第 5 章　多维数组和广义表

5.1　基本概念及运算

5.1.1　多维数组的概念及存储

多维数组是向量（一维数组）的推广，例如，二维数组 $A_{m\times n}$ 可以看成是 m 个行向量组成的向量，也可以看成是 n 个列向量组成的向量。因此，二维数组中的元素最多有两个直接前驱，最多有两个直接后继。这种规则可以推广到三维和三维以上的数组（多维数组）。

由上可知，多维数组是一种典型的非线性结构。多维数组在计算机中有两种存储形式：行优先存放、列优先存放。

行优先存放的规则是：最左边的下标变化最慢，最右边的下标变化最快，右边的下标从小到大变化一遍，与之相邻的左边下标才变化一次。C/C++语言中按此方式存储。

列优先存放的规则是：最右边的下标变化最慢，最左边的下标变化最快，左边的下标从小到大变化一遍，与之相邻的右边下标才变化一次。FORTRAN 语言中按此方式存储。

5.1.2　特殊矩阵及压缩存储

所谓特殊矩阵，是指矩阵中的元素分布呈现某种规律，或者一批元素的值完全相同。为了能节约内存空间，可以使特殊矩阵中多个元素共用一个内存单元，即可以将特殊矩阵压缩存储到一个存储空间比矩阵小的一维数组中，称这种存储为压缩存储。常用的特殊矩阵有对称矩阵、上三角矩阵、下三角矩阵、对角矩阵等。

5.1.3　稀疏矩阵及压缩存储

若矩阵的阶数很大，非零元的个数很少，非零元的分布无任何规律，称这样的矩阵为稀疏矩阵。为了节约内存空间，稀疏矩阵中的零元无需存储，只需存储矩阵中的非零元，称为稀疏矩阵的压缩存储。但稀疏矩阵中的非零元分布无任何规律，故在存储元素值时，还需要存储对应的位置（包含行号、列号）。常用的稀疏矩阵的压缩存储有三元组表法、带行指针的辅助链表法、十字链表法等。

5.1.4　广义表的存储及运算

广义表可以看成是线性表的推广，与线性表不同的是表中的元素可以是原子，也可以是子表，即广义表中的元素可以是原子、子表的混合，而线性表中的元素只能是原子。因此，广义表是线性表的推广。

线性表是一种线性结构，广义表是一种非线性结构，常用的广义表的存储方法有：单链表表示法、双链表表示法。

广义表的运算有：建立广义表、取表头、取表尾、求广义表的深度、输出广义表等。

5.2　习题及解答

5.2.1　配套教材中的习题

5-1. 按行优先存储方式，写出三维数组 A[3][2][4]在内存中的排列顺序及地址计算公式（假设每个数组元素占用 L 个字节的内存单元，a[0][0][0]的内存地址为 Loc(a[0][0][0])）。

解：A[3][2][4]按行优先的排列顺序为：A[0][0][0]、A[0][0][1]、A[0][0][2]、A[0][0][3]、A[0][1][0]、A[0][1][1]、A[0][1][2]、A[0][1][3]、A[1][0][0]、A[1][0][1]、A[1][0][2]、A[1][0][3]、A[1][1][0]、A[1][1][1]、A[1][1][2]、A[1][1][3]、A[2][0][0]、A[2][0][1]、A[2][0][2]、A[2][0][3]、A[2][1][0]、A[2][1][1]、A[2][1][2]、A[2][1][3]。

A[i][j][k]的地址计算公式为：

$$Loc(a[i][j][k])=Loc(a[0][0][0]+(i\times2\times4+j\times4+k)\times L$$

5-2. 按列优先存储方式，写出三维数组 A[3][2][4]在内存中的排列顺序及地址计算公式（假设每个数组元素占用 L 个字节的内存单元，a[0][0][0]的内存地址为 Loc(a[0][0][0])）。

解：A[3][2][4]按列优先的排列顺序为：A[0][0][0]、A[1][0][0]、A[2][0][0]、A[0][1][0]、A[1][1][0]、A[2][1][0]、A[0][0][1]、A[1][0][1]、A[2][0][1]、A[0][1][1]、A[1][1][1]、A[2][1][1]、A[0][0][2]、A[1][0][2]、A[2][0][2]、A[0][1][2]、A[1][1][2]、A[2][1][2]、A[0][0][3]、A[1][0][3]、A[2][0][3]、A[0][1][3]、A[1][1][3]、A[2][1][3]。

A[i][j][k]的地址计算公式为：

$$Loc(a[i][j][k])=Loc(a[0][0][0]+(k\times3\times2+j\times3+i)\times L$$

5-3．设有上三角矩阵 $A_{n\times n}$，它的下三角部分全为 0，将其上三角元素按行优先存入数组 B[m]中（m 足够大），使得 B[k]=a[i][j]，且有 k=f1(i)+f2(j)+c。试推出函数 f1、f2 及常数 c（要求 f1 和 f2 中不含常数项）。

解：将上三角矩阵 $A_{n\times n}$ 按行优先存放到一维数组 B[m]中，使得 B[k]=a[i][j]，这时 k 与 i、j 的对应关系为：

$$k=\begin{cases} -\frac{1}{2}i^2+\frac{1}{2}(n-\frac{1}{2})i+j & (i\leqslant j) \\ \frac{n(n+1)}{2} & (i>j) \end{cases}$$

因此，当 i⩽j 时，$f_1(i)=-\frac{1}{2}i^2+(n-\frac{1}{2})i$，$f_2(j)=j$，$c=0$；

当 i>j 时，$f_1(i)=0$，$f_2(j)=0$，$c=\frac{n(n+1)}{2}$。

5-4．若矩阵 $A_{m\times n}$ 中的某个元素 A[i][j]是第 i 行中的最小值，同时又是第 j 列中的最大值，则称此元素为该矩阵中的一个马鞍点。假设以二维数组存储矩阵 $A_{m\times n}$，试编写求出矩阵中所有马鞍点的算法，并分析你的算法在最坏情况下的时间复杂度。

解：算法描述如下：

```
void   chap5_4(int   a[m][n])
{
  int i,j,k,flag1,flag2,min,col;
  flag2=0;
```

```
    for(i =0; i <m; i ++)
    {
      min=a[i][0];
      for(j=0;j<n;j++)
      if(a[i][j]<min)
      {min=a[i][j];   col=j;}
      for(k=0,flag1=1;k<m&&flag1;k++)
      if(min<a[k][col])     flag1=0;
      if(flag1)
      {
        cout<< i<<"行"<<col<< "列是马鞍点，值为"<<min;
        "flag2=1;
      }
    }
    if(!flag2)
    cout<< "矩阵中无马鞍点"<<endl;
  }
```

该算法在最坏情形下的时间复杂度为 O(n×m)。

5-5．试写一个算法，查找十字链表中某一非零元素 x。

解：算法描述如下：

```
  linknode   *find(linknode *hm,elemtype x)
  {
    linknode   *p,*q;
    p=hm->k.next;
    while(p->k.next!=hm)
    {
      q=p->rptr;
      while(q!=p)
      if(q->k.v==x)   return q;
      else   q=q->rptr;
      p=p->k.next;
    }
    return NULL;        //找不到
  }
```

5-6．给定矩阵 A 如图 5-1 所示，写出它的三元组表和十字链表。

解：A 的三元组表如图 5-2 所示。

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 3 & 0 \\ 0 & 4 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

图 5-1　一个矩阵 A

i	j	v
0	0	1
1	2	2
1	3	3
2	1	4
2	4	5
4	4	6

图 5-2　矩阵 A 的三元组表

A 的十字链表如图 5-3 所示。

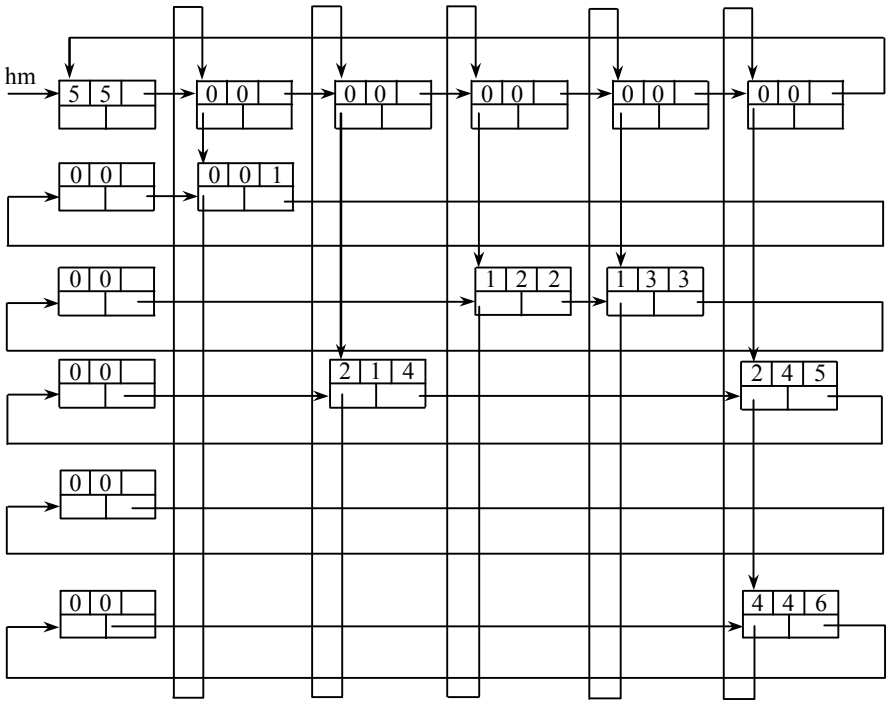

图 5-3　矩阵 A 的十字链表

5-7．对上题的矩阵，画出它的带行指针的链表，并给出算法来建立它。

解：带行指针的链表如图 5-4 所示。

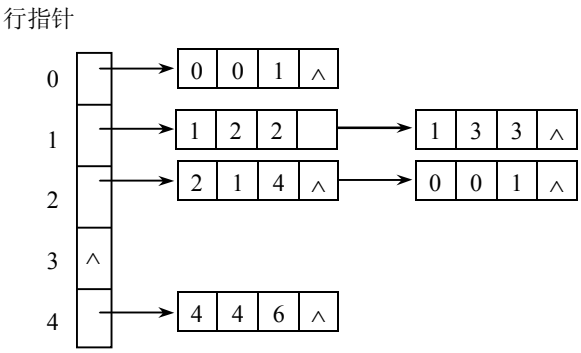

图 5-4　矩阵 A 的带行指针的链表表示

算法描述如下：

```
struct   linknode
{
  int   i,j;              //行列号
  elemtype   v ;          //非零元的值
  linknode   *next;
};
struct   node
{
  linknode   *link;
}a[6];
```

```
void   create(int n)
{
  int i,j,k;
  for(j=1;j<=n;j++)     a[j].next=NULL;
  for(k=1;k<=t;k++)                  //t 为三元组中非零元个数
  {
    cin>>i>>j>>v;              //输入一个三元组
    s=new   linknode;
    s->i =i ;   s->j=j ;   s->v=v;
    s->next=a[i].next;
    a[i].next=s;
  }
}
```

5-8．试编写一个以三元组形式输出、用十字链表表示的稀疏矩阵中非零元素及其下标的算法。

解：算法描述如下：

```
void   output(linknode   *hm)
{
  linknode   *p,*q;
  p=hm->k.next;
  while(p->k.next!=hm)
  {
    q=p->rptr;
    while(q!=p)
    {
        cout<<q->i<<" "<<q->j<< " "<<q->v<<endl;
        q=q->rptr;
    }
   p=p->k.next;
  }
}
```

5-9．给定一个稀疏矩阵，如图 5-5 所示。

$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 & -9 & 0 \\ 0 & 23 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 5 & 8 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 6 & 0 & 33 & 88 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 99 \\ 65 & 0 & 78 & 0 & 0 & 86 & 0 \end{bmatrix}$$

图 5-5　一个稀疏矩阵

用快速转置实现该稀疏矩阵的转置，写出转置前后的三元组表及开始的每一列第一个非零元的位置 pot[col]的值。

解：转置前后的三元组表分别如下：

转置前			转置后		
i	j	v	i	j	v
0	0	11	0	0	11
0	5	-9	0	4	1
1	1	23	0	7	65
1	4	7	1	1	23
2	2	5	1	4	6
2	3	8	2	2	5
2	6	2	2	5	4
4	0	1	2	7	78
4	1	6	3	2	8
4	3	33	3	4	33
4	4	88	4	1	7
5	2	4	4	4	88
6	6	99	5	0	-9
7	0	65	5	7	86
7	2	78	6	2	2
7	5	86	6	6	99

转置开始时的每一列第一个非零元的位置 pot[col]为：

pot[0]=0;

pot[1]=3;

pot[2]=5;

pot[3]=8;

pot[4]=10;

pot[5]=12;

pot[6]=14;

pot[7]=16

5-10．广义表是线性结构还是非线性结构？为什么？

解：广义表是非线性结构，因为表中的元素可以是子表。

5-11．求下列广义表的运算结果：

（1）HEAD((p,h,w))

（2）TAIL ((b,k,p,h))

（3）HEAD(((a,b),(c,d)))

（4）TAIL (((b),(c,d)))

（5）HEAD (TAIL(((a,b),(c,d))))

（6）TAIL (HEAD (((a,b),(c,d))))

（7）HEAD (TAIL (HEAD(((a,d),(c,d)))))

（8）TAIL (HEAD (TAIL (((a,b),(c,d)))))

解：（1）HEAD((p,h,w))=p

（2）TAIL ((b,k,p,h))=(k,p,h)

（3）HEAD(((a,b),(c,d)))=(a,b)

（4）TAIL (((b),(c,d)))=((c,d))

（5）HEAD (TAIL(((a,b),(c,d))))=(c,d)

（6）TAIL (HEAD (((a,b),(c,d))))=(b)

（7）HEAD (TAIL (HEAD(((a,d),(c,d)))))=b

（8）TAIL (HEAD (TAIL (((a,b),(c,d)))))=(d)

5-12．画出下列广义表的图形表示：

（1）A(b,(A,a,C(A)),C(A))

（2）D(A( ),B(e),C(a,L(b,c,d)))

解：（1）的广义表的图形表示如图 5-6 所示。

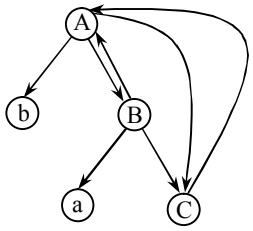

图 5-6 （1）的广义表的图形表示

（2）的广义表的图形表示如图 5-7 所示。

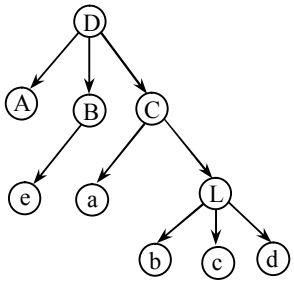

图 5-7 （2）的广义表的图形表示

5-13．画出第 12 题的广义表的单链表表示法和双链表表示法。

解：（1）的广义表的单链表表示法如图 5-8 所示。

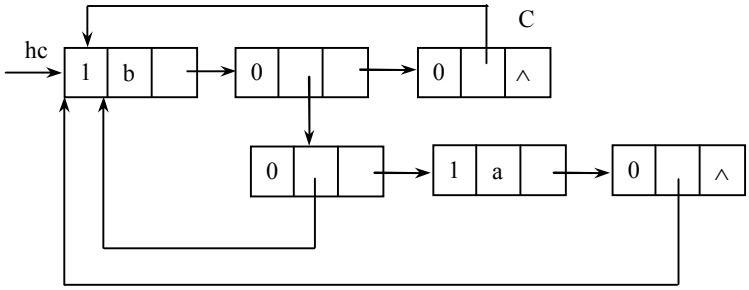

图 5-8 （1）的广义表的单链表表示法

（2）的广义表的单链表表示法如图 5-9 所示。

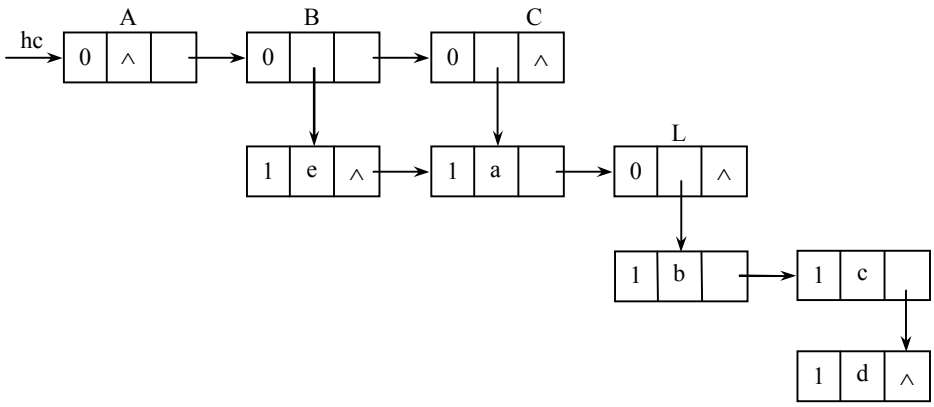

图 5-9　（2）的广义表的单链表表示法

（1）的广义表的双链表表示法如图 5-10 所示。

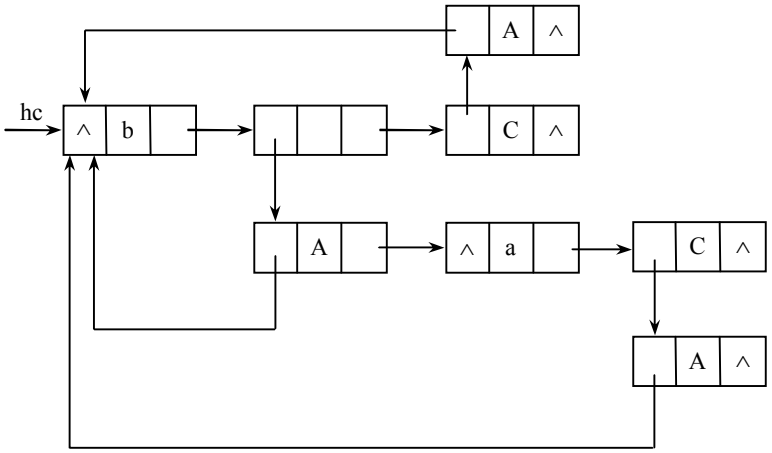

图 5-10　（1）的广义表的双链表表示法

（2）的广义表的双链表表示法如图 5-11 所示。

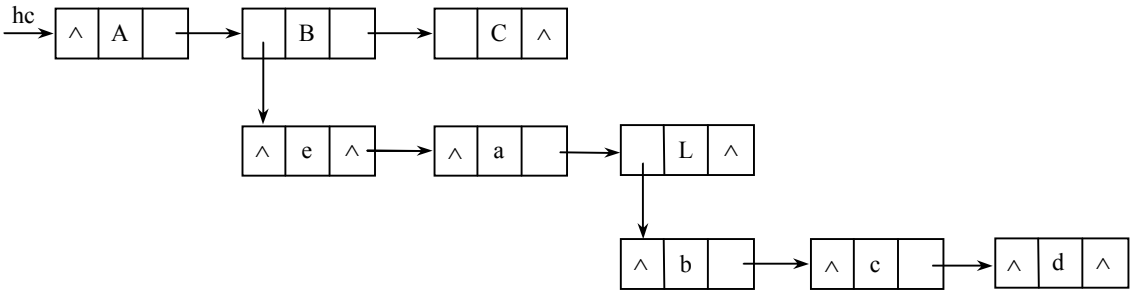

图 5-11　（2）的广义表的双链表表示法

5-14．假设一个准对角矩阵

$$\begin{bmatrix} a_{11} & a_{12} & & & & & & & \\ a_{21} & a_{22} & & & & & & & \\ & & a_{33} & a_{34} & & & & & \\ & & a_{43} & a_{44} & & & & & \\ & & & & \cdots & & & & \\ & & & & & a_{ij} & & & \\ & & & & & & \cdots & & \\ & & & & & & & a_{2m-1,2m-1} & a_{2m-1,2m} \\ & & & & & & & a_{2m,2m-1} & a_{2m,2m} \end{bmatrix}$$

按 $a_{11},a_{12},a_{21},a_{22},a_{33},a_{34},a_{43},a_{44},\ldots,a_{2m-1,2m-1},a_{2m-1,2m},a_{2m,2m-1},a_{2m,2m}$ 的顺序存放到一维数组 B[4m]中，试写出二维数组的下标(i,j)与一维数组下标 k 的对应公式。

解：当 i 为奇数时，k=i+j-1；当 i 为偶数时，k=i+j。

合并成一个公式，可以有 4 种形式的写法：

k=i+j-i%j

或 k=2*(i/2)+j

或 k=i+j+((-1)^i-1)/2

或 k=2*i-1+(1+(-1)^j)/2

5.2.2 综合题

5-1．设数组 R[n]的 n（n>1）个元素中有多个零元素，试设计一个算法，将 R 中所有非零元素依次移动到 R 数组的前端。

解：算法描述如下：

```
int    move( elemtype R[n])
{
   int k , j;
   k=-1;
   for(j=0;j<n;j++)
   if(R[j]!=0)
   {
      k++;
      R[k]=R[j];
      if(k!=j)    R[j]=0;
   }
   return    1;
}
```

5-2．已知 R[n+1]为整型数组，R[0]元素不用，试设计实现下列运算的递归算法：

（1）求数组 R 中的最大整数。

（2）求 n 个整数之和。

（3）求 n 个整数的平均值。

解：算法描述如下：

（1）求数组 R 中的最大整数的递归算法：

```
int     maxdata( int r[ ] ,   int n)
{
```

```
  if(n==1)     return   r[1];
  else if(r[n]>maxdata(r , n-1 ))
  return   r[n];
  else
  return   maxdata(r , n-1 ) ;
}
```

（2）求 n 个整数之和的递归算法：

```
int   count( int r[ ] , int n)
{
  if(n<0)   return   0;
  else
  return       count(r,n-1)+r[n] ;
}
```

（3）求 n 个整数的平均值的递归算法：

```
float   average(int r[ ] , int n)
{
  if(n==1)   return   r[1];
  else
return   1.0*r[n]/n+(1-1.0/n)*average(r , n-1) ;
}
```

5-3．试设计构造一个 n（n>1，n 为奇数）阶魔方阵的算法，所谓魔方阵是这样的一个方阵，它的每一行、每一列和对角线元素之和都相等，例如，三阶魔方阵如图 5-12 所示。

8　1　6
3　5　7
4　9　2

图 5-12　一个三阶魔方阵

解：算法描述如下：

```
void   mf( int r[16][16] , int n)
{
  int j,k,p,m,n;
  for(k=1;k<=n;k++)
     for(j=1;j<=n;j++)
        r[k][j]=0;
  j=n/2+1;
  r[1][j]=1;
  p=1;
  for(k=2;k<=n*n;k++)
  {
     p--;   j++;
     if((p<1)&&(j>n))
     { p+=2;j--; }
     else {
     if(p<1)     p=n;
     if(j>n)      j=1;
  }
  if(r[p][j]==0)   r[p][j]=k;
  else
  {
     p+=2;     j--;
     r[p][j]=k;
  }
}
```

```
for(p=1;p<=n;p++)
  {
   for(j=1;j<=n;j++)
   cout<<r[p][j]<< "    ";
   cout<<endl;
  }
}
```

5-4．试设计算法将数组 a[n]中的所有奇数移到所有偶数之前。要求不另外增加存储空间，且时间复杂度为 O(n)。

解：算法描述如下：

```
void   oddbefore( int a[ ] , int n)
{
    int c,k,j;
    k=0;j=n-1;
    while(k<j)
    {
        while((a[k]%2==1)&&(k<j))
        k++;
        while((a[k]%2==0)&&(k<j))
         j--;
         if(k<j)
           {
              c=a[k];
              a[k]=a[j];
              a[j]=c;
              k++;
              j--;
           }
    }
}
```

5-5．设有三对角矩阵 A，用一维数组 B 存放 A 中的对角线上的元素 a_{ij}（按行优先存放）。试设计由 A 确定 B 中元素的算法。

解：算法描述如下：

```
void   exstorge( int a[n][n ], int n)
{
    int b[3*n];
    int p,j,k;
    for(p=0;p<n;p++)
        for(j=0;j<n;j++)
        if(a[p][j]!=0)
        {
           k=2*p+j-2;
           b[k]=a[p][j];
        }
}
```

5-6．稀疏矩阵只存放其非零元素的行号、列号和数值，以一维数组顺序存放之（按行优先存放），行号为-1 作为结束，试写出两个稀疏矩阵相加的算法。

解：设有一个稀疏矩阵如图 5-13 所示。

$$\begin{bmatrix} 0 & 3 & 0 & 5 \\ 2 & 0 & 0 & 4 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

图 5-13　一个稀疏矩阵

显然，要将上述矩阵用一维数组存放，应存放非零元的行号、列号和值（共 16 个存储单元），存放形式为：0，1，3，0，3，5，

1，0，2，1，3，4，2，1，2，-1。

算法描述如下：

```
void     addmatrix(    int    a[ ], int b[ ], int c[ ] )
{
    int p,j,k,sum;
    p=j=k=0;
    while((a[p]!=-1)&&(b[j]!=-1))
     if(a[p]==b[j])
     {
        if(a[p+1]==b[j+1])
        {
           sum=a[p+2]+b[j+2];
           if(sum!=0)
           {
              c[k]=a[p];
              c[k+1]=a[p+1];
              c[k+2]=sum;
              k+=3;
           }
           p+=3;
           j+=3;
        }
        else
         if(a[p+1]<b[j+1])
            {
                c[k]=a[p];
                c[k+1]=a[p+1];
                c[k+2]=a[p+2];
                k+=3;
                p+=3;
            }
            else
            {
                c[k]=b[j];
                c[k+1]=b[j+1];
                c[k+2]=b[j+2];
                k+=3;    j+=3;
            }
     }
        else
            if(a[p]<b[j])
                {
                   c[k]=a[p];
                   c[k+1]=a[p+1];
                   c[k+2]=a[p+2];
                   k+=3;       p+=3;
                }
                else
                   {
                      c[k]=b[j];
                      c[k+1]=b[j+1];
                      c[k+2]=b[j+2];
```

```
                k+=3;
                j+=3;
              }
          if((a[p]==-1)&&(b[j]!=-1))
          {
             c[k]=b[j];
             c[k+1]=b[j+1];
             c[k+2]=b[j+2];
             k+=3;
             j+=3;
          }
          if((a[p]!=-1)&&(b[j]==-1))
          {
             c[k]=a[p];
             c[k+1]=a[p+1];
             c[k+2]=a[p+2];
             k+=3;
            p+=3;
          }
        c[k]=-1;
    }
```

5-7．设有两个多项式 $f(x)=f_nx^n+f_{n-1}x^{n-1}+\cdots+f_1x+f_0$ 和 $g(x)=g_mx^m+g_{m-1}x^{m-1}+\cdots+g_1x+g_0$，若采用数组来存储多项式的系数，即用数组的第 k 个元素存储多项式的 k 次幂项系数，例如 $f(x)=7x^6-9x^2+5x+3$，则用数组存储可表示为如图 5-14 所示的形式。

0	1	2	3	4	5	6	…	…
3	5	-9	0	0	0	7	…	…

图 5-14　一元多项式的存储表示

试写出两个多项式相乘的算法。

解：算法描述如下：

```
struct   multipoly
{
  int power;
  float coef[m];
};
void   multi(multipoly   f , multipoly   g ,   multipoly    p)
{
    int   k,j,low,n,high;
    float   temp;
    n=f.power+g.power;
    for(k=0;k<=n;k++)
    {
        if(k>=f.power)
        low=k-f.power;
        else     low=0;
        if(k>=g.power)
        high=g.power;
        else     high=k;
        temp=0;
        for(j=low;j<=high;j++)
```

```
            temp=f.coef[k-j]*g.coef[j]+temp;
            p.coef[k]=temp;
        }
        p.power=f.power+g.power;
    }
```

5-8．已知 A 和 B 是两个 n×n 的对称矩阵，输入时，对称矩阵只输入下三角部分的元素，并按行优先存入一个一维数组中，编写一个计算对称矩阵 A 和 B 乘积的算法（A 与 B 直接看成一维数组，得到的乘积看成二维数组，数组的下标都从 1 开始）。

解：根据对称矩阵的压缩存储，一维数组的下标 k 与二维数组的下标 i 和 j 的对应关系为：

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

于是，算法描述如下：

```
void    mult( int a[ ], int b[ ], int c[n][ ], int n)
{
  int    i,j,k,t1,t2,s;
  for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
      {
          s=0;
            for(k=1;k<=n;k++)
              {
                    if(i>=k)       t1= i*(i-1)/2+k;
                    else    t1= k*(k-1)/2+i;
                    if(k>=j)       t2= k*(k-1)/2+j;
                    else    t2= j(j-1)/2+k;
                      s+=a[t1]*b[t2];
              }
          c[i][j]=s;
      }
}
```

5-9．设有 succ(a)=a+1，pred(a)=a-1，即 succ(a)和 pred(a)分别是 a 的后继函数和前驱函数，利用 succ 函数和 pred 函数写出两个非负整数 x 和 y 相加的递归算法。

解：递归算法描述如下：

```
int    add( int x , int y )
{
  if( y==0)
  return    x;
  else
  return    succ(add( x , pred(y) ) ;
}
```