软件技术基础

王庆瑞 林金
蒋明 谭
编著

科学出版社

软件技术基础

王庆瑞　蒋　林　谭明金　编著

科学出版社

2001

内 容 简 介

本书将C语言程序设计、数据结构和软件工程等内容融为一体，旨在向读者系统地介绍软件技术中最基本的知识。内容包括C语言的基本概念、基本语句、数组、指针、函数、文件、编译预处理等，数据结构中最基本的表结构（包括链表结构）、树结构和图结构，算法设计中常用的递归、分治、动态规划、回朔等方法，面向对象的程序设计方法，软件工程的常用开发模型和工具，统一模拟语言UML等。内容丰富，叙述简练，每章都配有练习题。

本书可作为大学计算机软件技术基础课程教材或教学参考书，也可作为广大电脑爱好者学习程序设计方法的自学书籍。

图书在版编目(CIP)数据

软件技术基础/王庆瑞等编著. -北京：科学出版社，2001
ISBN 7-03-009340-2

Ⅰ.软… Ⅱ.王… Ⅲ.软件-基本知识 Ⅳ.TP31

中国版本图书馆CIP数据核字（2001）第25697号

科学出版社 出版
北京东黄城根北街16号
邮政编码：100717

新蕾印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2001年8月第 一 版 开本：787×1092 1/16
2001年8月第一次印刷 印张：16
印数：1—5 000 字数：374 000

定价：24.00元

（如有印装质量问题，我社负责调换〈环伟〉）

前　言

计算机基础教育是当前我国高校和社会教育的重要课题，其教学内容包括计算机文化基础、计算机软件基础和计算机硬件基础，以及计算机应用基础三个主要层次，是理工科大学生文化素质教育的重要组成部分。

计算机软件技术是一个庞大的概念群，其中软件技术的基础知识，一般应包括操作系统知识、程序设计知识（包括程序设计语言）、数据结构知识、软件工程知识，以及软件发展的新技术等诸多方面。要将这些内容十分广泛的知识全部容纳在一本书中，作为非计算机专业低年级大学生的教材，在有限的学时内使他们很好地掌握，的确有很大的难度。

考虑到上述因素，本书的基点放在软件技术最基本的知识，即程序设计和算法设计这条主线上，以当前较为普及的教学语言 C 语言作为程序设计的主体语言，数据结构内容则用于训练学生的基本程序设计方法。在此基础上，介绍一些算法设计中常用的基本算法，以加强对学生算法设计方法的训练。最后，再简单地介绍面向对象的程序设计和软件工程的基本概念。

本书是根据军队院校计算机软件技术基础课程教学大纲要求，参考地方高校相关课程的教学大纲，并结合目前软件技术发展趋势而编写的。全书共分十章，第一章介绍软件和算法的基本概念。第二章到第六章介绍 C 语言的基本语法及简单程序设计方法，包括基本数据类型和运算规则，条件语句、循环语句等常用语句用法，数据的输入输出方法，以及数组、指针、函数、文件、编译预处理等。第七章和第八章介绍 C 语言的结构类型和基本数据结构知识，包括顺序表、链表、树和图结构等等。第九章介绍简单而又常用的算法设计方法和面向对象的程序设计方法。第十章介绍软件工程的概念和常用开发模型。

本书作为解放军理工大学非计算机专业本科学生的计算机软件技术基础课程教材，是继计算机文化基础课程之后开设的一门计算机基础教育课程。希望通过本课程的教学，使学员更好地掌握计算机基础知识，培养学员的程序设计能力和简单算法设计能力。一般情况下，教学时数应安排 70 到 90 学时（含上机实践时间），可根据具体条件作适当增减，教学内容也可适当取舍，学时不足时，适当略去一部分标“*”号的章节内容。多上机练习是学好本书内容的捷径，希望读者尽可能多地动手编程并上机调试。

本书是在解放军理工大学机关的直接领导下进行编写的，其内容经有关专家组成的指导小组论证审定，由解放军理工大学机关组织编写小组。王庆瑞拟定编写题纲。蒋林编写第二章到第六章。谭明金编写第十章。王庆瑞编写第一章和第七、八、九章，并对全书各章内容进行认真细致的修改。最后由李健和王元元教授审阅定稿。在本书编写过程中，得到解放军理工大学及其各学院有关领导和人士的大力支持，在这里一并表示感谢。由于作者水平有限，书中难免有错误或不当之处，请读者批评指正。

编　者

目　录

第一章 概 论

1.1 计算机软件及其发展

人们在回顾计算机发展历史的时候，往往根据制造计算机硬件的原材料将计算机发展过程分成若干代，比如电子管代、晶体管代、集成电路代等等。众所周知，计算机系统由硬件系统和软件系统两大部分组成。在计算机硬件系统发展的同时，软件系统同样飞速发展着。

软件发展主要体现在编程语言的发展，操作系统的形成和发展，软件开发工具和环境的发展，以及软件工程的形成和发展等几个方面。软件又分成系统软件和应用软件两大类。通常把那些为其他软件提供运行环境，支持其他软件运行的软件称为系统软件。比如，操作系统，语言处理软件，网络管理软件以及一些服务性软件等都属于系统软件。其余的软件统称为应用软件。

本书不打算系统地介绍计算机软件的发展史，仅通过程序设计语言的发展和操作系统的发展情况作一点粗略的介绍。希望读者能从中体会到软件技术发展对计算机科学的推动作用。

1.1.1 程序设计语言的发展

根据冯·诺依曼体系，计算机是存储程序，由程序控制的自动化系统，使用计算机求解问题，就意味着要为计算机设计相当的处理程序。由于计算机的硬件(运算器和控制器等由逻辑电路构成的基本部件)只“认识”0 和 1，所以程序的最终形式都是由 0 和 1 组成的二进制代码形式。

早在计算机诞生之初，人们直接用二进制形式编程，但是这种在计算机看来十分明了的程序，在人看来却是一部“天书”。后来，人们又将三个二进制位合并在一起，这就形成了八进制，再后来，为了与字节对应，又将四个二进制位合在一起，就变成了十六进制。

不管八进制还是二进制，用数字表示程序都不直观，仍然很难读，于是人们便产生了用符号代表指令的想法，设计出了汇编语言。比如，用 ADD 表示加法指令，SUB 表示减法指令等等。事先设计一个能将汇编语言写的程序(叫源程序)翻译成机器指令的软件(叫做汇编程序)，装入计算机，人们用汇编语言编写出源程序之后，由计算机自动地将源程序翻译成计算机能够直接执行的二进制程序(称为目标程序)，这种二进制形式的语言称为机器语言。汇编程序就是将汇编语言的源程序翻译成机器语言程序的翻译器。一台计算机配上了汇编程序就相当于人们已经“教会”计算机认识汇编语言了。汇编程序是最早出现的计算机软件。直到现在，汇编语言仍然是开发最底层软件的常用工具之一。

有了汇编语言，减轻了人们的编程工作，但用起来仍然十分吃力。由于汇编源程序与机器语言程序有简单的对应关系，即一条汇编指令对应一条二进制的机器指令，不同种类的机器有不同的机器语言，不同类型的机器之间，“听不懂”对方的语言，所以用汇编语言

编写的程序缺乏通用性和可移植性。于是人们又设计出“高级程序设计语言”，相对地，机器语言和汇编语言便称为低级语言。

最早出现的高级程序设计语言是 FORTRAN 语言(Formula Translator，公式翻译程序语言)，由于早期计算机的主要应用是“数值计算”，所以在设计程序语言的时候，着重考虑如何充分发挥计算机的“计算”能力，因而 FORTRAN 语言是特别适合于数值计算的高级语言。接着出现了便于初学者使用的，有较强的人-机对话功能的“解释”性的语言 BASIC(Beginners All-purpose Symbolic Instruction Code，初学者通用符号指令码)和便于非程序员使用的商业语言 COBOL(Common Business-Oriented Language，面向商业的通用语言)等等。

FORTRAN 语言和 BASIC 语言在相当长的一段时期内占有“统治”地位，对计算机程序设计技术的普及和发展起到很大的作用。

在计算机发展过程中，人们逐步认识到“算法”的重要性。陆续设计出更加便于算法描述的 ALGOL(Algorithmic Language，算法语言)语言；汇集多种语言优点，被称作大型公共汽车的 PL/1 语言(Programming Language/1)；更加便于设计结构化程序的 Pascal 语言；以及便于设计系统程序的 C 语言等等。

为了满足不同行业的需要，人们还设计出各种各样的专用语言，这里不能一一列举。值得一提的是，随着计算机技术的发展，人们提出了面向对象的概念，并设计出面向对象的程序设计语言，比如早期的 Simula 语言，后来的 Smalltalk 语言、Ada 语言、Java 语言、C++语言等等。

正像将计算机分成第一代、第二代……一样，人们也将程序语言分级。随着计算机技术的发展，新开发的程序语言的功能也不断地增强，程序设计的“自动化”程度也越来越高，如果只分成低级和高级两类，不能完全反映程序语言的发展过程和适用范围。将与计算机硬件联系最为密切的机器语言和汇编语言称为一级，将 Pascal 和 C 等面向过程(或称过程式)的语言称为二级，将具有面向对象功能的 C++语言和 C++ Builder 等称为三级，将更为高级的语言，如 VB(Visual Basic)，Delphi 和 Power Builder 语言称为四级。由于 C 语言能直接对计算机硬件进行操作(所以它适合于编写系统软件)，也有人视其为介于低级语言和高级语言之间的一种语言。

需要说明的是，语言的级别越低，对使用者所掌握的计算机知识要求越高，像一级语言和二级语言，都属于“程序员级”的语言，即必须掌握足够的专门知识才能使用好这类语言。相反，语言的级别越高，对使用者的要求则越低，即使用这类语言编程无需掌握很多计算机专业知识。用较低级语言进行软件开发往往要(与用高级语言相比)花费更多的精力和时间，因为需要更为精细地控制计算机的操作。

语言的发展同时促进了编程技术的发展。首先，语言的“封装”性能越来越强，起初，ALGOL 语言第一次提出了复合语句和分程序的概念，将若干条相关语句封装起来成为一个整体。后来 Pascal 语言又将不同类型的相关数据封装在一起产生了记录类型，有了上述简单封装手段，大大地增强了程序的结构化。再后来的 C++等语言，进一步地将数据和处理这些数据的语句(程序段)封装在一起，产生了“类”和“对象”的概念。封装技术的发展和应用，不仅使程序具有良好的结构性，更主要的是促进了软件复用技术的形成和发展，以及软件工程的形成和发展。

1.1.2 操作系统的形成和发展

计算机操作系统(Operating System,简称 OS)的形成和发展对计算机应用也起着十分巨大的作用。操作系统是最典型的系统软件,它负责管理计算机的所有硬件和软件资源,控制计算机的工作流程,是人与计算机的界面(或称接口)。

人们形象地将仅由硬件组成的计算机称为"裸机",操作系统就像是给裸机穿上的内衣,是裹在裸机上的第一层软件。而程序语言(实际上是语言的编译系统)属于第二层软件,在操作系统和程序语言基础上开发的一些应用软件都属于高层软件。

早期,人们直接使用计算机裸机非常麻烦,先将编写好的程序穿成纸带或卡片,通过手工操作方式输入到计算机中。由于手工操作速度太慢,后来设计出批处理系统,将已穿成纸带或卡片的若干用户程序,通过主机或专门用来输入作业的卫星机读入计算机内存,由计算机自动地将其逐一执行,初步实现上机操作的自动化。后来又配备一些标准 I/O 子程序及一些子程序库。

随着硬件的发展,出现了"中断"和"数据通道",设计出控制多道程序并发执行的管理系统,当正在执行的一个用户程序需要输入输出数据时,主机启动外设,在外设进行输入输出操作期间,主机并不停止等待而是执行别的程序(外设的速度远远慢于主机的速度),这样主机与外设就可以平行地进行工作。从这一时期起,操作系统就逐步形成了。

以后又逐步开发出分时处理操作系统、实时操作系统等等。

一般来说,操作系统的处理模块分为处理机管理(又称进程管理和线程管理)、存储器管理、设备管理、文件管理和作业管理等五大部分。

处理机管理模块负责合理地利用 CPU 资源,使计算机各种设备很好地并发执行,充分利用处理机资源。存储器管理模块负责将内存空间划分成若干页或若干段,将多个用户程序同时装入不同的页或段中,充分利用内存资源。设备管理模块负责分配和调度计算机的外部设备,使每个用户程序都能正常工作。文件管理模块主要负责分配和管理磁盘等外部存储器,使用户的文件能安全而又方便地存储在磁盘上。至于作业管理主要用于批处理系统,当多个作业同时装入计算机时,由作业管理模块进行合理的调度,使每个作业都能正常运行。

微机的产生和发展在某些方面也推动着操作系统的发展。由于微机(又称个人计算机,或电脑)属于单用户机器(每个时刻只能由一个人使用),微机所配备的操作系统更多地考虑如何方便用户,所以微机操作系统一般都配有功能强大的文件管理系统和极为"友好"的用户界面。DOS 属于字符界面的操作系统,用户都是通过输入字符命令使用计算机的,非专业人员很不习惯。Windows 系统属于图形界面的操作系统,操作系统的命令都用图形(图标,按钮等等)展示在屏幕上,用户使用鼠标点击图标就完成相应的操作,一般人员无需很多计算机专业知识便能运用自如,所以用户界面十分友好。

随着计算机网络的发展,相应的产生了分布式操作系统,网络操作系统等等。

1.1.3 程序的一般结构

无论哪种程序语言,所提供的基本程序结构无非是顺序结构、分支结构和循环结构三大类,其中顺序结构是最基本的程序结构。采用不同的程序结构可以控制不同的程序流

程(即程序执行路线)。

所谓顺序结构是指程序中语句(汇编语言中将语句称作指令,语句和指令又统称为代码)的执行次序是按语句在程序中的自然次序(一句句地)依次执行。任何情况下,都是先执行(主程序中的)第一条语句,而后再逐一执行后面的语句。顺序结构如图 1.1(a)所示。

分支结构又称选择结构,这种结构的开头具有一个选择条件,根据选择条件,从几个分支中选择一个分支执行。其中,最简单的是二分支结构,即根据选择条件的成立与否从两个分支中选择一个分支执行。图 1.1(b)和(c)给出二分支结构示意图,Pascal 语言中的 CASE 语句和 C 语言中的 switch 语句都具有多分支功能。

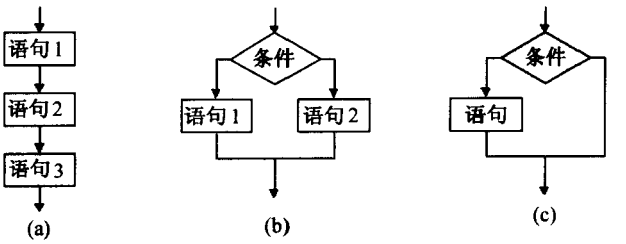

图 1.1 顺序结构和分支结构

第三种结构是循环结构(又称重复结构),这种结构都设立一个循环体和一个循环控制条件,循环体往往由一组语句(即程序段)组成。在循环条件的控制下,反复地执行循环体,直到循环控制条件达到某种状态为止,结束本循环语句的执行。

图 1.2(a)和(b)给出两种循环结构示意图,前者属于"当"型循环结构(当条件满足时就执行一次循环体),后者属于"直到"型循环(执行一次循环体后,再根据循环控制条件确定是否再执行一次循环体)。

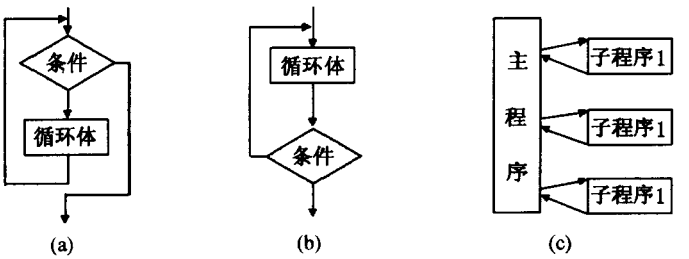

图 1.2 循环结构和子程序结构

三种结构可以互相嵌套,即循环结构中可以出现顺序结构、分支结构和循环结构,分支结构中也可以出现循环结构等等。

除了上述三种基本结构外,子程序结构(见图 1.2(c)),也是常用的程序结构。子程序是供主程序和其他子程序(统称为主调程序)调用的一个程序单位。当主调程序中出现

调用子程序的语句时，程序流程转向子程序，当子程序执行完毕后，再转回主调程序的调用点继续执行主调程序。

当然，子程序结构中也包含着顺序结构、分支结构和循环结构。几种结构相互嵌套，就构成了复杂的程序结构。

总之，顺序结构是最基本的程序结构，如果把每个子结构(分支结构，循环结构和子程序结构)都看成一条语句的话，那么整个程序宏观上是顺序结构的。

1.2 算法和数据结构概述

1.2.1 问题的求解过程

计算机技术的发展和应用，将人类社会推入一个全新的信息时代。人们已经习惯于让计算机做这做那。

大家都知道，计算机系统包括硬件系统和软件系统，硬件系统好比人的躯干，软件系统好比人的灵魂。为了能让计算机乖乖地为人类做更多的事，不仅要为它配备一套性能良好的硬件设备，还要为其设计出一套套能够解决各式各样问题的软件(即计算机程序)。要想让计算机代替人们解决某个问题，就得设计出求解该问题的程序，使计算机具有解此问题的能力。

程序设计过程实际上就是建模和解模过程。建模过程就是将所要求解的问题抽象成数学模型，解模过程就是要设计出求解这个模型的算法。

首先要对具体问题进行深入细致的分析，确定该问题所涉及到的有关数据，包括输入数据和输出数据。找出这些数据之间的内在联系，选择一种恰当的表示形式表示这些数据(即编码)。考虑用什么样的方法去处理这些数据才能达到求解目的，如何有效地将这些数据组成一个有机的整体(即下面所说的数据结构)，选择什么样的存储方式存储这些数据，并体现它们之间的联系，才能使程序处理起来更加方便，效率更高。

经过一番深思熟虑之后，开始起草解题方案和求解算法。再对算法的可行性进行论证，对算法的运行效率作比较客观地评估。在确认能够满足解题要求之后，才着手编程，并上机调试。这中间可能还要几经反复，在每个环节上都可能修改甚至推翻原方案，最终得到一个性能良好的计算机程序，交付使用。

要设计出性能良好的计算机程序，除了要弄清所求问题的来龙去脉，需求关系之外，还要具备程序设计知识，掌握一种或多种程序设计语言，同时还要具备一定的算法设计知识和数据结构知识。当然，如果参与较大型的软件设计，还应当具备软件工程知识，并掌握一定的软件开发技术和手段。

1.2.2 数据结构和算法的概念

数据是对客观事物的名称、数量、特征、性质的描述形式(即编码)，是计算机所能处理的一切符号的总称。数据既是计算机加工的对象，又是计算机的产品(计算结果)。

我们对那些单个的孤立的数据并不感兴趣，而着重研究由众多数据元素组成的数据集合，研究集合中数据元素之间存在怎样的内在联系，通常需要对数据和数据集合进行哪些运算(即对数据进行的处理)，如何提高运算效率等等。

在数据集合中，一个数据元素(data element)又叫做一个数据结点，简称结点(node)。同一个集合中，结点应当具有相同的数据类型。

一个数据结点由用来描述一个独立事物的名称、数量、特征、性质的一组相关信息组成。如果一个结点含有多个数据项(如记录型结点或结构型结点)，每个数据项叫做结点的一个域(field)，能够用来唯一标识结点的域称为关键字(key)。如果一个数据结点只含一个数据项(即单值结点)，不妨把结点看成一个整数。

例如，在设计处理学生成绩问题的程序时，每个学生有关的数据项(域)构成一个数据结点，可能包括学生的姓名、学号、各科考试成绩等等，学号可以作为结点的关键字。在处理库存商品问题时，一个数据结点对应一种商品的相关数据项，包括商品编号和名称、规格、数量、生产厂家、单价、入库日期等，商品编号可以作为关键字。

在描述算法时，为方便起见，常常用关键字代表结点。

一个结点集合，以及该集合中各结点之间的关系，组成一个数据结构(data structure)。

常见的数据结构有表结构、树结构和图结构等等。

如果结点之间存在某种简单的先后次序关系，那么这种结构就属于表结构。

如果结点之间存在着层次关系(或嵌套关系)，那么它就属于树结构。

如果结构之间存在复杂的“多对多”关系，那么它就属于图结构。

数据在计算机内的存储形式叫作数据的存储表示，也称为存储结构。在存储数据的同时，还必须体现出数据之间的关系。

用来存储一个数据结点的存储单元叫做一个存储结点。因为一个数据结点对应一个存储结点，所以通常存储结点也简称为结点，准备用来存储但尚未存储数据的存储结点叫空白结点，或曰空结点、自由结点。处理数据和数据结构的操作称为对这个数据结构进行的运算，查找、插入、删除、排序和遍历等都是很常见的运算。

算法记载了人们求解问题所采用的方法和步骤，是“程序的流程”。这里所说的“求解问题”是指为解此问题而设计计算机程序。因为有了程序，计算机就能“计算出”问题的答案，所以程序就是问题的解答。

算法有两种形式，一种是程序形式，另一种是描述形式。

程序形式(如 C 语言程序等)又称为算法的实现形式，是算法的最终形式。从这个意义上说，算法就是程序，程序中含有算法。

除了编写程序之外，人们常常用文字叙述形式，或画框图(即流程图)形式来介绍算法，我们把它叫做算法的描述形式。很多有关数据结构和算法书籍中，还用类语言(一种不精确的 C 语言或 PASCAL 语言等)形式描述算法，有人称为算法的伪代码形式。

文字叙述形式简单直观，易于理解。

例如，有一种非常简单的自然选择排序算法能将 n 个无规律的整数排列成由小到大的形式(即排序问题)，假定这 n 个数存储在数组 a[n]中，这个算法用文字叙述如下：

先从 a 的 n 个元素中选择一个最小元素(比如某个 a[j])，将它与 a[0]交换位置；

再从剩下的 n－1 个元素中选择一个最小元素，将它与 a[1]交换位置；

再从剩下的 n－2 个元素中选择一个最小元素，将它与 a[2]交换位置；

……

反复进行上述选择和交换工作，直到选择出最后一个元素，从而完成排序工作。

用 C 语言的循环语句描述排序过程：

```
for  (i=0; i<n-1; i++)
  { k=i;
    for  (j=i+1; j<n; j++) if  (a[j]<a[k]) k=j;
    w=a[i];  a[i]=a[k];  a[k]=w;
  }
```

虽然读者可能对 C 程序尚不熟悉，但有文字叙述形式相对照，也能猜得差不多。

用流程图(框图)描述算法也很直观。

例如，在已经排好序(由小到大)的整型数组 a[n]中，查找一个元素 x(整型变量)，如果 x 在数组 a 中出现，就用变量 i 记住它所在的数组下标，若 x 不出现，则 i = -1。

求解这个问题，有一个速度非常快的二分查找算法，用文字叙述该算法的执行步骤如下，算法中变量 left，right 和 mid 分别用来记住数组段的左右下标和“中值”点下标。

(1)置 left 和 right 的初值，即 left = 0；right = n - 1；

(2)如果 left > right，则查找不成功(即数组 a 中没有 x)，使 i = -1，算法终止。否则：

(3)计算中值地址，置下标 mid = (left + right)/2；

(4)比较 x 与 a[mid]的值：

(4.1)如果 x = a[mid]，则找到 x，使 i = mid，算法终止。否则：

(4.2)如果 x < a[mid]，则调整右指针 right，使 right = mid - 1，转到步(2)继续查找；

(4.3)如果 x > a[mid]，则调整左指针 left，使 left = mid + 1，转到步(2)继续查找。

这样，反复步(2)到步(4)，就能完成查找工作。

图 1.3 给出了上述二分查找的流程图。

与单纯的文字叙述相比，用流程图描述的算法结构更清晰，也更容易理解，更接近程序形式。只是画起来比较麻烦，画错了又不容易修改。

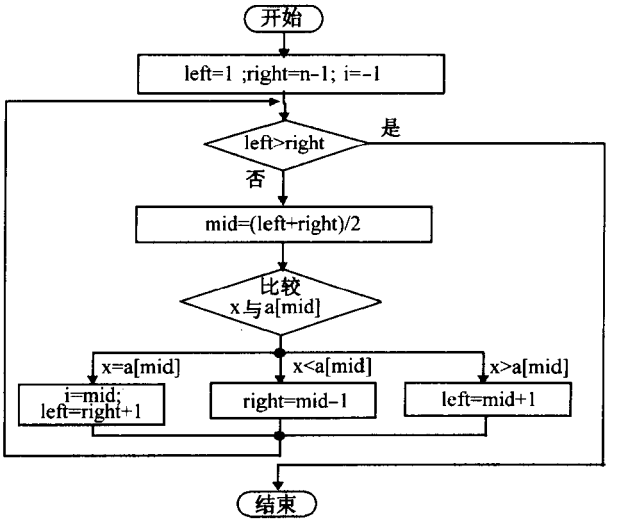

图 1.3　二分查找算法流程图

1.2.3 算法评价方法

评价一个算法的综合性能往往需要从几个不同的角度去考虑。主要是算法的正确性和算法的运行效率。

算法的正确性是最起码的,也是最重要的。

一个正确的算法(或程序)应当对所有合法的输入数据都能得到应该得到的结果。比如一个排序算法,只有对任意 n 个数据都能完成排序工作的算法才是正确的排序算法。

对于那些简单的算法(或程序),可以通过上机调试验证其正确与否。调试用的数据要精心挑选那些具有"代表性"的,甚至有点"刁钻性"的,以保证算法对"所有的"数据都正确。

但是,一般来说,调试并不能保证算法对所有数据都正确,只能保证算法对部分数据正确,调试只能验证算法有错,不能证明算法无错。就是说只要找出一组数据使算法失败(即计算结果不对),就能否定整个算法的正确性。但调试往往不能穷尽所有可能的情况,所以,即使算法有错,也不一定能通过调试在短时间内发现。不少大型软件在使用多年后,仍然还能发现其中的错误就是这个道理。

要保证算法的正确性,通常要用数学归纳法去证明。

评价算法性能另一个要考虑的因素就是算法的运行效率(或曰运行成本、费用),也就是要估计一下算法投入运行时,大致需要耗用多少时间,占用多少内存单元(即空间),其时间空间需求量能否满足客观要求。其中最主要的是算法运行时间的估算。

一般说来,一个算法的时间耗用量将随输入数据量的增大而增大。比如,排序算法,对 10 000 个元素排序所用时间要比对 100 个元素排序所用时间要长。要精确地算出一个算法所用时间是非常困难的。为了简单起见,通常根据算法的循环次数,递归调用次数等,粗略的估算所用时间与输入数据量之间的比例关系(即函数关系)。

设输入数据量为 n,若某算法的时间用量函数 T(n)差不多与 n 的平方成正比,我们就说这个算法的时间耗费是 n 的平方阶的,用符号表示成:

$$T(n) = O(n^2)$$

读作"Tn 是 On 平方的"。

在计算时间耗用函数 T(n)时,通常只估算到 T(n)最高项的阶,既不考虑低阶项,也不考虑常系数。比如,有一个算法 A,其时间耗用函数为:

$$T1(n) = 20n^2 + 100n$$

另一个算法 B,其时间耗用函数为:

$$T2(n) = 0.5n^2 - 3n + 18$$

那么,这两个算法的时间耗用函数的阶是一样的,都是 $O(n^2)$的。

一般说来,当 n 很大时,算法时间耗用量的阶越低,算法的运行速度越快,所以低阶算法(指时间耗用量的高低)比高阶算法好。

下面由低到高列出一些经常用于表示算法时间耗用函数的阶。

O(1)　　常数时间,即算法时间用量与输入数据量 n 的大小无关。

O(log n)　对数时间,对数阶通常不写底数,在计算机学科中,对数一般以 2 为底。

O(n)　　线性时间,即算法时间用量与 n 成正比。