计算机大专教材系列

数据结构

陈有祺　辛运帏　编著

南开大学出版社

数据结构

陈有祺　辛运帏　编著

南开大学出版社

内容提要

全书共分八章，依次介绍了数据结构的基本概念，线性表、栈、队列和数组，树结构和图结构，以及查找和排序等基本运算，其中主要算法都用C语言给出详细描述。

本书选材精炼，叙述深入浅出，尽量用实例来说明基本概念和方法，使初学者易于掌握。每章后面都附有习题，便于读者复习和检验所学知识。本书可作为大中专院校计算机类各专业的教材，也可作为广大计算机工作者提高程序设计水平的参考书。

数据结构

陈有祺　辛运帏　编著

南开大学出版社出版
（天津八里台南开大学校内）
邮编 300071　电话 23508542
新华书店天津发行所发行
南开大学印刷厂印刷

1996年5月第1版　1997年10月第2次印刷
开本：787×1092　1/16　印张：11
字数：267千　印数：8001－16000
ISBN 7-310-00913-4
TP·56 定价：12.00元

“计算机大专教材系列”编委会

出版说明

随着计算机应用的日益深入、普及，目前在我国正在兴起学习计算机专业知识的高潮，各种有关计算机的书籍如雨后春笋般涌现出来，使广大读者大有应接不暇之势。但是，已经出版的这些书籍中，有的偏深偏专，取材偏多偏全，适合有一定基础的计算机专业人员阅读参考；有的则是普及性读物，只适合急于入门的计算机爱好者使用；在为数不多的教材中，大都是为计算机专业本科生使用而编写的，不适合成人教育和大专类学生的需要。鉴于这种形势，我们决定编写一套适合于计算机类各专业大专学生和成人教育使用的教材。这套教材共有十种，虽然它还不能完全覆盖上述办学层次教学计划中的所有课程，但是它包括了培养一个计算机类专科生的主要教学内容。其中入门的教材有《计算机应用基础》和《C 语言程序设计》；属于专业基础的教材有《16 位微型计算机原理与接口》，《汇编语言程序设计》，《数据结构》和《操作系统》；应用性较强的《单片机及其应用》，《数据库系统教程》，《计算机网络基础》和《软件工程引论》。

这套教材贯彻了理论联系实际、学以致用的原则。在取材方面，不追求包罗万象、面面俱到，而着力保证把最基本、最实用的部分包含进来。在叙述方面，力求做到深入浅出，尽量用实例来说明基本概念和基本方法。我们希望这套教材不仅能适合课堂讲授的需要，也便于广大读者自学。这套教材由南开大学计算机与系统科学系的教师们编写而成，他们之中既有教学经验十分丰富的教授、副教授，也有活跃在计算机应用最前沿的青年教师。这些教师不仅具有教本科生、研究生的教学经验，也具有教大专生和成人教育的教学经验，这就使这套教材的质量有了基本的保证。但是由于我们初次编写这类教材，尚未经过实践的检验，缺点和不足之处在所难免，敬希同行专家和广大使用者批评指正。

前　言

随着计算机科学技术的飞速发展，计算机应用日益普及和深入。《数据结构》作为计算机大专系列教材之一，在计算机教育中起着“承前启后”的重要作用。初学者在初步掌握计算机基本知识并学会使用一种计算机语言之后，虽然也能编制一些应用程序，但在学习了数据结构之后，对于实际问题可以有意识地选取合适的数据结构，并在明确的算法思想指导下进行编程，从而使编出的程序结构更合理，运行效率更高。另一方面，它又是学习一些更高级课程的基础，如《操作系统》、《数据库系统》、《软件工程》等课程都需要数据结构的有关知识。

本书共分 8 章。第 1 章主要介绍数据结构的基本概念。第 2 章引入最基本、最常用的数据结构——线性表，给出线性表的定义、存储结构和基本运算。第 3、4 两章依次介绍几种特殊的线性表——栈、队列、数组和串，分别给出它们的定义、存储结构、基本运算和应用实例。第 5 章至第 8 章为本书的重点部分。第 5 章引入树型结构的概念，它是应用十分广泛的一种非线性结构，其中重点介绍了二叉树和一些应用实例。第 6 章引入比树更复杂的非线性结构——图结构，给出了它的各种存储结构，详细讨论了各种特殊图的典型应用。第 7 章和第 8 章分别讨论了两类最普遍的运算——查找和排序，给出了各种典型的查找和排序方法，并对它们的性能作出了比较。

本书是针对在校的计算机专业大专生和社会上广大的计算机应用人员编写的，因此在取材上力求少而精，只选择最基本、最实用的内容来讲述，对于抽象的概念和复杂的公式推导都尽量避免。在叙述上力求深入浅出，尽量通过实例来阐明基本概念和算法，对于重要的算法都用 C 语言给出比较详细的描述。这些用 C 语言描述的算法虽然不能简单地搬到计算机上运行，但对于学过 C 语言的读者来说，读懂了这些程序就能深入、具体地掌握算法。本书各章最后都附有习题，其中有些是属于基本概念的，有些是属于设计算法、编制程序的。我们认为，要学好数据结构，除熟悉书本知识外，必须有一定数量的上机实习，建议读者编一些有关的程序并上机调试。

由于编者的水平所限，书中难免有错误和不妥之处，恳请广大读者特别是同行专家批评指正。

编　者

1995 年 10 月

目　录

第 1 章　绪论

第 2 章　线性表及其应用

第 3 章　栈、队列和数组

第4章 串

第5章 树型结构

第6章 图结构

第7章 查找

第8章 排序

第1章 绪 论

数据结构(Data Structures)是一门随着计算机科学的发展而逐渐形成的学科,目前已成为计算机类各专业的基础课、核心课之一。本章的目的就是对数据结构的基本概念和基本内容作一概要介绍;可是,数据结构究竟是什么,并非三言两语所能说清楚的。要想讲明这个问题,还得从数据结构的发展历史讲起。

1.1 数据结构的发展历史

大家知道,在40年代计算机刚诞生时,计算机所能处理的数据非常简单,只是由0或1组成的二进制数,可以说谈不上什么结构。后来虽然可以处理十进制整数和十进制实数,也不过是由0到9这十个数字组成的序列;或者再加上一个小数点,其结构也是非常简单的,没有研究它的必要。

随着计算机的发展,从50年代到60年代的中期,各种高级程序语言纷纷出现,所能描述的数据类型也逐渐繁多。例如在FORTRAN语言中允许出现复数类型的量,实际上是用两个有序的实数〈X,Y〉表示一个复数,其中第一个实数表示复数的实部,第二个实数表示复数的虚部,这就有点"结构"的雏型了。另外,在BASIC,FORTRAN等语言中,都允许使用数组,其中最简单的一维数组,就是同类型元素的一个有限序列。这当然是一种结构。二维以上的数组,其结构更复杂些,这是本书下面要讨论的问题。当计算机的应用领域由数值计算扩充到非数值计算之后,各种新的结构应运而生,其中最重要的是在COBOL、PL/1等语言中出现的记录类型。它是把许多不同类型的数据组合起来构成一个新的类型,例如一个职工的基本情况可以包括姓名、性别、出生年月、工龄、工资等项内容,这些项目合起来就构成一个职工的记录。另外,大量的字符处理需要有字符串(简称为串)类型的量,SNOBOL语言就是对于串运算最方便的高级程序语言。后来,LISP语言又定义一种带有层次性的表结构,并定义了许多相应的运算。这种表已经是一种相当复杂的非线性结构,实际上是一种树型结构,这是我们在本书中要着重讨论的内容。

60年代中期以后,在数据结构的发展史上发生了几个重要的事件。首先,在美国计算机界出现了信息结构这一名称(后来改称数据结构)。然后在1968年,由美国计算机协会(ACM)颁发了建议性的计算机教学计划,其中规定数据结构作为独立的一门课,这在世界上还是第一次。同年,世界著名的计算机科学家、美国的D. E. Knuth教授的巨著《计算机程序设计的技巧》第一卷《基本算法》出版,这本书全面系统地论述了数据的逻辑结构和存储结构,并且给出了典型的各种算法,为数据结构奠定了理论基础。从60年代末到70年代初,出现了大型的程序和大规模的文件系统,结构程序设计成为程序设计方法学的主要内容。人们对数据结构越来

越重视，认为程序设计的实质就是对要处理的问题选择一种好的数据结构，同时在此结构上施加一种好的算法。著名计算机科学家N. Wirth写的《算法＋数据结构＝程序》一书正是体现了这种观点。

1.2 数据结构的基本术语和概念

数据 是能输入到计算机中并能被计算机处理的一切对象。这里必须强调指出，所谓数据绝不能仅理解为整数或实数这种狭义的“数”，必须作广义的理解。例如，一个用某种程序语言编写的程序，一篇文章，一张地图以及图画、声音等等都可视为“数据”。今后随着计算机的发展，还会不断扩充数据的范围。

数据元素 是数据的基本单位。由于数据范围非常广泛，因此基本单位可大可小，小到可以是一个字符，大到可以是一个国家的地图或一本书等等。对于较大的单位，还可以由称为“数据项”的较小单位组成。如一本书的目录卡片就可以包括书名、作者名、出版社名、出版日期等数据项。

数据对象 是具有相同性质的数据元素的集合，是数据的一个子集。因为计算机不可能同时处理一切类型的数据，总是对特定的问题处理一种或几种对象。例如用计算机求素数这一问题只涉及“整数”这种数据对象。

数据结构 是彼此具有一定关系的数据元素的集合。事实上，这些关系反映了客观世界事物之间的联系。由于客观事物之间存在着各种不同的联系形式，所以反映在数据关系上也就各不相同。比如，一个班的学生名单是一个接着一个排列的，这就用叫做“线性表”的结构来表示。又如，上级单位与多个下级单位的关系，父辈与子辈的关系，全体与部分的关系等，就需要用所谓的“树结构”来表示。再如，各城市之间的交通联系或电讯联系，一个大的工程项目的施工进度等，就要用更复杂的“图结构”来表示。

从另一个角度讲，数据结构又分为*逻辑结构*和*物理结构*两个方面。逻辑结构是指数据元素之间的逻辑(抽象)关系，如上面所举的线性结构、树结构、图结构等。物理结构又称存储结构，是指在计算机上如何具体表示数据的各种逻辑结构；同一种逻辑结构可以有几种不同的物理结构来实现它。在某种意义上，研究物理结构更为重要。

在引入上述基本概念之后，我们可以对数据结构这一学科的研究范围简单概括如下：

数据结构研究数据的逻辑结构和物理结构，并在这种结构上定义相关的运算，设计并实现相应的算法，分析算法的效率。

1.3 关于算法描述和算法分析

如上一节所述，研究数据结构必须和研究相应的算法结合起来，才能解决实际应用问题。什么叫做算法呢？简单地说，算法就是对特定问题求解步骤的一种描述。由于我们对数据结构的研究是不依赖于任何计算机程序设计语言的，因此我们可以用框图形式或者甚至用自然语言描述算法。但是这样描述有时太不精确，难以反映出算法的本质特点，所以我们还是要选择一种程序语言来描述算法。鉴于目前C语言最为流行，本书中所有需要精确描述的算法

都用C语言书写。当然这种用C语言书写的程序在语法上不一定十分严格，在上机作练习时不要简单地照抄这些算法，还要根据你上机时用的C语言版本的规定来编写你的真正的程序才行。

一个问题(例如对n个数进行排序)可以有许多算法，那么该怎样判断它们的优劣呢?当然判断算法优劣的标准很多，本书不能深入讨论。这里主要想讨论的是：一个算法除了正确性必须保证以外(一个有错误的算法根本不能称为算法!)，主要一个指标是它的效率。

算法的效率包括时间与空间两个方面，分别称为时间复杂度与空间复杂度。所谓时间复杂度是指一个算法所需运算次数的多少；所谓空间复杂度是指一个算法所需的辅助内存空间的大小。这两种度量都不是表示为绝对的量，而是表示为该算法所解决的问题中数据个数的一个函数关系。例如，我们设计一个对n个数排序的算法，这个算法对于n=10和n=10000时所用的时间显然是不同的，因此只能用一个n的函数f(n)来刻画算法的时间复杂度，f(n)的不同形式就区别了算法时间复杂度的高低。在空间复杂度方面也是类似的意思。

为了说明如何对一个算法的复杂度进行分析，我们举一个大家熟悉的例子。

例1.1 求两个n阶矩阵的乘积。用C语言写出算法如下：

```
for(i=0;i<n;i++)                                    (1)
  for(j=0;j<n;j++)                                  (2)
    {                                               (3)
      C[i][j]=0;                                    (4)
      for(k=0;k<n;k++)                              (5)
        C[i][j]=C[i][j]+a[i][k]*b[k][j];            (6)
    }                                               (7)
```

让我们对上述算法的时间复杂度作一分析。根据C语言中for语句的语义，算法第(1)行需执行n遍(每遍中具体的操作次数可从略)，而第(2)行的for是包含在整个for语句之中的，因此它需要执行n^2遍；同样，第(4)行的赋值语句也执行n^2遍；可是第(5)行的for需执行n^3遍；第(6)行的赋值语句也执行n^3遍。将各行的执行遍数加起来即为整个算法的时间复杂度

$$T(n)=2n^3+2n^2+n$$

它是一个n的三阶多项式，忽略低次项后，可写为

$$T(n)=O(n^3)$$

这里的$O(n^3)$表示它是与n^3同阶的一个量，也可以认为它是n^3的某个常数倍。以后我们描述复杂度时都用这种记法，如$O(1)$，$O(n)$，$O(n^2)$分别表示常量阶、线性阶和平方阶。其它可能出现的复杂度有对数阶$O(\log n)$，指数阶$O(2^n)$等。不同数量级的复杂度是很不相同的，它们的函数图像如图1-1所示。

从图中可以看出，对数阶的函数增长最慢，其次为多项式阶(在多项式中，当然是阶数越低越好)，而指数阶是增长最快的。事实上，一个指数函数2^n的增长率达到惊人的程度。例如当n=64时，这本不是一个很大的数目，可是2^{64}却达到10^{19}的数量级。如果一个算法需要2^{64}次运算的话，即使在每秒执行一亿次运算的计算机上也要几千年才能完成!这在实际上是不可能实现的，因此我们要避免使用那些计算复杂度达到指数阶的算法。

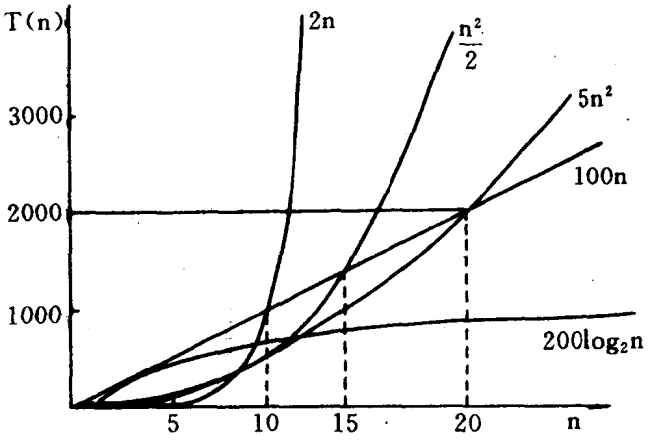

图 1-1　常见函数的增长率

习　题

1. 举例说明什么叫数据、数据元素和数据结构？

2. 数据结构研究的主要问题是什么？

3. 对于 n=5,10,50，分别计算出 n^2,n^3,logn,n·logn,2^n,3^n 的值。

4. 编写一个求一元多项式

$$P_n(x)=\sum_{i=0}^{n}a_i x^i$$

的值 $P_n(x_0)$ 的程序。该问题可以采用不同的算法，试分析不同算法的时间复杂度（写为 n 的函数）。

5. 已知 k 阶斐波那契序列的定义为：

$f_0=0$，$f_1=0$，…，$f_{k-2}=0$，$f_{k-1}=1$；

$f_n=f_{n-1}+f_{n-2}+\cdots+f_{n-k}$，　　$(n=k,k+1,\cdots)$

试编写求 k 阶斐波那契序列第 m 项的程序（函数形式，其中 k 和 m 为参数），并分析算法的时间复杂度和空间复杂度。

第2章 线性表及其应用

线性表(Linear List)是一种最基本、最常用的数据结构。本章将给出线性表的定义、基本运算以及它的几种存储结构，最后给出线性表的应用实例。

2.1 线性表的定义和基本运算

我们先举几个例子。

例 2.1 (3,8,5,7,10)是一个线性表，其中的数据元素是整数，按上述顺序排列，表中共有 5 个数据元素。

例 2.2 (A,B,C,D,…,X,Y,Z)是一个线性表，其中的数据元素是英文大写字母，按字母表的顺序排列，共有 26 个数据元素。

例 2.3 图 2-1 所表示的学生入学成绩表也是一个线性表。其中的数据元素是每个学生在表中所占的一行，包括姓名、准考证号、性别、年龄和总分等共 5 个数据项。通常把这种数据元素称为记录。表中所列出的学生人数就是该表中数据元素的个数。

姓名	准考证号	性别	年龄	总分
陈　东	941001	男	19	564
王会敏	941005	女	18	578
刘利平	941002	男	18	516
⋮	⋮	⋮	⋮	⋮
李民强	941015	男	19	485

图 2-1　学生入学成绩表

综合上述例子，我们给出线性表的定义：

一个线性表是 $n(n\geqslant 0)$个同类型数据元素 $a_1,a_2,\cdots,a_n$ 的有限序列。记作

$$(a_1,a_2,\cdots,a_n)$$

从定义可知，线性表强调两个特性。一个是强调每个数据元素 $a_i(i=1,\cdots,n)$必须是同类型的数据，例如，如果是记录，则必须是含有相同数据项的记录；如果是整数，则必须都是整数，不能将不同性质的数据并列在一个线性表内。另一个是强调数据元素的有序性，数据元素在表中的位置决定了它的序号。按照这个次序，元素 a_{i-1}称为 a_i 的直接前趋，反之，a_i 称为 a_{i-1}的直接后继$(i=2,\cdots,n)$。a_1 是表中第一个元素，因此它是没有前趋的；a_n 是表中最后一个元素，因此它是没有后继的。

表中数据元素的个数 n 称为线性表的长度。长度为 0 的表称为空表。

对于线性表，常用的运算有如下几种：

(1)存取　访问线性表的第 i 个元素，使用或改变该数据元素的值。

(2)查找　在线性表中查找满足某种条件的数据元素。

(3)插入　在线性表的第 i 个元素之前，插入一个同类型的元素。

(4)删除　删除线性表中第 i 个元素。

(5)求表长　求出线性表中元素的个数。

(6)置空表　建立一个空表。

(7)清除表　将已有线性表变为空表，即删除表中所有元素。

对线性表的其它运算还有将两个表合并，将表中元素按一定规则排序等。本章将重点讨论插入和删除两种运算。

2.2　线性表的顺序存储结构

在计算机内，可以用不同的方式来表示线性表，其中最简单和最常用的方式是用一组地址连续的存储单元依次存储线性表的元素。

假设线性表的每个元素需占用 l 个存储单元，则线性表

$$(a_1,a_2,\cdots,a_n)$$

的各个元素在内存中所占的位置如图 2-2 所示。

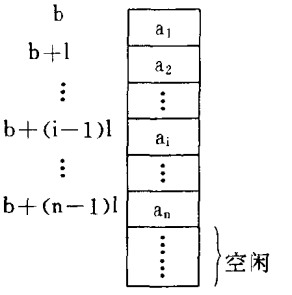

图 2-2　线性表顺序存储示意图

从图中可见，只要知道线性表的起始存储位置 b(也就是线性表第一个元素 a_1 的开始地址)，表中任一元素 a_i 的存储位置 LOC(a_i)就可用以下公式算出：

$$LOC(a_i)=LOC(a_1)+(i-1)*l$$

这种顺序存储结构，和许多程序语言中一维数组的结构极为相近；但是由于我们的线性表允许插入、删除等运算，它的长度是不断变化的，所以为适应这种情况，在 C 语言中用一个结构变量来表示。

```
Struct sqlist {
        anytype elem〔maxlen〕;
        int     last;
        }  ;
```

这里 anytype 代表线性表元素的类型；maxlen 是我们估计出的一个常数，它限定了线性表的最大长度；last 是 0 到 maxlen 之间的一个整数，表示线性表中实际元素个数(长度)。

在这种表示下，许多关于线性表的运算(如存取、求表长、置空表等)极易实现。现在主要讨论插入和删除运算。

插入。在表的第 i 个元素之前，插入同类型的元素 b。写成函数形式如下：

```
void
ins-sqlist(struct sqlist v,int i,anytype b)
{   if(i<1 ‖ i>v.last+1);
    error('Illegal value of i');
    else if(v.last==maxlen)
        error('Overflow');
```

```
        else { for(j=v.last,j>=i,j--)
                v.elem[j+1]=v.elem[j];
               v.elem[i]=b;
               v.last=v.last+1;
              }
}
```

删除。删除表中第 i 个元素。写成函数形式如下：

```
void
del-sqlist(struct sqlist v,int i)
{ if (i<1 || i>v.last)
    error('No this value of i');
  else { for(j=i+1,j<=v.last,j++)
          v.elem[j-1]=v.elem[j];
         v.last=v.last-1;
        }
}
```

从上面两种函数可见，插入和删除运算的时间主要耗费在元素的移动上，而移动次数的多少取决于 i 值。由于 i 的值是随机的，因此我们不好说 i 应当取什么值或者说 i 经常取什么值；一般来说，在多次的插入(或删除)运算中，i 取什么值都是可能的，而且是机会均等的。因此，在表长为 n 的线性表中，插入一个元素的平均移动次数为：

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

这也就是在线性表的顺序存储方式下，插入一个元素的时间复杂度。按照以上分析，删除一个元素的时间复杂度为$\frac{n-1}{2}$。粗略地表示它们的时间复杂度均为 O(n)。

2.3 线性表的链式存储结构

在线性表的顺序存储中，我们看出由于它的存储顺序与元素的逻辑顺序一致，所以确定任一元素的位置非常容易，访问任一元素也就非常方便。这是顺序存储方式的特点，也是它的主要优点。但是，这种存储方式也同时带来以下缺点：其一，在对线性表进行插入、删除操作时，需移动大量元素，这是很费时间的，特别是当每个数据元素包含庞大的数据时，这个问题更为突出；其二，我们对表的最大长度 maxlen 这个量难以事先准确估计出来。定得太大会浪费空间，定得太小则当一旦发生表满溢出时难以补救。鉴于上述问题，本节我们考虑线性表的另一种存储结构——链式存储结构。

2.3.1 线性链表

线性表的链式存储结构是用一组任意的存储单元存放数据元素，这组存储单元可以是连续的，也可以是不连续的。为了表示每个元素 a_i 和它的直接后继 a_{i+1} 之间的逻辑关系，对于数据元素 a_i 来说，除了要存储它本身的信息之外，还要存储指示它的直接后继的存储位置的信息。这两部分信息合起来对应着线性表中的一个数据元素，称之为结点。结点中存储数据元素