵的压缩存储

$$A=\begin{bmatrix} a_{00} & a_{01} & 0 & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 & 0 \\ 0 & a_{21} & a_{22} & a_{23} & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & a_{n-2n-3} & a_{n-2n-2} & a_{n-2n-1} \\ 0 & 0 & 0 & 0 & a_{n-1n-2} & a_{n-1n-1} \end{bmatrix}$$



- 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。
- 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]。
- 在三条对角线上的元素a_{ij} 满足

$$0 \le i \le n-1,\ i-1 \le j \le i+1$$

- 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。

稀疏矩阵 (Sparse Matrix)

$$A_{6\times7}=\begin{pmatrix}0&0&0&22&0&0&15\\0&11&0&0&0&17&0\\0&0&0&-6&0&0&0\\0&0&0&0&0&39&0\\91&0&0&0&0&0&0\\0&0&28&0&0&0&0\end{pmatrix}$$

非零元素个数远远少于矩阵元素个数

稀疏矩阵的抽象数据类型

```
template<class Type> class SparseMatrix;

template<class Type> class Trituple { //三元组
friend class SparseMatrix <Type>
private:
    int row, col;      //非零元素行号/列号
    Type value;        //非零元素的值
}
template <class Type> class SparseMatrix {
//稀疏矩阵类定义
```

```
    int Rows, Cols, Terms;  //行/列/非零元素数
    Trituple<Type> smArray[MaxTerms];
public:                                  //三元组表
    SparseMatrix (int MaxRow, int Maxcol);
    SparseMatrix<Type>& Transpose
      (SparseMatrix<Type>&);                    //转置
    SparseMatrix<Type>& Add (SparseMatrix
      <Type> a, SparseMatrix<Type> b);  //相加
    SparseMatrix<Type>& Multiply(SparseMatrix
      <Type> a,  SparseMatrix<Type> b);  //相乘
}
```

稀疏矩阵

$$\begin{pmatrix} 0 & 0 & 0 & 22 & 0 & 0 & 15 \\ 0 & 11 & 0 & 0 & 0 & 17 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 39 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	行 (row)	列 (col)	值 (value)
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	-6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28

转置矩阵

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 91 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 28 \\ 22 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 39 & 0 & 0 \\ 15 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	行 (row)	列 (col)	值 (value)
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	−6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

	行 (row)	列 (col)	値 (value)
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	−6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28

	行 (row)	列 (col)	値 (value)
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	−6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

- 假设以三元顺序存储结构来表示三元表，可得稀疏矩阵的一种压缩存储方式——三元组顺序表。
- //------稀疏矩阵的三元组顺序表存储表示------//

- #define MAXSIZE 12500//假设非零元素////的个数的最大值为12500
- Typedef struct {
- int i,j; //该非零元素的行下标和列下标
- ElemType e;
- } Triple;
- Typedef struct{
- Triple data[MAXSIZE+1];;
- int mu,nu,tu;//矩阵的行数、列数和非零元素个数
- }TSMatrix;

- 两个矩阵实现转制:
- (1)将矩阵的行列值相互交换;
- (2)将每个三元组的i和j相互调换;
- (3)重排三元组之间的次序便可实现矩阵的转制.

- 按照b.data中三元组次序依次在a.data中找到相应的三元组进行转换.换句话说,按照矩阵的M的列序来进行转制.为了找到M的每一列所有的非零元素,需要对其三元组标的a.data从第一行起整个扫描一遍,由于a.data是以M的行序为主序来存放每个非零元的,由此得到的恰是b.data应有的顺序.

```
Status TransposeSmtrix(TSMatrix M,TSMatrix &T){
        T.mu=M.nu; T.nu=M.nu; T.tu=M.tu;
        if(T.tu){
            q=1;
            for(col=1;col<=M.nu;++col)
                for(p=1;p<=M.tu;++p)
                    if(M.data[p].j==col){

    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;
                                        }
        }
}
```

广义表 (General Lists)

- **广义表的概念** n (≥ 0)个表元素组成的有限序列，记作

$$LS = (a_0, a_1, a_2, \ldots, a_{n-1})$$

LS是表名，a_i是表元素，它可以是表 (称为子表)，可以是数据元素(称为原子)。

- n为表的长度。 n = 0 的广义表为空表。
- n > 0时，表的第一个表元素称为广义表的表头(head)，除此之外，其它表元素组成的表称为广义表的表尾(tail)。

例如

- A=( ); //A是一个空表
- B=(e); //表B有一个原子
- C=(a,(b,c,d)); //两个元素为原子a和子表(b,c,d)
- D=(A,B,C); //有三个元素均为列表
- E=(a,E);//递归的列表

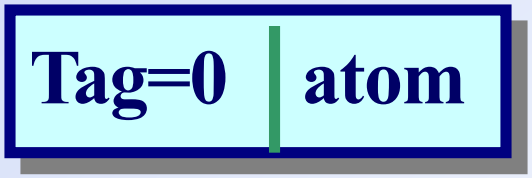
- 从上述定义和例子可推出列表的3个重要结论:
- (1)列表的元素可以是子表,而子表的元素还可以是子表
- (2)列表可为其他列表所共享.
- (3)列表可以是一个递归的表,即列表也可以是其本身的一个子表.

- GetHead(B)=e,GetTail(B)=()
- GetHead(D)=A;GetTail(D)=(B,C),
- 值得提醒的是列表()和(())不同.前者为空表,长度n=0;后者长度n=1,可分解得到其表头,可分解得到其表头,表尾均为空表().

广义表存储结构

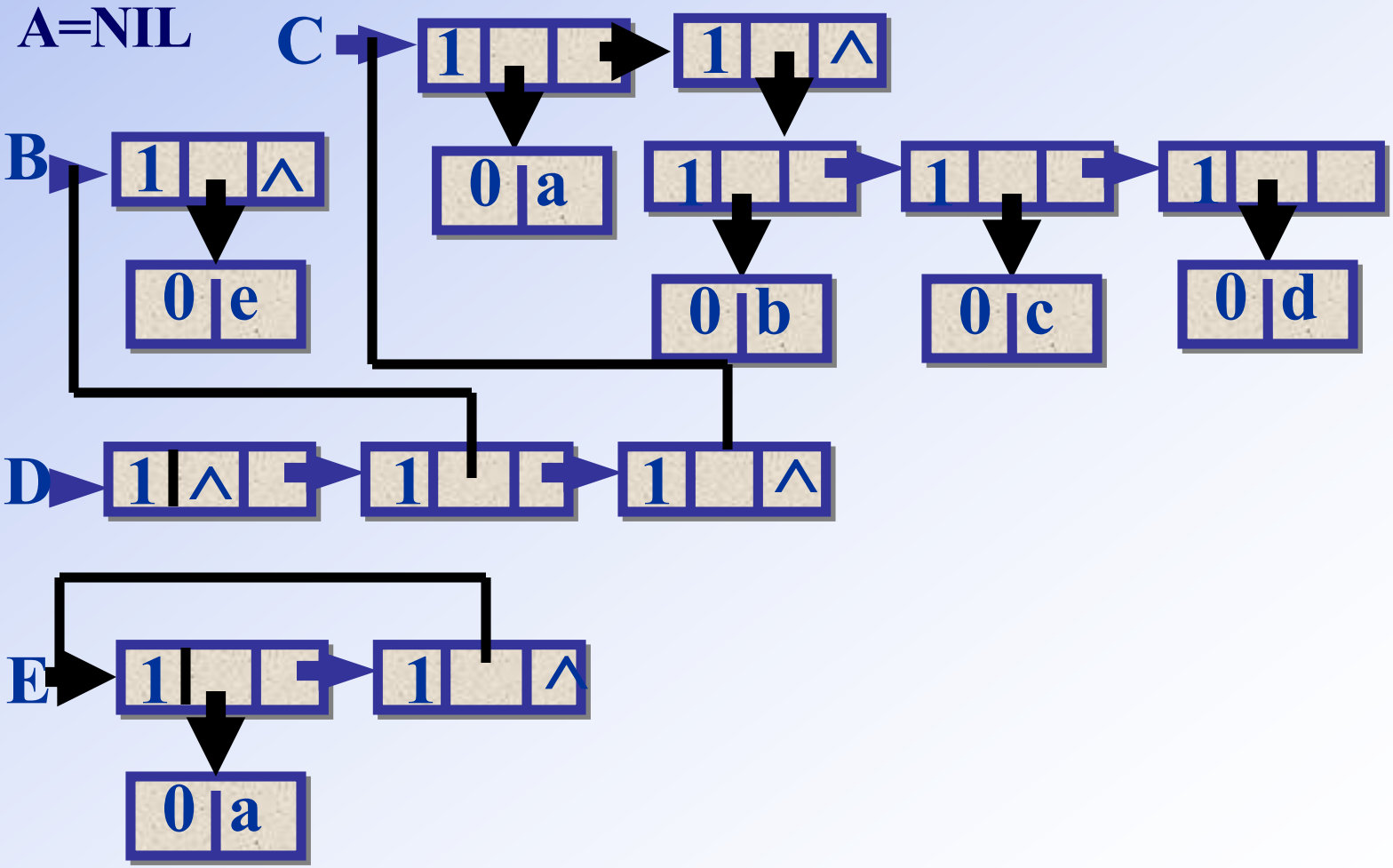
表结点

原子结点




```
typedef struct GLNode{
        int tag;
        union{
                char atom;
                struct {structGLNode hp,*yp;}ptr;
        };
}
```

方法一



方法二