**第五章 多维数组和广义表**

**多维数组和广义表是一种复杂的非线性结构。它们的逻辑特征是一个数据元素有多个直接的前驱和多个直接的后继。**

**5.1多维数组**

**多维数组是向量的推广，数组从二维开始称为多维数组，当然还有三维，四维等。我们只讨论二维数组，如：**

**5.2数组的顺序表示**

**数组通常是静态存储的，即一旦建立，其元素的个数和元素之间的关系就不再发生变化，因此通常采用顺序存储结构来存储数组。也因此数组一般不做插入和删除操作。**

**数组通常对应数学中的矩阵，即数组元素的两个下标分别代表其所在的行和列位置。**

**尽管二维数组呈非线性结构，但实际存储却是线性的，即按一维数组排列，于是出现下列分别。**

**数组元素的排列可以以“行”为主序，也可以以“列”为主序。如：**

**以“行”为主序的（像C语言、Pascal**语言等），元素存储排列为：**... ... ...**

**以“列”为主序的（像FORTRAN**语言等），元素存储排列为：**... ... ...**

**设每个数组的元素占t个存储单元，数组的下标从1开始，则元素的位置可以由下式确定：**

**Loc()=Loc()+[(i-1)\*n+(j-1)]\*t**

**其中：称为基地址元素。**

**在C语言中，数组的下标从0开始，则元素的位置可以由下式确定：**

**Loc()=Loc()+(i\*n+j)\*t**

**其中：称为基地址元素。**

**以下为了与C语言的下标一致，采用下标从0开始。**

**5.3矩阵的压缩存储**

**在科学与工程问题中矩阵是常使用的数学工具，可以用二维数组表示。实际情况有时矩阵阶数很高，但有些元素相同或者元素是0。为了节约存储空间，对这些矩阵进行压缩存储，通常的做法是：相同的元素只用一个空间存储，而0元素则不用空间存储。**

**由于这种矩阵的特殊性，进行压缩存放的过程通常是把二维数组元素放置到一维数组中。**

**5.3.1特殊矩阵的压缩存放**

**1．对称矩阵的压缩存放**

**若一个n阶方阵A的元素满足下述性质：**

** 0≤i,j≤n-1 ,称为对称矩阵。**

**由于元素的对称性，一对对称元素只须分配一个存储空间即可，这样一个n阶矩阵具有的个元素只须个空间存储（差不多节约一半）。**

**如：下列四阶矩阵需要16(4\*4)个元素的存储空间，压缩后只需要10个（4\*(4+1)/2）；**

**对上述的对称矩阵只需存储矩阵的下（上）三角部分即可**

**不失一般性存储对称矩阵可采用三角（可分为上三角或下三角）矩阵存储。**

**按行优先存储的对称矩阵用下（上）三角矩阵的形式分别如下：**

**对上述的对称矩阵，不妨用行为主序的“下三角”模式，将一个二维数组元素a[i][j]存储到一个对应的一维数组b[k]。**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **a[i][j]** | **a[0][0]** | **a[1][0]** | **a[1][1]** | **a[2][0]** | **…** | **a[n-1][n-1]** |
| **b[k]** | **b[0]** | **b[1]** | **b[2]** | **b[3]** | **…** | **b[n(n+1)/2-1]** |

**下标的对应关系如下：**



**其中：k（0）是一维数组b的下标；**

**i，j（）是二维数组a的下标（下同）；**

**2．三角矩阵的压缩存储**

**(1)上三角矩阵**

**其中：主对角线以下c所示的元素为常数或0，存储只需增加一个重复元素的空间即可。若c非零，存储数组多开一个单元，最后一个单元存c。**

**a[i][j]和b[k]的对应关系如下：**

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a[i][j]** | **a[0][0]** | **a[0][1]** | **a[2][0]** | **a[2][1]** | **…** | **a[n-1][n-1]** | **c** |
| **b[k]** | **b[1]** | **b[2]** | **b[3]** | **b[4]** | **…** | **b[n(n+1)/2-1]** | **c** |

**（2）下三角矩阵**

**a[i][j]和b[k]的对应关系如下：**

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a[i][j]** | **a[0][0]** | **a[1][0]** | **a[1][1]** | **a[2][0]** | **…** | **a[n-1][n-1]** | **C** |
| **b[k]** | **b[1]** | **b[2]** | **b[3]** | **b[4]** | **…** | **b[n(n+1)/2]** | **C** |

**3.对角（带状）矩阵的压缩存放**

**对角（带状）矩阵是指所有非零元素都集中在以主对角线为中心的带状区域中；**

**a[i][j]与b[k]的对应关系如下：**

**k=2+3(i-1)+j-(i-1)=2i+j**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **a[i][j]** | **a[0][0]** | **a[0][1]** | **a[1][0]** | **a[1][1]** | **…** | **a[n-1][n-1]** |
| **b[k]** | **b[1]** | **b[2]** | **b[3]** | **b[4]** | **…** | **b[3n-3]** |

**5.3.2 稀疏矩阵**

**当矩阵中非0元素远远少于0元素，而且分布没有规律，称为稀疏矩阵。**

**为了节约空间，存储时只存非0元素。**

**例如：以下矩阵非0和0元素的个数分别是7和18个**

**处理稀疏矩阵通常采用三元组表法和十字链表法。**

**1.三元组表法**

**将稀疏矩阵中的非0元素所在的行、列和其值列成表，称为三元组表法。**

**三元组表定义如下：**

**#define MAX 100**

**typedef int DataType;**

**typedef struct**

**{**

**int i,j;**

**DataType v;**

**}Element;**

**/\* i,j,v分别为非0元素所在的行、列和其值 \*/**

**typedef struct**

**{**

**int m,n;**

**Element data[MAX];**

**/\*第0行放置矩阵的行列数和非0个数\*/**

**}SpMatrix;**

**例如：处理矩阵M和它的转置N(M’)**

**M的三元组表（对非0元素）：**

|  |  |  |
| --- | --- | --- |
| **i(行)** | **j(列)** | **v(元素)** |
| **1** | **3** | **5** |
| **1** | **5** | **-6** |
| **2** | **3** | **12** |
| **2** | **5** | **15** |
| **3** | **1** | **2** |
| **3** | **3** | **7** |
| **5** | **2** | **11** |

**显然上表的排列呈现“行”优先序列，如果将矩阵M转置变为N（M’），只对M的三元组表简单交换行和列得到下列的三元组表，该表显然是“列”优先序列（不符合行优先序列）。**

**M’(N)的三元表是：**

|  |  |  |
| --- | --- | --- |
| **i（行）** | **j（列）** | **v（元素）** |
| **3** | **1** | **5** |
| **5** | **1** | **-6** |
| **3** | **2** | **12** |
| **5** | **2** | **15** |
| **1** | **3** | **2** |
| **3** | **3** | **7** |
| **2** | **5** | **11** |

**上述三元表对应的稀疏矩阵如下：**

**所以要想将三元组表由M应变为N（“行”优先），N的三元组表应是下列情况：**

|  |  |  |
| --- | --- | --- |
| i | j | v |
| 1 | 3 | 2 |
| 2 | 5 | 11 |
| 3 | 1 | 5 |
| 3 | 2 | 12 |
| 3 | 3 | 7 |
| 5 | 1 | -16 |
| 5 | 2 | 15 |

**将M三元组表变为N三元组表的算法是：**

**●对 M’(N)的三元表按行（下标i；若两个i值相同，对应的j值小的在前）进行升序排序。**

**P5-1.c用三元组方法实现矩阵转置**

**#define MAX 100**

**#include<stdio.h>**

**typedef int DataType;**

**typedef struct**

**{**

**int i,j;**

**DataType v;**

**}Element;**

**typedef struct**

**{**

**int m,n,t;**

**Element data[MAX+1];**

**/\* 第0行放置矩阵的行、列数和非0个数 \*/**

**}SpMatrix;**

**void TransMat(SpMatrix M,SpMatrix \*N)**

**{**

**int b,col,k;**

**N🡪m=M.n;N🡪n=M.m;N🡪t=M.t;**

**if(N🡪t>0)**

**{**

**b=1;**

**for(col=1;col<=M.n;col++)**

**for(k=1;k<=M.t;k++)**

**{**

**if(M.data[k].j==col)**

**{**

**N🡪data[b].i=M.data[k].j;**

**N🡪data[b].j=M.data[k].i;**

**N🡪data[b].v=M.data[k].v;**

**b++;**

**}**

**}**

**}**

**}**

**void main()**

**{**

**int s;**

**SpMatrix a={{5,5,7},{0,0,0},{1,3,5},**

**{1,5,-6},{2,3,12},{2,5,15},{3,1,2},**

**{3,3,7},{5,2,11}**

**};**

**SpMatrix \*p=&a;**

**printf("输出转置前的矩阵三元表：\n");**

**for(s=1;s<=a.t;s++)**

**printf("%5d%5d%5d\n",**

**a.data[s].i,a.data[s].j,a.data[s].v);**

**printf("\n\n");**

**TransMat(a,p);**

**printf("输出转置后的矩阵三元表：\n");**

**for(s=1;s<=a.t;s++)**

**printf("%5d%5d%5d\n",**

**a.data[s].i,a.data[s].j,a.data[s].v);**

**printf("\n\n");**

**}**

**2.十字链表法**

**三元组表法是采用顺序存储结构进行存储，当矩阵中非0元素的个数和位置经常发生变动，使用起来就不方便，采用十字链表法可以解决这个问题。**

**所谓“十字链表法”就是将矩阵中的非0元素用上下左右指针绑在十字链上。当非0元素变动时，调整有关指针，实际上进行的就是单链表的插入和删除操作。**

**下面介绍十字链表的结构:**

|  |  |  |
| --- | --- | --- |
| **i(行)** | **j(列)** | **e(元素的值)** |
| **down(向下指针)** | | **right(向右指针)** |

**定义如下：**

**typedef struct OLnode**

**{**

**int i,j;//非零元素的两个下标**

**ElemType e;//非零元素的值**

**struct OLnode \*right,\*down;**

**//行和列的后继指针**

**}OLnode,\*OLink;**

**typedef crossList**

**{**

**OLink \*rhead,\*chead;//行和列表头指针向量**

**int mu,nu,tu;**

**//稀疏矩阵的行数、列数和非零元素个数**

**}CrossList;**

**CrossList M;**

**例如稀疏矩阵：**

****

1 1 7

1 3 9

2 4 -1

3 1 5

3 3 2

M.rhead

M.chead

**p5-2.c 十字链表处理矩阵**

**数据在文件中**

**3 4 5 //三行四列和五个非0元素的矩阵参数**

1. **1 7 //矩阵的三元表**
2. **3 9**
3. **4 -1**
4. **1 5**

**3 3 2**

**#define MAXSIZE 100**

**#include<stdio.h>**

**typedef int DataType;**

**typedef struct node**

**{**

**int i,j;**

**union**

**{**

**DataType v;**

**struct node \*next;**

**}val;**

**struct node \*right,\*down;**

**}MLink;**

**MLink \*CreatMatrix( )**

**{**

**MLink \*p,\*s,\*hm,\*head[MAXSIZE];**

**int i,j,k,m,n,t,max;**

**DataType v;**

**FILE \*fp;**

**fp=fopen("P5-2.dat","r");**

**fscanf(fp,"%d%d%d",&m,&n,&t);**

**printf("行数=%d,列数=%d,非0元素的个数=%d\n\n",m,n,t);**

**if(m>n)max=m;**

**else max=n;**

**hm=(MLink \*)malloc(sizeof(MLink));**

**hm🡪i=m;**

**hm🡪j=n;**

**head[0]=hm;**

**for(k=1;k<=max;k++)/\* 生成表头结点的循环链表 \*/**

**{**

**s=(MLink \*)malloc(sizeof(MLink));**

**s🡪i=0;**

**s🡪j=0;**

**s🡪right=s;**

**s🡪down=s;**

**head[k]=s;**

**head[k-1]🡪val.next=s;**

**}**

**head[max]🡪val.next=hm;**

**printf("矩阵三元表：\n");**

**for(k=1;k<=t;k++)**

**{**

**fscanf(fp,"%d%d%d",&i,&j,&v);**

**printf("%d\t%d\t%d\n",i,j,v);**

**s=(MLink \*)malloc(sizeof(MLink));**

**s🡪i=i;**

**s🡪j=j;**

**s🡪val.v=v;**

**p=head[i];**

**while(p🡪right!=head[i]&&p🡪right🡪j<j)p=p🡪right;**

**s🡪right=p🡪right;**

**p🡪right=s;**

**p=head[j];**

**while(p🡪down!=head[j]&&p🡪down🡪i<i)**

**p=p🡪down;**

**s🡪down=p🡪down;**

**p🡪down=s;**

**}**

**fclose(fp);**

**return hm;**

**}**

**void main()**

**{**

**MLink \*head;**

**head=CreatMatrix();**

**}**