第1章

绪　论

1.1　引言

电子计算机是20世纪科学技术最卓越的成就之一。自1946年第一台电子计算机问世以来，计算机产业和应用的发展远远超出了人们对它的预料。如今，计算机的应用已不再局限于科学计算，而更多地用于数据处理、信息管理、实时控制等非数值计算的各个方面。用数字计算机解决任何问题都离不开程序设计。为了编制“好”的程序，必须分析程序处理的数据的特性及数据之间的关系，这就是“数据结构”这门学科形成和发展的背景。

数据结构主要研究非数值应用问题中数据之间的逻辑关系和对数据的操作，同时还研究如何将具有逻辑关系的数据按一定的存储方式存放在计算机内。分析数据之间的逻辑关系和确定数据在计算机内的存储结构是程序设计前两个必须完成的任务。

处理非数值计算问题和数值计算问题的解决方案不同。例如，求解梁架结构中应力的数学模型为线性方程组，预报人口增长情况的数学模型为微分方程。但还有更多的非数值计算问题是无法用数学方程加以描述的。

例1-1　某单位职工档案的管理。

表1.1中的职工档案表就是一个数据结构。计算机档案管理的主要功能包括查找、浏览、插入、修改、删除、统计等。如果把表中的一行看成一个记录并称为一个结点，则在此表中，结点和结点之间的关系是一种最简单的线性关系。

表1.1　某单位职工档案表

工号	姓名	性别	出生年月	婚否	学历	进厂日期
0001	张丽萍	女	08/21/1962	已	大专	09/01/1984
0005	李小明	男	04/06/1972	未	大学	04/10/1996
0006	王冠英	男	06/06/1942	已	高中	03/12/1961
⋮	⋮	⋮	⋮	⋮	⋮	⋮

例 1-2 某学校教师的名册。虽然可以用例 1-1 中的二维表格将全校教师的名单列出,但采用图 1.1 所示的结构更好。它像一棵根在上而倒挂的树,清晰地描述了教师所在的系和教研组,这样一来可以从树根沿着某系某教研组很快找到某个教师,查找的过程就是从树根沿分支到某个叶子的过程。类似于树这样的数据结构可以描述家族的家谱、企事业单位中的人事关系,甚至可用树来反映人机下棋的动态过程等。

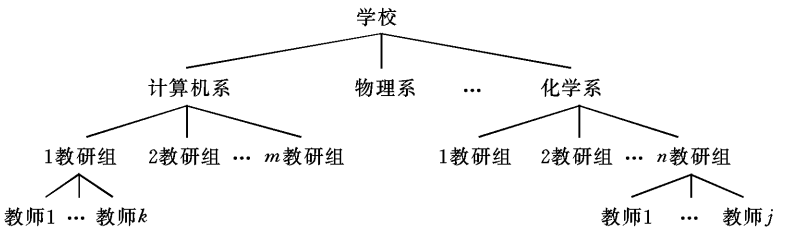
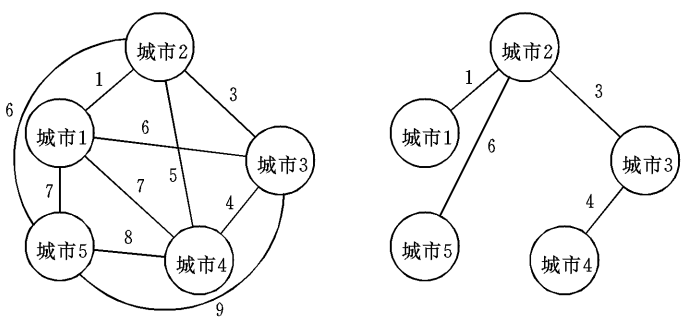


图 1.1 某学校教师名册

例 1-3 在 n 个城市之间建立通信网络,要求在其中任意两个城市之间都有直接的或间接的通信线路,在已知某些城市之间直接通信线路预算造价的情况下,使网络的造价最低。

当 n 很大时,这样的问题只能用计算机来求解。我们可以用图 1.2(a)中描述的关系来说明:图中的小圆圈表示一个城市,两个圆圈之间的连线表示对应城市之间的通信线路,连线上的数值表示该通信线路的造价。这一描述的结构为图状结构,利用计算机可以求出满足要求的最小造价通信网络,如图 1.2(b)所示。



(a) 通信网络中的数据　　(b) 最小造价通信网络

图 1.2 用图描述通信网络问题

通过上面三个例子可以看出:数据结构中元素和元素之间存在着逻辑关系,而线性表,树,图是三种基本的逻辑结构,其他各类的数据结构都是由这三种基本结构派生的。数据结构就是解决如何分析数据元素之间的关系、如何确立合适的逻辑结构、如何存储这些数据,并对为完成数据操作所设计的算法作出时间和空间的分析。“数据结构”在计算机科学中是一门综合性的专业基础课,它不仅是一般程序设计(特别是非数值计算的程序设计)的基础,而且也是设计和实现编译程序、操作系统、数据库系统及大型应用程序的重要基础。

简单来说，数据结构是研究程序设计中非数值计算的数据以及它们之间的关系和操作等的一门课程，重点分析数据之间抽象的相互关系，不涉及数据的具体内容。

1.2 基本概念和术语

数据(data) 是指所有能输入到计算机中并被计算机程序处理的符号的总称。计算机输入和处理的数据除数值外，还有字符串、表格、图像甚至声音等，它们都是数字编码范畴。

数据元素(data element) 数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。一个数据元素可以由若干个数据项组成，也可以只由一个数据项组成。数据元素又被称为元素、结点(node)或记录(record)。

数据项(data item) 是指数据的不可分割的、含有独立意义的最小单位，数据项有时也称字段(field)或域。上面的职工档案表格是要存放到计算机中进行处理的数据，表中每一行记录了一个职工的档案信息，在数据操作中作为一个整体考虑，对应为一个数据元素，又称为一个记录。这个记录中包含有工号、姓名、学历等若干个数据项。操作的基本单位是记录，如职工的插入或删除一定是作用于一个职工的全部信息即一个记录，而不可能是作用于其中的某个数据项。设想删除操作只删除某个职工的姓名或工号，将引起数据的不完整等严重后果。每个数据项(如职工的姓名或工号)均有独立的含义，但在档案管理这个实际问题中并无完整的意义，而组合在一个记录中构成职工的档案，就具有了完整的实际意义。数据、数据元素、数据项实际上反映了数据组织的三个层次：数据可由若干个数据元素构成，而数据元素又可以由一个或若干个数据项组成。

数据逻辑结构(data logical structure) 数据结构主要是研究数据元素之间的关联方式。数据元素之间存在的一种或多种特定的关系被称为数据的逻辑结构。通常有集合、线性结构、树形结构和图状结构四类基本结构。见图 1.3。

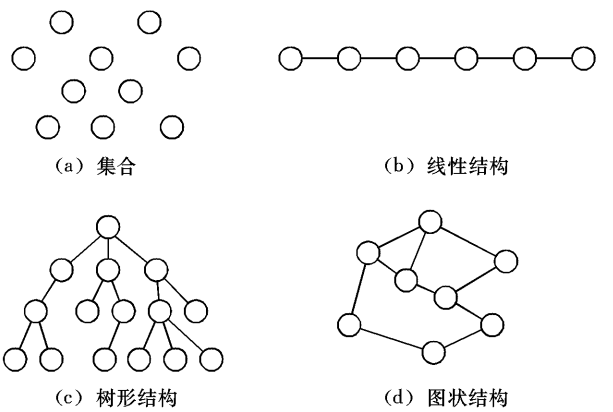


图 1.3 四类基本逻辑结构的示意图

数据物理结构(data physical structure) 数据在计算机中的存放方式称为数据的物理结构，又称存储结构。数据的存储结构是逻辑结构在计算机存储器中的实现。数据元

素在计算机中主要有两种不同的存储方法，即顺序存储结构和链式存储结构。顺序存储的特点是在内存中开辟一组连续的空间（高级语言中的数组）来存放数据，数据元素之间的逻辑关系通过元素在内存中存放的相对位置来确定，又称向量存储。链式存储的特点是通过指针反映数据元素之间的逻辑关系，又称动态存储。

数据的逻辑结构和物理结构是数据结构的两个密切相关的方面，同一逻辑结构可以对应不同的存储结构。算法的设计取决于数据的逻辑结构，而算法的实现依赖于指定的存储结构。

例如，10以内的奇数1,3,5,7,9用顺序存储结构的方式依次存放在以300为起地址的内存向量中，并且两个字长存放一个奇数，如图1.4(a)所示。在顺序存储结构中，如要读取第三个奇数，它的地址可以通过起地址和要读取奇数的位置序号计算得到。若本例中的奇数改用链式存储结构存放，那么，第一个奇数存放在地址为300的内存单元中，第二个奇数存放的地址和第一个奇数存放的地址无关，但第二个奇数所在的地址存放在第一个奇数相关的指针中，后面奇数的存放也按如此的规律。设两个字长存放一个奇数，一个字长存放一个指针，如图1.4(b)所示。在链式存储结构中，如要读取第三个奇数，只能从第一个奇数所在的地址开始，通过第一个奇数关联的指针得到第二个奇数存放的地址105，再找到第三个奇数存放的地址400，才能读出。从以上的例子和分析看，数据结构主要就是研究数据的逻辑结构、相应的存储结构以及完成数据操作的算法设计。

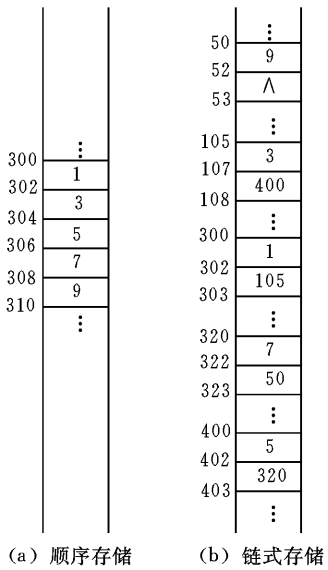


图1.4 两种存储结构的示意图

数据类型（data type） 和数据结构密切相关的一个概念，在用高级程序设计语言编写的程序中，每个变量、常量或表达式都对应一个确定的数据类型。数据类型可分为两类：一类是非结构的原子类型，如C语言中的基本类型（整型、实型、字符型等）、指针类型和空类型；另一类是结构类型，它的成分可以由多个结构类型组成，并可以分解。结构类型的成分中可以是非结构的，也可以是结构的。例如数组的值由若干分量组成，每个分量可以是整数，也可以是数组等结构类型。

本书在讨论各种数据结构时，针对其逻辑结构和具体的存储结构给出对应的数据类型，进一步在确定的数据类型上实现各种操作。

算法（algorithm） 是指解决特定问题的一种方法或一种描述。

1968年，美国唐·欧·克努特教授开创了数据结构的最初体系，他所著的《计算机程序设计技巧》第一卷《基本算法》是第一部较系统地阐述数据的逻辑结构、存储结构及算法的著作。20世纪70年代初，大型程序出现，软件业飞速发展，结构化程序设计成为程序设计方法学的主要内容，人们越来越重视数据结构，认为程序设计的实质是确定数据的结构，加上设计一个好的算法，也就是人们所说的“程序＝数据结构＋算法”。

1.3 算法描述

程序设计人员需要对程序处理的问题准确理解，只有准确理解问题后才能研究出解决问题的方法。算法是指解决问题的一种方法或过程描述。如果将问题看作函数，那么算法能把输入转化为输出。解决一个问题可以有多种算法，但一个给定的算法只能解决一个特定的问题。例如对一组数据的排序，可给出5种甚至更多种的排序算法。可以用多种算法求解问题的优点在于：根据问题的具体限定条件，可以选用合适的算法求解。例如，有的排序算法适合于元素个数少的序列，有的算法适合于元素个数多的序列，有的算法则适合于定长数值型数据的排序。计算机程序就是用某种程序设计语言去具体地实现一个算法，或称为代真。本书中主要介绍各种算法，并给出一部分算法对应的C语言程序。当然使用其他的程序设计语言也可以实现算法的代真。综上所述，问题、算法、程序是三个互相关联的不同概念。

1.3.1 算法的重要特性

- 正确性　它必须解决具体的问题，完成所期望的功能，给出正确的输出。
- 确定性　算法执行的每一步和下一步必须确定，不能有二义性。
- 有限性　一个算法必须由有限步组成。无限步组成的算法无法用计算机程序来实现。因此算法必须可以终止，不能进入死循环。
- 输入　一个算法有零个或多个输入。
- 输出　一个算法有一个或多个输出。

1.3.2 数据结构上的基本操作

基本操作主要有以下几种：

- 查找　寻找满足特定条件的数据元素所在的位置。
- 读取　读出指定位置上数据元素的内容。
- 插入　在指定位置上添加新的数据元素。
- 删除　删去指定位置上对应的数据元素。
- 更新　修改某个数据元素的值。

根据操作的结果可将操作分为两种基本类型：

- 加工型操作　其操作改变了原逻辑结构的“值”，如数据元素的个数、某数据元素的内容等(一般不考虑改变逻辑结构的类型)。上面基本操作中的后三种操作均为加工型操作。
- 引用型操作　其操作不改变原逻辑结构的“值”，只是查找或读取。

1.3.3 算法的描述方法

算法的描述方法有很多，根据描述算法语言的不同，可将算法分为以下四种：

- 框图算法描述　这种描述方法在算法研究的早期曾流行过。它的优点是直观、易

懂，但用来描述比较复杂的算法就显得不够方便，也不够清晰简洁。

- 非形式算法描述　用中文语言，同时还使用一些程序设计语言中的语句来描述算法，这称为非形式算法描述。
- 类 C 语言算法描述　类 C 语言算法又称为伪语言算法。这种算法不能直接在计算机上运行，但专业设计人员经常使用类 C 语言来描述算法，它容易编写、阅读和统一格式。
- C 语言编写的程序或函数　这是可在计算机上运行并获得结果的算法，使给定问题能在有限时间内被求解，通常这种算法也称为程序。

下面以求两个整数 $m,n(m\geqslant n)$ 的最大公因子为例来看看不同的算法描述的方法。

(1) 该问题的框图描述如图 1.5 所示。

(2) 非形式算法描述：

a. [求余数] 以 n 除 m，并令 r 为余数 $(0\leqslant r<n)$；

b. [余数是零否] 若 $r=0$ 则结束算法，n 就是最大公因子；

c. [替换并返回 a] 若 $r\neq 0$ 则 $m\leftarrow n$，$n\leftarrow r$ 返回 a。

(3) C 语言函数描述：

```
int max_common_factor (int m, int n)
{
   int r ;
   r = m % n;
   while (r != 0)
      {m = n ; n = r ; r = m % n;}
   return n ;
}
```

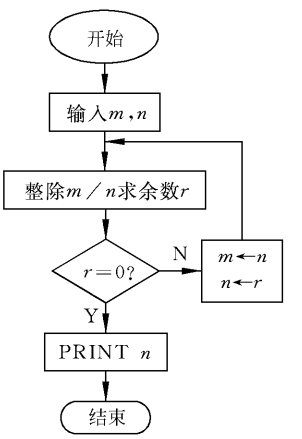


图 1.5　框图描述法

本书主要介绍算法的思路和实现过程，且尽可能地将算法对应的 C 语言函数或程序提供给读者阅读或上机运行，以便更好地理解算法。C 语言函数或程序用到的数据类型说明的详细列表见附录 A。

1.4　算法分析

1.4.1　算法设计的要求

设计一个“好”的算法应考虑以下几个方面：

- 易读性　算法应易于阅读和理解，以便于调试、修改和扩充。
- 健壮性　正确的输入能得到正确的输出这是算法必须具有的特性之一。但当遇到非法输入时，算法应能作出反应或处理(如提示信息等)，而不会产生不需要的或不正确的结果。
- 高效率　即达到所需的时空性能。一个算法的时空性能是指该算法的时间性能(时间效率)和空间性能(空间效率)。解决同一问题如果有多个算法，执行时间短

的算法时间效率高，而存储量和辅助空间量少的算法空间效率高。这两者都和问题的规模有关。

1.4.2 算法时间效率的度量分析

本节重点介绍算法的时间效率分析的基础知识。算法运行的时间分析和程序运行的时间分析有区别。同一算法由不同的编程员所编出来的程序有优劣之分，程序运行的时间也就有所不同；程序在不同的机器上运行的速度又和机器本身的速度有关。我们感兴趣的是对解决问题的算法作时间上的度量分析，或对解决同一问题的两种或两种以上的算法运行的时间加以比较。我们称这种度量分析为算法的时间复杂度分析。它可以估算出当问题的规模变大时，算法运行时间增长的速度。这种分析实际上是一种数学化了的估算方法。

估算算法运行时间的基本考虑是：确定问题的“规模”和确定算法执行“基本操作”的次数。一个算法的“规模”和“基本操作”要视具体算法而定。“规模”一般是指输入量的数目，比如在排序问题中，问题的规模可以是被排序的元素数目。“基本操作”一般是指在某个数据类型上的“标准操作”，比如两个整数相加、比较两个整数的大小等都可以视为是基本操作。

例 1-4 查找一维 n 元整数数组中最大元素的算法。该算法从数组中下标为 0 的元素开始，遍历数组中的所有元素，在遍历过程中，将当前最大元素保存在变量 currlarge 中。下面是对应的算法：

```
int largest (int * array, int n)
{
    int currlarge, i ;
    currlarge = array[ 0 ] ;
    for(i = 1; i < n ; i++)
        if (array[i] > currlarge) currlarge = array[i];
    return currlarge ;
}
```

其中，数组 array 中存放有 n 个整数，则问题的规模为 n。基本操作是“比较”，即将数组中的一个整数和现有的最大整数作比较。影响算法运行时间的最主要的因素就是输入规模 n，我们经常把运行算法所需要的时间 T 写成输入规模 n 的函数，记作 $T(n)$。

我们把 largest 函数中比较一个元素所需要的时间记作 c_1，“比较”这一基本操作是对数组中每一个元素都要做的工作。算法主要考虑这一基本操作所花的时间，忽略当找到一个新的最大元素时要做的工作的时间，也不考虑函数初始化时所需要的时间，这样就能得到运行该算法的一个合理的近似时间。因此，运行 largest 函数的总时间可近似地认为是 $c_1 n$。largest 函数运行的时间代价可以用下面的等式来表示：$T(n)=c_1 n$。这个等式表明了顺序检索数组中最大元素的算法的时间是随着 n 的增长而线性增长。

例 1-5 将一个整数数组的第一个元素值赋给另一个变量。完成这一功能所需要的时间是固定的。无论这个数组有多大，复制一个元素值的时间总是确定的，记作 c_2。因此该算法运行时间代价的等式就是 $T(n)=c_2$。输入规模即 n 的大小对运行时间不产生影

响。这个等式称为常数运行时间。

例 1-6 再看下面一个算法段：

```
sum = 0;
for (i = 1; i <= n; i++)
  for(j = 1; j <= n; j++)
sum++;
```

显然，随着 n 的增大，其运行时间也会增大。本例的基本操作是 sum 的累加，假设这个基本操作所需要的时间为 c_3，忽略了初始化 sum 的时间和循环变量 i 和 j 累加的时间，基本操作总次数为 n^2，因此，该算法运行时间代价可用下面的等式来表示：$T(n)=c_3 \cdot n^2$。

增长率的概念是非常重要的。它可以帮助我们比较算法的运行效率。图 1.6 给出了五个常见的运行时间函数的曲线，每一条曲线反映出某种算法的时间代价，显示了不同算法的增长率。标记为 $100n$ 的函数为一直线，增长率称为线性增长率。这说明，当 n 增大时，算法的运行时间线性增大。如果算法的运行时间函数中含有如 n^2 这样的高次项，则称为二次增长率，图中标有 $5n^2$ 的曲线就代表二次增长率。标有 2^n 的曲线属于指数增长率。

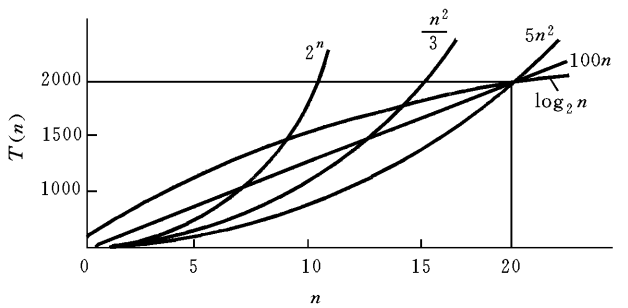


图 1.6 常见函数的增长率

由于算法的时间复杂度分析只考虑相对于问题规模 n 的增长率，因而，在难以精确计算基本操作执行次数的情况下，只要求出它关于 n 的增长率即可。我们可以在计算任何算法运行时间代价时，忽略所有的常数和低次项，用 O 表示法来表示算法的时间复杂度。例 1-4 中算法的时间复杂度为 $O(n)$，例 1-5 中算法的时间复杂度为 $O(1)$，例 1-6 为 $O(n^2)$，分别称为线性阶、常量阶和平方阶。如果某算法的运行时间代价为 $O(5n^2+n)$，可以忽略其中的低次项和常数，而视该算法的时间复杂度为 $O(n^2)$。算法的时间复杂度还有对数阶 $O(\log_2 n)$、指数阶 $O(2^n)$等。

1.5 应用举例及分析

例 1-7 用 C 语言描述下列算法，并给出算法的时间复杂度。

(1) 求一个 n 阶方阵的所有元素(正整数)之和。对应的算法如下：

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

本算法的时间复杂度为 $O(n^2)$。

此算法中的基本操作是“s=s+A[i][j];”语句，问题的规模和 n 有关。算法中含有双重 for 循环语句，其中外循环 n 次。对每一次外循环，内循环执行“s=s+A[i][j];”语句 n 次。总的“s=s+A[i][j];”语句共执行 n^2 次，时间复杂度为 $O(n^2)$。

(2) 对于输入的任意 n 个整数，输出其中的最大和最小元素。对应的算法如下：

```
void maxmin ( int A[], int n, int &max, int &min)
{
    int i;
    max = min = A[0];
    for ( i = 1; i < n; i++)
        {if (A[i] > *max) max = A[i];
         if (A[i]< *min) min = A[i];
        }
}
```

本算法的时间复杂度为 $O(n)$。

算法中只含有一个 for 循环语句，共执行 n 次。每一次或执行“if(A[i]>*max)max=A[i];”语句，或执行“if(A[i]<*min)min=A[i];”语句。时间复杂度为 $O(n)$。

例 1-8 判断以下叙述的正确性。

(1) 数据元素是数据的最小单位。

错误。数据元素是数据的基本单位，在计算机程序中通常作为一个整体来考虑和处理。一个数据元素可以由若干个数据项组成，也可以只有一个数据项组成。数据项是数据的不可分割的有独立含义的最小单位。

(2) 数据对象是由一些类型相同的数据元素构成的。

正确。在数据对象上实现某些操作，首先要确定数据结构。而数据对象必须是由类型相同的数据元素所构成。

(3) 数据的逻辑结构与数据元素在计算机中如何存储有关。

错误。数据在计算机中如何存储称为数据的物理结构，又称存储结构。数据的逻辑结构和物理结构是数据结构的两个密切相关的方面，同一逻辑结构的数据可以对应不同的存储结构。

(4) 逻辑结构相同的数据，可以采用多种不同的存储方法。

正确。

例 1-9 当为解决某一问题而选择数据对象的数据结构时，应从哪些方面考虑？

从两方面考虑,第一是以此结构为基础实现的算法所需的存储空间量,第二是算法所需的时间。时间考虑包括程序运行时所需输入的数据总量;程序中指令重复执行的次数,也就是估算当问题的规模变大时,算法运行时间增长的速度。

习　题

1-1　简述下列术语:数据、数据元素、数据项、数据逻辑结构、数据存储结构、数据类型、算法。

1-2　分析下面语句段执行的时间复杂度。

(1)
```
for (i = 1; i <= n ; i ++)
    for (j = 1; j <= n ; j ++)
      s ++;
```

(2)
```
for (i = 1; i <= n ; i ++)
    for (j = i; j <= n ; j ++)
      s ++;
```

(3)
```
for (i = 1; i <= n ; i ++)
    for (j = 1; j <= i; j ++)
      s ++;
```

(4)
```
i = 1; k = 0;
while (i <= n - 1) {
   k += 10 * i;
   i ++;
}
```

1-3　试画出与下列程序段等价的框图。

(1)
```
p = 1; i = 1;
while ( i <= n ) {
     p * = i;
     i ++;
}
```

(2)
```
i = 0;
do {
   i ++;
}while (( i !=n) && ( a[i] != x ));
```

1-4　按 n 的增长率由小至大顺序排列下列各函数。

$$(2/3)^n,\ (3/2)^n,\ n^2,\ n^n,\ n!,\ 2^n,\ n,\ \log_2 n,\ n^3$$

1-5　写一算法,自大至小依次输出顺序读入的三个整数 X、Y 和 Z 的值。

1-6　编一程序,输出所有小于等于 n(n 为一个大于 2 的正整数)的素数。

实 训 题

1-7 举出一个数据结构的例子，叙述其逻辑结构、存储结构及在其结构上的操作内容。

1-8 判断以下叙述的正确性。

(1) 数据项是数据的最小单位。

(2) 数据的物理结构是指数据在计算机内的实际的存储形式。

(3) 顺序存储方式只能用于存储线性结构。

(4) 逻辑结构不相同的数据，要采用不同的存储方法来存储。

1-9 编一程序，计算任一输入的正整数的各位数字之和。

第2章

线 性 表

数据结构分线性结构和非线性结构。第2章、第4章、第5章分别介绍各种线性的数据结构，包括线性表、栈、队列、数组和串。线性结构的特点是在数据元素的非空有限集合中：

- 存在唯一的“第一个”数据元素；
- 存在唯一的“最后一个”数据元素；
- 除第一个数据元素之外，集合中的每一个数据元素都只有一个前驱；
- 除最后一个数据元素之外，集合中的每一个数据元素都只有一个后继。

2.1 线性表的定义及逻辑结构

线性表是线性结构中最常用而又最简单的一种数据结构。可以简单定义为：一个线性表是 n 个数据元素的有限序列。例如从1～30的质数可以放在一个线性表中：(1,2,3,5,7,11,13,17,19,23,29)。一个星期中的七天可放在一个线性表中：(星期一、星期二、……星期六、星期日)。又如，一个学校的学生健康情况登记表如表2.1所示，表中每一个学生的情况为一个记录，它由姓名、学号、性别、年龄、班级和健康状况等六个数据项组成。

表2.1 学生健康情况登记表

姓 名	学 号	性 别	年 龄	班 级	健康状况
王 林	950631	男	18	计18	健康
蔡 明	950632	男	20	计18	一般
张立华	950730	女	19	计19	健康
吴 红	950731	女	21	计19	一般
⋮	⋮	⋮	⋮	⋮	⋮

从上面的例子中可以看出，线性表中的数据元素可以是各种各样的，但同一表中的元素必定具有相同特性。表中的一个数据元素可以由若干个数据项组成，也可以只由一个数据项组成，通常把数据元素称为记录，有大量记录的线性表称为文件。

线性表的长度 $n(n\geqslant 0)$就是表中数据元素的个数。$n=0$ 时，称为空表，$n>0$ 时，线性表的表示形式为$(a_1,a_2,\cdots,a_n)$。

线性表具有线性结构的特点，表中 a_i 元素的直接前驱元素是 a_{i-1}，a_i 元素的直接后继元素是 a_{i+1}。数据元素在线性表中的位置只取决于它的序号。

线性表	$(a_1,$	$a_2,$	$a_3,$	$\cdots,$	$a_{n-1},$	$a_n)$
序号	1	2	3		$n-1$	n

2.2 线性表的基本操作

(1) INITIATE(L)　初始化操作函数。生成一个空的线性表 L。

(2) LENGTH(L)　求表长度的函数。函数值为线性表 L 中数据元素的个数。

(3) GET(L, i)　取表中元素的函数。当 $1\leqslant i\leqslant$ LENGTH(L)时，函数值为线性表 L 中第 i 个数据元素，否则返回一特殊值。i 是该数据元素在线性表中的位置序号。

(4) LOCATE(L, x)　定位函数。给定值 x，在线性表 L 中若存在和 x 相等的数据元素，则函数返回和 x 相等的数据元素的位置序号，否则返回 0。若线性表中存在一个以上的和 x 相等的数据元素，则函数返回多个位置序号中的最小值，也就是表中第一个和 x 相等的元素的位置序号。

(5) INSERT(L, b, i)　插入操作。在给定线性表 L 中第 i(1≤i≤LENGTH(L)+1)个数据元素之前插入一个新的数据元素 b，使原来线性表的长度 n 变成 $n+1$。

(6) DELETE(L, i)　删除操作。删除在给定线性表 L 中第 i(1≤ i≤LENGTH(L))个数据元素，使原来线性表的长度 n 变成 $n-1$。

(7) EMPTY(L)　判空表函数。若 L 为空表，则返回布尔值“真”，否则返回布尔值“假”。

(8) CLEAR(L)　表置空操作。不管原来的线性表 L 是空表还是非空表，操作结果将 L 表置空。

以上基本操作中，(1)，(5)，(6)，(8) 是加工型操作，其他都是引用型操作。

2.3 线性表的顺序存储结构

线性表的顺序存储是计算机中最简单、最常用的一种存储方式，即用一组地址连续的存储单元依次存放线性表的元素。由于同一线性表中元素的类型相同，可设定一个元素占用 b 个存储单元，表中第一个元素存放的地址作为线性表的存储起地址 LOC(a_1)，用 h 来表示。

线性表顺序存储的特点是：表中相邻的元素 a_i 和 a_{i+1} 所对应的存储地址 LOC(a_i) 和地址 LOC(a_{i+1})也是相邻的。也就是说表中元素的物理关系和逻辑关系是一致的。只要知道线性表的起始地址 LOC(a_1)$=h$ 和一个元素占用的存储单元 b，表中任意一个元素的存储起地址可用公式得到：LOC(a_i)=LOC(a_1)+($i-1$)b(1≤i≤n)。本书中将顺

序存储结构的线性表称为顺序表。

下面是顺序表的逻辑表示和对表中元素存储地址计算的分析示意：

逻辑表示	$(a_1,$	$a_2,$	$a_3,$	…,	$a_i,$	…,	$a_{n-1},$	$a_n)$
元素在表中的位置序号	1	2	3		i		$n-1$	n
存储地址	h	$h+b$	$h+2b$		$h+(i-1)b$			$h+(n-1)b$

从计算公式可以看出，计算顺序表中每一个元素的存储起地址的时间是相同的，读取表中元素所花的时间也是一样的。顺序表中任一元素都可以随机存取，所以线性表的顺序存储结构是一种随机存取的存储结构。在这种结构上很容易实现线性表的某些操作，如随机存取表中第 i 个元素等。但是，从下面对表中插入元素和删除元素的操作中可看到，因这些操作需要移动元素而要花去大量的时间。

2.4 基本操作在顺序表上的实现

2.4.1 顺序表上元素的插入

插入操作是指在长度为 n 的线性表中第 i（$1 \leqslant i \leqslant n$）个元素之前插入一个元素 x，使长度为 n 的线性表变为长度为 $n+1$ 的线性表。顺序表的数据类型描述如下：

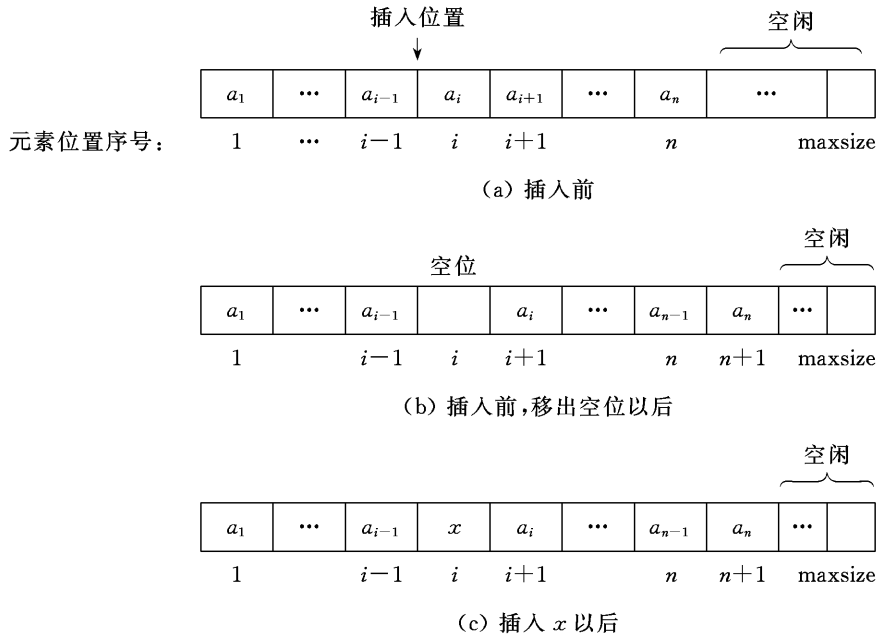
```
#define   DATATYPE1   int
#define   MAXSIZE     100

typedef struct
{ DATATYPE1 datas[MAXSIZE];
  int last;
}SEQUENLIST;
```

算法的思路是：因为要使插入元素后的线性表仍具有线性表的结构特征，必须将元素 a_i，…，a_n 逐一向后移动一个位置，腾出第 i 个位置，然后，再将 x 置入该位置中，表长加1。这里移动次序十分重要，从逻辑上考虑，只能按 a_n，a_{n-1}，…，a_i 的次序进行，先将 a_n 向后移一位，再将 a_{n-1} 移到 a_n 原来的位置上，依次类推，直到将 a_i 移到 a_{i+1} 原来的位置上，才能将 x 置入原 a_i 所在的位置中。这样，原线性表在插入新元素以后仍为一线性表。图 2.1给出了顺序表在插入元素前后的状况图。并给出了在顺序表中插入元素的算法。这里要特别注意的是：C 语言中数组的下标从“0”开始，因此，若 a 是 SEQUENLIST 类型的顺序表，则表中第 i 个元素在 C 语言数组中对应 a. datas[i－1]。

顺序表初始化函数如下：

```
void init_sequelist (SEQUENLIST a)
{
    a.last = 0 ;
    return ;
}
```



(a) 插入前

(b) 插入前，移出空位以后

(c) 插入 x 以后

图 2.1　顺序表插入一个元素的过程示意图

顺序表中插入一个元素的算法如下：

```
int insert (SEQUENLIST a, DATATYPE1 x, int i)
// 将新元素 x 插在顺序表 a 的第 i (1 ≤ i ≤ a.last + 1)个元素的前面
{
  int k;
  if (i < 1 || i > a.last + 1 || a.last >= MAXSIZE)
     rerurn 0;
  else {
        for (k = a.last ; k >= i ; k --)
             a.datas[ k ] = a.datas[k - 1];
        a.datas[i - 1] = x;
        a.last = a.last + 1;
        return 1; }
}
```

现在分析上面插入算法的时间复杂度。设表长 a.last 为 n。该插入算法的基本操作是元素后移操作。执行元素后移的次数是 $n-i+1$。可以看到移动元素的次数不仅和表长有关，而且还与插入元素的位置 i 有关。当 $i=n+1$ 时，无须移动元素，当 $i=1$ 时，则元素后移将执行 n 次，也就是说该算法在最好情况下时间复杂度是 $O(1)$，最坏情况下时间复杂度是 $O(n)$。进一步分析算法的平均性能：考虑在长度为 n 的线性表中插入一个元素，令 E_{is} 为移动元素的平均次数，在表中第 i 个元素前插入一个元素要移动元素的次数为 $n-i+1$，故

$$E_{is} = \sum_{i=1}^{n+1} P_i(n-i+1)$$

P_i 表示在表中第 i 个位置前插入一个元素的概率，假设在表中任何有效位置前($1 \leqslant i \leqslant n+1$)插入元素的机会是均等的，则

$$P_i = \frac{1}{n+1}$$

因此

$$E_{is} = \sum_{i=1}^{n+1}(n-i+1)/(n+1) = \frac{n}{2}$$

也就是说，在顺序表上做插入操作，平均要移动表的一半元素。就数量级而言，它是线性阶的，算法的平均时间复杂度为 $O(n)$。

对后面的各种算法，我们不再详细分析和推导算法的时间复杂度，对有些算法则只给出推导的结果。

2.4.2 顺序表上元素的删除

在顺序表上实现删除操作也必须移动元素才能使删除后的线性表仍具有线性结构的特征。这里移动次序也十分重要，从逻辑上考虑，只能按 $a_{i+1}, a_i, \cdots, a_n$ 的次序进行，先将 a_{i+1} 向前移一位，从而覆盖掉被删除元素，再将 a_{i+2} 移到 a_{i+1} 原来的位置上，依次类推，直到将 a_n 移到 a_{n-1} 原来的位置上，并将表长减 1。下面是在顺序表上删除第 i 个元素的算法：

```
int delete (SEQUENLIST a , int i)
// 在顺序表 a 中删除第 i 个元素
{
int k ;
if (i < 1 || i > a. last || a. last == 0)
   return 0;
else { for (k = i ; k < a. last ; k ++)
          a. datas[k - 1] = a. datas[k];
       a. last --;
       return 1; }
}
```

如果希望被删除元素在删除前保留起来以备后用，可在上面的程序中，在移动元素 for 语句前加一条语句：x=a. datas[i]；程序的修改和调试留给读者完成。

2.4.3 顺序表上元素的定位

这是一个引用型操作，对线性表中的元素只有访问没有变动，所以无移动元素的操作。基本操作是判定表中元素值是否和给定值 x 相等。算法如下：

```
int locate (SEQUENLIST a, DATATYPE1 x)
// 顺序表 a 中存在和给定值 x 相等的元素，返回该元素在表中的位置(从 1 算起)，
// 否则返回 0
```

```
{
    int k;
    k = 1;
    while (k <= a.last && a.datas[k - 1] != x)
        k++;
    if (k <= a.last)
        return k ;
    else return 0;
}
```

在顺序表上做元素的删除操作和定位操作的算法的时间复杂度均为 $O(n)$。

2.5 应用举例及分析

例 2-1 已知顺序表中元素值递增有序。用算法实现将元素 x 插到表中适当的位置上，以保持顺序表的有序性，且分析算法的时间复杂度。

```
void insert_x_seqlist (SEQUENLIST a, int x)
{
    int i,m ;
    i = a.last;
    while ((i >= 1) && ( x < a.datas[i])) i--;
    for(m = a.last; m >= i + 1; m--) a.datas[m + 1] = a.datas[m];
    a.datas[i + 1] = x;
    a.last ++;
}
```

算法的时间复杂度均为 $O(n)$。算法中虽有两个循环语句，但并不嵌套，查找插入位置 i 的语句是“while((i>=1)&&(x<a.datas[i]))i--;”，最多执行 a.last 次，移动元素的语句是“for(m = a.last; m >= i + 1; m--)a.datas[m + 1] = a.datas[m];”，最多也执行 a.last 次，所以算法的时间复杂度为 O(a.last)+O(a.last)=O(2 * a.last)->O(n)。

例 2-2 将所有在顺序表 lb 中存在而在顺序表 la 中不存在的数据元素插入到表 la 中。

这个例子实现的思路是：从顺序表 lb 中依次取出每一个元素，并在顺序表 la 中查访，若在表 la 中不存在，则可插到表 la 中。而且每个插入到表 la 中的元素均统一规定插在表 la 的尾部，这样可节省算法执行的时间。过程中的查访和插入可调用前面的 locate 和 insert 函数。对应的算法如下：

```
void unite (SEQUENLIST la, SEQUENLIST lb)
{
    int i ;
    for ( i = 1 ; i <= lb.last ; i ++)
        if ( !locate (la , lb.datas[i -1] ))
            insert ( la, lb.datas[i -1], la.last + 1);
}
```

例 2-3 已知顺序表 la 和 lb 中的元素依值非递减有序排列,编写一算法将表 la 和 lb 归并到新的顺序表 lc 中,表 lc 中的元素也依值非递减有序排列。例如:

la=(3,6,9,11)

lb=(4,6,8,11,13,17,20)

则 lc=(3,4,6,6,8,9,11,11,13,17,20)

从例子中可看到,表 lc 中的元素不是表 la 中的元素,就是表 lb 中的元素,表 lc 中元素的个数是 la 中元素个数和 lb 中元素个数之和。为了使表 lc 中元素也依值非递减有序排列,可设三个指针 i, j, k 分别指向 la、lb 和 lc 中对应的元素,当 la 的元素放入 lc 时,i, k 指针加 1,当 lb 的元素放入 lc 时,j, k 指针加 1,三个指针的初值均为 1。对应的算法如下:

```
void merge_sqlist(SEQUENLIST la,SEQUENLIST lb,SEQUENLIST lc)
{
  int i , j , k ;
  i = j = k = 1 ;
  while( i <= la. last && j <= lb. last )        //la 和 lb 均不空
    if( la. datas[i - 1] <= lb. datas[j - 1])
      {lc. datas[k - 1] = la. datas[i - 1] ;
       k++ ;
       i++ ; }
    else
      {lc. datas[k - 1] = lb. datas[j - 1] ;
       k++ ;
       j++ ; }
  while( i <= la. last )                         //lb 已空,la 非空
    {  lc. datas[k - 1] = la. datas[i - 1] ;
       k++ ;
       i++ ;}
  while( j <= lb. last )                         //la 已空,lb 非空
    {  lc. datas[k - 1] = lb. datas[j - 1] ;
       k++ ;
       j++ ;}
  lc. last = k - 1;                              //lc 表长
  return;
}
```

例 2-4 一元多项式相加。

数学上,n 阶一元多项式可以用下面的式子表示:$A(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x^1+a_0x^0(a_n\neq 0)$,$A(x)$的阶数为 n。一个 n 阶一元多项式中含有 $n+1$ 项系数,并唯一确定,分别对应 x^n 的系数到 x^0 的系数。在计算机中,描述一个一元多项式,可用线性表表示 $A=(n,a_n,a_{n-1},a_{n-2},\cdots,a_1,a_0)$,其中第一个元素是阶数,后面都是系数,按阶次逐一递减排列,A 的长度是 $n+2$。数学上,一元多项式的另一种表示方法是:$A(x)=b_mx^{e_m}+b_{m-1}x^{e_{m-1}}+\cdots+b_1x^{e_1}$,式中每一项都是非零系数项,$b_i$ 是非零系数,e_i 具有递减性。这种表示法对稀疏型多项式特别适合。在计算机中,可用线性表表示 $A=(m,e_m,b_m,e_{m-1},$

b_{m-1},…,e_1,b_1),其中,第一个元素是多项式中非零系数的项数,后面是每个项对应的阶次和系数,每两项 e_i,b_i 对应多项式中的某一非零系数项的指数和系数,A 的长度是 $2m+1$。

例如,多项式 $X^4+10X^3+3X^2+1$ 对应上述第一种表示方法的线性表是(4,1,10,3,0,1),对应上述第二种表示方法的线性表是(4,4,1,3,10,2,3,0,1)。多项式 $X^{1000}+1$ 对应上述第一种表示方法的线性表是(1000,1,0,0,0,0,…,0,1),表中的 0 的个数是 999 个,对应上述第二种表示方法的线性表是(2,1000,1,0,1)。

下面给出多项式相加的算法,多项式用上述第二种表示方法对应的线性表表示。例如,多项式 $A(x)=5x^{17}+8x^7-3x^5+2x-4$ 和 $B(x)=4x^9-22x^7+3x^5+6$ 对应的两个线性表是 a=(5,17,5,7,8,5,−3,1,2,0,−4)和 b=(4,9,4,7,−22,5,3,0,6),多项式相加得到 $C(x)=A(x)+B(x)=5x^{17}+4x^9-14x^7+2x+2$,对应的顺序表 c=(5,17,5,9,4,7,−14,1,2,0,2)。为了算法清晰,设定 C 语言中开设的数组的 0 单元空闲不用。a 表的长度为 a.datas[1] * 2+1。b 表的长度为 b.datas[1] * 2+1。表 a、表 b 和表 c 中元素存放的示意图如图 2.2 所示。

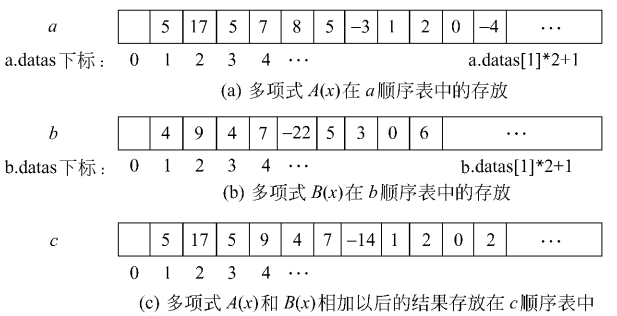


图 2.2　多项式相加结构示意图

按照多项式相加原则,当表 a 和表 b 中存在有两个阶数相同的项式时,则将两项的系数相加。相加得到的项式的系数不为零,则产生一个新的项式放入表 c 中。由于表中只存放非零系数的项式,阶数具有递减性,可设三个指针 p,q,r 分别指向表 a、表 b 和表 c 中对应的项式。三个指针初值均为 2。每处理一个非零系数项,指针加 2。表 c 中的非零系数项的个数要到最后相加结束时才能计算得到,并存入 c.datas[1]中。对应的算法如下:

```
void polynomial_add( SEQUENLIST a,SEQUENLIST b,SEQUENLIST c)
//多项式 a,b,c 用顺序存储结构,0 单元空闲不用//
{
    int m , n;
    int p , q ,r;
    m = a.datas[1];      //表 a 非零系数项个数
    n = b.datas[1];      //表 b 非零系数项个数
    p = q = r = 2;
    while ( p <= 2 * m && q <= 2 * n)
      {if(a.datas[p] == b.datas[q])      //指数相等,则系数相加
        {c.datas[r+1] = a.datas[p+1] + b.datas[q+1];
```

```
            if(c.datas[r+1] ! = 0)
              {c.datas[r] = a.datas[p];
              r = r + 2;}
            p = p + 2;
            q = q + 2;}
        else if(a.datas[p] > b.datas[q])              //指数不等
              {c.datas[r+1] = a.datas[p+1];
               c.datas[r] = a.datas[p];
               p = p + 2;
               r = r + 2;}
            else {c.datas[r+1] = b.datas[q+1];     //指数不等
               c.datas[r] = b.datas[q];
               q = q + 2;
               r = r + 2;}
          }
        while(p < 2 * m)        //如果表 b 元素已处理完,表 a 中还有非零系数项要处理
          {c.datas[r] = a.datas[p];
           c.datas[r+1] = a.datas[p+1];
           p = p + 2;
           r = r + 2;}
        while(q < 2 * n)        //如果表 a 元素已处理完,表 b 中还有非零系数项要处理
          {c.datas[r] = b.datas[q];
           c.datas[r+1] = b.datas[q+1];
           q = q + 2;
           r = r + 2;}
        c.datas[1] = r / 2 - 1   //计算表 c 中非零系数项的个数
}
```

习　题

2-1　一个顺序表元素值有序递增,编写算法,删除顺序表中值相同的多余元素。

2-2　一个顺序表元素值无序,编写算法,删除顺序表中值相同的多余元素。

2-3　编写程序,将顺序表 A 中的元素逆置,要求算法所用的辅助空间尽可能地少。

2-4　编写程序,输出已知顺序表 A 中元素的最大值和次最大值。

2-5　设有两个按元素值递增有序的顺序表 A 和表 B,编写程序将表 A 和表 B 归并成一个新的递增有序的顺序表 C(值相同的元素均保留在表 C 中)。此题的核心算法即是本章的例 2-3。

2-6　已知两个顺序表 A 和表 B,同一表中无重复元素,编写程序实现表 A 和表 B 的并集运算,并集结果放在表 A 中。

实 训 题

2-7　已知一顺序表 A,设计一个算法删除顺序表中值为 item 的数据元素。

2-8　已知一顺序表 A,表中都是不相等的整数。设计一个算法,把表中所有的奇数移到