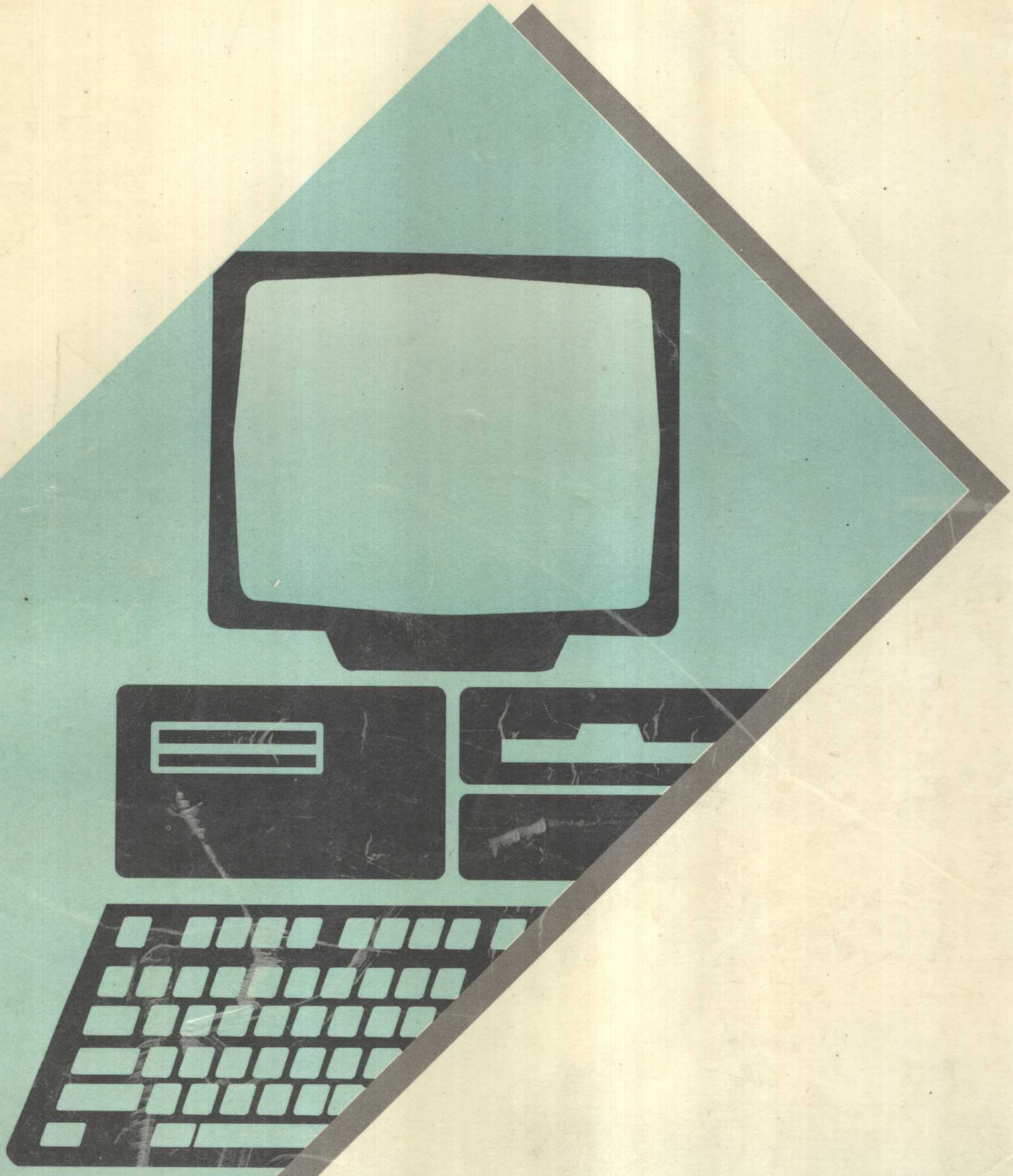
科学出版社



数据结构实用教程

杨秀金　主编

数据结构实用教程

杨秀金　主编

科学出版社

1996

（京）新登字 092 号

内 容 简 介

本书系统地介绍了各种数据结构的特点、存储结构和有关的算法。书中采用流行实用的 C 语言描述算法。主要内容包括：数据结构的基本概念、算法描述和算法分析初步；线性表、堆栈、队列、数组、树、图等结构；排序、查找和文件组织等。每章后面配有练习题，最后一章介绍了算法书写和上机实习规范并给出了若干程序实例。全书叙述清晰、深入浅出、注重应用，便于教学与实践。

本书既可作为计算机专业的教材，也可以作为非计算机专业学生的选修课或辅修课的教材，还可作为计算机应用工作者和工程技术人员的自学参考书。

数据结构实用教程

杨秀金 主编

责任编辑：彭 斌

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

上海竟成印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1996 年 3 月第 1 版 开本：787×1092 1/16

1996 年 3 月第 1 次印刷 印张：11

印数：1－4000 字数：278 千字

ISBN 7-03-005429-6/TP・609

定价：14.00 元

前　言

数据结构是计算机专业教学中一门重要的专业基础课与核心课程之一。数据结构是从事计算机科学及其应用的科技人员必须具备的重要基础知识。

为适应我国计算机应用的发展，进一步提高计算机专业数据结构课程的教学质量，我们根据多年的教学经验，在分析国内多种同类教材的基础上，博采众长，编写了这本书，奉献给读者。本书除具有教材的一般特点外，还具有以下特色：

1. 由浅入深，通俗易懂。就数据结构本身而言，它具有很强的理论性，本书从应用出发，在不影响连贯性的前提下，略去一些理论推导和证明，每个新概念的引入均以应用实例开始，对各种基本算法描述尽量详细，叙述清晰。

2. 用类 C 语言描述数据结构和算法。近年来 C 语言普及发展很快，应用广泛，引人注目。本教材采用 C 语言描述算法，但描述算法的函数并不拘泥于语法细节，突出算法思路与框架。

3. 用系统科学方法指导教学。因为数据结构课程具有一定的难度，有很强的系统性和逻辑性，所以，用科学的方法指导学生学习显得尤为重要。本书编写过程中贯穿了系统科学方法的思想，旨在指导学习全过程，以便提高教与学的质量与效果。

4. 突出学以致用。本书在基本概念、基本理论阐述方面注重科学严谨，在算法介绍方面注重理论与实践相结合。为了巩固所学理论知识，每章后面都附有一定数量的练习题，供学生书面和上机作业选用。在某些重要章节补充了一些示例和算法，例如二叉树建立的算法，供学生编写上机程序参考。数据结构课程的一个重要任务是培养学生进行复杂程序设计的能力，因此本书第十一章给出了算法书写规范和上机步骤规范，并配有上机实习示范例题，目的在于培养学生进行模块化、结构化、程序设计的能力和进行规范化程序设计的素养。

5. 有所探索。本书大部分内容符合当前教学大纲，考虑到计算机应用的发展、数据结构的发展，适当增加了一些内容。例如 B 树用于外部查找、面向对象的程序设计等。

本书可作为高校计算机专业的教材、非计算机专业学生选修课教材，也可作为计算机应用工作者和工程技术人员的参考书。

本书由杨秀金主编，曾恺平副主编，杜舜国主审。其中第一章、第六章、第九章由杨秀金编写；第二章、第三章第 1 节由李翊编写；第三章第 2 节、第七章由曾恺平编写；第四章由陈雁编写；第五章、第十章第 1、2 节，第十一章第 1 节由李斌编写；第八章，第十章第 3、4 节，第十一章第 2 节由张红梅编写。全书由杨秀金、曾恺平、张红梅统编。

本书在编写过程中，始终得到有关院校的大力支持，林美雄副教授、杜舜国教授审阅了全书，并提出很多宝贵意见，为本书的出版付出了大量心血。林美雄还为本书设计了全部版面，并用计算机绘制了全部图形。在此一并表示敬意和衷心的感谢！

由于作者水平有限，书中不妥与疏漏之处难免，敬请读者批评指正。

杨秀金

1996.1

目　录

第一章 绪 论

§1.1 数据结构的基本概念和术语

1.1.1 引言

1946 年世界上第一台计算机问世以后，计算机迅速发展，大规模集成电路、微电子、激光技术、计算机网络与通讯技术成为信息处理的有效工具和手段，使信息处理日新月异，因此数据量小、结构简单、形式统一的数值计算再也满足不了信息发展的需要。由于数据的内涵极大的丰富，只有把大量数据的内在联系分清楚，才能进行有效处理，“数据结构”就是在这种情况下诞生和发展的。

60 年代的数据结构，几乎和图论，特别是表与树的理论是同义语，而且将离散数学结构的内容融在一起。1968 年，美国的 D. E. Knuth 教授开创了数据结构的最初体系，他的名著《计算机程序设计技巧》较为系统地阐述数据的逻辑结构和存储结构及其操作。随着计算机科学的飞速发展，数据结构的基础研究也日臻成熟。到 70 年代后期，我国高校就把数据的逻辑结构、存储结构，以及对每种结构所定义的操作与算法描述组成了“数据结构”的主要内容，并且陆续开设该课程。

数据结构主要解决非数值方面的问题，解决怎样合理地组织数据及建立合适的数据结构，并对所提出的算法所花费的时间与所占用的空间作确切的分析。它是一门研究在非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作的等等学科。

目前，《数据结构》是计算机专业中的一门专业基础课，也是必修课。数据结构涉及到各方面的知识，如计算机硬件范围的存储装置和存取方法；在计算机软件范围中的文件系统数据的动态存储与管理，信息检索；数学范围的许多算法知识，还有一些综合性的知识，如编码理论、算子关系、数据类型、数据表示、数据运算、数据存取等各方面的知识。可见，“数据结构”不仅是一般程序设计的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

下面请看三个例子：

例 1.1 某单位职工档案表

表 1.1 职工档案表

编号	姓名	性别	出生日期	婚否	学历	党团
1	安 丽	女	08/21/62	.T.	大专	党员
2	马 俊	男	04/06/70	.F.	大学	团员
3	闫 明	男	12/08/42	.T.	高中	党员
…	…	…	…	…	…	…

上述职工档案表就是一个数据结构，表中的每一行称为一个结点，如果我们希望查找某一名职工的档案，则只能以编号为关键字段（因姓名可能重复），也就是说一个编号能唯一确

定一个职工。在此表中，结点和结点之间存在着的是一种最简单的线性关系，其中有且只有一个表头(第一个结点)，有且只有一个表尾(最后一个结点)；除表头外，每个结点都有唯一的前趋；除表尾外，每个结点都有唯一的后续，象这类模型就称为线性的数据结构。

例 1.2 一个学校的教师花名册可以采用如图 1.1 所示的结构，它的形态似一棵倒悬的"树"，给出了教师所属的系和专业，就很容易找到所要找的教师。"树根"是学校，而所有的"叶子"就是全校所有的教师。查找的过程就是从树根沿树叉到某个叶子的过程。树可以是某些非数值计算问题的模型，它也是一种数据结构。

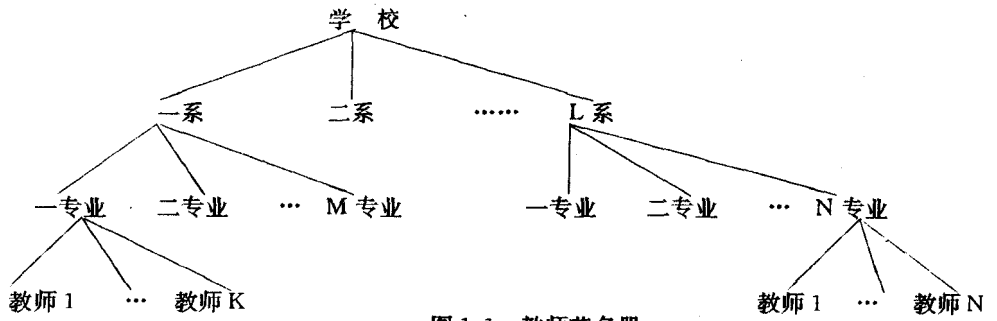


图 1.1 教师花名册

例 1.3 计算机系学生所学的某些课程之间的先后关系。左边列出的是课程名及先修课程表，右边图 1.2 是直观表示课程之间优先关系的一个有向图。

课程代号	课程名称	先修课程
C1	高等数学	
C2	程序设计	
C3	离散数学	C1,C2
C4	数据结构	C2,C3
C5	算法语言	C2
C6	编译技术	C4,C5
C7	操作系统	C4,C9
C8	普通物理	C1
C9	计算机原理	C8

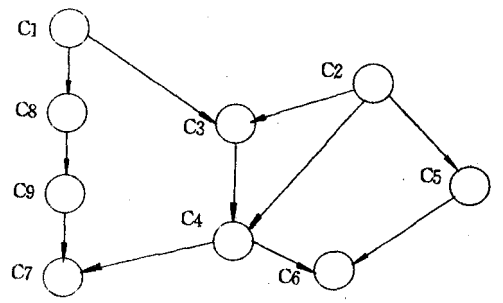


图 1.2 学生课程安排图

通常，这类问题的模型是一种称谓"图"的数据结构。其中，《数据结构》在计算机科学中是一门综合性的专业基础课，它需要《程序技术》和《离散数学》作为基础，如图 1.2 中 C2→C4 和 C3→C4。而在《编译程序》和《操作系统》中都涉及到数据元素在存储器中的分配问题，所以《数据结构》又是《编译技术》和《操作系统》的先修课程，如图 1.2 中 C4→C6 和 C4→C7。

通过以上三个例子，可以知道数据结构中元素和元素之间存在着相互的关系。而线性表、树和图是三种基本的数据结构(严格地说是数据的逻辑结构)，其他种类的数据结构多是由这三种基本结构派生的。

1.1.2 数据结构有关概念及术语

数据(data)是描述客观事物的数、字符以及所有能输入到计算机中并被计算机程序处理的符号的集合。计算机输入和输出的数据除了数字以外，还有字符串，即用英文、汉字或其它语种字母组成的词组、语句，还有表示图形和声音的符号等。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。例如，例 1.2 中的“树”中的一位教师，例 1.3 中的“图”中的一个圆圈都被称为一个数据元素。数据元素除了可以是一个数或一个字符串以外，它也可以由一个或多个数据项组成，例如，例 1.1 中每个职工的档案作为一个数据元素，在表中占一行，每个数据元素由编号、姓名、性别、出生日期、婚否、学历、党团七个数据项组成，数据项(data item)是有独立含义的数据的最小单位，数据项有时也称为域(field)。

在数据结构中，往往涉及数据类型(data type)和数据对象的概念。数据类型是指某种程序设计语言所允许使用的变量种类，如在 C 语言中，有整型等简单数据类型，也有数组等复杂数据类型及指针类型。一个数据类型不仅定义了相应变量可以设定的值的集合和存储方法，而且还规定了对变量允许进行的一组运算及其规则。所以，可以把数据类型看作是程序设计语言中已经实现了的数据结构。

数据对象(data object) 是性质相同的数据元素的集合，是数据的一个子集。例如，整数数据对象是集合 N＝{0，±1，±2，…}，字母字符数据对象是集合 C＝{'A'，'B'，…，'Z'}。

数据结构(data structure) 是带有结构的元素的集合，它是指数据之间的相互关系，即数据的组织形式。我们把数据元素间的逻辑上的联系，称之为数据的逻辑结构，如线性表、树、图等，它是数据元素间的抽象化的相互联系，逻辑结构不考虑数据在计算机中具体的存储方式，是独立于计算机的。然而，讨论数据结构的目的是为了在计算机中实现对它的操作，因此还需要研究如何在计算机中表示它。数据的逻辑结构在计算机存储设备中的映象被称为数据的存储结构，也可以说数据的存储结构是逻辑结构在计算机存储器里的实现，又称物理结构。数据的存储结构是依赖于计算机的，无论在形式上还是在顺序上都可能和数据的逻辑结构不同，例如有顺序结构、链表结构等。数据结构课程主要研究数据的逻辑结构、相应的存储结构以及定义在它们之上的一组运算。

1.1.3 运用系统科学方法学习数据结构

数据结构课程本身系统性逻辑性较强，在具体的学习过程中要深入分析，掌握好基本概念、基本方法。在每一章结束时要自觉地总结归纳。

本书在讨论某种数据结构的某种运算时，着重研究算法本身，算法的主要思路，常以函数形式写出类 C 语言的算法描述。

数据结构又是一门实践性很强的课程，学习数据结构不仅要掌握它的基本内容，同时还要从中提高自己使用计算机解决实际问题的能力，特别是要培养自己使用计算机解决实际问题的能力。充分地分析和理解问题本身，弄清楚要求做什么(而不是怎样做)，限制条件是什么，按照以数据结构为中心的原则划分模块，要综合考虑系统功能，分析每从此过程和函数，列出过程(或函数)之间的调用关系，最后综合得到一个完整的程序。

本书详细介绍了线性表、栈和队列、串、数组和广义表、树和二叉树以及图等几种基本的数据结构，以及在程序设计中经常遇到的两个问题——查找和排序。本书对每一种结构力求从数据元素之间固有的关系出发给出恰当的描述，即以逻辑结构→存储结构→算法分析为主线贯穿各章。在上机题目中既有重要的、基本的算法练习，也有难度较大(并不属于基本要求)的算法实验。可用“黑箱”方法把难易、主次不同的算法模块分别按“黑箱”、“灰箱”和“白箱”来处理。即难度大的算法模块直接引用书上提供的函数，而基本算法模块则自己设计。这

样，对一些重要算法，不论其难易程度程度如何，均可通过上机实验增强感性认识，提高程序设计技巧。

§1.2 算法描述与分析

1.2.1 什么是算法

如上所述，在确定了数据的存储结构之后，需进一步研究与之相关的一组操作（也称运算）。主要操作有：插入、删除、排序、查找等。为了实现某种操作（例如查找），常常需要设计一种算法。算法（algorithm）是对特定问题求解步骤的一种描述，它是指令或语句的有限序列。故算法需要用一种语言来描述。一个算法一般具有下列五个重要特性：

（1）有穷性：一个算法必须总是在执行有穷步之后结束。

（2）确定性：算法中的每一条指令必须有确切的含义，不能产生多义性。

（3）可行性：算法中的每一条指令必须是切实可行的，即原则上是可以通过已经实现的基本运算执行有限次来实现的。

（4）输入：一个算法有零个或多个输入，这些输入取自于特定对象的集合。

（5）输出：一个算法有一个或多个输出，这些输出是同输入有某个特定关系的量。

1.2.2 算法描述工具——类C语言

如何选择描述数据结构和算法的语言是十分重要的问题。近年来，在计算机的科学研究、教学和开发中，C语言的使用越来越广泛。因此，本教材采用类C语言进行算法描述。实际上，它是对C语言的一种简化，不但保留了C语言的精华，且与标准C语言兼容。类C语言就是为了实现这一目的而设计的，本书并不很注意语言的细节，类C语言在描述算法时有如下的约定：

1. 问题的规模尺寸用MAXSIZE表示，约定在宏定义中已经预先定义，例如：

```
#define MAXSIZE 100。
```

2. 数据元素的类型一般写成ELEMTP，可以认为在宏定义中预先定义过，例如：

```
#define ELEMTP int
```

也可以在上机实验时根据需要临时用某具体的类型标识符来代替。

3. 一个算法要以函数形式给出：

```
类型标识符  函数名(带类型说明的形参表)。
        {语句组}
```

例如：

```
int sum (int a,int b)
    {c=a+b;
     return(c);
    }
```

除了形参类型说明放在圆括号中之外，函数中其它变量的类型说明省略不写，这样使算法的处理过程更加突出明了。

4. 关于数据存储结构的类型定义以及全局变量的说明等均应在写算法之前进行说明。

下面的例子给出了书写算法的一般步骤。

例 1.4 有 n 个整型数据，将它们按由大到小的顺序排序，并且输出。

分析：n 个数据的逻辑结构是线性表：$(a_1,a_2,\cdots,a_n)$；选用一维数组作存储结构。每个数组元素两个域：一个是数据的序号域，一个是数据的值域：

```
struct ar{ int num;          /*序号域*/
          int data;          /*数据域*/
          }
```

算法描述：

```
Void simsort(struct ar a[MAXSIZE],int n)      /*n,a 数组数据由主程序提供*/
{ for (i=1;i<n;i++)
  for (j=i;j<=n;j++)
    if (a[i].data<a[j].data)
      {m:=a[i];a[i]:=a[j];a[j]:=m; }
  for (i=1;i<=n;i++)
    printf("%8d%8d%10d",i,a[i].num,a[i].data);
}
```

1.2.3 算法分析技术初步

著名的计算机科学家 N. 沃思提出了一个有名的公式：算法＋数据结构＝程序。由此可见，数据结构和算法是程序的两大要素，二者相辅相成，缺一不可。打个通俗的比方，一本菜谱介绍各种烹调方法，对每一种菜肴来说，需要说明用什么原料，然后介绍操作步骤。一种数据结构的优劣是由实现其各种运算的算法体现的。评价一个算法主要看这个算法所要占用机器资源的多少。而在这些资源中时间和空间是两个最主要的方面，因此算法分析中最关心的也就是算法所需要的时间代价和空间代价。

(1) 空间

所谓算法的空间代价(或称空间复杂度)指的是：当问题的规模以某种单位由 1 增至 n 时，解决该问题的算法实现所占用的空间也以某种单位由 1 增至 f(n)。则称该算法的空间代价是 f(n)。

(2) 时间

1. 语句频度(frequency count)：指的是该语句重复执行的次数。

2. 算法的渐近时间复杂度(asymptotic time complexity)：算法中基本操作重复执行的次数依据算法中最大语句频度来估算，它是问题规模 n 的某个函数 f(n)，算法的时间量度记作 T(n)=O(f(n))，表示随问题规模 n 的增大，算法执行时间的增长率和 f(n)的增长率相同，称作算法的渐近时间复杂度，简称时间复杂度。时间复杂度往往不是精确的执行次数，而是估算的数量级，它着重体现的是随着问题规模 n 的增大，算法执行时间的变化趋势。

例如：在下列三个程序段中

```
(a) x=x+1;
(b) for (i=1; i<=n; i++) x=x+1;
(c) for (j=1; j<=n; j++)
        for (k=1; k<=n; k++) x=x+1;
```

语句 x=x+1 的频度分别为 1，n 和 n^2，则(a)的时间复杂度可记为 O(1)；在(b)中，赋值语句于 for 循环之中，故要执行 n 次，其执行时间和 n 成正比，时间复杂度应记为 O(n)；在(c)中，赋值语句要执行 n^2次，其执行时间和 n^2成正比，则时间复杂度应记为 $O(n^2)$。

现在来分析例 1.4 的时间复杂度。算法中有一个二重循环，if 语句的执行频度为：

$$n+(n-1)+(n-2)+\cdots+3+2+1=\frac{n(n+1)}{2}$$

它的数量级为 $O(n^2)$。算法中输出语句的语句频度为 n，数量级为 O(n)。该算法的时间复杂度以 if 语句执行频度来估算(忽略输出部分)，则记为 $O(n^2)$。通常将这些时间复杂度分别称为常量阶，线性阶和平方阶，算法还可能呈现的时间复杂度有指数阶等。不同数量级时间复杂度的性状如图 1.3 所示。

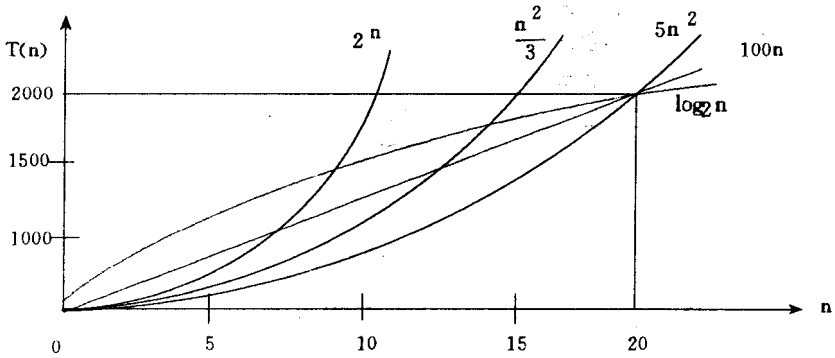


图 1.3　各种数量级的时间复杂度

从图中可见，随着问题规模的增大，其时间消耗也在增大，但它们的增长趋势明显不同。如果对于一个问题所设计的两种不同算法，算法 1 的时间复杂度为 $O(n^2)$，算法 2 的时间复杂度为 $O(\log_2 n)$。由图可知，随着问题规模 N 的增大，算法 1 所消耗时间迅速增大，而算法 2 增大趋向平缓，显然，算法 2 运行速度较快，可以认为算法 2 优于算法 1。

习　题　一

1. 简述下列术语：

   数据元素，数据，数据对象，数据结构，存储结构和算法。

2. 试写一算法，自大至小依次输出顺序读入的三个整数 x，y 和 z 的值。

3. 举出一个数据结构的例子，叙述其逻辑结构、存储结构、运算等三方面的内容。

4. 分析下面的语句组所代表的算法的时间复杂度。

```
{ for (i=1;i<=n;i++)
  for (j=1;j<=n;j++)
    for (k=1;k<=j;k++)
      { s=i+j+k; printf("%d",s); }
```

第二章　线性表

§2.1　线性表的定义及其运算

2.1.1　线性表的定义

线性表是最简单、最常用的一种数据结构。一个线性表是 n(n≥0)个数据元素的有限序列。表中，元素之间存在着线性的逻辑关系：表中有且仅有一个开始结点；有且仅有一个终端结点；除开始结点外，表中的每个结点均只有一个前趋(predecessor)；除终端结点外，表中的每个结点均只有一个后继(successor)。根据它们之间的关系可以排成一个线性序列：

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

这里，a_1为开始结点；a_n为终端结点；a_{i-1}是 a_i的前趋；a_i是 a_{i-1}的后继，n 称为该表的长度。长度 n＝0 的表称为空表。例如，英文字母表：A、B、C、D、…、X、Y、Z；一周中的七天：星期日、星期一、星期二、星期三、星期四、星期五、星期六，都是线性表。

2.1.2　各种运算简介

对线性表要经常进行的运算有以下几种：

① 取线性表中的第 i 个元素；
② 修改线性表中的第 i 个元素；
③ 删除线性表中的第 i 个元素；
④ 在线性表中第 i 个元素之后(或之前)插入一个新元素；
⑤ 按某种要求重排线性表中各元素的顺序；
⑥ 按某个特定值查找线性表中的元素。

§2.2　线性表的顺序存储结构(向量)

2.2.1　顺序存储结构(向量)

在计算机内，可以用不同的方法来存储数据信息，最常用的方法是顺序存储。顺序存储结构也称为向量存储。向量是内存中一批地址连续的存储单元。由于向量的所有元素属于同一类型，所以每个元素在存储器中占用的空间大小相同，假设向量的第一个元素存放的位置为 $LOC(k_1)$，每个元素占用的空间大小为 L，则元素 k_i的存放位置为：

$$LOC(k_i)=LOC(k_1)+L*(i-1)$$

线性表的这种机内表示称做线性表的顺序存储结构或顺序映象(sequential mapping)，只要确定了存储线性表的起始位置，线性表中任一数据元素都可随机存取，所以线性表的顺序存储结构是一种随机存取的存储结构。

在高级语言中讨论顺序存储结构，常把一维数组当做向量使用。在类 C 描述中为更好地体现信息隐蔽原则以及数据抽象原则，向量存储结构定义为：

```
struct sequnce
{ ELEMTP elem[MAXSIZE];
  int len;          /*线性表长度域*/
}
```

2.2.2 向量中基本运算的实现

（1）简单插入

向量的插入操作是指在线性表的第 i－1 个数据元素和第 i 个数据元素之间插入一个新的数据元素，就是使长度为 n 的线性表$(a_1, \cdots, a_{i-1}, a_i, \cdots, a_n)$变成长度为 n＋1 的线性表$(a_1, \cdots, a_{i-1}, b, a_i, \cdots, a_n)$，如图 2.1 所示。

图 2.1　向量的插入

插入过程可用如下算法来实现：

```
void insert(struct sequnce *p,int i,ELEMTP x)
{ v=*p;
  if (i<1)||(i>v.len+1)  printf("Overflow");
  else {
    for (j=v.len; j>=i; j    ) v.elem[j+1]=v.elem[j];
    v.elem[i]=x;v.len=v.len+1;
   }
}
```

（2）简单删除

向量的删除操作和插入操作类似，对于长度为 n 的线性表$(a_1, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$变为长度为 n－1 的线性表$(a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$，如图 2.2 所示。具体的删除算法如下：

```
void delete(struct sequnce *p,int i)
{ v=*p;
  if (i<1) || (i>v.len)  printf("Not exist");
  else {
    for (j=i; j<=v.len  1; j++)
      v.elem[j]=v.elem[j+1];
    v.len=v.len  1;
   }
}
```

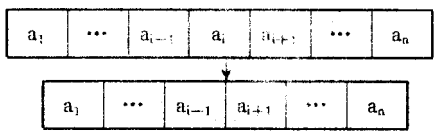


图 2.2　向量删除

（3）两种运算的时间复杂度分析

从向量插入的算法和向量删除的算法可见，当在顺序存储结构的线性表中某个位置上插入或删除一个向量时，其时间主要耗费在移动元素上，而移动元素的个数取决于插入或删除元素的位置。

假设 pi 是在第 i 个元素之前插入一个元素的概率，则在长度为 n 的线性表中插入一个元素时所需移动元素次数的平均次数为：

$$Ein = \sum_{i=1}^{n+1} p_i(n-i+1)$$

假设 q_i 是删除第 i 个元素的概率，则在长度为 n 的线性表中删除一个元素时所需移动元素次数的平均次数为：

$$Ede = \sum_{i=1}^{n} q_i(n-i)$$

如果在表的任何位置上插入或删除元素的概率相等，即

$$p_i = \frac{1}{(n+1)} \qquad q_i = \frac{1}{n}$$

则

$$Ein = \frac{1}{n+1}\sum_{i=1}^{n}(n-i+1) = \frac{n}{2}, \qquad Ede = \frac{1}{n}\sum_{i=1}^{n}(n-i) = \frac{(n-1)}{2}$$

可见，在顺序表中插入或删除一个元素时，平均约移动表中的一半元素，若表长为 n，则上述算法的时间复杂度均为 O(n)。

(4) 其他运算

在向量中除了上述两种基本运算外，还有一些较为复杂的运算，比如在非递减有序表中插入一个数据元素 x，使线性表仍保持非递减有序；在非递减有序表中删除所有值为 x 的元素等等，在这里就不作详细介绍了。

§2.3 线性表的链表存储结构

从上一节的讨论可以看出：采用顺序存储方式的线性表，在插入和删除操作时往往造成大量信息的移动，效率较低，同时，顺序存储必须占用一片地址连续的存储空间，存储分配只能预先进行。如果插入数据量超出预先分配的存储空间，要临时扩大有很大困难，本节将讨论另一种存储结构——链式存储结构，由于它不要求逻辑上相邻的元素在物理位置上也相邻，因此它没有顺序存储结构所具有的弱点。

2.3.1 单链表与指针

线性表的链式存储结构的特点是可用内存空间中一组不连续的任意的存储单元存储线性表的数据元素，分配给每个结点的存储单元一般分为两个域：其中存储数据元素信息的域称为数据域，该域可以是一个简单类型域，也可是包含较多信息的结构类型；另一个存储直接后继结点地址的域称为指针域，指针域中存储的信息称做指针。n 个结点(a_i(1≤i≤n)的存储映象)链接成一个链表。由于此链表的每个结点中只包含一个指针域，故又称单链表。例如图 2.3 所示为线性表$(a_1,a_2,\cdots,a_n)$的线性链表存储结构，整个链表的存取必须从头指针 h 开始进行，头指针 h 指示链表中第一个结点的存储位置，同时，由于最后一个结点没有后继结点，它的指针域为空(NULL)，用∧表示。若线性表为空，则头指针 h=NULL。每个结点的存储结构定义为：

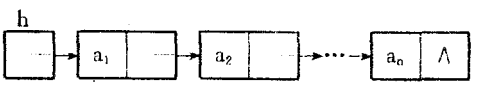


图 2.3 单链表的存储结构示意图

```
struct node
  { ELEMTP data;      /*数据域*/
    struct node *next;  /*指针域*/
    }
```

假设h,p,q为指针变量,可用下列语句来说明:

```
struct node *h,*p,*q;
```

变量p被定义为指针型,但未指向任何实际结点,即p变量中没有某实际结点所占空间的地址值。假设h已指向链表第一个结点,通过"p=h;",则p也指向链表第一个结点。使用语句:"p=(struct node*)malloc (sizeof(struct node));"可令p指向一个新的结点。如果要把结点归还系统,则用函数free(p)实现。一个指针变量占4个单元,而它所指向的结点则根据数据元素结构不同,占用存储单元数不同,一般一个结点占用较多单元。因此图2.3中指针变量h对应一个小方框,画图时省去指针变量的方框,而它所指的结点是一个大方框。

结点p的数据域用p->data来表示,它的指针域用p->next来表示。指针变量的常用操作如图2.4所示。

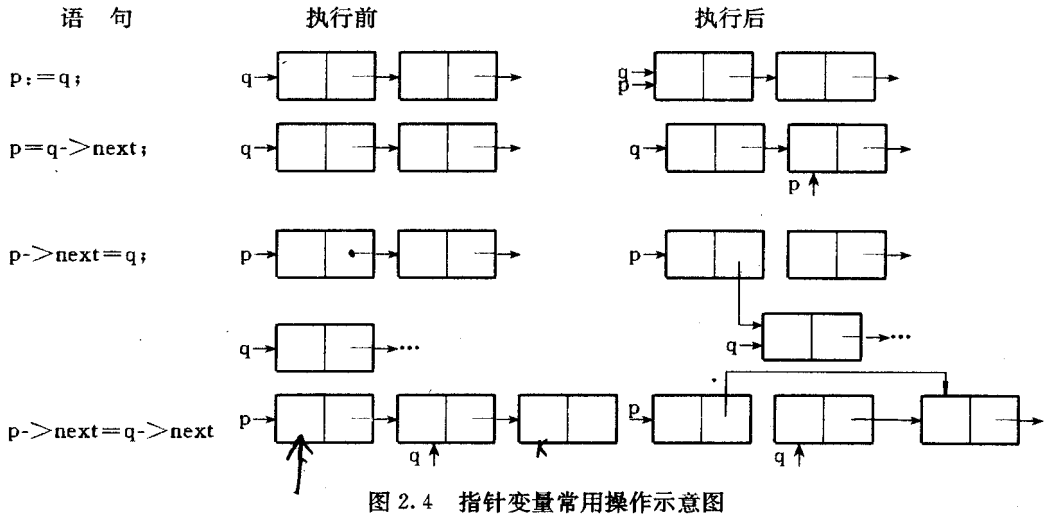


图2.4 指针变量常用操作示意图

2.3.2 单链表的基本运算

(1) 查找

已知头指针h,查找值为x的结点,返回指针值,如图2.5所示,算法如下:

```
struct node *search (struct node *head,ELEMTP x)
{ P= head ;
  while (p ! = NULL && p->data! =x)  p = p-> next;
  return (p);
  }
```

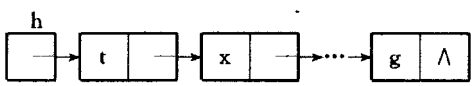


图2.5 单链表查找示意图

由算法可以看出,为了找到链表中的某结点,需用一个指针变量从头指针处逐步向后移动查找,由于移动次数不确定,所以用while循环或do-while循环结构,而不用for循环。