第四章 多维数组与字符串

**一、思考题（CDDBC ABBDB BDABB A）**

1.串是一种特殊的线性表，其特殊性体现在( ) 。

(A)可以顺序存储 (B)可以用链表存储

(C)数据元素是一个字符 (D)数据元素可以是多个字符

2.串是（ ）。

(A)少于一个字母的序列 (B)任意个字母的序列

(C)不少于一个字符的序列 (D)有限个字符的序列

3.串的长度是（ ）。

(A)串中不同字母的个数 (B)串中不同字符的个数

(C)串中所含字符的个数，且大于0　　 (D)串中所含字符的个数

4.设有两个串p和q,求q在p中首次出现的位置的运算( ).

(A)连接　　 (B)模式匹配

(C)求子串　 (D)求串长

5.若某串的长度小于一个常数，则采用( )存储方式最为节省空间。

(A)链式　 (B)堆结构　 (C)顺序

6.串中任意多个连续字符组成的子序列称为该串的子串( ).

(A)正确 (B)不正确

7.如果两个串含有相同的字符集，则说两者相等( ).

(A)正确 (B)不正确

8.存取数组中任一元素的时间都是相等的，这种存取方式为（ ）存取方式。

(A)顺序 (B)随机 (C)线性 (D)非线性

9.设一个一维数组第一个元素的存储单元的地址是100，每个元素的长度是6，则它的第5个元素的地址是（ ）。

(A)130 (B)105 (C)106 (D)124

10.设n阶方阵是一个上三角矩阵，则需要存储的元素个数是（　）。

(A)n2/2 (B)n(n+1)/2 (C)n (D)n2

11.对一些特殊矩阵采用压缩存储的目的主要是为（ ）。

(A)表达变得简单 (B)减少不必要的存储空间的开销

(C)去掉矩阵中的多余元素 (D)对矩阵元素的存取变得简单

12.三元组表不包括（ 　）。

(A) 行数 (B) 列数 (C) 元素值 (D) 元素总数

13.设已知一个稀疏矩阵的三元组如下：(1,2,3),(1,6,1), (3,1,5),(3,2,-1),(4,5,4),(5,1,-3),则其转置矩阵的三元组表中第3个三元组为（ ）。

(A) (2,1,3) (B) (3,1,5)

(C) (3,2,-1) (D) (2,3,-1)

14.若采用三元组压缩技术存储稀疏矩阵，只要把每个元素的行下标和列下标互换，就完成了对该矩阵的转置运算，这种观点（ ）

(A)正确 (B)不正确

15.两维数组是一种非线性结构。（ ）

(A)正确 (B)不正确

16.数组A三维的长度分别为b3,b2,b1；每个数组元素占一个存储单元；LOC[0，0，0]为基址。若以行序为主序，则元素A[i][j][k]的地址为( )(其中0<=i<b3,0<=j<b2,0<=k<b1)

(A)LOC[0,0,0]+i\*b2\*b1+j\*b1+k

(B)LOC[0,0,0]+i\*b3\*b2+j\*b1+k

(C)LOC[0,0,0]+b3\*i+b2\*j+k

(D)LOC[0,0,0]+b3\*i\*j+b2\*j+k

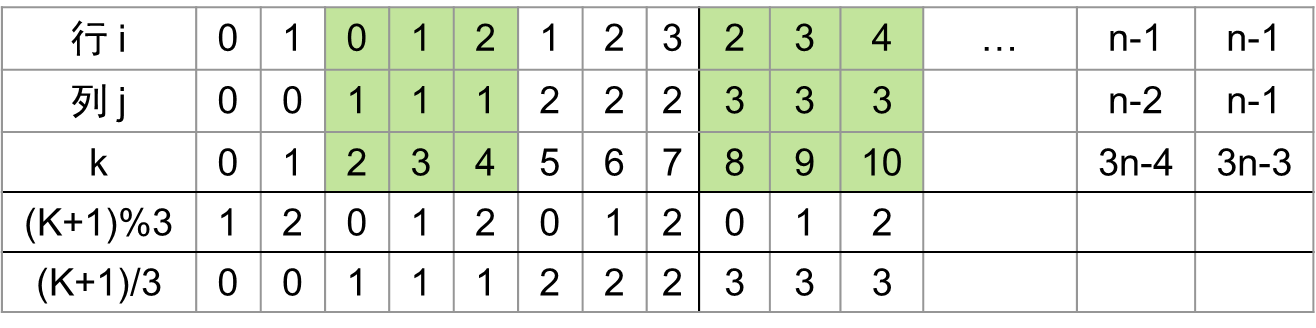
**二、应用题**

**1.设三对角矩阵An×n按列优先顺序，压缩存储在数组B[3\*n-2]之中，求：  
(1)用i,j表示k的下标变换公式；  
(2)用k表示i,j的下标变换公式。**

(1)求k

思路：设k=ai+bj+c，代入具体点a00、a10、a01。

解得：a=1,b=2,c=0,故**k=i+2j**（0≤i＜n; 0≤ j＜n）



**j=(k+1)/3** 0 ≤ k ＜ 3n-2

**i=(k+1)/3+ (k+1) % 3-1** [or **i=k-2j=k-((k+1)/3)\*2** ]

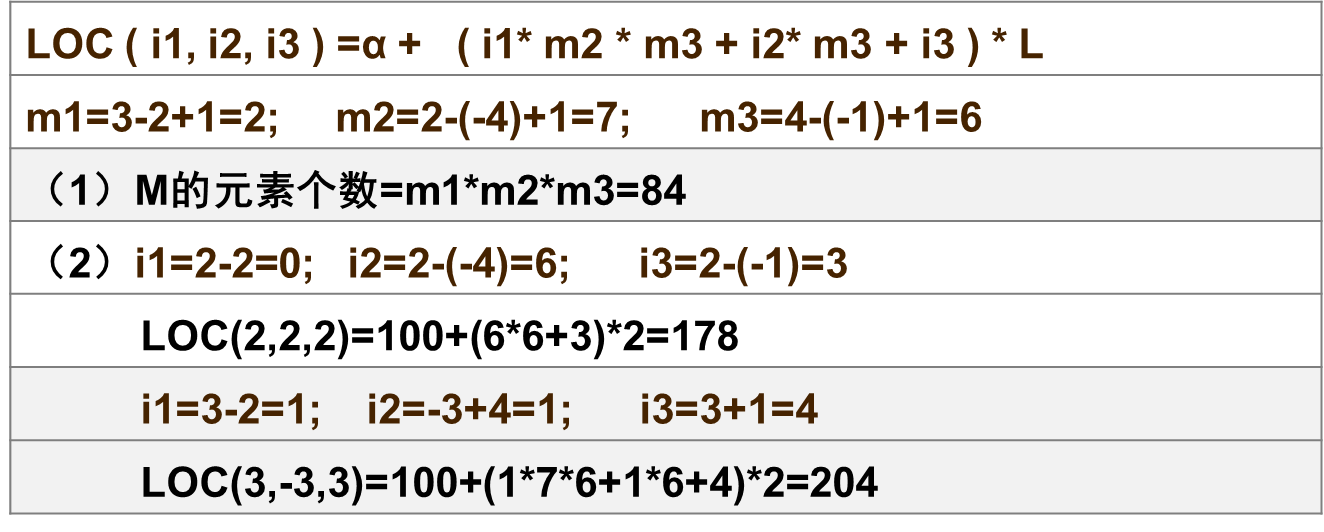
**2.已知三维数组M[2…3, - 4…2, -1…4]，且每个元素占用2个存储单元，起始地址为100，按行下标优先顺序存储，求：**

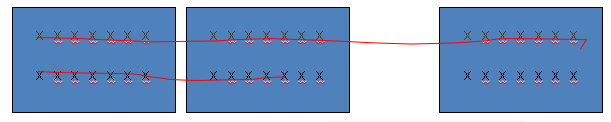
**(1) M所含的数据元素个数；**

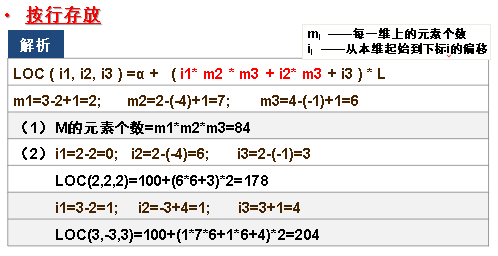
**(2) M[2,2,2]，M[3,-3,3]的存储地址。**

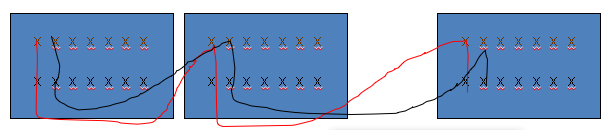
**解析：**

****

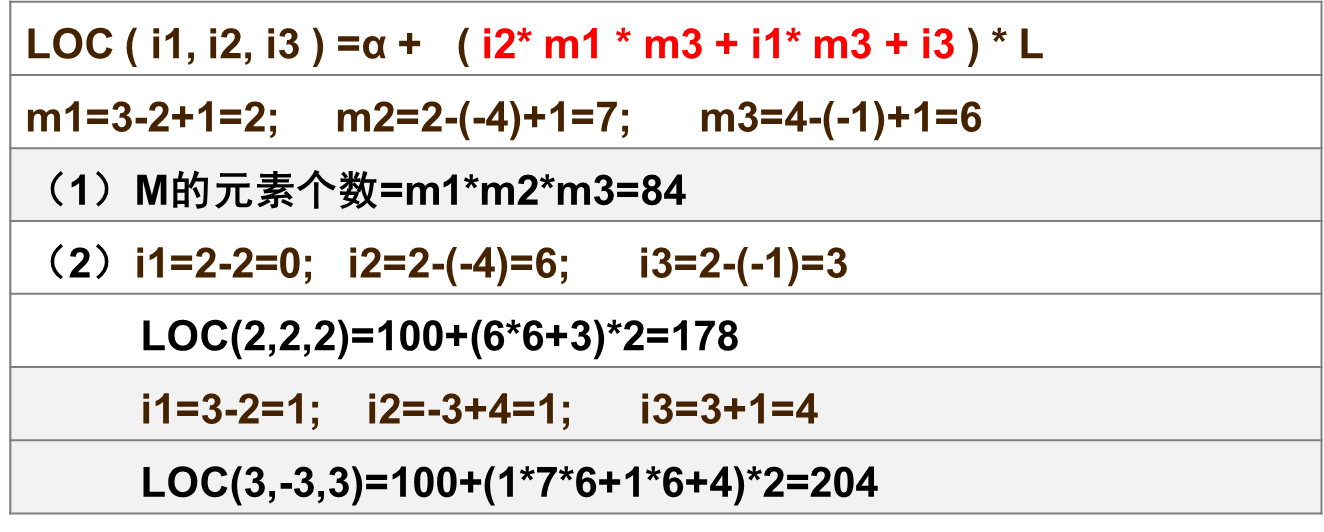
****

****

****

****

* **按列存放**



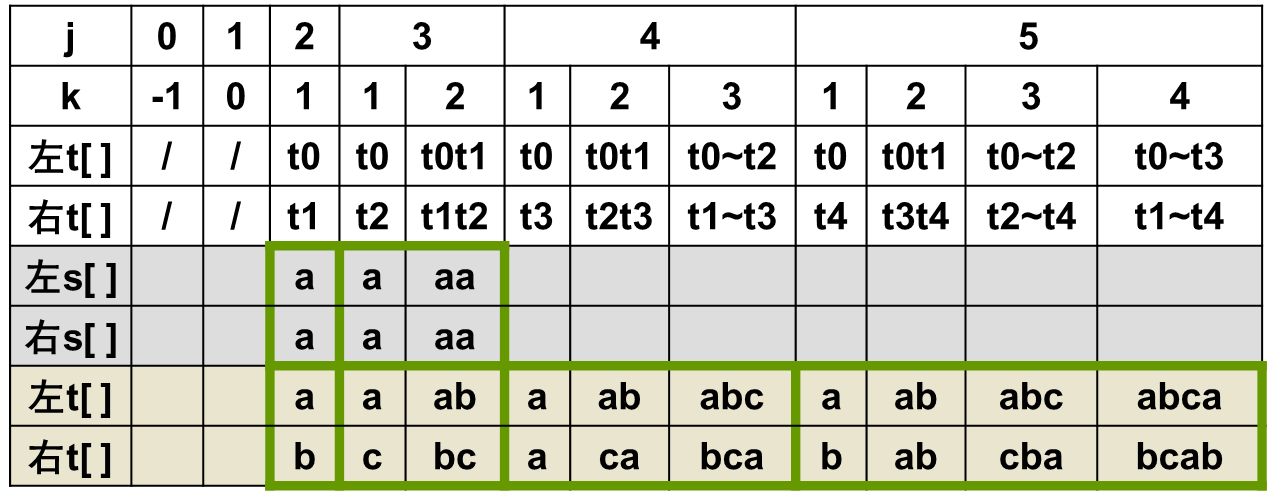
元素个数为：2 \* 7 \* 6 = 84

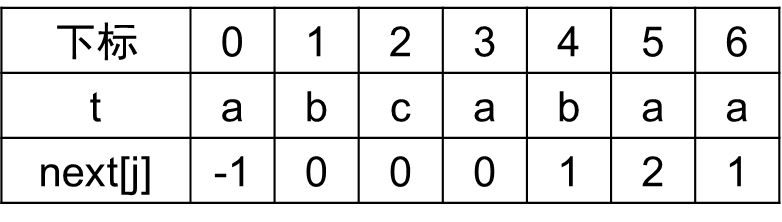
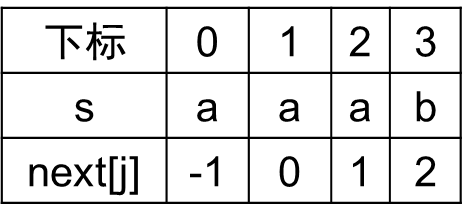
100 + 2 \* （6 \* 6 + 3） = 178

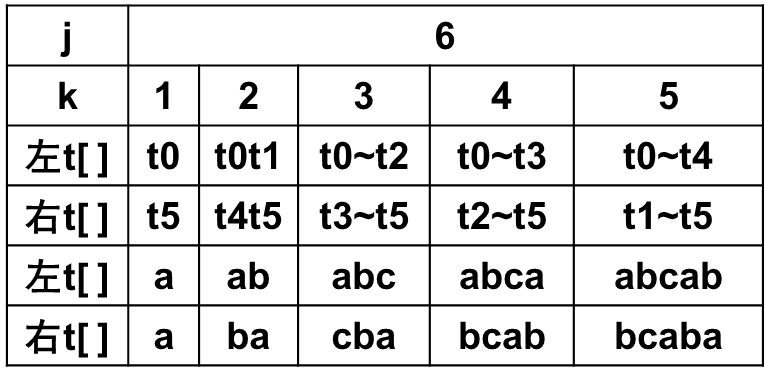
100 + 2 \* （1 \* 7 \* 6 + 1 \* 6 + 4 ） = 204

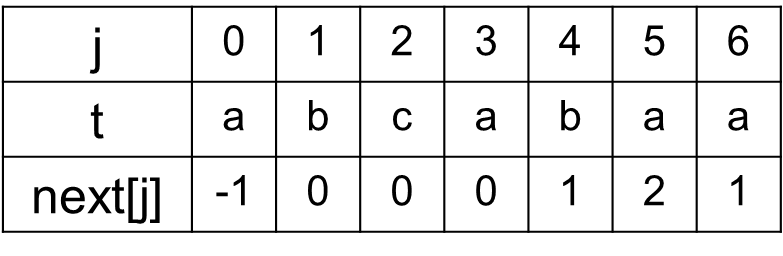
**3. 令s=“aaab”，t=“abcabaa”，试分别求出它们的next函数值。**

**解析：**

****

****

****

****

**三、上机题**

**4．编写一个程序，生成大于10000的随机整数。输出该整数，然后以英文单词的形式输出整数中的各个数字。例如，如果生成的整数是345678，则输出应该是：**

**The value is 345678**

**three four five six seven eight**

**数据结构设计：**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 字符串值 | zero | one | two | three | four | five | six | seven | eight | nine |

**Char \*data[10]={"zero","one","two","three","four","five","six","seven","eight","nine"}**

**算法描述：**

|  |
| --- |
| 伪代码 |
| 读入随机整数n |
| 利用整除和求余运算得到当前n的最高位bits |
| 输出data[bits] |

**8、若在矩阵****Am×n中存在一个元素A[i-1][j-1]满足：A[i-1][j-1]是第i行元素中最小值，且又是第j列元素最大值，则称此元素为该矩阵的一个马鞍点。假设以二维数组存储矩阵Am×n，设计求出矩阵中所有的马鞍点的算法，并分析所设计算法在最坏情况下的时间复杂度。**

**解析：**

**数据结构：**

根据马鞍点的定义，马鞍点一定等于其所在行的最小值，因此，只需判断和每行最小值相等的元素是否为马鞍点即可;根据上述性质，若已找到一个马鞍点saddle,则其他马鞍点的值-定等于saddle,此时先比较一行中的最小值是否等于saddle,若不等，则该行不存在马鞍点;若相等，需进一步判断和该行最小值相等的元素是否是其所在行的最大值

**算法设计：**

|  |
| --- |
| **伪代码** |
| **辅助数组min[]和max[],分别用来存放每行的最小元素值和每列的最大元素值；变量saddle用来存放马鞍点的值，变量f用来表示是否已找到一个马鞍点。** |
| **扫描求出每行的最小值，存入min数组** |
| **扫描求出每列的最大值，存入max数组** |
| **若f=1**  **if(min[i]==saddle) false:不存在马鞍点**  **true：min[i]是否为所在列最大值 false：不是马鞍点**    **true：是马鞍点**  **若f=0**  **Min[i]所有元素是否为所在行的最大值 false：不是马鞍点**  **true：令f=1，saddle=A[i][j]** |

**算法：**

void Minmax(int A[ ][n],int m,int n)

{

f =0;

for (i=0;i<m;i++)/\*求每行的最小元素值，并存入min[ ]数组中\*/

{

min[i]=A[i][0];

for(j=1; j<n; j++)

{

if(A[i][j]<min[i]) min[i]=A[i][j];

}

for(j=0; j<n; j++)/\*求每列的最大元素值，并存入max[ ]数组中stiy

{

max[j]=A[0][j];

for(i=1;i<m;i++)

if(A[i][j]>max[j]) max[j]=A[i][j];

}

for(i=0;i<m;i++)/\*找马鞍点\*/

{

if(f==1)/\*若已找到一个马鞍点\*/

if(min[i]!=saddle) continue;/\*若该行的最小元素值不等于saddle,查找下行\*/

for(j=o;j<n; j++)/\*未发现马鞍点，或已发现马鞍点且该行的最小元索值等于saddle\*/

{

if(A[i][j]==min[i])/\* A[i][j]等于该行最小值\*/

if(A[i][j]==max[j])/\* A[i][j]等于该列最大值\*/

{

if(f==0)

{

f=1;

saddle=A[i][j];

}

printf(“found a saddle %d\n”,A[i][j]);

}

}

}

If(f==0)

printf(“there is not saddle.”);

}

**性能分析：**

该算法的基本操作仍然是元素的比较。第一遍扫描矩阵求每行最小值的比较次数是m\*n;第二次扫描矩阵求每列最大值的比较次数是m\*n;第三次扫描矩阵找马鞍点的比较次数:无马鞍点的情况下，比较次数为m\*n, 存在马鞍点的情况下，最好情况下(只有一个马鞍点且存在第行)，比较次数为n+m-1;最坏情况下(矩阵中的元素都是马鞍点)，比较次数为m\*2n;

因此，该算法在最坏情况下的比较次数为m\*n+m\*n+m\*2n=4m\*n,时间复杂度为0(m\*n)。该算法用了两个数组作为辅助空间，大小分别是m和n,因此空间复杂度是0(m+n)。