第2章 数组

**一、复习要点**

### 本章主要讨论数组抽象数据类型及利用数组实现的顺序表、字符串等数据结构。它们都是线性结构。但数组是直接存取结构，可以根据数组元素的下标直接在数组中存取该元素，而利用它实现的顺序表是顺序存取结构，所有数据元素集中存储于表的前端。字符串是顺序表的特化。

本章复习的要点：

**1、**基本知识点

理解作为抽象数据类型定义的数组类，掌握在C++ 中数组的定义和初始化方法，明确静态数组和动态数组的不同特点和使用，特别需要注意的是数组的存储结构不一定是一个连续的存储空间，当数组存放于一个连续的存储空间时叫做数组的顺序存储方式。要求掌握一维数组、二维数组、三维数组的地址计算方法。需要明确：数组是一种实现级的结构。

其次，需要理解顺序表的定义和特点，顺序表的类定义及其主要操作，如搜索、插入和删除的实现，掌握对它们的性能估计，包括搜索算法的平均搜索长度，插入与删除算法中的对象平均移动次数。还要掌握顺序表实例的定义和使用事例。

接着，需要掌握稀疏矩阵的三元组表示，并理解稀疏矩阵的转置运算的实现及其性能。

此外，需要理解字符串的定义，字符串抽象数据类型的类定义，字符串中各种重载操作的实现和使用，简单的模式匹配算法和匹配事例。

**2、**算法设计

⮚ 静态数组对象的定义，动态数组对象的定义

⮚ 数组中数组元素的原地逆置

⮚ 递归计算数组长度、数组中所有元素的和及平均值

⮚ 在顺序表中搜索值为item的元素，在有序顺序表中搜索值为item的元素

⮚ 在有序顺序表中插入新元素item到第i个位置

⮚ 在有序顺序表中删除第i个元素

⮚ 两个有序顺序表的合并，m个有序顺序表的合并

**二、难点与重点**

1、作为抽象数据类型的数组：数组的定义、数组的按行顺序存储与按列顺序存储

⮚ 确定数组元素的三要素：行号、列号、元素值

⮚ 数组元素的存放地址计算

2、顺序表：顺序表的定义、搜索、插入与删除

⮚ 顺序表搜索算法、平均比较次数的计算

⮚ 插入与删除算法、平均移动次数的计算

3、字符串：字符串的定义及其操作的实现

⮚ 串重载操作的定义与实现

**三、教材中习题的解析**

2-1 设n个人围坐在一个圆桌周围，现在从第s个人开始报数，数到第m个人，让他出局；然后从出局的下一个人重新开始报数，数到第m个人，再让他出局，……，如此反复直到所有的人全部出局为止。下面要解决的Josephus问题是：对于任意给定的n, s和m，求出这n个人的出局序列。请以n = 9, s = 1, m = 5为例，人工模拟Josephus的求解过程以求得问题的解。

【解答】

出局人的顺序为5, 1, 7, 4, 3, 6, 9, 2, 8。

2-2 试编写一个求解Josephus问题的函数。用整数序列1, 2, 3, ……, n表示顺序围坐在圆桌周围的人，并采用数组表示作为求解过程中使用的数据结构。然后使用n = 9, s = 1, m = 5，以及n = 9, s = 1, m = 0，或者n = 9, s = 1, m = 10作为输入数据，检查你的程序的正确性和健壮性。最后分析所完成算法的时间复杂度。

【解答】函数源程序清单如下：

**void** Josephus( **int** A[ ], **int** n, s, m ) **{**

**int** i, j, k, tmp**;**

**if** ( m == 0 ) **{**

**cerr** << "m = 0是无效的参数！" << **endl;**

**return;**

**}**

**for** ( i = 0**;** i < n**;** i++ ) A[i] = i + 1**;** //初始化，执行n次

i = s - 1**;** //报名起始位置

**for** ( k = n**;** k > 1**;** i-- ) **{** //逐个出局，执行n-1次

**if** ( i == k ) i = 0**;**

i = ( i + m - 1 ) % k**;** //寻找出局位置

**if** ( i != k-1 ) **{**

tmp = A[i]; //出局者交换到第k-1位置

**for** ( j = i**;** j < k-1**;** j++ ) A[j] = A[j+1]**;**

A[k-1] = tmp**;**

**}**

**}**

**for** ( k = 0**;** k < n / 2**;** k++ ) **{** //全部逆置, 得到出局序列

tmp = A[k]**;**

A[k] = A[n-k+1]**;**

A[n-k+1] = tmp**;**

**}**

**}**

例：n = 9, s = 1, m = 5

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
| k = 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 第5人出局, i = 4 |
| k = 8 | 1 | 2 | 3 | 4 | 6 | 7 | 8 | 9 | 5 | 第1人出局, i = 0 |
| k = 7 | 2 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 5 | 第7人出局, i = 4 |
| k = 6 | 2 | 3 | 4 | 6 | 8 | 9 | 7 | 1 | 5 | 第4人出局, i = 2 |
| k = 5 | 2 | 3 | 6 | 8 | 9 | 4 | 7 | 1 | 5 | 第3人出局, i = 1 |
| k = 4 | 2 | 6 | 8 | 9 | 3 | 4 | 7 | 1 | 5 | 第6人出局, i = 1 |
| k = 3 | 2 | 8 | 9 | 6 | 3 | 4 | 7 | 1 | 5 | 第9人出局, i = 2 |
| k = 2 | 2 | 8 | 9 | 6 | 3 | 4 | 7 | 1 | 5 | 第2人出局, i = 0 |
|  | 8 | 2 | 9 | 6 | 3 | 4 | 7 | 1 | 5 | 第8人出局, i = 0 |
| 逆置 | 5 | 1 | 7 | 4 | 3 | 6 | 9 | 2 | 8 | 最终出局顺序 |

例：n = 9, s = 1, m = 0

报错信息 m = 0是无效的参数！

例：n = 9, s = 1, m = 10

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
| k = 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 第1人出局, i = 0 |
| k = 8 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 第3人出局, i = 1 |
| k = 7 | 2 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 1 | 第6人出局, i = 3 |
| k = 6 | 2 | 4 | 5 | 7 | 8 | 9 | 6 | 3 | 1 | 第2人出局, i = 0 |
| k = 5 | 4 | 5 | 7 | 8 | 9 | 2 | 6 | 3 | 1 | 第9人出局, i = 4 |
| k = 4 | 4 | 5 | 7 | 8 | 9 | 2 | 6 | 3 | 1 | 第5人出局, i = 1 |
| k = 3 | 4 | 7 | 8 | 5 | 9 | 2 | 6 | 3 | 1 | 第7人出局, i = 1 |
| k = 2 | 4 | 8 | 7 | 5 | 9 | 2 | 6 | 3 | 1 | 第4人出局, i = 0 |
|  | 8 | 4 | 7 | 5 | 9 | 2 | 6 | 3 | 1 | 第8人出局, i = 0 |
| 逆置 | 1 | 3 | 6 | 2 | 9 | 5 | 7 | 4 | 8 | 最终出局顺序 |

当m = 1时，时间代价最大。达到( n-1 ) + ( n-2 ) + ∙∙∙∙∙∙ + 1 = n(n-1)/2 ≈ O(n2)。

2-3 设有一个线性表 (e0, e1, …, en-2, en-1) 存放在一个一维数组A[arraySize]中的前n个数组元素位置。请编写一个函数将这个线性表原地逆置，即将数组的前n个原址内容置换为 (en-1, en-2, …, e1, e0)。

【解答】

**template<class Type> void** inverse ( **Type** A[ ], **int** n ) **{**

**Type** tmp**;**

**for** ( **int** i = 0**;** i <= ( n-1 ) / 2**;** i++ ) **{**

tmp = A[i]**;**  A[i] = A[n-i-1]**;** A[n-i-1] = tmp**;**

**}**

**}**

2-4 假定数组A[arraySize]中有多个零元素, 试写出一个函数, 将A 中所有的非零元素依次移到数组A的前端A[i]（0≤ i ≤ arraySize）。

【解答】

因为数组是一种直接存取的数据结构，在数组中元素不是像顺序表那样集中存放于表的前端，而是根据元素下标直接存放于数组某个位置，所以将非零元素前移时必须检测整个数组空间，并将后面变成零元素的空间清零。函数中设置一个辅助指针free，指示当前可存放的位置，初值为0。

先给出作为抽象数据类型数组的类声明。

**#include** <iostream.h> //在头文件“array.h”中

**#include** <stdlib.h>

**const int** DefaultSize = 30**;**

**template <class Type> class** Array **{**

//数组是数据类型相同的n（size）个元素的一个集合, 下标范围从0到n-1。对数组中元素

//可按下标所指示位置直接访问。

**private:**

**Type** \*elements**;** //数组

**int** ArraySize**;** //元素个数

**public:**

Array ( **int** Size = DefaultSize )**;** //构造函数

Array ( **const** Array**<Type> &** x )**;** //复制构造函数

~Array ( ) **{ delete** [ ] elements**; }** //析构函数

Array**<Type>** **&** **operator** = ( **const** Array**<Type>** **&** a )**;** //数组整体赋值 (复制)

**Type& operator** [ ] ( **int** i )**;** //按下标访问数组元素

**int** Length ( ) **const { return** ArraySize**; }** //取数组长度

**void** compact ( )**; //**数组压缩

**void** ReSize ( **int** sz )**;** //修改数组长度

**}**

**template<class Type> void** Array**<Type> ::** compact( ) **{**

**int** free = 0**;**  //非零元素存放地址

**for** ( **int** i = 0**;** i < ArraySize**;** i++ ) //检测整个数组

**if** ( elements[i] != 0 ) **{** //发现非零元素

elements[free] = elements[i]**;** //前移

free++**;** elements[i] = 0**;**

**}**

**}**

2-5 顺序表的插入和删除要求仍然保持各个元素原来的次序。设在等概率情形下, 对有127个元素的顺序表进行插入, 平均需要移动多少个元素? 删除一个元素, 又平均需要移动多少个元素?

【解答】

若设顺序表中已有n = last+1个元素，last是顺序表的数据成员，表明最后表项的位置。又设插入或删除表中各个元素的概率相等，则在插入时因有n+1个插入位置(可以在表中最后一个表项后面追加)，每个元素位置插入的概率为1/(n+1)，但在删除时只能在已有n个表项范围内删除，所以每个元素位置删除的概率为1/n。

插入时平均移动元素个数AMN(Averagy Moving Number )为

 删除时平均移动元素个数AMN为

根据上述推导，插入时平均移动 127 /2 = 63.5个元素，删除时平均移动 (127-1)/2 = 63个元素。

2-6 若矩阵Am×n中的某一元素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的一个鞍点。假设以二维数组存放矩阵，试编写一个函数，确定鞍点在数组中的位置（若鞍点存在时），并分析该函数的时间复杂度。

【解答】

**int** minmax ( **int** A[ ][ ], **const int** m, **const int** n ) {

//在二维数组A[m][n]中求所有鞍点, 它们满足在行中最小同时在列中最大

**int** \*row = **new int**[m]**; int** \* col = **new int**[n]**;**

**int** i, j**;**

**for** ( i = 0**;** i < m**;** i++ ) **{** //在各行中选最小数组元素, 存于row[i]

row[i] = A[i][0]**;**

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

**if** ( A[i][j] < row[i] ) row[i] = A[i][j]**;**

**}**

**for** ( j = 0**;** j < n**;** j++ ) **{** //在各列中选最大数组元素, 存于col[j]

col[i] = A[0][j]**;**

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

**if** ( A[i][j] > col[j] ) col[j] = A[i][j]**;**

**}**

**for** ( i = 0**;** i < m**;** i++ ) **{** //检测矩阵，寻找鞍点并输出其位置

**for** ( j = 0**;** j < n**;** j++ )

**if** ( row[i] == col[j] )

**cout** << “The saddle point is : (” << i << “, ” << j << “)” << **endl;**

**delete** [ ] row**; delete** [ ] col**;**

**}**

此算法有3个并列二重循环，其时间复杂度为O(m×n)。

2-7 设有一个二维数组A[m][n]，假设A[0][0]存放位置在644(10)，A[2][2]存放位置在676(10)，每个元素占一个空间，问A[3][3](10)存放在什么位置？脚注(10)表示用10进制表示。

【解答】

设数组元素A[i][j]存放在起始地址为Loc ( i, j ) 的存储单元中。

∵ Loc ( 2, 2 ) = Loc ( 0, 0 ) + 2 \* n + 2 = 644 + 2 \* n + 2 = 676.

∴ n = ( 676 - 2 - 644 ) / 2 = 15

∴ Loc ( 3, 3 ) = Loc ( 0, 0 ) + 3 \* 15 + 3 = 644 + 45 + 3 = 692.

2-8 利用顺序表的操作，实现以下的函数。

(1) 从顺序表中删除具有最小值的元素并由函数返回被删元素的值。空出的位置由最后一个元素填补，若顺序表为空则显示出错信息并退出运行。

(2) 从顺序表中删除第i个元素并由函数返回被删元素的值。如果i不合理或顺序表为空则显示出错信息并退出运行。

(3) 向顺序表中第i个位置插入一个新的元素x。如果i不合理则显示出错信息并退出运行。

(4) 从顺序表中删除具有给定值x的所有元素。

(5) 从顺序表中删除其值在给定值s与t之间（要求s小于t）的所有元素，如果s或t不合理或顺序表为空则显示出错信息并退出运行。

(6) 从有序顺序表中删除其值在给定值s与t之间（要求s小于t）的所有元素，如果s或t不合理或顺序表为空则显示出错信息并退出运行。

(7) 将两个有序顺序表合并成一个新的有序顺序表并由函数返回结果顺序表。

(8) 从顺序表中删除所有其值重复的元素，使表中所有元素的值均不相同。

【解答】

顺序表的类定义

**#ifndef** SEQLIST\_H //定义在头文件“seqlist.h”中

**#define** SEQLIST\_H

**#include** < iostream.h>

**#include** <stdlib.h>

**template <class Type> class** SeqList **{**

**private:**

**Type** \*data**;** //顺序表的存放数组

**int** MaxSize**;** //顺序表的最大可容纳项数

**int** last**;** //顺序表当前已存表项的最后位置

**int** current; //顺序表的当前指针（最近处理的表项）

**public:**

SeqList ( **int** MaxSize )**;** //构造函数

~SeqList ( ) **{ delete** [ ] data**; }** //析构函数

**int** MaxLength ( ) **const { return** MaxSize**; }** //求表的最大长度

**int** Length ( ) **const** **{ return** last+1**;** **}** //计算表长度

**int** Find ( **Type** x ) **const;** //定位函数**:** 找x位置，置为当前表项

**void** Locate ( **int** i ) **{** i >= 0 **&&** i <= last ? current = i **:** exit(1); **}**

**//**定位函数**:** 第i项置为当前表项

**int** IsIn ( **Type** x )**;** //判断x是否在表中，不置为当前表项

**Type** \* GetData ( ) **{ return** current == -1？NULL **:** data[current]**; }** //取当前表项的值

**int** Insert ( **Type** x )**;** //插入x在表当前表项之后，置为当前表项

**int** Append ( **Type** x )**;** //追加x到表尾，置为当前表项

**Type** \* Remove ( )**;** //删除当前表项，置下一表项为当前表项

**Type** \*First ( )**;** //取表中第一个表项的值，置为当前表项

**Type** \*Next ( ) **{** **return** current < last ? **&**data[++current] **:** NULL**; }**

//取当前表项后继表项的值，置为当前表项

**Type** \*Prior ( ) **{** **return** current > 0 ? **&**data[--current] **:** NULL**; }**

//取当前表项前驱表项的值，置为当前表项

**int** IsEmpty ( ) **{ return** last == -1**;** **}** //判断顺序表空否, 空则返回1; 否则返回0

**int** IsFull ( ) **{ return** last == MaxSize-1**;** **}** //判断顺序表满否, 满则返回1; 否则返回0

**}**

**#endif**

(1) 实现删除具有最小值元素的函数如下：

**template<Type>** **Type** \* DelMin ( SeqList<**Type**> \*L ) **{**

**if** ( L->Length( ) *==* 0 ) //表空, 中止操作返回

**{ cerr** << “ List is Empty! ” << **endl; exit**(1)**; }**

**Type** min = \*L->First( ), \* temp**;** //假定0号元素的值最小

**for** ( **int** i = 1**;** i < L->Length( )**;** i++ ) **{** //循环, 寻找具有最小值的元素

temp **=** L->Next( )**; //**下一个表项定位为当前表项

**if** ( \*temp < min ) min = \* temp**;** //让min记忆当前具最小值的元素

**}**

L->Find( min )**;** temp =L->Remove( )**; //**定位于min所在表项，删除之

**return** temp**;**

**}**

(2) 实现删除第i个元素的函数如下（设第i个元素在data[i], i=0,1,…,last）：

**template<Type>** **Type** \* DelNo#i ( SeqList<**Type**> \*L, **int** i ) **{**

**if** ( L->Length( ) == 0 || i < 0 || i >= L->Length( ) ) //表空或i不合理, 中止操作

**{ cerr** << “ List is Empty or Parameter is out range! ” << **endl; exit**(1)**; }**

L->Locate( i )**; //**定位于第i个元素

**Type** \* temp = L->Remove( )**;** //删除

**return** temp**;**

**}**

(3) 实现向第i个位置插入一个新的元素x的函数如下（设第i个元素在data[i], i=0,1,…,last）：

**template<Type>** **void** InsNo#i ( SeqList<**Type**> \*L, **int** i, **Type** x ) **{**

**if** ( L->IsFull( )|| i < 0 || i >= L->Length( ) ) //表满或参数i不合理, 中止操作返回

**{ cerr** << “ List is Full or Parameter is out range! ” << **endl; exit**(1)**; }**

L->Locate( i )**; //**定位于第i个元素

L->Insert( x )**;** //在第i个位置插入

**}**

(4) 从顺序表中删除具有给定值x的所有元素。

**template<Type>** **void** DelValue ( SeqList<**Type>** \*L**, Type** x ) **{**

**int** i = 0**; Type** \* temp = L->First( )**;**

**while** (i < L->Length( ) ) //循环, 寻找具有值x的元素并删除它

**if** ( \* temp == x ) L->Remove( )**;** //删除具有值x的元素

**else {** temp = L->Next( )**;** i++**; }**

**}**

(5) 实现删除其值在给定值s与t之间（要求s小于t）的所有元素的函数如下：

**template<Type>** **void** DelNo#sto#t ( SeqList<**Type>** \*L**, Type** s, **Type** t ) **{**

**if** ( L->Length( ) == 0 || s >= t )

**{ cerr** << “List is empty or parameters are illegal!” << **endl; exit**(1)**; }**

**int** i = 0**; Type** \*temp = L->First( )**;**

**while** (i < L->Length( ) ) //循环, 寻找界于s与t之间的元素并删除它

**if** ( \*temp >= s **&&** \*temp <= t ) //删除满足条件的元素

L->Remove( )**;**

**else {** temp = L->Next( )**;**  i++**; }**

**}**

(6) 实现从有序顺序表中删除其值在给定值s与t之间的所有元素的函数如下：

**template<Type>** **void** DelNo#sto#t1 ( SeqList<**Type>** \*L**, Type** s, **Type** t ) **{**

**if** ( L->Length( ) == 0 || s >= t )

**{ cerr** << “List is empty or parameters are illegal!” << **endl; exit**(1)**; }**

**int** i = 0**; Type** \*temp = L->First( )**;**

**while** (i < L->Length( ) ) //循环, 寻找值 ≥s 的第一个元素

**if** ( \* temp >= s ) **break;** //退出循环时, 该元素成为当前表项

**else {** temp = L**-**>Next( )**;**  i++**; } //**否则, 继续寻找

**while** ( i < L->Length( ) **&&** \* temp <= t )

**{** L->Remove( )**;** temp = L->GetData( )**; }**

**}**

(7) 实现将两个有序顺序表合并成一个新的有序顺序表的函数如下：

**template<Type>** SeqList**<Type>** Merge ( SeqList<**Type>** A, SeqList<**Type>** B ) **{**

//合并有序顺序表A与B成为一个新的有序顺序表并由函数返回

SeqList<**Type**> temp**;**

**if** ( A.Length( ) + B.Length( ) > temp.MaxLength( ) )

**{ cerr** << “The summary of The length of Lists is out MaxSize!” << **endl; exit**(1)**; }**

**Type** \*value1 = A.First ( )**,**  \*value2 = B.First ( )**;**

**int** i = 0**,** j = 0**;**

**while** (i < A.length ( ) **&&** j < B.length ( ) ) **{**  //循环, 两两比较, 小者存入结果表

**if** ( \*value1 <= \*value2 )

**{** temp.Append( \*value1 )**;** \*value1 = A.Next ( )**;** i++**; }**

**else {** temp.Append( \*value2 )**;** \*value2 = B.Next ( )**;** j++**; }**

**}**

**while** ( i < A.Length ( ) ) //当A表未检测完, 继续向结果表传送

**{** temp.Append(\* value1)**;** \*value1 = A.Next ( )**;** i++**;** **}**

**while** ( j < B.Length ( ) ) //当B表未检测完, 继续向结果表传送

**{** temp.Append(\* value2 )**;** \*value2 = B.Next ( )**;** j++**;** **}**

**return** temp**;**

**}**

(8) 实现从表中删除所有其值重复的元素的函数如下：

**template<Type>** **void** SeqList<**Type> ::** DelDouble ( ) **{**

**if** ( L->IsEmpty( ) ) **{ cerr** << “List is empty!” << **endl; exit**(1)**; }**

**int** i = 0**,** j**; Type** \*temp1 = L->First( ), \*temp2**;**

**while** (i < L->Length( )-1 ) **{** //循环检测

j = i + 1**;** temp2 = L->Next( )**;**

**while** ( j < L->Length( ) ) //对于每一个i, 重复检测一遍后续元素

**if** ( \*temp1 == \*temp2 ) //如果相等, 删除

**{** L->Remove( )**;** \*temp2 = L->GetData( )**; }**

**else {** \*temp2 = L->Next( )**;** j++**; }**

L->Locate( ++i )**;** \*temp1 = L->GetData( )**;**

**}**

**}**

2-9 设有一个n×n的对称矩阵A，如图(a)所示。为了节约存储，可以只存对角线及对角线以上的元素，或者只存对角线或对角线以下的元素。前者称为上三角矩阵，后者称为下三角矩阵。我们把它们按行存放于一个一维数组B中，如图(b)和图(c)所示。并称之为对称矩阵A的压缩存储方式。试问：

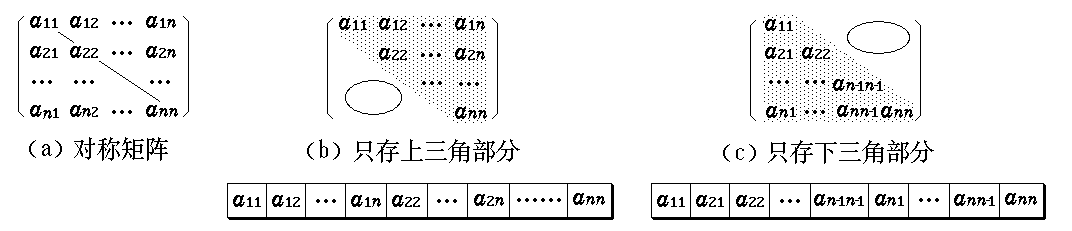
(1) 存放对称矩阵A上三角部分或下三角部分的一维数组B有多少元素？

(2) 若在一维数组B中从0号位置开始存放，则如图(a)所示的对称矩阵中的任一元素aij在只存上三角部分的情形下(图(b))应存于一维数组的什么下标位置？给出计算公式。

(3) 若在一维数组B中从0号位置开始存放，则如图(a)所示的对称矩阵中的任一元素aij在只存下三角部分的情形下(图(c))应存于一维数组的什么下标位置？给出计算公式。

【解答】

(1) 数组B共有n + ( n-1 ) +∙∙∙∙∙∙ + 1= n \* ( n+1 ) / 2个元素。



(2) 只存上三角部分时，若i ≤ j，则数组元素A[i][j]前面有i-1行（1~i-1，第0行第0列不算)，第1行有n个元素，第2行有n-1个元素，∙∙∙∙∙∙，第i-1行有n-i+2个元素。在第i行中，从对角线算起，第j号元素前面有j-i个元素，因此，数组元素A[i][j]在数组B中的存放位置为

n + (n-1) + (n-2) + ∙∙∙∙∙∙ + (n-i+2) + j-i

= (2n-i+2) \* (i-1) / 2 + j-i

= (2n-i) \* (i-1) / 2 + j -1

若i > j，数组元素A[i][j]在数组B中没有存放，可以找它的对称元素A[j][i]。在

数组B的第 (2n-j) \* (j-1) / 2 + i -1位置中找到。

如果第0行第0列也计入，数组B从0号位置开始存放，则数组元素A[i][j]在数组B中的存放位置可以改为

当i ≤ j时，= (2n-i+1) \* i / 2 + j - i = ( 2n - i - 1 ) \* i / 2 + j

当i > j时，= (2n - j - 1) \* j / 2 + i

(3) 只存下三角部分时，若i ≥ j，则数组元素A[i][j]前面有i-1行（1~i-1，第0行第0列不算)，第1行有1个元素，第2行有2个元素，∙∙∙∙∙∙，第i-1行有i-1个元素。在第i行中，第j号元素排在第j个元素位置，因此，数组元素A[i][j]在数组B中的存放位置为

1 + 2 + ∙∙∙∙∙∙ + (i-1) + j = ( i-1)\*i / 2 + j

若i < j，数组元素A[i][j]在数组B中没有存放，可以找它的对称元素A[j][i]。在

数组B的第 (j-1)\*j / 2 + i位置中找到。

如果第0行第0列也计入，数组B从0号位置开始存放，则数组元素A[i][j]在数组B中的存放位置可以改为

当i ≥ j时，= i\*(i+1) / 2 + j

当i < j时，= j\*(j+1) / 2 + i

2-10 设A和B均为下三角矩阵，每一个都有n行。因此在下三角区域中各有n(n+1)/2个元素。另设有一个二维数组C，它有n行n+1列。试设计一个方案，将两个矩阵A和B中的下三角区域元素存放于同一个C中。要求将A的下三角区域中的元素存放于C的下三角区域中，B的下三角区域中的元素转置后存放于C的上三角区域中。并给出计算A的矩阵元素aij和B的矩阵元素bij在C中的存放位置下标的公式。

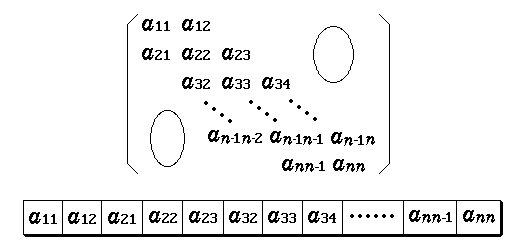
【解答】





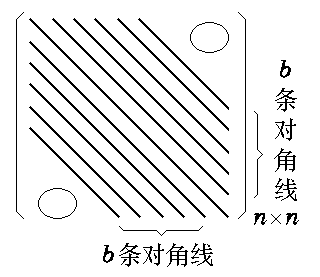
计算公式



2-11 在实际应用中经常遇到的稀疏矩阵是三对角矩阵，如右图所示。在该矩阵中除主对角线及在主对角线上下最临近的两条对角线上的元素外，所有其它元素均为0。现在要将三对角矩阵A中三条对角线上的元素按行存放在一维数组B中，且a11存放于B[0]。试给出计算A在三条对角线上的元素aij （1≤ i ≤ n, i-1 ≤ j ≤ i+1）在一维数组B中的存放位置的计算公式。

【解答】

在B中的存放顺序为 [ a11, a12, a21, a22, a23, a32, a33, a34, …, an n-1, ann ]，总共有3n-2个非零元素。元素aij在第i行，它前面有3(i-1)-1个非零元素，而在本行中第j列前面有j-i+1个，所以元素aij在B中位置为2\*i+j-3。

2-12 设带状矩阵是n×n阶的方阵，其中所有的非零元素都在由主对角线及主对角线上下各b条对角线构成的带状区域内，其它都为零元素。试问：

(1) 该带状矩阵中有多少个非零元素？

(2) 若用一个一维数组B按行顺序存放各行的非零元素，且设a11存放在B[0]中，请给出一个公式，计算任一非零元素aij在一维数组B中的存放位置。

【解答】

(1) 主对角线包含n个非零元素，其上下各有一条包含n-1个非零元素的次对角线，再向外，由各有一条包含n-2个非零元素的次对角线，……，最外层上下各有一条包含n-b个非零元素的次对角线。则总共的非零元素个数有

 n + 2(n-1) + 2(n-2) + … + 2(n-b) = n + 2( (n-1) + (n-2 ) + … + (n-b) )

(2) 在用一个一维数组B按行顺序存放各行的非零元素时，若设b≤n/2，则可按各行非零元素个数变化情况，分3种情况讨论。

① 当1≤i≤b+1时，矩阵第1行有b+1个元素，第2行有b+2个元素，第3行有b+3个元素，……，第i行存有b+i个元素，因此，数组元素A[i][j]在B[ ]中位置分析如下：

第i行(i≥1)前面有i-1行，元素个数为 (b+1)+(b+2)+…+(b+i-1) = (i-1)\*b+i\*(i-1)/2，在第i行第j列(j≥1)前面有j-1个元素，则数组元素A[i][j]在B[ ]中位置为

② 当b+1<i≤n-b+1时，各行都有2b+1个元素。因为数组A[ ][ ]前b行共有b\*b+(b+1)\*b/2 = b\*(3\*b+1)/2个元素，所以数组元素A[i][j]在B[ ]中位置为

③ 当n-b+1<i≤n时，各行元素个数逐步减少。当i=n-b+1时有2b个非零元素，当

i = n-b+2时有2b-1个非零元素，当i=n-b+3时有2b-2个非零元素，…，当i=n时有b+1个非零元素。因为前面n-b行总共有b\*(3\*b+1)/2+(n-2\*b)\*(2\*b+1)个非零元素，所以在最后各行数组元素A[i][j]在B[ ]中位置为

2-13 稀疏矩阵的三元组表可以用带行指针数组的二元组表代替。稀疏矩阵有多少行，在行指针数组中就有多少个元素：第i个元素的数组下标i代表矩阵的第i行，元素的内容即为稀疏矩阵第i行的第一个非零元素在二元组表中的存放位置。二元组表中每个二元组只记录非零元素的列号和元素值，且各二元组按行号递增的顺序排列。试对右图所示的稀疏矩阵，分别建立它的三元组表和带行指针数组的二元组表。



【解答】

二元组表 data

|  |  |  |
| --- | --- | --- |
|  | col | value |
| 0 | 0 | 12 |
| 1 | 2 | 11 |
| 2 | 5 | 13 |
| 3 | 6 | 14 |
| 4 | 1 | -4 |
| 5 | 5 | 3 |
| 6 | 3 | 8 |
| 7 | 1 | -9 |
| 8 | 4 | 2 |

行指针数组

row

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 3 |
| 2 | 4 |
| 3 | 6 |
| 4 | 7 |
| 5 | 7 |

2-14 字符串的替换操作replace (String **&**s, String **&**t, String **&**v)是指：若t是s的子串，则用串v替换串t在串s中的所有出现；若t不是s的子串，则串s不变。例如，若串s为“aabbabcbaabaaacbab”，串t为“bab”，串v为“abdc”，则执行replace操作后，串s中的结果为“aababdccbaabaaacabdc”。试利用字符串的基本运算实现这个替换操作。

【解答】

下面先给出字符串的类定义。

**#ifndef** String1\_H //定义在string1.h中

**#define** String1\_H

**#define** maxLen 128 //字符串的最大长度

**class** String **{**

//对象**:** 零个或多个字符的一个有限序列。

**private:**

**int** curLen**;** //串的长度

**char** \*ch**;** //串存放数组

**public:**

String ( **const** String**&** ob )**;**

//复制构造函数, 由一个已有的字符串对象ob构造一个新字符串。

String ( **const char** \*init )**;**

//构造函数, 构造一个最大长度为*maxLen*, 由init初始化的新字符串。

String ( )**;**

//构造函数, 构造一个最大长度为maxLen, 实际长度为0的字符串。

~String ( ) **{ delete** [ ] ch**; }**

//析构函数，释放动态分配的串空间。

**int** Length ( ) **const {** **return** curLen**; }**

//函数返回串**\*this**的长度。

String**& operator** ( ) ( **int** pos, **int** len )**;**

//当 0 ≤ pos ≤ *maxLen*且0 ≤ len且pos+len < maxLen 时, 则在串**\*this**中从pos所指出

//位置开始连续取len个字符组成子串返回。

**int operator** == ( **const** String**&** ob ) **const { return** strcmp ( ch, ob.ch ) == 0**; }**

//判是否串相等。若串**\*this**与ob相等, 则函数返回1, 否则函数返回0。

**int operator** != ( **const** String**&** ob ) **const { return** strcmp ( ch, ob.ch ) != 0**; }**

//判是否串不相等。若串**\*this**与ob不相等, 则函数返回1, 否则函数返回0。

**int operator** ! ( ) **const** **{ return** curLen == 0**; }**

//判是否串空。若串**\*this**为空, 则函数返回1, 否则函数返回0。

String**& operator** = ( **const** String**&** ob )**;**

//串ob赋值给当前串\***this**

String**& operator** += ( **const** String**&** ob )**;**

//若length (**\*this**)+length (ob) ≤ maxLen, 则把串ob接在串**\*this**后面。

**char& operator** [ ] ( **int** i )**;**

//取**\*this**的第i个字符。

String**&** Replace ( String**&** t**,** String**&** v)

**int** Find ( String**&** pat ) **const;**

//若串pat与串**\*this**中的某个子串匹配, 则函数返回第1次匹配时子串在串\***this**中的

//位置。若串pat为空或在串**\*this**中没有匹配子串, 则函数返回-1。

**}**

**#endif**

String**&** String **::** Replace ( String**&** t**,** String**&** v) **{**

**if** ( ( **int** id = Find ( t ) ) == -1 )//没有找到，当前字符串不改，返回

**{ cout <<** "The (replace) operation failed." << **endl;** **return \*this; }**

String temp( ch )**;** //用当前串建立一个空的临时字符串

ch[0] = '\0'**;** curLen = 0**;**  //当前串作为结果串，初始为空

**int** j, k = 0, len**;**  //存放结果串的指针

**while** ( id != -1 ) **{**

**for** (j = 0**;** j < id**;** j++) ch[k++] = temp.ch[j]**;** //摘取temp.ch中匹配位置

**if** ( curLen+ id + v.curLen <= maxLen )

len = v.curLen**;**  //确定替换串v传送字符数l

**else** len = maxLen - curLen - id**;**

**for** ( j = 0**;** j < len**;** j++ ) ch[k++] = v.ch[j]**;** //连接替换串v到结果串ch后面

curLen += id + len**;** //修改结果串连接后的长度

**if** ( curLen == maxLen ) **break;** //字符串超出范围

**for** ( j = id + t.curLen**;** j < temp.curLen**;** j++ )

temp.ch[j- id - t.curLen] = temp.ch[j]**;** //删改原来的字符串

temp.curLen -= id + t.curLen**;**

id = temp.Find ( t )**;**

**}**

**return \*this;**

**}**

2-15 编写一个算法frequency，统计在一个输入字符串中各个不同字符出现的频度。用适当的测试数据来验证这个算法。

【解答】

**include** <iostream.h>

**include** "string1.h"

**void** frequency( String**&** s, **char** A[ ]**,** **int** C[ ]**, int &**k ) **{**// s是输入字符串，数组A[ ]中记录字符串中有多少种不同的字符，C[ ]中记录每

//一种字符的出现次数。这两个数组都应在调用程序中定义。k返回不同字符数。

**int** i, j, len = s.length( )**;**

**if** ( !len ) { **cout** << "The string is empty. " << **endl;**  k = 0**; return; }**

**else {**  A[0] = s[0]**;** C[0] = 1**;** k = 1**;** //语句s[i]是串的重载操作

**for** ( i = 1**;** i < len**;** i++ ) C[i] = 0**;** //初始化

**for** ( i = 1**;** i < len**;** i++ ) **{** //检测串中所有字符

j = 0**;while** ( j < k **&&** A[j] != s[i] ) j++**;** //检查s[i]是否已在A[ ]中

**if** ( j == k ) **{** A[k] = s[i]**;** C[k]++**;** k++ **}** //s[i]从未检测过

**else** C[j]++**;** //s[i]已经检测过

**}**

**}**

**}**

测试数据 s = "cast cast sat at a tasa\0"

测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | c | a | s | t | ~~b~~ |
| C | 2 | 7 | 4 | 5 | 5 |

【另一解答】

**include** <iostream.h>

**include** "string1.h"

**const int** charnumber = 128; //ASCII码字符集的大小\*/

**void** frequency( String**&** s**,** **int** C[ ] ) **{**// s是输入字符串，数组C[ ]中记录每一种字符的出现次数。

**for** ( **int** i = 0**;** i < charnumber**;** i++ ) C[i] = 0**;** //初始化

**for** ( i = 0**;** i < s.length ( )**;** i++ ) //检测串中所有字符

C[ **atoi** (s[i]) ]++； //出现次数累加

**for** ( i = 0**;** i < charnumber**;** i++ ) //输出出现字符的出现次数

**if** ( C[i] > 0 ) **cout** << "( " << i << " ) : \t" << C[i] << "\t"**;**

**}**

2-16 设串s为“aaab”，串t为“abcabaa”，串r为“abcaabbabcabaacbacba”，试分别计算它们的失效函数f (j)的值。

【解答】

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 |  | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| s | a | a | a | b |  | t | a | b | c | a | b | a | a |
| f (j) | -1 | 0 | 1 | -1 |  | f (j) | -1 | -1 | -1 | 0 | 1 | 0 | 0 |

**四、其他练习题**

2-17 填空题

一维数组的逻辑结构是( ① )，存储结构是( ② )。对于二维数组，有( ③ )和( ④ )两种不同的存储方式。对于一个二维数组A[m][n]，若采取按行存放（行优先存储）的方式，则任一数组元素A[i][j]相对于A[0][0]的地址为( ⑤ )。

【解答】

① 线性结构 ② 顺序存储表示 ③ 行优先顺序 ④ 列优先顺序

⑤ n\*i+j

2-18 判断下列叙述的对错。如果正确，在题前打“√”，否则打“×”。

(1) 线性表的逻辑顺序与物理顺序总是一致的。

(2) 线性表的顺序存储表示优于链式存储表示。

(3) 线性表若采用链式存储表示时所有存储单元的地址可连续可不连续。

(4) 二维数组是其数组元素为线性表的线性表。

(5) 每种数据结构都应具备三种基本运算：插入、删除和搜索。

【解答】

(1) × (2) × (3) √ (4) × (5) √

2-19 设有一个10×10的对称矩阵A[10][10]，采取按行压缩存储的方式存放于一个一维数组B[ ]中，则数组B[ ]的容量应有多大？若设A[0][0]为第一个元素，存放于B[0]，且数组A[ ][ ]的每一个数组元素在数组B[ ]中占一个数组元素位置，则A[8][5]在数组B[ ]中的地址是多少？

【解答】

数组B共应有55个元素。对于上三角矩阵，A[8][5] = A[5][8] = (2n-i-1)\*i / 2 + j = 43，对于下三角矩阵， A[8][5] = ( i+1)\*i / 2 + j = 41。

2-20 设有上三角矩阵A[n][n]，将其上三角元素逐行存储到一维数组B[m]中，使得B[k] = A[i][j]，且k = f1 (i) + f2 (j) + C。试推导出函数f1 (i)、f2 (j)和常数C，要求f1 (i)和f2 (j)中不包含常数项。

【解答】

f (i) = (2n-i-1)\*i / 2，f (j) = j，C = 0

2-21 设有三对角矩阵A[n][n]，将其三条对角线中的元素逐行存储到一维数组B[3n-2 ]中，使得B[k] = A[i][j]。试求：

(1) 用i, j表示k的地址转换公式；

(2) 用k表示i, j的地址转换公式；

【解答】

(1) k = 2\*i+j (2) i = ⎣(k+1)/3⎦, j = k-2\*i

2-22 设定整数数组B[m+1][n+1]的数据在行、列方向上都按从小到大的顺序排序，且整型变量x中的数据在B中存在。试设计一个算法，找出一对满足B[i][j] *==* x的i, j值。要求比较次数不超过m+n。

【解答】

算法的思想是逐次二维数组右上角的元素进行比较。每次比较有三种可能的结果：若相等，则比较结束；若右上角的元素小于x，则可断定二维数组的最上面一行肯定没有与x相等的数据，下次比较时搜索范围可减少一行；若右上角的元素大于x，则可断定二维数组的最右面一列肯定不包含与x相等的数据，下次比较时可把最右一列剔除出搜索范围。这样，每次比较可使搜索范围减少一行或一列，最多经过m+n次比较就可找到要求的与x相等的数据。

**void** find ( **int** B[ ][ ], **int** m, **int** n, **int** x, **int&** i, **int&** j ) **{**

//在二维数组B[m][n]中寻找与x相等的元素, 找到后, 由i与j返回该数组元素的位置

i = 0**;** j = n**;**

**while** ( B[i][j] != x )

**if** ( B[i][j] < x ) i++**;**

**else** j--**;**

**}**

2-23 试编写一个函数，将一个有n个非零元素的整数一维数组A[n] 拆分为两个一维数组，使得A[ ] 中大于零的元素存放在B[ ] 中，小于零的元素存放在C[ ] 中。

【解答】

算法采用两个指针pb和pc，分别作为数组B[ ] 和C[ ] 的存放指针，以引用方式传递其结果。

**void** fenjie( **int** A[ ]**, int** B[ ]**, int** C[ ]**, int** n**, int&** pb**, int&** pc ) **{**

pb = pc = -1**;**

**while** ( **int** k = 0**;** k < n**;** k++ )

**if** ( A[k] > 0 ) B[++pb] = A[k]**;**

**else** C[++pc] = A[k]**;**

**}**

2-24 已知在一维数组A[m+n] 中依次存放着两个顺序表 (a0, a2, …, am-1) 和 (b0, b2, …, bn-1)。试编写一个函数，将数组中两个顺序表的位置互换，即将 (b0, b2, …, bn-1) 放在 (a0, a2, …, am-1) 的前面。

【解答】

#**define** mpn 20 //数组长度，要求大于或等于n+m

**typedef int** DataType**;** //数组元素的数据类型

**void** inverse ( DataType A[ ], **int** st, **int** ed )**;** //逆转序列所有元素位置

**void** exchange ( DataType A[ ]**, int** m**, int** n ) **{**

//数组A[m+n]中，从0到m-1存放顺序表(a0, a2, …, am-1)，从m到m+n-1存放顺序表

// (b0, b2, …, bn-1)。

inverse ( A, 0, m+n-1 )**;** //逆转 (bn-1, bn-2, …, b0, an-1, an-2, …, a0 )

inverse ( A, 0, m-1 )**;** //逆转 (b0, b1, …, bm-1, an-1, an-2, …, a0 )

inverse ( A, m, m+n-1 )**;** //逆转 (b0, b1, …, bm-1, a0, a1, …, an-1 )

**}**

**void** inverse ( DataType A[ ], **int** st, **int** ed ) **{** //逆转(ast, ast+1, …, aed )为( aed, aed-1, …, ast )

**int** md = ( st + ed ) / 2**;**

**for** ( **int** i = 0**;** i <= md-st**;** i++ )

**{** DataType temp = A[st+i]**;**  A[st+i] = A[ed-i]**;**  A[ed-i] = temp**; }**

**}**

2-25 试编写一个函数，以不多于3n/2的平均比较次数，在一个有 n个整数的顺序表A中找出具有最大值和最小值的整数。

【解答】

**void** FindMaxMin ( SeqList A**, int&** Max**, int&** Min ) **{**

//在顺序表中一趟扫描，分别找最大值Max与最小值Min。

Max = Min = A.Get[0]**;**

**for** ( **int** i = 1**;** i <= A.length( )**;** i++ ) **{**

**if** ( A.Get(i) > Max ) Max = A.Get(i)**;**

**else if** ( A.Get(i) < Min ) Min = A.Get(i)**;**

**}**

**}**

最坏情况：当表中n个整数按从大到小非递增排列时，数据比较次数为2(n-1) 次；最好情况：当表中n个整数按从小到大递增排列时，数据比较次数为n-1次；数据平均比较次数为

(2(n-1) + (n-1) ) / 2 = 3n/2 - 3/2 < 3n/2

2-26 设A = (a1, a2, …, am) 和B = (b1, b2, …, bn)均为顺序表，A’ 和B’ 分别是除去最大公共前缀后的子表。如A = ( b, e, i, j, i, n, g )，B = ( b, e, i, f, a, n, g )，则两者的最大公共前缀为b, e, i，在两个顺序表中除去最大公共前缀后的子表分别为A’ = ( j, i, n, g )，B’ = ( f, a, n, g )。若A’ = B’ = 空表，则A = B；若A’ = 空表且B’ ≠ 空表，或两者均不空且A’的第一个元素值小于B’的第一个元素的值，则A < B；否则A > B。 试编写一个函数，根据上述方法比较A和B的大小。

【解答】

**char** Compare ( SeqList**&** A, SeqList**&** B ) **{**

**int** i = 0, alen = A.length( ), blen = B.length( )**;**

**while** ( i <= alen **&&** i <= blen ) //寻找公共前缀

**if** ( A.Get(i) == B.Get(i) ) i++**;**

**else break;**

**if** ( i > alen **&&** i > blen ) **return** '='**;** //比较

**else if** ( a > alen || A.Get(i) < B.Get(i) ) **return** '<'**;**

**else return** '>'**;**

**}**

2-27 若采用数组来存储多项式的系数，即用数组的第i个元素存放多项式的i次幂项的系数，如对于多项式 f(x) = 6x6 + 7x4 -10x2 + 5x + 3，可用数组表示为：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | … | … |
| coef | 3 | 5 | -10 | 0 | 7 | 0 | 6 | … | … |

(1) 试编写一个算法，求两个多项式的和。

(2) 试编写一个算法，求两个多项式的乘积。

【解答】

#**define** defaultSize 100 //默认数组大小

#**include** <iostream.h>

**class** Polynomial **{**  //多项式类的定义

**private:**

**float** \* coef**;** //多项式数组指针

**int** MaxSize**,** degree**;** //多项式数组大小及多项式最高阶

**public:**

Polynomial ( **int** size = defaultSize ) //构造函数

**:** MaxSize ( size ), degree ( 0 ) **{** coef = **new float**[MaxSize]**; }**

~Polynomial ( ) **{ delete** [ ] coef**; }** //析构函数

**float** GetCoef ( e ) **{ return** ( e >= 0 **&&** e <= degree ) **?** coef[e] **:** 0**; }** //取系数

**void** setCoef ( **int** c**, int** e ) **{ if** ( e >= 0 **&&** e <= degree )coef[e] = c**; }** //改系数

**float** GetDegree ( ) **{ return** degree**; }** //取多项式的阶

**void** setDegree ( **int** d) **{ if** ( d >= 0 **&&** d <= MaxSize-1 )degree = d**; }** //改多项式的阶

**friend** Polynomial **operator** + ( Polynomial**,** Polynomial)**;**  //加法

**friend** Polynomial **operator** \* ( Polynomial**,** Polynomial)**;**  //乘法

**friend istream&** **operator** >> ( **istream&**, Polynomial**&** )**;** //建立多项式

**};**

**istream& operator** >> ( **istream&** in, Polynomial**&** pl ) **{**

**int** m = 0**; float** c**; int** e**;**

**cout** << "Enter the coef, exp and end with e *==* -1 :\n?"**;**

in>> c >> e **;**

**while** ( e != -1 ) **{**

**if** ( e <= MaxSize ) **{** coef [e] = c**; if** ( e > m ) m = e**; }**

**else cout <<** "Degree " << e << " out of range : " << MaxSize << **endl;**

in >> c >> e**;**

**}**

degree = m**;**

**return** in**;**

**}**

Polynomial **operator** + ( Polynomial ah**,** Polynomial bh ) **{**

//两个按升幂排列的多项式相加，返回结果多项式指针

Polynomial temp**;**

temp.setDegree ( ah.getDegree( ) + bh.getdegree ( ) )**;**

**for** ( **int** i = 0**;** i < getDegree ( )**;** i++ )

temp.setCoef ( ah.getCoef( i ) **+** bh.getCoef( i )**,** i )**;**

**return** temp**;**

**}**

Polynomial **operator** \* ( Polynomial ah**,** Polynomial bh ) **{**

Polynomial temp**; float** val**; int** i**,** j**,** k**,** m**,** n**,** low**,** high**;**

m = ah.getDegree ( )**;** n = bh.Degree ( )**;** k = m + n**;**

**for** ( i = 0**;** i <= k**;** i++ ) **{**

low = ( i <= m ) **?** 0 **:** i - m**;**

high = ( i <= n ) **?** i **:** n**;**

val = 0.0**;**

**for** ( j = low**;** j <= high**;** j++ )

val += ah.getCoef ( i - j ) \* bh.getCoef ( j )**;**

temp.setCoef ( val, i )**;**

**}**

temp.setDegree ( k )**;**

**return** temp**;**

**}**

若设有两个多项式ah和bh，它们分别为

*Am*(*x*) = *a*0 + *a*1*x* + *a*2*x*2 + *a*3*x*3 + … + *amxm*

*Bn*(*x*) = *b*0 + *b*1*x* + *b*2*x*2 + *b*3*x*3 + … + *bnxn*

则两个多项式相加，它们的和为

*Sk*(*x*) = *Am*(*x*) + *Bn*(*x*) = *s*0 + *s*1*x* + *s*2*x*2 + *s*3*x*3 + … + *skxk*

其中，*k* = max ( *m*, *n* )，*si* = *ai* + *bi* ( *i* = 0, 1, …, *k* )

两个多项式相乘，它们的乘积为

*Pk*(*x*) = *Am*(*x*) \* *Bn*(*x*) = *p*0 + *p*1*x* + *p*2*x*2 + *p*3*x*3 + … + *pkxk*

其中，*k* = *m* + *n*， ( *i* = 0, 1, …, *k* )

例如，若m = 3, n = 8，则有*A*3(*x*) = *a*0 + *a*1*x* + *a*2*x*2 + *a*3*x*3，*B*8(*x*) = *b*0 + *b*1*x* + *b*2*x*2 + *b*3*x*3 + *b*4*x*4 +

*b*5*x*5 + *b*6*x*6 + *b*7*x*7 + *b*8*x*8。它们的乘积为

*Pk*(*x*) = (*a*0 + *a*1*x* + *a*2*x*2 + *a*3*x*3) \* (*b*0 + *b*1*x* + *b*2*x*2 + *b*3*x*3 + *b*4*x*4 + *b*5*x*5 + *b*6*x*6 + *b*7*x*7 + *b*8*x*8) =

= *a*0*b*0 + (*a*0*b*1 + *a*1*b*0) *x* + (*a*0*b*2 + *a*1*b*1 + *a*2*b*0) *x*2 + (*a*0b3 + *a*1*b*2 + *a*2*b*1 + *a*3*b*0) *x*3 +

+ (*a*0*b*4 + *a*1*b*3 + *a*2*b*2 + *a*3*b*1) *x*4 + (*a*0*b*5 + *a*1*b*4 + *a*2*b*3 + *a*3*b*2) *x*5 + (*a*0*b*6 +*a*1*b*5 + *a*2*b*4 + *a*3*b*3) *x*6 +

+ (*a*0*b*7 + *a*1*b*6 + *a*2*b*5 + *a*3*b*4) *x*7 + (*a*0*b*8 + *a*1*b*7 + *a*2*b*6 + *a*3*b*5) *x*8 + (*a*1*b*8 + *a*2*b*7 + *a*3*b*6) *x*9 +

+ (*a*2*b*8 + *a*3*b*7) *x*10 + *a*3*b*8*x*11 =







因此，在实现算法中，当i = 0, 1, …, 3 (= m) 时，求累加和循环的下上界为：low = 0, high = i；当 i = 4, 5, …, 8 (= n) 时，求累加和循环的下上界为：low = i-m, high = i；当 i = 9, …, 11 (= m+n) 时，求累加和循环的下上界为：low = i-m, high = n。