第 1 章

绪　　论

计算机科学是一门研究数据表示和数据处理的科学。数据是计算机化的信息，是计算机可以直接处理的最基本和最重要的对象。无论是进行科学计算、数据处理、过程控制，还是对文件的存储和检索及数据库技术的应用，都是对数据进行加工处理的过程。因此，要设计出一个结构好、效率高的程序，必须研究数据的特性及数据间的相互关系及其对应的存储表示，并利用这些特性和关系设计出相应的算法和程序。

本章要点

(1) 数据、数据元素、数据结构、数据的逻辑结构与物理结构的概念以及逻辑结构与物理结构间的关系。

(2) 数据类型的概念。

(3) 算法的定义、特性以及算法的时间复杂度和空间复杂度。

(4) 用 C 语言描述算法的方法，使用 C 语言编写算法程序。

1.1　数据结构概念

数据结构是计算机科学与技术专业的专业基础课，是十分重要的核心课程。所有的计算机系统软件和应用软件都要用到数据结构。因此，要想更好地运用计算机来解决实际问题，仅掌握几种计算机程序设计语言是难以应对众多复杂的课题的。要有效地使用计算机，充分发挥计算机的性能，还必须学习和掌握好数据结构的有关知识。具备数据结构的扎实基础，对于学习计算机专业的其他课程，如操作系统、编译原理、数据库管理系统、软件工程、人工智能等都是十分有益的。

1.1.1　为什么要学习数据结构

在计算机发展的初期，人们使用计算机的目的主要是处理数值计算问题。使用计算

机解决一个具体问题时,一般需要经过下列步骤。首先要从具体问题中抽象出一个适当的数学模型,然后设计或选择一个求解此数学模型的算法,最后编出程序进行调试、测试,直至得到最终的解答,如图 1-1 所示。例如,求解梁架结构中应力的数学模型的线性方程组,该方程组可以使用迭代算法来求解。

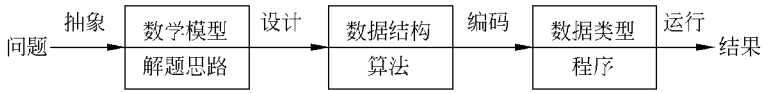


图 1-1 计算机解决问题的一般过程

由于当时所涉及的运算对象是简单的整型、实型或布尔型数据,所以程序设计者的主要精力集中于程序设计的技巧上,而无需重视数据结构。随着计算机应用领域的扩大和软、硬件的发展,非数值计算问题显得越来越重要。据统计,当今处理非数值计算性问题占用了 90%以上的机器时间。这类问题涉及的数据结构更为复杂,数据元素之间的相互关系一般无法用数学方程式描述。因此,解决这类问题的关键不再是数学分析和计算方法,而是要设计出合适的数据结构。下面所列举的实例就是属于这一类的具体问题。

例 1.1 学生信息检索系统。当我们需要查找某个学生的有关情况,或者想查询某个专业或年级的学生的有关情况时,只要建立相关的数据结构,按照某种算法编写相关程序,就可以实现计算机自动检索。由此,可以在学生信息检索系统中建立一个按学号顺序排列的学生信息表,以及分别按姓名、专业、年级顺序排列的索引表,如图 1-2 所示。由这 4 个表构成的文件便是学生信息检索的数学模型,计算机的主要操作便是按照某个特定要求(如给定姓名)对学生信息文件进行查询。

诸如此类的还有电话自动查号系统、考试查分系统、仓库库存管理系统等。在这类文档管理的数学模型中,计算机处理的对象之间通常存在着一种简单的线性关系,因此,这类数学模型称为线性数据结构。

例 1.2 教学计划编排问题。一个教学计划包含许多课程,在这些课程中,有些课程必须按规定的先后次序进行学习,有些则没有次序要求。即有些课程之间有先修和后续的关系,有些课程可以任意安排次序。这种各个课程之间的次序关系可用称之为图的数据结构来表示,如图 1-3 所示。有向图中的每一个顶点表示一门课程,如果从顶点 v_i 到 v_j 之间存在有向边 $\langle v_i, v_j \rangle$,则表示课程 i 必须先于课程 j 进行。

由以上两个例子可见,描述这类非数值计算问题的数学模型不再是数学方程,而是诸如表、树、图之类的数据结构。因此,可以说数据结构课程主要是研究非数值计算的程序设计问题中所出现的计算机操作对象,以及它们之间的关系和操作的学科。

学号	姓名	性别	专 业	年 级
980001	吴承志	男	计算机科学与技术	1998 级
980002	李淑芳	女	信息与计算科学	1998 级
990301	刘　丽	女	数学与应用数学	1999 级
990302	张会友	男	信息与计算科学	1999 级
990303	石宝国	男	计算机科学与技术	1999 级
000801	何文颖	女	计算机科学与技术	2000 级
000802	赵胜利	男	数学与应用数学	2000 级
000803	崔文靖	男	信息与计算科学	2000 级
010601	刘　丽	女	计算机科学与技术	2001 级
010602	魏永鸣	男	数学与应用数学	2001 级

(a) 学生信息表

崔文靖	8
何文颖	6
李淑芳	2
刘　丽	3,9
石宝国	5
魏永鸣	10
吴承志	1
赵胜利	7
张会有	4

(b) 姓名索引表

计算机科学与技术	1,5,6,9
信息与计算科学	2,4,8
数学与应用数学	3,7,10

(c) 专业索引表

2000 级	6,7,8
2001 级	9,10
1998 级	1,2,3
1999 级	4,5

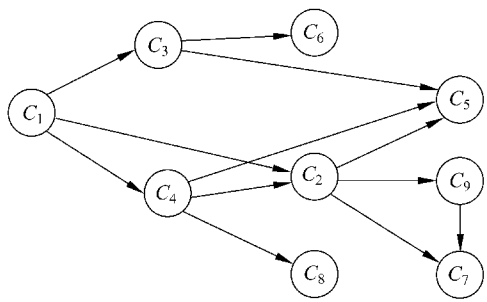
(d) 年级索引表

图 1-2　学生信息查询系统中的数据结构

学习数据结构的目的是了解计算机处理对象的特性，将实际问题中所涉及的处理对象在计算机中表示出来，并对它们进行处理。与此同时，通过算法训练提高学生的思维能力，通过程序设计的技能训练促进学生的综合应用能力和专业素质的提高。

课程编号	课程名称	先修课程
C_1	计算机导论	无
C_2	数据结构	C_1,C_4
C_3	汇编语言	C_1
C_4	C程序设计语言	C_1
C_5	计算机图形学	C_2,C_3,C_4
C_6	接口技术	C_3
C_7	数据库原理	C_2,C_9
C_8	编译原理	C_4
C_9	操作系统	C_2

(a) 计算机专业的课程设置



(b) 课程之间优先关系的有向图

图 1-3 教学计划编排问题的数据结构

1.1.2 有关概念和术语

在系统地学习数据结构知识之前,先对一些基本概念和术语赋予确切的定义。

数据(data)是信息的载体,它能够被计算机识别、存储和加工处理。它是计算机程序加工的原料,应用程序可以处理各种各样的数据。它可以是数值数据,也可以是非数值数据。数值数据包括整数、实数或复数,主要用于工程计算、科学计算和商务处理等;非数值数据包括字符、文字、图形、图像、语音等。

数据元素(data element)是数据的基本单位,在计算机程序中通常作为一个整体进行考虑和处理。一个数据元素可由若干个数据项组成。在不同的条件下,数据元素又可称为元素、结点、顶点、记录等。例如,学生信息检索系统中学生信息表中的一个记录,教学

计划编排问题中的一个顶点等，都被称为一个数据元素。

数据项(data item)是不可分割的、含有独立意义的最小数据单位，有时也称为字段(field)或域。例如，学籍管理系统中学生信息表的每一个数据元素就是一个学生记录，它包括学生的学号、姓名、性别、籍贯、出生年月、成绩等数据项。这些数据项可以分为两种，一种是初等项，如学生的性别、籍贯等，这些数据项是在数据处理时不能再分割的最小单位；另一种是组合项，如学生的成绩，它可以再划分为数学、物理、化学等更小的项。通常，在解决实际应用问题时把每个学生记录当做一个基本单位进行访问和处理。

数据结构(data structure)是指相互之间存在一种或多种关系的数据元素的集合。在任何问题中，数据元素之间都不是孤立的，都会存在着这样或那样的关系，这种数据元素之间的关系称为结构。根据数据元素之间关系的不同特性，通常有以下 4 类基本的结构(数据的逻辑结构)。

(1) 集合结构

在集合结构中，数据元素间的关系是“属于同一个集合”。数据元素之间除了同属一个集合外，不存在其他关系。集合是元素关系中极为松散的一种结构。

(2) 线性结构

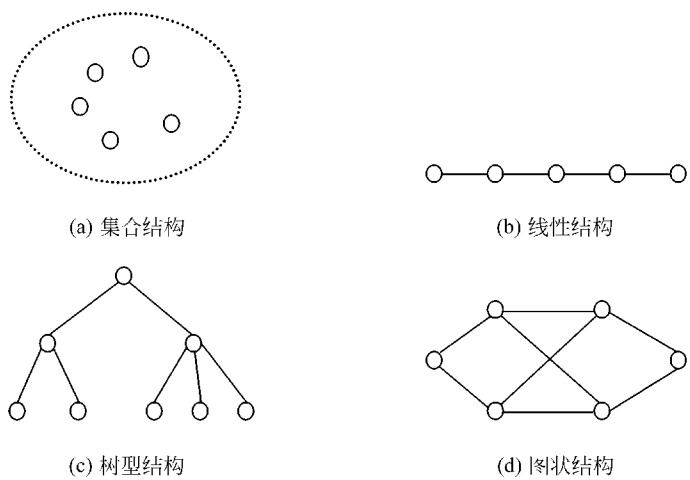
线性结构的数据元素之间存在一对一的关系。

(3) 树型结构

树型结构的数据元素之间存在一对多的关系。

(4) 图状结构

图状结构的数据元素之间存在多对多的关系。图状结构也称为网状结构。图 1-4 为上述 4 类基本结构的示意图。



(a) 集合结构　(b) 线性结构

(c) 树型结构　(d) 图状结构

图 1-4　4 类基本结构示意图

由于集合是数据元素之间关系极为松散的一种结构,因此也可用其他结构来表示它。

由数据结构概念中可以知道,一个数据结构有两个要素,一个是数据元素的集合,另一个是关系的集合。在形式上,数据结构通常可以采用一个二元组来表示。

数据结构的形式定义为:数据结构是一个二元组

$$\text{Data\_Structure} = (D,R)$$

其中,D 是数据元素的有限集,R 是 D 上关系的有限集。

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构可以看做是从具体问题抽象出来的数学模型,它与数据的存储无关。研究数据结构的目的是在计算机中实现对它的操作,为此还需要研究如何在计算机中表示一个数据结构。数据结构在计算机中的表示(又称为映像)称为数据的物理结构,或存储结构。它所研究的是数据结构在计算机中的实现方法,包括数据结构中数据元素的表示和数据元素间关系的表示。

数据的存储结构可以采用顺序存储或链式存储方法。

顺序存储方法是把逻辑上相邻的元素存储在物理位置相邻的存储单元中,由此得到的存储表示称为顺序存储结构。顺序存储结构是一种最基本的存储表示方法,通常借助于程序设计语言中的数组来实现。

链式存储方法对逻辑上相邻的元素不要求其物理位置相邻,元素间的逻辑关系通过附设的指针字段来表示,由此得到的存储表示称为链式存储结构,链式存储结构通常借助于程序设计语言中的指针类型来实现。

除了通常采用的顺序存储方法和链式存储方法外,有时为了查找方便,还采用索引存储方法和散列存储方法。

数据类型(data type)是一个与数据结构密切相关的概念。在高级程序设计语言中,用于限制变量取值范围和可能进行的操作的总和称为数据类型。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,它可以由多个结构类型组成,也可以分解。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相应的数据类型,进一步在确定的数据类型上实现各种操作。

算法(algorithm)是解决特定问题的一种方法或步骤的描述。

数据结构作为一门独立的课程起始于1968年,但在此之前与其有关的内容已散见于编译原理及操作系统中。20世纪60年代中期,美国的一些大学开始设立相关课程,但当时的课程名称并不叫数据结构。1968年,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1卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。从20世纪60年代末到70年代初,出现了大型程

序，软件也相对独立，结构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，再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”。

1.2　算法描述

算法与数据结构的关系紧密，在算法设计时，先要确定相应的数据结构，而在讨论某一种数据结构时也必然会涉及相应的算法。下面从算法特性、算法描述、算法性能分析与度量等三个方面对算法进行介绍。

1.2.1　算法特性

算法是对特定问题求解步骤的一种描述，是指令的有限序列。其中，每一条指令表示一个或多个操作。一个算法应该具有下列特性。

(1) 有穷性。一个算法必须在有穷步之后结束，即必须在有限时间内完成。

(2) 确定性。算法的每一步必须有确切的定义，无二义性，且在任何条件下算法只有惟一一条执行路径，即对于相同的输入只能得出相同的输出。

(3) 可行性。算法中的每一步都可以通过已经实现的基本运算的有限次执行得以实现。

(4) 输入。一个算法具有零个或多个输入，这些输入取自特定的数据对象集合。

(5) 输出。一个算法具有一个或多个输出，这些输出与输入之间存在某种特定的关系。

算法的含义与程序十分相似，但又有区别。一个程序不一定满足有穷性。例如，对于操作系统，只要整个系统不遭到破坏，就永远不会停止工作，即使没有作业需要处理，它仍处于动态等待中。因此，操作系统不是一个算法。另一方面，程序中的指令必须是计算机可执行的，而算法中的指令则无此限制。算法代表了对问题的解，而程序则是算法在计算机上特定的实现。一个算法若用程序设计语言来描述，它就是一个程序。

算法与数据结构是相辅相成的。解决某一特定类型问题的算法可以选定不同的数据结构，而选择恰当与否将直接影响算法的效率。反之，一种数据结构的优劣由各种算法的执行来体现。

要设计一个好的算法通常应考虑以下要求。

(1) 正确。算法的执行结果应当满足预先规定的功能和性能要求。

(2) 可读。一个算法应当思路清晰、层次分明、简单明了、易读易懂。

(3) 健壮。当输入不合法数据时，应能做适当处理，不至于引起严重后果。

(4) 高效。能够有效地使用存储空间和具有较高的时间效率。

一般来说,数据结构的基本操作主要有以下几种。

(1) 查找。寻找满足特定条件的数据元素所在的位置。

(2) 读取。读出指定位置上数据元素的内容。

(3) 插入。在指定位置上添加新的数据元素。

(4) 删除。删除指定位置上对应的数据元素。

(5) 更新。修改某个数据元素的值。

1.2.2 算法描述

算法的描述方法很多,根据描述算法语言的不同,可将算法描述分为以下4种。

(1) 框图算法描述

框图算法描述(程序流程图、N-S图等)在算法研究的早期曾经流行过。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。

(2) 非形式算法描述

非形式算法描述用人类自然语言(如中文,英文等),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。

(3) 伪语言算法描述

如类C语言算法描述。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。

(4) 高级语言编写的程序或函数

这是可以在计算机上运行并获得结果的算法,使给定的问题能够在有限的时间内求解。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,考察不同算法的描述方法。

(1) 该问题的框图描述(程序流程图)如图1-5所示。

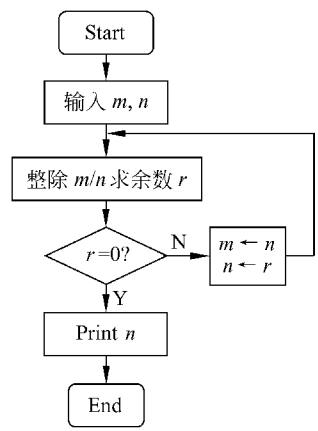


图1-5 程序流程图

(2) 非形式算法描述如下:

① 求余数。以 n 除 m,并令 r 为余数$(0\leqslant r<n)$。

② 余数是零否。若 $r=0$ 则结束算法,n 就是最大公因子。

③ 替换并返回步骤①。若 $r\neq 0$ 则 $m\leftarrow n$,$n\leftarrow r$ 返回步骤①。

(3) C 语言函数描述如下：

```
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;
}
```

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(或类 C 语言算法描述)提供给读者阅读或上机运行，以便更好地理解算法。

1.3　算法分析

下面介绍算法性能的分析和度量。可以从算法的时间复杂度和空间复杂度来评价算法的优劣。

将一个算法转换成程序并在计算机上执行时，其运行所需要的时间取决于下列因素。

(1) 硬件的速度。例如，使用 486 机还是使用 P4 机，算法的执行时间是不同的。

(2) 实现算法的程序设计语言。实现语言的级别越高，其执行效率就越低。

(3) 编译程序所生成目标代码的质量。对于代码优化较好的编译程序，其所生成的程序质量较高。

(4) 问题的规模。例如，求 100 以内的素数与求 1000 以内的素数的执行时间自然是不相同的。

显然，在各种因素都不能确定的情况下，很难比较出算法的执行时间。也就是说，使用执行算法的绝对时间来衡量算法的效率是不合适的。为此，可以将上述各种与计算机相关的软、硬件因素都确定下来，这样一个特定算法的运行工作量就只依赖于问题的规模(通常用正整数 n 表示)，或者说它是问题规模的函数。

1. 时间复杂度

程序的时间复杂度(time complexity)是指程序运行从开始到结束所需要的时间。

一个算法是由控制结构和原操作构成的，其执行时间取决于两者的综合效果。为了便于比较同一问题的不同算法，通常的做法是：从算法中选取一种对于所研究的问题来说是基本运算的原操作，以该原操作重复执行的次数作为算法的时间度量。一般情况下，算法中原操作重复执行的次数是规模 n 的某个函数 $T(n)$。

多数情况下，要精确地计算 $T(n)$是困难的。引入渐进时间复杂度，可以在数量上估

计一个算法的执行时间,也能够达到分析算法的目的。

定义 如果存在两个正常数 c 和 n_0,使得对所有的 $n(n \geqslant n_0)$,有

$$T(n) \leqslant c * f(n)$$

则有

$$T(n) = O(f(n))$$

例如,一个程序的实际执行时间为 $T(n)=2.7n^3+3.8n^2+5.3$,则 $T(n)=O(n^3)$。

使用大 O 记号表示的算法时间复杂度,称为算法的渐进时间复杂度。通常,用 $O(1)$ 表示常数计算时间。常见的渐进时间复杂度为

$$O(1) < O(\ln n) < O(n) < O(n\ln n) < O(n^2) < O(n^3) < O(2^n)$$

例 1.3 两个 $n \times n$ 阶的矩阵相乘的程序中的主要语句及其重复次数如下。

```
                                                    原操作语句的执行频度
for  (i=0;i<n;i++)
    for  (j=0;j<n;j++)
      {  s[i][j]=0;                                        n^2
         for  (k=0;k<n;k++)
              s[i][j]=s[i][j]+a[i][k] * b[k][j];           n^3
      }
```

该段程序的时间复杂度 $T(n)=cn^3+n^2=O(n^3)$(其中 c 为常量,表示算术运算时间是简单赋值运算时间的常数倍)。

2. 空间复杂度

程序的空间复杂度(space complexity)是指程序运行从开始到结束所需要的存储量,记作

$$S(n) = O(f(n))$$

其中,n 为问题的规模(或大小)。一个上机执行的程序,除了需要存储空间来寄存本身所用的指令、常数、变量和输入数据外,还需要对数据进行操作的工作单元和存储为实现计算所需信息的辅助空间。如果输入数据所占空间只取决于问题本身,与算法无关,则只需要分析除输入和程序之外的额外空间,否则应同时考虑输入本身所需空间(与输入数据的表示形式有关)。如果额外空间相对于输入数据量来说是常数,则称此算法为原地工作。如果所占空间量依赖于特定的输入,则除特别指明外,均按最坏的情况来分析。

小结

(1) 要求理解的概念有数据、数据元素、数据结构和数据类型。数据结构概念应从数据结构的逻辑结构、物理结构和相关操作三个方面进行讨论。它反映了数据结构设计的

不同层次，即逻辑结构属于问题解决范畴，而物理结构是逻辑结构在计算机中的存储方式。

(2) 对于算法的概念和简单的算法性能分析方面，应用算法要求明确算法的时间和空间复杂度。因此，必须理解算法的定义和算法的 5 个特性，掌握简单的时间复杂度估计和空间复杂度估计方法。

(3) 必须清楚了解算法的定义、特性，以及对算法编制的质量要求。此外，要求掌握算法性能(时间、空间)的简单分析，特别是程序步数的估计和大 O 表示法，这对于算法的评价和选择是至关重要的。通过本章的学习，要求基本掌握 C 语言的基本概念和用 C 语言编写应用程序的基本技术。

习题

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

2. 举出一个数据结构的例子，叙述其逻辑结构、存储结构和结构上的操作内容。

3. 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++;
}
```

4. 试画出与下列程序段等价的程序流程图。

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

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

5. 试写一算法,自大至小依次输出顺序读入的3个整数 X,Y 和 Z 的值。

6. 编写算法,求一元多项式 $P_n(x)=a_0+a_1x+a_2x_2+a_3x_3+\cdots+a_nx^n$ 的值 $P_n(x_0)$,要求算法的时间复杂度尽可能小。

第 2 章

线 性 表

线性表是最简单、最基本，也是最常用的一种线性结构。它有两种存储方法：顺序存储和链式存储，它的主要基本操作是插入、删除和检索等。

本章要点

(1) 线性表的逻辑结构特性和线性表的两种存储实现方式。

(2) 线性表的概念、特点，以及顺序表的定义与实现，包括搜索、插入、删除算法的实现及其平均比较次数的计算。

(3) 链表有单链表、循环单链表和双向链表之分。

(4) 单链表的结构和特点，掌握单链表的定义、单链表的插入与删除算法，理解带首结点的单链表的优点。

(5) 循环链表的特点，循环链表的定义，以及用循环链表解决问题的方法。

(6) 双向链表的特点，双向链表的定义及相关操作的实现，用双向链表解决问题的方法。

2.1 线性表的逻辑结构

2.1.1 线性表的定义

线性表是一种线性结构。线性结构的特点是数据元素之间一对一的线性关系，数据元素“一个接一个地排列”。在同一个线性表中，数据元素的类型是相同的。或者说，线性表是由同一类型的数据元素构成的线性结构。在实际问题中线性表的例子是很多的，如学生情况信息表是一个线性表，表中数据元素的类型为学生类型；一个字符串也是一个线性表，表中数据元素的类型为字符型，等等。

综上所述，线性表定义如下。

线性表是具有相同数据类型的 $n(n\geqslant 0)$个数据元素的有限序列,通常记为

$$(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$$

其中,n 为表长。当 $n=0$ 时称为空表。

表中相邻元素之间存在顺序关系。将 a_{i-1}称为 a_i的直接前趋,a_{i+1}称为 a_i的直接后继。就是说对于 a_i,当 $i=2,3,\cdots,n$ 时,有且仅有一个直接前趋 a_{i-1};当 $i=1,2,\cdots,n-1$ 时,有且仅有一个直接后继 a_{i+1};a_1是表中第一个元素,它没有前趋,a_n是最后一个元素,无后继。

需要说明的是,a_i是序号为 i 的数据元素($i=1,2,\cdots,n$),本书将它的数据类型抽象为 datatype,而在实际应用中,datatype 可根据具体应用问题代之不同的数据类型。例如在学生情况信息表中,它是用户自定义的学生类型;而在字符串中,它是字符型,等等。

2.1.2 线性表的基本操作

在第1章中提到,数据结构的运算是定义在逻辑结构层次上的,而运算的具体实现是建立在存储结构上的。因此,下面定义的线性表的基本运算作为逻辑结构的一部分,而每一个操作的具体实现只有在确定了线性表的存储结构之后才能完成。

线性表上的基本操作如下。

(1) 线性表初始化

格式 Init_List(L)

初始条件 线性表 L 不存在。

操作结果 构造一个空的线性表 L。

(2) 求线性表的长度

格式 Length_List(L)

初始条件 线性表 L 存在。

操作结果 返回线性表中的所含元素的个数。

(3) 取表元

格式 Get_List(L,i)

初始条件 线性表 L 存在且 $1\leqslant i\leqslant$Length_List(L)。

操作结果 返回线性表 L 中的第 i 个元素的值或地址。

(4) 按值查找

格式 Locate_List(L,x)(x 是给定的数据元素)

初始条件 线性表 L 存在。

操作结果 在表 L 中查找值为 x 的数据元素。若其结果返回在 L 中首次出现的值为 x 的数据元素的序号或地址,称为查找成功;否则,在 L 中未找到值为 x 的数据元素,返回一特殊值表示查找失败。

(5) 插入操作

格式　Insert_List(L,i,x)

初始条件　线性表 L 存在，插入位置正确($1 \leqslant i \leqslant n+1$，$n$ 为插入前的表长)。

操作结果　在线性表 L 的第 i 个位置上插入一个值为 x 的新元素。即原序号为 i，$i+1,\cdots,n$ 的数据元素的序号插入后变为 $i+1,i+2,\cdots,n+1$，插入后表长＝原表长＋1。

(6) 删除操作

格式　Delete_List(L,i)

初始条件　线性表 L 存在，$1 \leqslant i \leqslant n$。

操作结果　在线性表 L 中删除序号为 i 的数据元素。删除后，原序号为 $i+1,i+2,\cdots,n$ 的元素的序号变为 i，$i+1,\cdots,n-1$，新表长＝原表长－1。

下面两点需要说明：

(1) 某数据结构上的基本运算不是它的全部运算，而是一些常用的基本运算，每一个基本运算在实现时也可能根据不同的存储结构派生出一系列相关的运算。例如，线性表的查找在链式存储结构中还会有按序号查找；插入运算也可能是将新元素 x 插入到适当位置上等，不可能也没有必要全部定义出它的运算集。待掌握了某一数据结构上的基本运算后，其他运算可以通过基本运算来实现，也可以直接去实现。

(2) 在上面各操作中定义的线性表 L 仅仅是一个抽象在逻辑结构层次的线性表，尚未涉及其存储结构，因此每个操作在逻辑结构层次上尚不能用具体的某种程序设计语言写出具体的算法，而算法只有在存储结构确立之后才能用具体程序设计语言实现之。

2.2　线性表的顺序存储及操作实现

2.2.1　顺序表

线性表的顺序存储是指在内存中用地址连续的一块存储空间顺序存放线性表中的各数据元素，用这种存储形式存储的线性表称之为顺序表。因为内存中的地址空间是线性的，因此用相邻的物理地址实现数据元素之间的逻辑相邻关系既简单又自然。线性表的顺序存储如图 2-1 所示。设 a_1 的存储地址为 $\mathrm{Loc}(a_1)$，每个数据元素占 d 个存储地址，则第 i 个数据元素的地址为

$$\mathrm{Loc}(a_i) = \mathrm{Loc}(a_1) + (i-1)d \qquad 1 \leqslant i \leqslant n$$

这就是说，只要知道顺序表首地址和每个数据元素所占地址单元的个数，就可求出第 i 个数据元素的地址来，这也是顺序表具有按数据元素的序号随机存取的特点。

在程序设计语言中，一维数组在内存中占用的存储空间就是一组连续的存储区域，因此，用一维数组来表示顺序表的数据存储区域是非常合适的。考虑到线性表的运算有插

入、删除等运算,即表长是可变的,因此,数组的容量需设计得足够大,假设用 data[MAXSIZE]来表示,其中 MAXSIZE 是一个根据实际问题定义的足够大的整数,线性表中的数据从 data[0] 开始依次顺序存放,但当前线性表中的实际元素个数可能未达到 MAXSIZE 个,因此需用变量 last 记录当前线性表中最后一个元素在数组中的位置,即 last 起一个指针的作用,始终指向线性表中最后一个元素,因此,表空时 last=-1。这种存储思想的具体描述可以是多样的。例如可以是

```
datatype data[MAXSIZE];
int last;
```

所表示的顺序表如图 2-1 所示。其中,表长为 last+1,数据元素分别存放在 data[0]~data[last]中。这样使用简单方便,但有时不便管理。

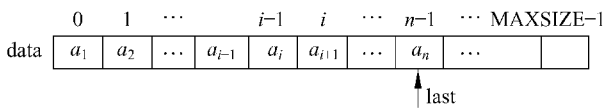


图 2-1 线性表的顺序存储示意图

从结构性上考虑,通常将 data 和 last 封装成一个结构作为顺序表的类型,即

```
typedef struct
    { datatype data[MAXSIZE];
        int last;
    } SeqList;
```

定义一个顺序表: SeqList L;

所表示的线性表如图 2-2(a) 所示。表长=L. last+1,线性表中的数据元素 a_1~a_n分别存放在 L. data[0]~L. data[L. last]中。

由于后面的算法用 C 语言描述,根据 C 语言中的规则,为方便起见可以定义一个指向 SeqList 类型的指针:

SeqList *L;

其中,L 是一个指针变量,线性表的存储空间通过 L=malloc(sizeof(SeqList))操作来获得。L 中存放的是顺序表的地址,这样表示的线性表如图 2-2(b)所示。表长表示为 (*L). last+1 或 L->last+1,线性表的存储区域为L->data ,线性表中数据元素的存储空间为

L->data[0] ~ L->data[L->last]

在后面的算法中多用上述方法表示,读者在读算法时注意相关数据结构的类型说明。

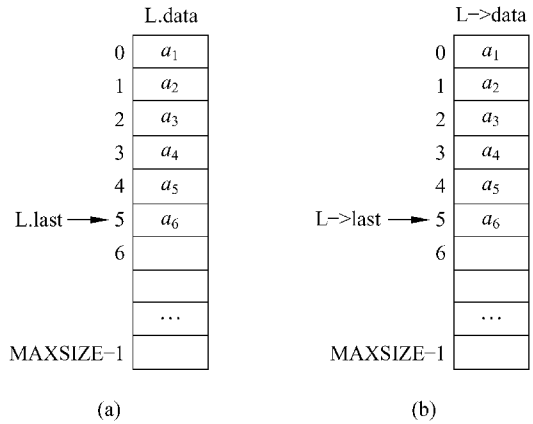


图 2-2　线性表的顺序存储示意图

2.2.2　顺序表的基本操作实现

1. 顺序表的初始化

顺序表的初始化即构造一个空表。将 L 设为指针参数，首先动态地分配存储空间，然后将表中 last 指针置为－1，表示表中没有数据元素，算法如下：

```
SeqList * init_SeqList( )
  { SeqList * L;
    L=malloc(sizeof(SeqList));
    L->last=-1; return L;
  }
```

设调用函数为主函数，主函数对初始化函数的调用如下：

```
main()
{ SeqList * L;
  L=init_SeqList();
   …
}
```

2. 插入操作

(1) 插入算法

线性表的插入是指在表的第 i 个位置上插入一个值为 x 的新元素，插入后使原表长为 n 的表(如 $a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n$)成为表长为 $n+1$ 表($a_1,a_2,\cdots,a_{i-1},x,a_i,$

a_{i+1},…,a_n)。

i 的取值范围为 $1 \leqslant i \leqslant n+1$。

在顺序表上完成这一运算所通过的步骤如下:

① 将 $a_i \sim a_n$ 顺序向下移动,为新元素让出位置;

② 将 x 置入空出的第 i 个位置;

③ 修改 last 指针(相当于修改表长),使之仍指向最后一个元素。

插入算法如下:

```
int Insert_SeqList(SeqList * L,int i,datatype x)
{ int j;
  if (L->last==MAXSIZE-1)
       { printf("表满"); return(-1); }      /* 表空间已满,不能插入 */
  if (i<1 || i>L->last+2)                  /* 检查插入位置的正确性 */
    { printf("位置错");return(0); }
  for(j=L->last;j>=i-1;j--)
       L->data[j+1]=L->data[j];            /* 结点移动 */
  L->data[i-1]=x;                          /* 新元素插入 */
  L->last++;                               /* last 仍指向最后元素 */
  return (1);                              /* 插入成功,返回 */
}
```

插入算法中应注意的问题如下:

① 顺序表中的数据区域有 MAXSIZE 个存储单元,所以在向顺序表中插入新元素时应先检查表空间是否已满。在表满的情况下不能再插入,否则会产生溢出错误。

② 应检验插入位置的有效性。这里,i 的有效范围是 $1 \leqslant i \leqslant n+1$,$n$ 为原表长。

③ 应注意数据的移动方向。

(2) 插入算法的时间复杂度分析

顺序表上的插入运算时间主要消耗在数据的移动上。在第 i 个位置上插入 x,从 a_i 到 a_n 都要向下移动一个位置,共需要移动 $n-i+1$ 个元素,而 i 的取值范围为 $1 \leqslant i \leqslant n+1$,即有 $n+1$ 个位置可以插入,如图 2-3 所示。设在第 i 个位置上插入的概率为 P_i,则平均移动数据元素的次数为

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

设 $P_i=1/(n+1)$,即为等概率情况,则有

$$E_{\text{in}} = \sum_{i=1}^{n+1} P_i(n-i+1) = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{n}{2}$$

这说明,在顺序表上做插入操作需移动表中 50% 的数据元素。显然,时间复杂度为 $O(n)$。

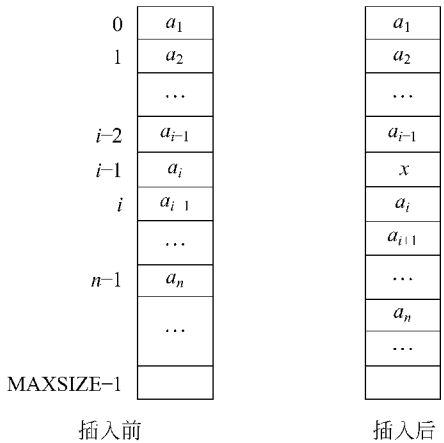


图 2-3　顺序表中的插入运算

3. 删除操作

线性表的删除运算是指将表中第 i 个元素从线性表中除去，删除后使原表长为 n 的线性表(a_1，a_2，…，a_{i-1}，a_i，a_{i+1}，…，a_n)成为表长为 $n-1$ 的线性表(a_1，a_2，…，a_{i-1}，a_{i+1}，…，a_n)。i 的取值范围为 $1 \leqslant i \leqslant n$。

在顺序表上删除运算的步骤如下：

(1) 将元素 $a_{i+1} \sim a_n$顺序向上移动。

(2) 修改 last 指针(相当于修改表长)，使之仍指向最后一个元素。

图 2-4 为顺序表中删除操作示意图。

(1) 删除算法

```
int Delete_SeqList(SeqList * L;int i)
{ int j;
  if(i<1 || i>L->last+1)                    /* 检查空表及删除位置的合法性 */
        { printf ("不存在第 i 个元素"); return(0); }
  for(j=i;j<=L->last;j++)
    L->data[j-1]=L->data[j];                /* 向上移动 */
  L->last--;
  return(1);                                /* 删除成功 */
}
```

删除算法应注意的问题如下：

① 删除第 i 个元素，i 的取值为 $1 \leqslant i \leqslant n$，否则第 i 个元素不存在。因此，应检查删除位置的有效性。

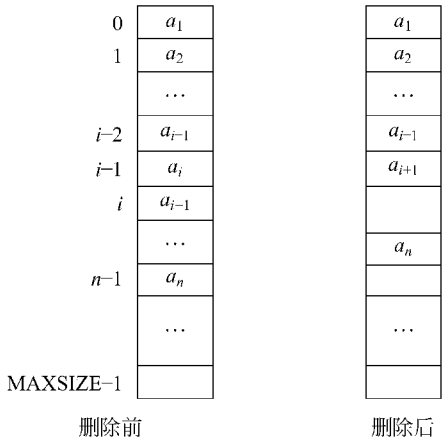


图 2-4 顺序表中的删除运算

② 表空时不能做删除运算。因为表空时,L->last 的值为-1,条件(i<1 || i>L->last+1)包括了对表空的检查。

③ 删除元素 a_i 之后,该数据已不存在,如果需要,先取出 a_i,再做删除运算。

(2) 删除算法的时间复杂度分析

与插入运算相同,其时间主要消耗在移动表中元素上。删除第 i 个元素时,其后面的元素 $a_{i+1}\sim a_n$ 都要向上移动一个位置,共移动了 $n-i$ 个元素,所以平均移动数据元素的次数为

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

在等概率情况下,$P_i=1/n$,则有

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

这说明,在顺序表上做删除运算大约需要移动表中 50% 的元素,显然该算法的时间复杂度为 $O(n)$。

4. 按值查找

线性表中的按值查找是指在线性表中查找与给定值 x 相等的数据元素。在顺序表中完成该运算最简单的方法是,从第一个元素 a_1 起依次与 x 比较,直到找到一个与 x 相等的数据元素,则返回它在顺序表中的存储下标或序号(二者相差 1);如果查遍整个表都没有找到与 x 相等的元素,则返回-1。

按值查找算法如下: