CHAPTER 5

第5章 数组和稀疏矩阵

5.1 问答题及其参考答案

5.1.1 问答题

1. 为什么说数组是线性表的推广或扩展，而不说数组就是一种线性表呢？

2. 为什么数组一般不使用链式存储结构？

3. 如果某个一维整数数组 A 的元素个数 n 很大，存在大量重复的元素，且所有值相同的元素紧跟在一起，请设计一种压缩存储方式使得存储空间更节省。

4. 有一个 5×6 的二维数组 a，起始元素 $a[1][1]$的地址是 1000，每个元素的长度为 4。

(1) 若采用按行优先存储，给出元素 $a[4][5]$的地址。

(2) 若采用按列优先存储，给出元素 $a[4][5]$的地址。

5. 一个 n 阶对称矩阵存入内存，在采用压缩存储和非压缩存储时占用的内存空间分别是多少？

6. 一个 6 阶对称矩阵 $\boldsymbol{A}$ 中主对角线以上部分的元素已按列优先顺序存放于一维数组 B 中，主对角线上的元素均为 0。根据以下 B 的内容画出 $\boldsymbol{A}$ 矩阵。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
B:	2	5	0	3	4	0	0	1	4	2	6	3	0	1	2

7. 设 $\boldsymbol{A}[0..9,0..9]$是一个 10 阶对称矩阵，采用按行优先将其下三角＋主对角线部分的元素压缩存储到一维数组 B 中。已知每个元素占用两个存储单元，其第一个元素 $\boldsymbol{A}[0][0]$的存储位置为 1000，求以下问题的计算过程及结果：

(1) 给出 $\boldsymbol{A}[4][5]$的存储位置。

(2) 给出存储位置为 1080 的元素的下标。

8. 设 n 阶下三角矩阵 $\boldsymbol{A}[0..n-1,0..n-1]$已压缩存储到一维数组 $B[1..m]$中，若按行为主序存储，则 $\boldsymbol{A}[i][j](i\geqslant j)$元素对应的 B 中存储位置为多少？给出推导过程。

9. 用十字链表表示一个有 k 个非零元素的 $m\times n$ 的稀疏矩阵，则其总的结点数为多少？

10. 特殊矩阵和稀疏矩阵哪一种压缩存储后失去随机存取特性？为什么？

5.1.2　问答题参考答案

1. 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，所以二维数组可以看成线性表，三维及以上维的数组亦如此。数组的基本运算不同于线性表，数组的主要操作是存取元素，不含插入和删除等运算，所以数组可看作线性表的推广，但数组不等同于线性表。

2. 答：因为数组的主要操作是存取元素，通常没有插入和删除操作，在使用链式存储结构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。

3. 答：采用元素类型形如“{整数，个数}”的结构体数组压缩存储，例如，数组 A 为{1,1,1,5,5,5,5,3,3,3,3,4,4,4,4,4,4}，共有 17 个元素，对应的压缩存储为{{1,3},{5,4},{3,4},{4,6}}。在压缩数组中只有 8 个整数。从中看出，重复元素越多，采用这种压缩存储方式越节省存储空间。

4. 答：(1) $m\times n$ 的二维数组 a(行、列下标从 c1、c2 开始)按行优先存储时元素 $a[i][j]$的地址计算公式是 $\mathrm{LOC}(a[i][j])=d+((i-\mathrm{c1})\times(n-\mathrm{c2}+1)+(j-\mathrm{c2}))\times k$。这里 $m=5,n=6,k=4,d=1000,\mathrm{c1}=\mathrm{c2}=1$，所以有 $\mathrm{LOC}(a[4][5])=1000+((4-1)\times(6-1+1)+(5-1))\times 4=1000+88=1088$。

(2) $m\times n$ 的二维数组 a(行、列下标从 c1、c2 开始)按列优先存储时元素 $a[i][j]$的地址计算公式是 $\mathrm{LOC}(a[i][j])=d+((j-\mathrm{c2})\times(m-\mathrm{c1}+1)+(i-\mathrm{c1}))\times k$。这里 $m=5,n=6,k=4,d=1000,\mathrm{c1}=\mathrm{c2}=1$，所以有 $\mathrm{LOC}(a[4][5])=1000+((5-1)\times(5-1+1)+(4-1))\times 4=1000+92=1092$。

5. 答：若采用压缩存储，其容量为 $n(n+1)/2$，若不采用压缩存储，其容量为 n^2。

6. 答：对应的 $\boldsymbol{A}$ 对称矩阵如下。

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

7. 答：(1) $\boldsymbol{A}[4][5]$元素前面有 4 行，元素个数为 1+2+3+4=10，在第 4 行中，$\boldsymbol{A}[4][5]$元素前面有 5 个元素，则前面元素的总数=10+5=15。

$\mathrm{LOC}(\boldsymbol{A}[4][5])=\mathrm{LOC}(\boldsymbol{A}[0][0])+15\times 2=1030$。

(2) 存储位置为 1080，则元素在压缩数组中的序号为(1080−1000)/2=40。设该元素

为$\boldsymbol{A}[i][j]$,则有$i(i+1)/2+j=40$且$0\leqslant i,j\leqslant 9$,从而得到$i=8,j=4$,所以对应的元素为$\boldsymbol{A}[8][4]$或者$\boldsymbol{A}[4][8]$。

8. 答:$\boldsymbol{A}$的下标从0开始,对于$\boldsymbol{A}[i][j](i\geqslant j)$元素,前面有$0\sim i-1$共$i$行,各行的元素个数分别为1、2、…、$i$,计$i(i+1)/2$个元素,在第$i$行中,$\boldsymbol{A}[i][j]$元素前面的元素有$\boldsymbol{A}[i,0..j-1]$,计$j$个元素,所以在$\boldsymbol{A}$中$\boldsymbol{A}[i][j]$元素之前共存储$i(i+1)/2+j$个元素。而$B$的下标从1开始,所以对应的$B$中存储位置是$i(i+1)/2+j+1$。

9. 答:十字链表有一个十字链表表头结点,MAX(m,n)个行列表头结点。另外,每个非零元素对应一个结点,即k个元素结点,所以共有MAX(m,n)$+k+1$个结点。

10. 答:特殊矩阵$\boldsymbol{A}$指值相同的元素或常元素在矩阵中的分布有一定规律,可以将这些特殊值的元素压缩存储在一维数组B中,即将$\boldsymbol{A}[i][j]$元素值存放在$B[k]$中,下标k和下标i、j的关系用函数$k=f(i,j)$表示,该函数的计算时间为$O(1)$,因此对于$\boldsymbol{A}[i][j]$找到存储值$B[k]$的时间为$O(1)$,所以仍具有随机存取特性。

而稀疏矩阵是指非零元素个数t和矩阵容量相比很小($t\ll m\times n$),且非零元素分布没有规律。在用十字链表存储时自然失去了随机存取特性,即便用三元组顺序存储结构存储,存取下标为i和j的元素时也需要扫描整个三元组表,平均时间复杂度为$O(t)$。因此稀疏矩阵$\boldsymbol{A}$无论采用三元组还是十字链表存储,查找$\boldsymbol{A}[i][j]$元素值的时间不再是$O(1)$,所以失去了随机存取特性。

5.2 算法设计题及其参考答案

5.2.1 算法设计题

1. 设计一个算法,将含有n个整数元素的数组$a[0..n-1]$循环右移m位,要求算法的空间复杂度为$O(1)$。

2. 有一个含有n个整数元素的数组$a[0..n-1]$,设计一个算法求其中最后一个最小元素的下标。

3. 设a是一个含有n个元素的double型数组,b是一个含有n个元素的整数数组,其值介于$0\sim n-1$,且所有元素不相同。现设计一个算法,要求按b的内容调整a中元素的顺序,比如当$b[2]=11$时,要求将$a[11]$元素调整到$a[2]$中。如$n=5$,$a[]=\{1,2,3,4,5\}$,$b[]=\{2,3,4,1,0\}$,执行本算法后$a[]=\{3,4,5,1,2\}$。

4. 设计一个算法,实现m行n列的二维数组a的就地转置,当$m\neq n$时返回false,否则返回true。

5. 设计一个算法,求一个m行n列的二维整型数组a的左上角-右下角和右上角-左下角两条主对角线元素之和,当$m\neq n$时返回false,否则返回true。

5.2.2 算法设计题参考答案

1. 解:设a中元素为xy(x为前$n-m$个元素,y为后m个元素)。先将x逆置得到$x^{-1}y$,再将y逆置得到$x^{-1}y^{-1}$,最后将整个$x^{-1}y^{-1}$逆置得到$(x^{-1}y^{-1})^{-1}=yx$。对应的算法如下:

```
void Reverse(int a[],int i,int j)                    //逆置 a[i..j]
{ int i1=i,j1=j;
  while (i1<j1)
  {  swap(a[i1],a[j1]);
     i1++; j1--;
  }
}
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);
}
```

2. 解：设最后一个最小元素的下标为 mini，初值为 0。i 从 1 到 $n-1$ 循环，当 a[i]<=a[mini]时置 mini=i，最后返回 mini。对应的算法如下：

```
int FindMin(int a[],int n)
{ int mini=0;
  for (int i=1;i<n;i++)
     if (a[i]<=a[mini])
        mini=i;
  return mini;
}
```

3. 解：建立一个临时动态数组 c，其大小为 n。用 i 扫描 b 数组，将 $a[b[i]]$放到 $c[i]$中，再将数组 c 复制到 a 中。对应的算法如下：

```
void Rearrange(double a[],int b[],int n)
{ double *c=new double[n];
  for (int i=0;i<n;i++) c[i]=a[b[i]];
  for (int i=0;i<n;i++) a[i]=c[i];                   //将 c 复制到 a 中
  delete [] c;
}
```

4. 解：当 $m\neq n$ 时返回 false。i 从 0 到 $m-1$、j 从 0 到 $i-1$ 循环，将 $a[i][j]$与 $a[j][i]$元素交换，最后返回 true。对应的算法如下：

```
bool Trans(int a[M][N],int m,int n)
{ if (m!=n) return false;
  for (int i=0;i<m;i++)
     for (int j=0;j<i;j++)
        swap(a[i][j],a[j][i]);
  return true;
}
```

5. 解：当 $m\neq n$ 时返回 false。置 s 为 0，用 s 累加 a 的左上角-右下角元素 $a[i][i]$ $(0\leqslant i<m)$之和，再累加 a 的右上角-左下角元素 $a[j][n-j-1]$$(0\leqslant j<n)$之和。当 m 为奇数时，两条对角线有一个重叠的元素 $a[m/2][m/2]$，需从 s 中减去；当 m 为偶数时，没有重叠的元素。最后返回 true。对应的算法如下：

```
bool Diag(int a[M][N],int m,int n,int &s)
{ if (m!=n) return false;
  s=0;
  for (int i=0;i<m;i++) s+=a[i][i];
  for (int j=0;j<=n;j++) s+=a[j][n-j-1];
  if (m%2==1)                                    //m 为奇数时
     s-=a[m/2][m/2];
  return true;
}
```

5.3 基础实验题及其参考答案

5.3.1 基础实验题

1. 编写一个实验程序,给定一个 m 行 n 列的二维数组 a,每个元素的长度 k,数组的起始地址 d,该数组按行优先还是按列优先存储,数组的初始下标 c1(假设 a 的行、列初始下标均为 c1),求元素 $a[i][j]$的地址,并用相关数据进行测试。

2. 编写一个实验程序,给定一个 n 阶对称矩阵 $\boldsymbol{A}$,采用压缩存储存储在一维数组 B 中,指出存储下三角+主对角部分的元素还是上三角+主对角部分的元素,按行优先还是按列优先,$\boldsymbol{A}$ 的初始下标 c1 和 B 的初始下标 c2,求元素 $\boldsymbol{A}[i][j]$在 B 中的地址 k,并用相关数据进行测试。

3. 编写一个实验程序,假设稀疏矩阵采用三元组压缩存储,设计相关基本运算算法,并用相关数据进行测试。

5.3.2 基础实验题参考答案

1. 解:二维数组 a 的存储结构参见《教程》第5章中的5.1.2节。对应的程序如下:

```
#include<iostream>
using namespace std;
int addr()
{ int m,n,k,d,i,j,c1,c2,flag,loc;
  printf("  m n k: ");
  scanf("%d%d%d",&m,&n,&k);
  printf("  起始地址: ");
  scanf("%d",&d);
  printf("  1-按行优先  2-按列优先: ");
  scanf("%d",&flag);
  printf("  初始下标: ");
  scanf("%d",&c1);
  c2=c1;
  printf("  元素 i j: ");
  scanf("%d%d",&i,&j);
  if (flag==1)
     loc=d+((i-c1)*(n-c2+1)+(j-c2))*k;
  else
     loc=d+((j-c2)*(m-c1+1)+(i-c1))*k;
```

```
    printf("  元素 a[%d][%d]的地址:%d\n",i,j,loc);
}
int main()
{ printf("\n  计算二维数组元素地址\n");
  addr();
  return 0;
}
```

图 5.1　第 5 章基础实验题 1 的执行结果

上述程序的一次执行结果如图 5.1 所示。

2. 解：假设 n 阶对称矩阵 $\boldsymbol{A}$ 的行、列起始地址为 c1，压缩存放的一维数组 B 的起始地址为 c2，分 4 种情况讨论。

(1) 采用按行优先方式压缩存储下三角＋主对角部分元素。

对于元素 a_{ij} ($i \geqslant j$)，前面存放 c1～$i-1$ 的行元素，共 $i-$c1 行，第 c1 行有一个元素，第 c1＋1 行有两个元素，…，第 $i-1$ 行有 $i-$c1 个元素，共有 $(1+i-\text{c1})(i-\text{c1})/2$ 个元素。在第 i 行中前面存放的元素是 $a[i][\text{c1}..j-1]$，共 $j-$c1 个元素，则 $k=(1+i-\text{c1})(i-\text{c1})/2+j-\text{c1}+\text{c2}$。

对于元素 a_{ij} ($i<j$)，对应的地址 $k=(1+j-\text{c1})(j-\text{c1})/2+i-\text{c1}+\text{c2}$。

(2) 采用按列优先方式压缩存储下三角＋主对角部分元素。

对于元素 a_{ij} ($i \geqslant j$)，前面存放 c1～$j-1$ 的列元素，共 $j-$c1 列，第 c1 列有 n 个元素，第 c1＋1 列有 $n-1$ 个元素，…，第 $j-1$ 列有 $n-j+\text{c1}+1$ 个元素，共有 $(2n-j+\text{c1}+1)(j-\text{c1})/2$ 个元素。在第 j 行中前面存放的元素是 $a[i-1..j][j]$，共 $i-j$ 个元素，则 $k=(2n-j+\text{c1}+1)(j-\text{c1})/2+i-j+\text{c2}$。

对于元素 a_{ij} ($i<j$)，对应的地址 $k=(2n-i+\text{c1}+1)(i-\text{c1})/2+j-i+\text{c2}$。

(3) 采用按行优先方式压缩存储上三角＋主对角部分元素。

对于元素 a_{ij} ($i \leqslant j$)，前面存放 c1～$i-1$ 的行元素，共 $i-$c1 行，第 c1 行有 n 个元素，第 c1＋1 行有 $n-1$ 个元素，…，第 $i-1$ 行有 $n-i+\text{c1}+1$ 个元素，共有 $(2n-i+\text{c1}+1)(i-\text{c1})/2$ 个元素。在第 i 行中前面存放的元素是 $a[i][i..j-1]$，共 $j-i$ 个元素，则 $k=(2n-i+\text{c1}+1)(i-\text{c1})/2+j-i+\text{c2}$。

对于元素 a_{ij} ($i>j$)，对应的地址 $k=(2n-j+\text{c1}+1)(j-\text{c1})/2+i-j+\text{c2}$。

(4) 采用按列优先方式压缩存储上三角＋主对角部分元素。

对于元素 a_{ij} ($i \leqslant j$)，前面存放 c1～$j-1$ 的列元素，共 $j-$c1 列，第 c1 列有一个元素，第 c1＋1 列有两个元素，…，第 $j-1$ 列有 $j-$c1 个元素，共有 $(1+j-\text{c1})(j-\text{c1})/2$ 个元素。在第 j 列中前面存放的元素是 $a[\text{c1}..i-1][j]$，共 $i-$c1 个元素，则 $k=(1+j-\text{c1})(j-\text{c1})/2+i-\text{c1}+\text{c2}$。

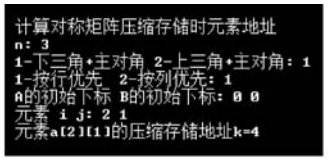
对于元素 a_{ij} ($i>j$)，对应的地址 $k=(1+i-\text{c1})(i-\text{c1})/2+j-\text{c1}+\text{c2}$。

对应的程序如下：

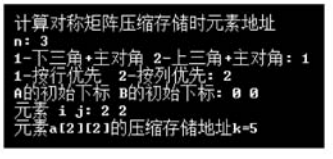
```
#include<iostream>
using namespace std;
int addr()
{ int n,k,i,j,c1,c2,tag,flag;
  printf("  n: ");
  scanf("%d",&n);
```

```
    printf("  1－下三角＋主对角 2－上三角＋主对角: ");
    scanf("%d",&tag);
    printf("  1－按行优先  2－按列优先: ");
    scanf("%d",&flag);
    printf("  A 的初始下标 B 的初始下标: ");
    scanf("%d%d",&c1,&c2);
    printf("  元素 i j: ");
    scanf("%d%d",&i,&j);
    if (tag==1)
    {  if (flag==1)                                          //(1)
       {  if (i>=j) k=(1+i-c1)*(i-c1)/2+j-c1;
          else k=(1+j-c1)*(j-c1)/2+i-c1;
       }
       else                                                  //(2)
       {  if (i>=j) k=(2*n-j+c1+1)*(j-c1)/2+i-j+c2;
          else k=(2*n-i+c1+1)*(i-c1)/2+j-i+c2;
       }
    }
    else
    {  if (flag==1)                                          //(3)
       {  if (i<=j) k=(2*n-i+c1+1)*(i-c1)/2+j-i+c2;
          else k=(2*n-j+c1+1)*(j-c1)/2+i-j+c2;
       }
       else                                                  //(4)
       {  if (i<=j) k=(1+j-c1)*(j-c1)/2+i-c1+c2;
          else k=(1+i-c1)*(i-c1)/2+j-c1+c2;
       }
    }
    printf("  元素 a[%d][%d]的压缩存储地址 k=%d\n",i,j,k);
}
int main()
{  printf("\n  计算对称矩阵压缩存储时元素地址\n");
   addr();
   return 0;
}
```

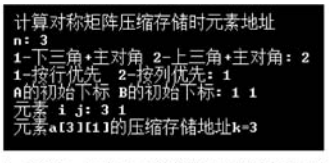
上述程序执行的 4 种情况及其结果如图 5.2 所示。



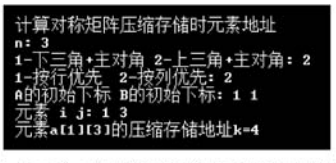
(a) 下三角+主对角+行优先+下标从0开始



(b) 下三角+主对角+列优先+下标从0开始



(c) 上三角+主对角+行优先+下标从1开始



(d) 上三角+主对角+列优先+下标从1开始

图 5.2　第 5 章基础实验题 2 的执行结果

3. 解：稀疏矩阵三元组压缩存储结构及其基本运算算法设计参见《教程》中的第 5.3.1 节。对应的程序如下：

```
#include<iostream>
using namespace std;
const int MAXR=20;                          //稀疏矩阵的最大行数
const int MAXC=20;                          //稀疏矩阵的最大列数
const int MaxSize=100;                      //三元组顺序表的最大元素个数
struct TupElem                              //单个三元组元素的类型
{ int r;                                    //行号
  int c;                                    //列号
  int d;                                    //元素值
  TupElem() {}                              //构造函数
  TupElem(int r1,int c1,int d1)             //重载构造函数
     r=r1;
     c=c1;
     d=d1;
  }
};
class TupClass                              //三元组存储结构类
{ int rows;                                 //行数
  int cols;                                 //列数
  int nums;                                 //非零元素的个数
  TupElem *data;                            //稀疏矩阵对应的三元组顺序表
public:
  TupClass()                                //构造函数
  {  data=new TupElem[MaxSize];             //分配空间
     nums=0;
  }
  ~TupClass()                               //析构函数
  {
     delete [] data;                        //释放空间
  }
  void CreateTup(int A[][MAXC],int m,int n)  //创建三元组
  {  rows=m; cols=n; nums=0;
     for (int i=0;i<m;i++)
        for (int j=0;j<n;j++)
           if (A[i][j]!=0)                  //只存储非零元素
           {  data[nums]=TupElem(i,j,A[i][j]);
              nums++;
           }
  }
  bool Setvalue(int i,int j,int x)          //三元组元素赋值 A[i][j]=x
  {  if (i<0 || i>=rows || j<0 || j>=cols)
        return false;                       //下标错误时返回 false
     int k=0,k1;
     while (k<nums && i>data[k].r)
        k++;                                //查找第 i 行的第一个非零元素
     while (k<nums && i==data[k].r && j>data[k].c)
        k++;                                //在第 i 行中查找第 j 列的元素
```

```
        if (data[k].r==i && data[k].c==j)          //找到了这样的元素
            data[k].d=x;
        else                                       //不存在这样的元素时插入一个元素
        {   for (k1=nums-1; k1>=k;k1--)            //后移元素以便插入
            {   data[k1+1].r=data[k1].r;
                data[k1+1].c=data[k1].c;
                data[k1+1].d=data[k1].d;
            }
            data[k].r=i; data[k].c=j; data[k].d=x;
            nums++;
        }
        return true;                               //赋值成功时返回 true
    }
    bool GetValue(int i,int j,int &x)              //将指定位置的元素值赋给变量 x=A[i][j]
    {   if (i<0 || i>=rows || j<0 || j>=cols)
            return false;                          //下标错误时返回 false
        int k=0;
        while (k<nums && data[k].r<i)
            k++;                                   //查找第 i 行的第一个非零元素
        while (k<nums && data[k].r==i && data[k].c<j)
            k++;                                   //在第 i 行中查找第 j 列的元素
        if (data[k].r==i && data[k].c==j)          //找到了这样的元素
            x=data[k].d;
        else
            x=0;                                   //在三元组中没有找到表示是零元素
        return true;                               //取值成功时返回 true
    }
    void DispMat()                                 //输出三元组
    {   if (nums<=0) return;                       //没有非零元素时返回
        cout << "\t" << rows << "\t" << cols << "\t" << nums << endl;
        cout << "\t--------------------\n";
        for (int i=0;i<nums;i++)
            cout << "\t" << data[i].r << "\t" << data[i].c << "\t" << data[i].d << endl;
    }
};
int main()
{   TupClass t,tb;
    int x;
    int a[MAXR][MAXC]={{0,0,1,0,0,0,0},{0,2,0,0,0,0,0},{3,0,0,0,0,0,0},
                       {0,0,0,5,0,0,0},{0,0,0,0,6,0,0},{0,0,0,0,0,7,4}};
    int m=6,n=7;
    printf("\n  稀疏矩阵 A:\n");
    for (int i=0;i<m;i++)
    {   for (int j=0;j<n;j++)
            printf("%4d",a[i][j]);
        printf("\n");
    }
    t.CreateTup(a,6,7);
    cout << "  三元组 t 表示:\n"; t.DispMat();
    cout << "  执行 A[4][1]=8\n";
    t.Setvalue(4,1,8);
```

```
    cout << "   三元组 t 表示:\n"; t.DispMat();
    cout << "   求 x=A[4][1]\n";
    t.GetValue(4,1,x);
    cout << "   x=" << x << endl;
    return 0;
}
```

上述程序的执行结果如图 5.3 所示。

图 5.3　第 5 章基础实验题 3 的执行结果

5.4　应用实验题及其参考答案

5.4.1　应用实验题

1. 给定 $n(n\geqslant 1)$个整数的序列用整型数组 a 存储，要求求出其中的最大连续子序列之和。例如序列(−2,11,−4,13,−5,−2)的最大连续子序列和为 20，序列(−6,2,4,−7,5,3,2,−1,6,−9,10,−2)的最大连续子序列和为 16。分析算法的时间复杂度。

2. 求马鞍点问题。如果矩阵 $\boldsymbol{a}$ 中存在一个元素 $a[i][j]$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”，则称之为该矩阵的一个马鞍点。设计一个程序，计算出 $m\times n$ 的矩阵 $\boldsymbol{a}$ 的所有马鞍点。

3. 对称矩阵压缩存储的恢复。一个 n 阶对称矩阵 $\boldsymbol{A}$ 采用一维数组 a 压缩存储，压缩方式是按行优先顺序存放 $\boldsymbol{A}$ 的下三角和主对角线部分的各元素。完成以下功能：

(1) 由 $\boldsymbol{A}$ 产生压缩存储 a。

(2) 由一维数组 b 来恢复对称矩阵 $\boldsymbol{A}$。

通过相关数据进行测试。

5.4.2 应用实验题参考答案

1. 解：含有 n 个整数的序列为 $a[0..n-1]$，这里提供3种解法求其中的最大连续子序列之和。

解法1：枚举所有连续子序列 $a[i..j]$($i \leqslant j$)，求出它的所有元素之和 thisSum，并通过比较将最大值存放在 maxSum 中，最后返回 maxSum。对应的程序如下：

```
#include<iostream>
using namespace std;
int maxSubSum(int a[],int n)
{ int maxSum=a[0],thisSum;
  for (int i=0;i<n;i++)                          //三重循环穷举所有的连续子序列
  {  for (int j=i;j<n;j++)
     {  thisSum=0;
        for (int k=i;k<=j;k++)
           thisSum+=a[k];
        if (thisSum>maxSum)                      //通过比较求最大连续子序列之和
           maxSum=thisSum;
     }
  }
  return maxSum;
}
void disp(int a[],int n)                         //输出 a
{ for (int i=0;i<n;i++) printf(" %d",a[i]);
  printf("\n");
}
int main()
{ int a[]={-2,11,-4,13,-5,-2};
  int n=sizeof(a)/sizeof(a[0]);
  int b[]={-6,2,4,-7,5,3,2,-1,6,-9,10,-2};
  int m=sizeof(b)/sizeof(b[0]);
  printf("\n  a:"); disp(a,n);
  printf("  a的最大连续子序列和: %d\n",maxSubSum(a,n));
  printf("\n  b:"); disp(b,m);
  printf("  b的最大连续子序列和: %d\n",maxSubSum(b,m));
  return 0;
}
```

上述程序的执行结果如图5.4所示。maxSubSum()采用三重 for 循环，对应的时间复杂度为 $O(n^3)$。

图5.4 第5章应用实验题1的执行结果

解法 2：设置前缀和数组 sum，sum[i]=a[0]+a[1]+…+a[i]，枚举 i 和 j($i\leqslant j$)求 a[i..j]中的所有元素之和，即 sum[j]−sum[$i-1$]，比较求出最大值 ans，最后输出 ans。对应的程序如下：

```
#include<iostream>
using namespace std;
int maxSubSum(int a[],int n)
{ int *sum= new int[n];                          //存放前缀和
  sum[0]=a[0];
  for(int i=1;i<n;i++)
     sum[i]=a[i]+sum[i-1];
  int ans=a[0];                                  //ans 保存最大子序列之和
  for (int i=0;i<n;i++)
     for(int j=i;j<n;j++)
     {  int s=sum[j]-sum[i-1];
        ans=max(ans,s);
     }
  delete [] sum;
  return ans;
}
void disp(int a[],int n)                         //输出 a
{ for (int i=0;i<n;i++) printf(" %d",a[i]);
  printf("\n");
}
int main()
{ int a[]={-2,11,-4,13,-5,-2};
  int n=sizeof(a)/sizeof(a[0]);
  int b[]={-6,2,4,-7,5,3,2,-1,6,-9,10,-2};
  int m=sizeof(b)/sizeof(b[0]);
  printf("\n  a:"); disp(a,n);
  printf("  a 的最大连续子序列和: %d\n",maxSubSum(a,n));
  printf("\n  b:"); disp(b,m);
  printf("  b 的最大连续子序列和: %d\n",maxSubSum(b,m));
  return 0;
}
```

上述 maxSubSum()采用两重 for 循环，对应的时间复杂度为 $O(n^2)$。

解法 3：修改 a[i]表示以位置 i 结尾的最大连续子序列之和。对于序列 a，用 ans 表示其中的最大连续子序列之和，显然 ans 至少为 0(因为 0 个元素也是其中的一个连续子序列，其和为 0)，当 a[0]<0 的修改为 a[0]=0，用 i 迭代，若 a[$i-1$]≤0 则舍弃前面的子序列，从 a[i]开始计，若 a[$i-1$]>0，则该连续子序列再加上 a[i]称为更大连续子序列，即 a[i]+a[$i-1$]。求出最大的 a[i]为 ans，最后输出 ans。对应的程序如下：

```
#include<iostream>
using namespace std;
int maxSubSum(int a[],int n)
{ if (a[0]<0) a[0]=0;                    //修改 a[i]表示以位置 i 结尾的最大连续子序列之和
  int ans=a[0];
  for(int i=1;i<n;i++)
```

```
    {   if (a[i-1]>0) a[i]+=a[i-1];
        else a[i]+=0;
        ans=max(ans,a[i]);
    }
    return ans;
}
void disp(int a[],int n)                    //输出 a
{ for (int i=0;i<n;i++)
      printf(" %d",a[i]);
  printf("\n");
}
int main()
{ int a[]={-2,11,-4,13,-5,-2};
  int n=sizeof(a)/sizeof(a[0]);
  int b[]={-6,2,4,-7,5,3,2,-1,6,-9,10,-2};
  int m=sizeof(b)/sizeof(b[0]);
  printf("\n  a:"); disp(a,n);
  printf("  a 的最大连续子序列和: %d\n",maxSubSum(a,n));
  printf("\n  b:"); disp(b,m);
  printf("  b 的最大连续子序列和: %d\n",maxSubSum(b,m));
  return 0;
}
```

上述 maxSubSum()采用一重 for 循环,对应的时间复杂度为 $O(n)$。

2. 解:对于二维数组 $a[m][n]$,先求出每行的最小值元素放入 min 数组中,再求出每列的最大值元素放入 max 数组中。若 min[i]=max[j],则元素 $a[i][j]$便是马鞍点,找出所有这样的元素并输出。对应的实验程序 Exp2-2.cpp 如下:

```
#include<iostream>
#include<vector>
using namespace std;
#define MAXM 10
#define MAXN 10
vector<vector<int>> MinMax(int a[MAXM][MAXN],int m,int n)    //求所有马鞍点
{ int *mind=new int[m];                          //存放每行的最小元素
  int *maxd=new int[n];                          //存放每列的最大元素
  for (int i=0;i<m;i++)                          //计算每行的最小元素,放入 mind[i]中
  {   mind[i]=a[i][0];
      for (int j=1;j<n;j++)
          if (a[i][j]<mind[i])
              mind[i]=a[i][j];
  }
  for (int j=0;j<n;j++)                          //计算每列的最大元素,放入 maxd[j]中
  {   maxd[j]=a[0][j];
      for (int i=1;i<m;i++)
          if (a[i][j]>maxd[j])
              maxd[j]=a[i][j];
  }
  vector<vector<int>> ans;
  for (int i=0;i<m;i++)                          //判定是否为马鞍点
```

```
        for (int j=0;j<n;j++)
            if (mind[i]==maxd[j])                    //找到一个马鞍点
            {   vector<int> e;
                e.push_back(i);
                e.push_back(j);
                e.push_back(a[i][j]);
                ans.push_back(e);
            }
    return ans;
}
void disp(int a[MAXM][MAXN],int m,int n)             //输出二维数组
{ for (int i=0;i<m;i++)
    {   for (int j=0;j<n;j++) printf("%4d",a[i][j]);
        printf("\n");
    }
}
int main()
{ int a[MAXM][MAXN]={{10,3,3,4},{15,10,1,3},{4,5,3,6}};
    int m=3,n=4;
    printf("\n a:\n"); disp(a,m,n);
    printf(" 所有马鞍点:\n");
    vector<vector<int>> ans;
    ans=MinMax(a,m,n);
    for (int i=0;i<ans.size();i++)
        printf("   (%d,%d): %d\n",ans[i][0],ans[i][1],ans[i][2]);
    return 0;
}
```

上述程序的执行结果如图 5.5 所示。

```
a:
 10   3   3   4
 15  10   1   3
  4   5   3   6
所有马鞍点:
  (0,2): 3
  (2,2): 3
```

图 5.5　第 5 章应用实验题 2 的执行结果

3. 解：设 $\boldsymbol{A}$ 为 n 阶对称矩阵，若其压缩数组 a 中的 m 个元素，则有 $n(n+1)/2=m$，即 $n^2+n-2m=0$，求得 $n=$(int)(−1+sqrt(1+8m))/2。

若 $\boldsymbol{A}$ 的下三角或者主对角线 $\boldsymbol{A}[i][j](i\geqslant j)$ 元素值存放在 $b[k]$ 中，则 $k=i(i+1)/2+j$，显然 $i(i+1)/2\leqslant k$，可以求出 $i\leqslant(-1+\text{sqrt}(1+8k))/2$，则 $i=\text{int}((-1+\text{sqrt}(1+8k))/2)$，$j=k-i(i+1)/2$。由此设计由 k 求出 i、j 下标的算法 getij(k)。

对应的实验程序 Exp2-3.cpp 如下：

```
#include<iostream>
#include<cmath>
using namespace std;
#define MAXM 10
#define MAXN 10
void disp(int A[MAXM][MAXN],int n)                   //输出二维数组 A
{ for (int i=0;i<n;i++)
    {   for (int j=0;j<n;j++) printf("%4d",A[i][j]);
        printf("\n");
    }
}
void compression(int A[MAXM][MAXN],int n,int a[])    //将 A 压缩存储到 a 中
```

```
{ for (int i=0;i<n;i++)
      for (int j=0;j<=i;j++)
      {   int k=i*(i+1)/2+j;
          a[k]=A[i][j];
      }
}
void getij(int k,int &i,int &j)                    //由 k 求出 i、j 下标
{ i=int((-1+sqrt(1+8*k))/2);
  j=k-i*(i+1)/2;
}
void Restore(int b[],int s,int C[MAXM][MAXN],int &n)   //由 b 恢复成 C
{ int i,j;
  n=int((-1+sqrt(1+8*s))/2);
  for (int k=0;k<s;k++)                            //求主对角线和下三角部分元素
  {   getij(k,i,j);
      C[i][j]=b[k];
  }
  for (i=0;i<n;i++)                                //求上三角部分元素
      for (j=i+1;j<n;j++)
          C[i][j]=C[j][i];
}
int main()
{ printf("\n ********** 测试 1 ********** \n");
  int n=3,s=n*(n+1)/2;
  int A[MAXM][MAXN]={{1,2,3},{2,4,5},{3,5,6}};
  int C[MAXM][MAXN];
  int *a=new int[s];
  printf(" A:\n"); disp(A,n);
  printf(" A 压缩得到 a\n");
  compression(A,n,a);
  printf(" a:\n");
  for (int i=0;i<s;i++) printf("  %d",a[i]);
  printf("\n");
  printf(" 由 a 恢复得到 C\n");
  Restore(a,s,C,n);
  printf(" C:\n"); disp(C,n);
  printf("\n ********** 测试 2 ********** \n");
  n=4;
  s=n*(n+1)/2;
  int B[MAXM][MAXN]={{1,2,3,4},{2,5,6,7},{3,6,8,9},{4,7,9,10}};
  int D[MAXM][MAXN];
  int *b=new int[s];
  printf(" B:\n"); disp(B,n);
  printf(" B 压缩得到 b\n");
  compression(B,n,b);
  printf(" b:\n");
  for (int i=0;i<s;i++) printf("  %d",b[i]);
  printf("\n");
  printf(" 由 b 恢复得到 D\n");
  Restore(b,s,D,n);
  printf(" D:\n"); disp(D,n);
```

```
    return 0;
}
```

上述程序的执行结果如图 5.6 所示。

图 5.6　第 5 章应用实验题 3 的执行结果