

5.1 练习题5及参考答案

5.1.1 练习题5

1. 单项选择题

- (1) 有一个三维数组 $A[-2..2][-4..5][2..6]$, 其元素个数是()。
- A. 60 B. 250 C. 144 D. 396
- (2) 设二维数组 $A[1..5][1..8]$, 若按行优先的顺序存放数组的元素, 则 $A[4][6]$ 元素的前面有()个元素。
- A. 6 B. 28 C. 29 D. 40
- (3) 设二维数组 $A[1..5][1..8]$, 若按列优先的顺序存放数组的元素, 则 $A[4][6]$ 元素的前面有()个元素。
- A. 6 B. 28 C. 29 D. 40
- (4) 一个 n 阶对称矩阵 A 采用压缩存储方式, 将其下三角部分按行优先存储到一维数组 B 中, 则 B 中元素个数是()。
- A. n B. n^2
C. $n(n+1)/2$ D. $n(n+1)/2+1$
- (5) 一个 n 阶对称矩阵 $A[1..n, 1..n]$ 采用压缩存储方式, 将其下三角部分按行优先存储到一维数组 $B[1..m]$ 中, 则 $A[i][j] (i \geq j)$ 元素在 B 中的位置 k 是()。
- A. $j(j-1)/2+i$ B. $j(j-1)/2+i-1$
C. $i(i-1)/2+j$ D. $i(i-1)/2+j-1$
- (6) 一个对称矩阵 $A[1..10, 1..10]$ 采用压缩存储方式, 将其下三角部分按行优先存储到一维数组 $B[0..m]$ 中, 则 $A[8][5]$ 元素在 B 中的位置 k 是()。
- A. 32 B. 37 C. 45 D. 60

(7) 一个对称矩阵 $A[1..10, 1..10]$ 采用压缩存储方式, 将其下三角部分按行优先存储到一维数组 $B[0..m]$ 中, 则 $A[5][8]$ 元素值在 B 中的位置 k 是()。

- A. 18 B. 32 C. 45 D. 60

(8) 一个对称矩阵 $A[1..10, 1..10]$ 采用压缩存储方式, 将其上三角部分按行优先存储到一维数组 $B[1..m]$ 中, 则 $A[8][5]$ 元素值在 B 中的位置 k 是()。

- A. 10 B. 37 C. 45 D. 60

(9) 一个 n 阶上三角矩阵 A 按列优先顺序压缩存放在一维数组 B , 则 B 中元素个数是()。

- A. n B. n^2 C. $n(n+1)/2$ D. $n(n+1)/2+1$

(10) 一个 10 阶下三角矩阵 $A[0..9, 0..9]$ 按行优先压缩存放在一维数组 $B[0..m]$ 中, 则 $A[3][2]$ 在 B 中的位置 k 是()。

- A. 1 B. 8 C. 10 D. 21

(11) 对特殊矩阵采用压缩存储的目的主要是为了()。

- A. 表达变得简单 B. 对矩阵元素的存取变得简单
C. 去掉矩阵中的多余元素 D. 减少不必要的存储空间

(12) 稀疏矩阵是指()的矩阵。

- A. 非零元素较多且分布无规律 B. 非零元素较少且分布无规律
C. 总元素个数较少 D. 不适合用二维数组表示

(13) 稀疏矩阵一般的压缩存储方法有两种, 即()。

- A. 二维数组和三维数组 B. 三元组和散列
C. 三元组和十字链表 D. 散列和十字链表

(14) 一个稀疏矩阵采用压缩后, 和直接采用二维数组存储相比会失去()特性。

- A. 顺序存储 B. 随机存取 C. 输入输出 D. 以上都不对

(15) 一个 m 行 n 列的稀疏矩阵采用十字链表表示时, 其中总的头结点的个数为()。

- A. $m+1$ B. $n+1$
C. $m+n+1$ D. $\text{MAX}\{m, n\}+1$

2. 填空题

(1) 三维数组 $A[c_1..d_1, c_2..d_2, c_3..d_3]$ ($c_1 \leq d_1, c_2 \leq d_2, c_3 \leq d_3$) 共含有()个元素。

(2) 已知二维数组 $A[m][n]$ 采用行序为主序存储, 每个元素占 k 个存储单元, 并且第一个元素的存储地址是 $\text{LOC}(A[0][0])$, 则 $A[i][j]$ 的地址是()。

(3) 二维数组 $A[10][20]$ 采用列序为主序存储, 每个元素占一个存储单元, 并且 $A[0][0]$ 的存储地址是 200, 则 $A[6][12]$ 的地址是()。

(4) 二维数组 $A[10..20][5..10]$ 采用行序为主方式存储, 每个元素占 4 个存储单元, 并且 $A[10][5]$ 的存储地址是 1000, 则 $A[18][9]$ 的地址是()。

(5) 有一个 10 阶对称矩阵 A , 采用压缩存储方式(以行序为主存储下三角部分, 且 $A[0][0]$ 存放在 $B[1]$ 中), 则 $A[8][5]$ 在 B 中的地址是()。

(6) 设 n 阶下三角矩阵 $A[1..n][1..n]$ 已压缩到一维数组 $B[1..n(n+1)/2]$ 中, 若按行序为主存储, 则 $A[i][j]$ 对应的 B 中的存储位置是()。

(7) 稀疏矩阵的三元组表示中,每个结点对应于稀疏矩阵的一个非零元素,它包含三个数据项,分别表示该元素的()。

3. 简答题

- (1) 简述数组的主要基本运算。
- (2) 为什么说数组是线性表的推广或扩展,而说不说数组就是一种线性表呢?
- (3) 为什么数组一般不采用链式结构存储?
- (4) 如果一维数组 A 中元素个数 n 很大,存在大量重复的元素,且所有元素值相同的元素紧挨在一起,请设计一种压缩存储方式使得存储空间更节省。

4. 算法设计题

- (1) 假定数组 $A[0..n-1]$ 的 n 个元素中有多个零元素,设计一个算法将 A 中所有的非零元素全部移到 A 的前端。
- (2) 有一个含有 n 个整数元素的数组 $a[0..n-1]$,设计一个算法通过比较求 $a[i..j]$ 中的第一个最小元素的下标。
- (3) 设计一个算法,求一个 $n \times n$ 的二维整型数组 A 的下三角和主对角部分的所有元素之和。
- (4) 设计一个算法,给定一个 $n \times n$ 的二维整型数组 A ,按位置输出其中左上-右下和左下-右上两条对角线的元素。

5.1.2 练习题 5 参考答案

1. 单项选择题

- | | | | | |
|--------|--------|--------|--------|--------|
| (1) B | (2) C | (3) B | (4) C | (5) C |
| (6) A | (7) B | (8) B | (9) D | (10) B |
| (11) D | (12) B | (13) C | (14) B | (15) D |

2. 填空题

- (1) $(d_1 - c_1 + 1) \times (d_2 - c_2 + 1) \times (d_3 - c_3 + 1)$
- (2) $\text{LOC}(A[0][0]) + (n \times i + j) \times k$
- (3) 326
- (4) 1208
- (5) 42
- (6) $i(i-1)/2 + j$
- (7) 行下标、列下标和元素值

3. 简答题

- (1) 答:数组的主要基本运算如下。
 - ① 取值运算:给定一组下标,读取其对应的数组元素。
 - ② 赋值运算:给定一组下标,存储或修改与其相对应的数组元素。
- (2) 答:从逻辑结构的角度看,一维数组是一种线性表;二维数组可以看成数组元素为一维数组的一维数组,所以二维数组是线性结构,可以看成是线性表,但就二维数组的形状

而言,它又是非线性结构,因此将二维数组看成是线性表的推广更准确。三维及以上维的数组也是如此。

(3) 答:因为数组使用链式结构存储时需要额外占用更多的存储空间,而且不具有随机存取特性,使得相关操作更复杂。

(4) 答:设数组的元素类型为 ElemType,采用一种结构体数组 B 来实现压缩存储,该结构体数组的元素类型如下。

```
struct
{   ElemType data;           //元素值
    int length;             //重复元素的个数
}
```

如数组 $A[] = \{1, 1, 1, 5, 5, 5, 5, 3, 3, 3, 3, 4, 4, 4, 4, 4\}$, 共有 17 个元素,对应的压缩存储 B 如下。

$\{\{1, 3\}, \{5, 4\}, \{3, 4\}, \{4, 6\}\}$

压缩数组 B 中仅有 8 个整数。从中看出,如果重复元素越多,采用这种压缩存储方式越节省存储空间。

4. 算法设计题

(1) 解:从前向后找为零的元素 $A[i]$,从后向前找非零的元素 $A[j]$,将 $A[i]$ 和 $A[j]$ 进行交换。对应的算法如下。

```
void move(ElemType A[], int n)
{   int i=0, j=n-1;
    ElemType tmp;
    while (i < j)
    {   while (A[i] != 0) i++;           //从前向后找零元素 A[i]
        while (A[j] == 0) j--;           //从后向前找非零元素 A[j]
        if (i < j)                       //A[i]与 A[j]交换
        {   tmp = A[i]; A[i] = A[j];
            A[j] = tmp;
        }
    }
}
```

(2) 解:当参数错误时算法返回 0; 否则先置 $mini = i$, k 从 $i+1$ 到 j 循环:比较将最小元素的下标放在 $mini$ 中。对应的算法如下。

```
int Minij(int a[], int n, int i, int j, int &mini)
{   int k;
    if (i < 0 || j >= n || i > j || j >= n)
        return 0;                       //参数错误返回 0
    mini = i;
    for (k = i+1; k <= j; k++)
        if (a[k] < a[mini])
            mini = k;
    return 1;                             //成功找到返回 1
}
```

(3) 解: 用 sum 记录 a 中下三角和主对角部分的所有元素和(初始为 0), i 从 0 到 $n-1$, j 从 0~ i 循环, 执行 $\text{sum} += a[i][j]$, 最后返回 sum。对应的算法如下。

```
int LowDiag(int a[][N], int n)
{
    int i, j, sum=0;
    for (i=0; i<n; i++)
        for (j=0; j<=i; j++)
            sum+=a[i][j];
    return sum;
}
```

(4) 解: 对于二维数组 A , 左上-右下对角线元素 $a[i][j]$ 满足 $i==j$, 左下-右上对角线元素 $a[i][j]$ 满足 $i+j==n-1$ 。 i 从 0~ $n-1$, j 从 0~ $n-1$ 循环, 输出满足条件的元素值。对应的算法如下。

```
void Output(int a[][N], int n)
{
    int i, j;
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
            if (i==j || i+j==n-1)
                printf("%5d", a[i][j]);
            else
                printf(" ");
        printf("\n");
    }
}
```

5.2 上机实验题 5 及参考答案

5.2.1 上机实验题 5

1. 基础实验题

(1) 有一个 $m \times n$ 的 C/C++ 整型数组 $A = \{a_{i,j}\} (0 \leq i < m, 0 \leq j < n)$ 。设计一个算法, 分别采用以行序为主序和以列序为主序时 a_{ij} 元素前面的元素个数是多少。并用相关数据进行测试。

(2) 一个 n 阶整数对称矩阵 $A = \{a_{i,j}\} (0 \leq i < m, 0 \leq j < n)$ 进行压缩存储, 采用一维数组 $B = \{b_k\}$ 按列优先顺序存放其上三角和主对角线的各元素。编写一个程序将 A 压缩存放在 B 中, 输出 B 的元素并通过 B 输出 A 的所有元素。以 $n=4$ 为例用相关数据进行测试。

2. 应用实验题

(1) 设有一个整型数组 a , 设计一个算法, 使 $a[i]$ 的值变为 $a[0] \sim a[i-1]$ 中小于原 $a[i]$ 值的个数。例如 $a = (5, 2, 1, 4, 3)$, 这样转换后 $a = (0, 0, 0, 2, 2)$ 。并用相关数据进行测试。

(2) 设计一个算法, 将含有 n 个元素的数组 A 的元素 $A[0 \cdots n-1]$ 循环右移 m 位。要求算法的空间复杂度为 $O(1)$ 。并用相关数据进行测试。

(3) 已知一个 $n \times n$ 矩阵 A (元素为整数) 按行优先存于一维数组 $B[0..n(n-1)]$ 中, 试给出一个算法将原矩阵转置后仍存于数组 B 中。并用相关数据进行测试。

(4) 如果矩阵 A 中存在这样的元素 $A[i][j]$ 满足条件: $A[i][j]$ 是第 i 行中值最小的元素, 且又是第 j 列中值最大的元素, 则称为该矩阵的一个马鞍点。设计一个算法求出 $m \times n$ 的矩阵 A 的所有马鞍点。并用相关数据进行测试。

(5) 设有二维数组 $A[m][n]$, 其元素为整数, 每行每列都按从小到大有序, 试给出一个算法求数组中值为 x 的元素的行号 i 和列号 j 。设值 x 在 A 中存在, 要求比较次数不多于 $m+n$ 次。并用相关数据进行测试。

(6) 假设稀疏矩阵采用三元组表示, 设计一个算法求所有左上-右下的对角线元素之和, 若稀疏矩阵不是方阵, 返回 0; 否则求出结果并返回 1。并用相关数据进行测试。

5.2.2 上机实验题 5 参考答案

1. 基础实验题

(1) 解: 采用以行序为主序时, 对于 $a_{i,j}$ 元素, 前面有 i 行, 每行 n 个数, 第 i 行中前面有 j 个元素, 合计前面的元素个数为 $i \times n + j$ 。

采用以列序为主序时, 对于 $a_{i,j}$ 元素, 前面有 j 列, 每列 m 个数, 第 j 列中前面有 i 个元素, 合计前面的元素个数为 $j \times m + i$ 。

对应的实验程序如下。

```
#include <stdio.h>
#define M 10
#define N 10
int Prenums1(int m, int n, int i, int j)           //以行序为主序
{
    int k;
    if (i >= 0 && i < m && j >= 0 && j < n)
    {
        k = i * n + j;
        return k;
    }
    else return -1;
}
int Prenums2(int m, int n, int i, int j)         //以列序为主序
{
    int k;
    if (i >= 0 && i < m && j >= 0 && j < n)
    {
        k = j * m + i;
        return k;
    }
    else return -1;
}
void main()
{
    int m = 3, n = 4, i, j;
    int a[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
    printf("以行序为主序\n");
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            printf(" a[%d][%d]前面的元素个数: %d\n", i, j, Prenums1(m, n, i, j));
```

```

printf("以列序为主序\n");
for (i=0;i<m;i++)
    for (j=0;j<n;j++)
        printf("  a[%d][%d]前面的元素个数:%d\n",i,j,Prenums2(m,n,i,j));
}

```

上述程序的执行结果如图 5.1 所示。

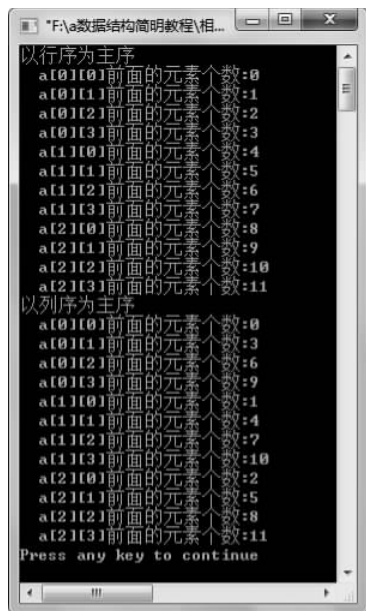


图 5.1 实验程序的执行结果

(2) 解: 对于 A 中上三角和主对角线的元素 $a_{i,j}$, 有 $i \leq j$, 按列优先顺序存放时, 元素 $a_{i,j}$ 前面有 $0 \sim j-1$ 列, 存储的元素个数分别是 $1, 2, \dots, j$, 计 $j(j+1)/2$ 个元素, 在第 j 列表, 前面存储的元素有 $a_{0,j} \sim a_{i-1,j}$, 计 i 个元素。这样 $a_{i,j}$ 在 B 中的存放位置为 $k = j(j+1)/2 + i$ 。

对于下三角部分的元素 $a_{i,j}$, 有 $i > j$, 它的元素值等于 $a_{j,i}$ 。

归纳起来, 对于对称矩阵 $A = \{a_{i,j}\}$ 中的任何元素, 在 B 数组中下标 k 的关系如下。

$$k = j(j+1)/2 + i \quad \text{当 } i \leq j$$

$$k = i(i+1)/2 + j \quad \text{当 } i > j$$

对应的实验程序如下。

```

#include <stdio.h>
#define N 4
int findk(int i, int j)                //由 i,j 求 k
{
    if (i <= j)
        return(j * (j+1)/2 + i);
    else
        return(i * (i+1)/2 + j);
}
int Compress(int A[N][N], int B[])    //将 A 压缩存储到 B 中

```

```

{   int i,j,k=0;
    for (j=0;j<N;j++)
        for (i=0;i<=j;i++)
            {   B[k]=A[i][j];
                k++;
            }
    return k;
}

void dispA(int B[]) //通过 B 输出 A 的所有元素
{   for (int i=0;i<N;i++)
    {   for (int j=0;j<N;j++)
        printf("%4d",B[findk(i,j)]);
        printf("\n");
    }
}

void dispB(int B[],int s) //输出 B
{   for (int k=0;k<s;k++)
    printf("%4d",B[k]);
    printf("\n");
}

void main()
{   int A[4][4]={ {1,2,3,4},{2,5,6,7},{3,6,8,9},{4,7,9,10}};
    int B[10],s; //s 为 B 中元素个数
    printf("原来的 A:\n");
    for (int i=0;i<N;i++)
    {   for (int j=0;j<N;j++)
        printf("%4d",A[i][j]);
        printf("\n");
    }
    s=Compress(A,B);
    printf("输出 B:\n");
    dispB(B,s);
    printf("通过 B 输出 A:\n");
    dispA(B);
}

```

上述程序的执行结果如图 5.2 所示。



图 5.2 实验程序的执行结果

2. 应用实验题

(1) 解: 这样转换后, $a[i]$ 为其前面小于原来 $a[i]$ 的元素个数。 i 从 $n-1$ 到 0 循环: 累计 $a[0..i-1]$ 中大于 $a[i]$ 的元素个数 c , 置 $a[i]$ 为 c 。对应的实验程序如下。

```
#include <stdio.h>
void Trans(int a[], int n)           //转换算法
{
    int i, j, c;
    for (i = n - 1; i >= 0; i--)
    {
        c = 0;
        for (j = 0; j < i; j++)
            if (a[j] < a[i]) c++;
        a[i] = c;
    }
}

void main()
{
    int a[] = {5, 2, 1, 4, 3};
    int n = sizeof(a) / sizeof(a[0]);
    printf("转换前 a:");
    for (int i = 0; i < n; i++)
        printf("%3d", a[i]);
    printf("\n");
    Trans(a, n);
    printf("转换后 a:");
    for (int j = 0; j < n; j++)
        printf("%3d", a[j]);
    printf("\n");
}
```

上述程序的执行结果如图 5.3 所示。



图 5.3 实验程序的执行结果

(2) 解: 设 A 中元素为 ab (a 为前 $n-m$ 个元素, b 为后 m 个元素)。先将 a 逆置得到 $a^{-1}b$, 再将 b 逆置得到 $a^{-1}b^{-1}$, 最后将整个 $a^{-1}b^{-1}$ 逆置得到 $(a^{-1}b^{-1})^{-1} = ba$ 。本算法的时间复杂度为 $O(n)$, 空间复杂度为 $O(1)$ 。对应的实验程序如下。

```
#include <stdio.h>
void Reverse(int A[], int i, int j)   //逆置 A[i..j]
{
    int k, tmp;
    for (k = 0; k < (j - i + 1) / 2; k++)
    {
        tmp = A[i + k];
        A[i + k] = A[j - k]; A[j - k] = tmp;
    }
}

void Rightmove(int A[], int n, int m) //将 A[0..n-1] 循环右移 m 个元素
```

```

{   if (m > n) m = m % n;
    Reverse(A, 0, n - m - 1);
    Reverse(A, n - m, n - 1);
    Reverse(A, 0, n - 1);
}

void display(int A[], int n, int m)           //输出测试结果
{   printf("  移动前:");
    for (int i = 0; i < n; i++)
        printf("%3d", A[i]);
    printf("\n");
    printf("  循环右移 %d 个元素\n", m);
    Rightmove(A, n, m);
    printf("  移动后:");
    for (int j = 0; j < n; j++)
        printf("%3d", A[j]);
    printf("\n");
}

void main()
{   int a[] = {1, 2, 3, 4, 5, 6};
    int n1 = sizeof(a) / sizeof(a[0]);
    int m1 = 4;
    printf("测试 1\n");
    display(a, n1, m1);
    printf("测试 2\n");
    int b[] = {1, 2, 3, 4, 5, 6};
    int n2 = sizeof(b) / sizeof(b[0]);
    int m2 = 20;
    display(b, n2, m2);
}

```

上述程序的执行结果如图 5.4 所示。



图 5.4 实验程序的执行结果

(3) **解**: 矩阵转置是将矩阵中第 i 行第 j 列的元素与第 j 行第 i 列的元素互换位置。因此应先确定矩阵 A 与一维数组 B 的映射关系: $a_{i,j}$ 在一维数组 B 中的下标为 $i \times n + j$, $a_{j,i}$ 在一维数组 B 中的下标为 $j \times n + i$, 当 A 采用 B 存储时, A 的转置相当于交换 $B[i \times n + j]$ 和 $B[j \times n + i]$ 元素。对应的实验程序如下。

```

#include <stdio.h>
#define N 4
void swap(int &x, int &y)           //交换 x 和 y

```

```

    {   int tmp=x;
        x=y; y=tmp;
    }
void trans(int B[])                //转置算法
{   int i,j;
    for (i=0;i<N;i++)
        for (j=0;j<i;j++)
            swap(B[i * N+j],B[j * N+i]); //交换
}
void Save(int A[][N],int B[])      //A 按行优先存于 B 中
{   int k=0;
    for (int i=0;i<N;i++)
        for (int j=0;j<N;j++)
        {   B[k]=A[i][j];
            k++;
        }
}
void Rest(int B[],int A[][N])      //由 B 恢复为 A
{   int k=0;
    for (int i=0;i<N;i++)
        for (int j=0;j<N;j++)
        {   A[i][j]=B[k];
            k++;
        }
}
void dispA(int A[][N])             //输出 A
{   for (int i=0;i<N;i++)
    {   for (int j=0;j<N;j++)
        {   printf("%4d",A[i][j]);
            printf("\n");
        }
    }
}
void dispB(int B[])               //输出 B
{   for (int i=0;i<N*N;i++)
    {   printf("%3d",B[i]);
        printf("\n");
    }
}
void main()
{   int A[][N]={ {1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
    int B[N*N];
    printf("(1)矩阵 A:\n"); dispA(A);
    printf("(2)A 存放在 B 中\n");
    Save(A,B);
    printf("   B:"); dispB(B);
    printf("(3)对 B 转置\n");
    trans(B);
    printf("   B:"); dispB(B);
    printf("(4)B 恢复为 A\n");
    Rest(B,A);
    printf("   矩阵 A:\n"); dispA(A);
}

```

上述程序的执行结果如图 5.5 所示。

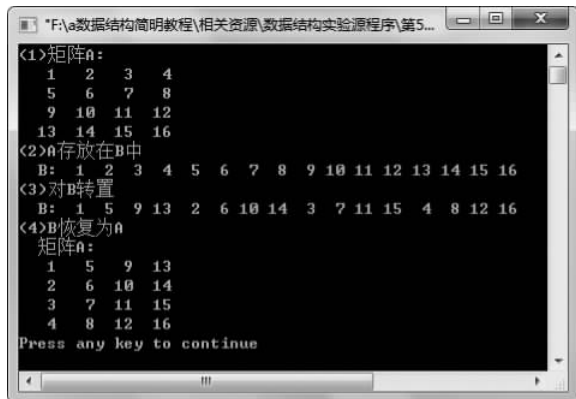


图 5.5 实验程序的执行结果

(4) 解: 先求出数组 A 每行的最小值元素, 放入 $\min[m]$ 之中, 再求出每列的最大值元素, 放入 $\max[n]$ 之中, 若某元素既在 $\min[i]$ 中, 又在 $\max[j]$ 中, 则该元素 $A[i][j]$ 便是马鞍点, 找出所有这样的元素, 即找到了所有马鞍点。对应的实验程序如下。

```
#include <stdio.h>
#define m 3
#define n 4
int MinMax(int A[m][n], int &mini, int &maxj)
{
    int i, j;
    int min[m], max[n];
    for (i=0; i<m; i++) //计算出每行的最小值元素, 放入 min[m] 之中
    {
        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++) //计算出每列的最大值元素, 放入 max[1..n] 之中
    {
        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++) //判定是否为马鞍点
    {
        for (j=0; j<n; j++)
            if (min[i] == max[j])
            {
                mini = i; maxj = j;
                return 1; //找到马鞍点返回 1
            }
    }
    return 0; //没有找到马鞍点返回 0
}

void main()
{
    int a[m][n] = {{2, 3, 5, 1}, {10, 5, 8, 6}, {6, 2, 6, 8}};
    int mini, maxj;
```

```

    if (MinMax(a, mini, maxj))           //调用 MinMax()找马鞍点
        printf("马鞍点: a[%d][%d] = %d\n", mini, maxj, a[mini][maxj]);
    else
        printf("没有马鞍点\n");
}

```

上述程序的执行结果如图 5.6 所示。

(5) 解: 由于算法要求比较次数不多于 $m+n$ 次, 因此不能按行扫描数组的每一元素, 否则比较次数在最坏情况下可达 $m \times n$ 次。根据该数组的特点可从矩阵右上角(简单理解为中间位置元素)向左下角查找。



图 5.6 实验程序的执行结果

首先 $i=0, j=N-1$, 在 $i < m$ 且 $j \geq 0$ 时循环: 若 $a[i][j]=x$, 查找成功返回 1; 否则若 $x < a[i][j]$, 只能在同行前面列中找到 x , 即 $j--$; 若 $x > a[i][j]$, 只能在同列后面行中找到 x , 即 $i++$ 。这样比较次数不多于 $m+n$ 次。对应的实验程序如下。

```

#include <stdio.h>
#define M 3
#define N 4
int Findx(int a[M][N], int x, int &i, int &j)    //求解算法
{
    int flag=0;
    i=0; j=N-1;
    while (i < M && j >= 0)
    {
        if (a[i][j] != x)
        {
            if (a[i][j] > x) j--;                //修改列号
            else i++;                            //修改行号
        }
        else                                     //a[i][j] == x
        {
            flag=1;
            break;
        }
    }
    return flag;
}

void display(int a[M][N], int x)                //输出测试结果
{
    int i, j;
    if (Findx(a, x, i, j))
        printf("a[%d][%d] = %d\n", i, j, x);
    else
        printf("查找 %d 失败\n", x);
}

void main()
{
    int a[M][N] = {{1, 5, 12, 20}, {2, 7, 15, 25}, {4, 9, 18, 30}};
    int x;
    x=2; printf("测试 1: x=%2d ", x); display(a, x);
    x=15; printf("测试 2: x=%2d ", x); display(a, x);
    x=9; printf("测试 3: x=%2d ", x); display(a, x);
    x=10; printf("测试 4: x=%2d ", x); display(a, x);
}

```

上述程序的执行结果如图 5.7 所示。

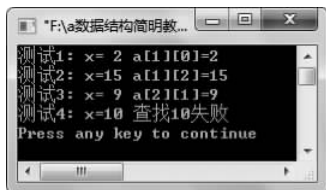


图 5.7 实验程序的执行结果

(6) 解: 对于稀疏矩阵三元组表示 t , 若其行号不等于列号, 返回 0; 否则通过扫描 t .data 数组累加行、列下标相同的元素值并返回 1。对应的实验程序如下。

```
# define M 5
# define N 5
# include "TSMatrix.cpp"           //包含稀疏矩阵三元组的基本运算函数
int Diagonal(TSMatrix t, ElemType &s) //求解算法
{
    int i;
    if (t.rows!=t.cols)
        return 0;                  //不是方阵
    s=0;
    for (i=0;i<t.nums;i++)
        if (t.data[i].r==t.data[i].c) //行号等于列号
            s+=t.data[i].d;
    return 1;
}
void main()
{
    TSMatrix t;
    int a[N][N] = {{0,0,1,0,0},{0,2,0,0,3},
                   {0,0,4,0,0},{0,0,0,0,5},{0,0,0,0,6}};
    CreatMat(t,a);                  //创建三元组 t
    printf("三元组 t\n");
    DispMat(t);                     //输出三元组 t
    int s;
    if (Diagonal(t,s))
        printf("主对角线元素之和=%d\n",s);
    else
        printf("不是方阵\n");
}
```

上述程序的执行结果如图 5.8 所示。



图 5.8 实验程序的执行结果