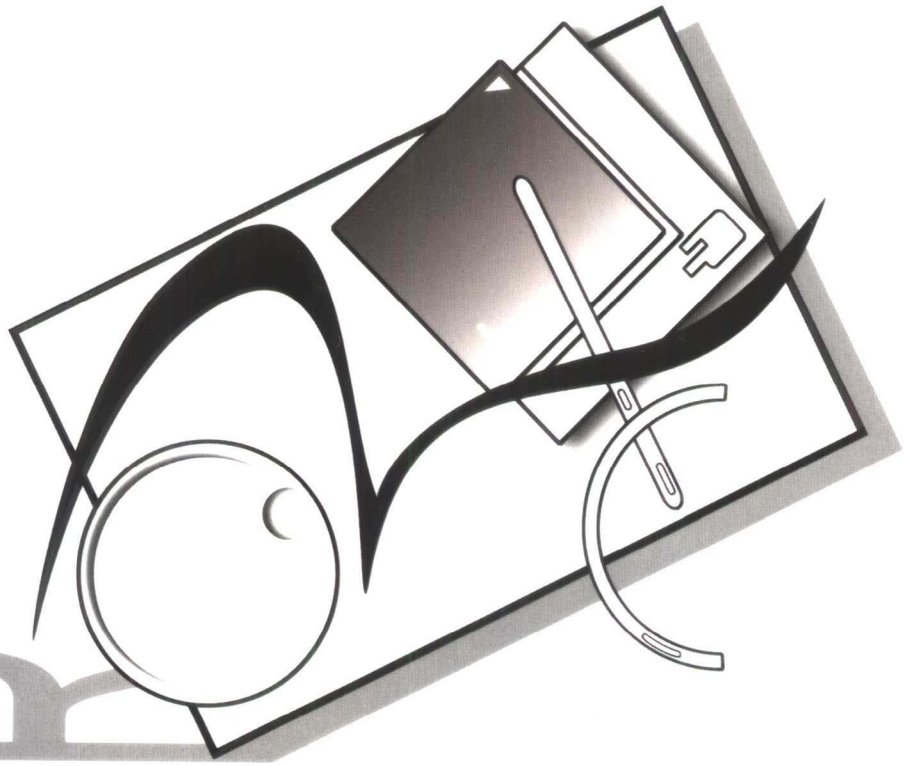
高等职业教育计算机专业推荐教材



COMPUTER

数据结构与算法基础

王庆瑞 [illegible] 编著

机械工业出版社
CHINA MACHINE PRESS

高等职业教育计算机专业推荐教材

数据结构与算法基础

王庆瑞　陈卫卫　编著

机械工业出版社

本书以线性表、栈、队、链表、树、图结构以及排序算法为主线，以基本数据结构的常见运算——查找、插入、删除为基础，介绍了算法设计中最基本的概念和方法，用通俗的语言和结构优美的程序，深入浅出地阐明了算法设计常用的方法和技巧，旨在培养学生程序设计的良好习惯，提高程序设计能力，使他们逐步学会编写具有一定难度的高质量程序。

书中每章最后均给出这一章算法的配套程序和习题，以供学生实践。

本书主要用作高等职业学校计算机专业教材，也可作为广大电脑爱好者学习程序设计方法的入门性科技读物。

图书在版编目(CIP)数据

数据结构与算法基础/王庆瑞，陈卫卫编著. —北京：机械工业出版社，2005.8

(高等职业教育计算机专业推荐教材)

ISBN 7-111-17179-9

Ⅰ.数… Ⅱ.①王… ②陈… Ⅲ.①数据结构—高等学校：技术学校—教材 ②算法分析—高等学校：技术学校—教材

Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字(2005)第 092179 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)

策　　划：胡毓坚

责任编辑：陈振虹

责任印制：杨　曦

北京蓝海印刷有限公司印刷

2005 年 9 月第 1 版·第 1 次印刷

787mm×1092mm　1/16·17.25 印张·424 千字

0001 - 5000 册

定价：24.00 元

高等职业教育计算机专业推荐教材
编委会成员名单

编 者 的 话

根据有关部门对我国信息产业发展的客观需求及劳动力市场现状的调查，在计算机应用和软件专业领域培养技能型紧缺人才，是当务之急。近年来，不仅高等职业技术类院校，而且相当一部分本科类工程技术院校（特别是相当数量高等学校的二级学院、民办院校），都把招收和培养计算机专业技能型紧缺人才列为教育改革的重要举措。为一些院校提供“适时、适度、优选、优质”的计算机专业的高等职业教育系列教材，正是我们组织编写这套“高等职业教育计算机专业推荐教材”（以下简称“推荐教材”）的目标。“推荐教材”由四个模块的30多本教材组成。这些模块是：基础知识模块、程序设计模块、实用技术模块、实践模块。

这套“推荐教材”是“适时”的，因为它努力适应我国信息产业发展和劳动力市场的客观需求，适应计算机行业技术的现状，强调教学内容的先进性和实用性。这套教材十分关注信息技术的最新发展，突出本专业领域的新知识、新技术、新流程和新方法。其中程序设计模块和实用技术模块充分体现了这一特色，所涉及的19本教材既有基础的平台、语言，如《Linux操作系统》、《C语言程序设计与实践》，也有最新的《Visual C#.NET面向对象程序设计教程》、《XML实用教程》、《JSP应用教程》等工具，还有十分接近实际工作需要的《Oracle数据库应用教程》、《计算机网络管理》、《电子商务概论》等实用教材。

这套“推荐教材”是“适度”的，因为它不是简单地摒弃基础理论，而是注意强调理论联系实际，努力做到专业技能型人才能从中学习到必要和相对系统的基础理论知识，把各种能力的培养和全面素质的提高放在首要的位置。“推荐教材”中基础知识模块的设置，充分体现了这一特色，它囊括了从数学基础、电子基础、硬件技术基础到系统软件基础、应用技术基础、网络技术基础、信息安全基础等10本教材。

这套“推荐教材”是“优选”的，因为充分考虑了现有高中毕业生的认知水平和已有知识，为学生提供适应劳动力市场需要和有职业发展前景的、模块化的教材体系。在学习内容、教学组织等方面留给教师和学生选择和创新的空间，便于教师组织和构建开放式的课程体系，适应学生个性化发展的需要，在灵活的模块化课程结构中自由发展。“推荐教材”的四个模块对重要内容都安排了看似重复的多种教材，供教师和学生去选择。例如，可以在《C语言程序设计教程》、《Java程序设计教程》中任意选择一到两门；也可以在《ASP基础及应用教程》，《JSP应用教程》中任选一门。

这套“推荐教材”是“优质”的，因为它们的作者多数是从事高等职业教育的计算机专业教师，具有长期的计算机实际工作和教育工作经验。这套教材的优质，还体现在它的改革和创新精神上。其中《计算机电路基础》对传统的电路、模拟电路和数字电路课程教材作了重大的改变，《计算机组装与维修教程》则是一门纯实践的课程教材。我们欢迎使用这套教材的师生，指出教材中存在的问题并提出修改意见。

高等职业教育计算机专业推荐教材
编委会

前　言

本书以高等职业学校计算机专业学生，以及广大电脑爱好者作为主要读者对象，旨在培养他们的计算机程序设计能力和初级算法设计能力。与目前市面上各种计算机软件应用方面的书籍不同，本书不但要求读者能熟练地操作计算机，更强调编写性能良好的计算机程序。

考虑到本书的主要读者对象，在内容编选上，以最基本的表、树、图结构，以及排序算法为主线，介绍数据结构和算法的基本原理，以及算法设计的基本方法。

全书共分 6 章，每章内容的后面都附有本章内容的小结、本章算法的配套程序，以及习题。

第 1 章介绍数据结构和算法的有关概念，以及算法的表现形式（描述和实现）和评价算法性能的方法。

第 2 章介绍线性表在顺序存储下完成基本运算（查找、插入、删除等）的算法，以及进栈、退栈、进队、出队等运算算法。

第 3 章介绍链表的设计方法。内容包括链表的基本形式，链表的构造和输出方法，有序链表和无序链表的构造、查找、插入、删除算法。

第 4 章介绍树结构。主要内容有树和二叉树的基本概念，二叉树的遍历和构造算法，遍历运算的简单应用，检索树的概念，检索树的构造、查找、插入、删除算法设计方法，以及哈夫曼树的概念，构造哈夫曼树的哈夫曼算法和哈夫曼树的应用。

第 5 章介绍图结构，主要内容有图的基本概念，图的邻接矩阵及其存储方法，图的深度优先搜索算法，以及求最小生成树的 Kruskal 算法和 Prim 算法，求最短路径的 Dijkstra 算法。

第 6 章介绍几种常见的排序算法，包括插入排序、冒泡排序、快速排序、堆排序等。

最后，给出 4 个附录。其中，附录 A 是关于每章算法的配套程序的说明。附录 B 介绍 C++ 的传引用（即引用型的参数）的用法。附录 C 和附录 D 分别介绍 VC（即 Visual C++ 6.0）和 TC（即 Turbo C 2.0 和 Turbo C++ 3.0）的简单用法。虽然只是附录，却含有学习本书内容必须掌握的预备知识，以及上机操作必备知识。所以，这些附录（尤其是附录 B）应当作为本书的先修内容。另外，书中打 * 号的小节可根实际教学需要取舍。

本书在介绍算法时，以文字叙述形式为主，并且给出相应的 C/C++ 语言函数。这里，所谓的“C/C++ 语言”指的是在 C 语言基础上，引入了 C++ 的两种语法成分，一是使用了“行注释”；二是有些函数（算法编号后面带“++”符号者）使用了引用型的形式参数。引入前者仅仅是为了录入方便（不影响程序的功能）；引入后者是为了用引用型的参数代替指针型的参数，从而达到“净化”程序结构的目的（详见第 1 章和附录 B 的有关部分）。

考虑到有些读者可能会对引用型形式参数的用法一时难以理解，所以，书中凡需要使用引用型形式参数的函数，都有两个副本，其一是含有引用型形式参数，其二是不含有引用型形式参数（用指针型的形参代替它们）。

为了帮助读者加深对算法的理解，本书中所有算法（指实现算法的 C/C++ 函数）均给出相配套的、能够直接上机运行的完整程序，将其附在各章正文后面，供读者参考。实际上，这些配套的完整程序是算法的一种测试环境，运行时，读者只要按照程序输出的提示信息，输入符合要求的数据，程序便会调用相应的函数，做出处理，并输出处理结果。读者可以通过观察输出结果，加深对算法的原理和应用情况的理解，或检查算法是否正确。每个完整的配套程序后

面都给出运行实例，这些运行实例，就含有输入数据和输出结果。读者可以比照运行实例，学习如何按照自我设计的步骤对程序进行测试。

书中所有配套程序均在 Visual C++6.0 环境下上机调试通过。运行实例，也是在 Visual C++ 6.0 环境下产生的。

这些配套程序也都能在 Turbo C++（3.0 或更高版本）环境下安全运行，只是，有可能显示不出汉字。

凡是算法编号后面不带“++”标记的（++标记表示用到了引用型参数），和其相应的配套程序，只要将行注释改为段注释（或去掉注释），也能在 Turbo C 2.0 环境下安全运行。同样，也有可能显示不出汉字。

换言之，这些配套程序在 Visual C++6.0 环境下运行最完美。为此，建议读者学会使用 Visual C++6.0。

对于那些已经习惯使用 TC（包括 Turbo C 2.0，Turbo C++3.0 等）的读者，如果用 Turbo C++3.0 作为上机实验环境，最好将汉字换成西文；如果用 Turbo C 2.0 作为上机实验环境，不仅要将汉字换成西文，还要将行注释改成段注释，同时将引用型参数改为指针型参数，在第 1 章中给出了修改示例。

学好、掌握好本书内容的最好方法是多多上机编程练习。在每章末的习题中，带“△”标记的是作者建议作为上机练习的题目。带“*”标记的题目难度较大，可选做。

作　者

目　录

第1章 引 论

1.1 基本概念

当人们着手准备为求解某给定问题设计计算机程序时，首先要对这个问题进行认真细致地分析，头脑里不时地思考着：这个问题涉及到哪些数据，这些数据之间有什么关系，用什么方法（即算法）去处理这些数据，如何存储这些数据，并且体现出它们之间的关系，才能使程序处理起来更加方便。

实际上，这里面涉及到针对具体问题，选用适当的算法和数据结构方面的知识。

在正式动手编程之前，通常还要对所用算法的可行性和效率进行论证和评估。在确认“问题不大”后，再着手编程，并上机调试。这中间还会几经反复，最后才能得到性能良好的计算机程序。

数据结构和算法（data structures and algorithms）是研究数据和数据之间的关系（relationship）、数据集合上的运算（operation）、数据的存储形式、完成运算的算法设计方法，以及对算法性能进行综合评价的一门学科。

在计算机科学中，计算机能够处理的一切信息统统称为数据。

比如，用 Word 进行文件编辑时，文件中的所有内容，包括文字、数值、表格、图片、图像、录音等都作为数据，尽管它们有不同的类型，不同的外部表现形式和不同的内存编码形式。

再如，对 C 语言源程序进行编译时，在编译系统看来，源程序是输入数据，而最终产生的可执行程序是输出的结果数据。

总之，数据是对客观事物的名称、数量、特征、性质等信息的描述及其编码形式，是计算机加工的对象和计算结果。

我们对那些单个的孤立的数据并不感兴趣，只关注由众多数据元素组成的数据元素集合。一个数据元素（data element）又叫做一个数据结点，简称结点（node）。

结点是由用来描述单一独立事物的名称、数量、特征、性质等组成的一组信息，每种信息作为结点的一个数据项，也称域（field），也就是说，结点的类型通常是结构类型。

比如，描述学生情况的数据结构中，一名学生的相关信息构成一个结点，结点包含学生的姓名、学号、班级、性别、年龄、若干门功课的成绩等数据项。由众多学生的数据结点组成一个结点集合。各结点既互相独立，又互相关联在一起。

本书为了简化程序结构和文字叙述，通常把结点看成一个整数，于是，结点类型可定义为：

```
typedef int node_type; //定义结点类型名
```

用 node_type 作为结点类型名，而不直接用 int 作为结点类型名，这只是一种变通方法，主要是为了便于将一般 int 类型变量与结点变量区分开来。当然，你也可以不这样做，而直接用 int 表示结点类型。

如果实际应用中，结点是其他类型，只要修改上面的类型定义。比如，用于描述学生信息的结点类型名 std _ node，可作如下定义：

```
typedef struct
  { char name[M]; //姓名域，M 是已定义常数
  int number,math,eng,computer,total ;//学号、数学、英语、计算机成绩和总分域
} std _ node;   //结点类型名
```

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。

数据结构的种类繁多，通常可分为表结构、树结构、图结构等几大类。

图 1-1 为常见数据结构的示意图。图中，圆圈表示结点，线条表示结点之间的关系。

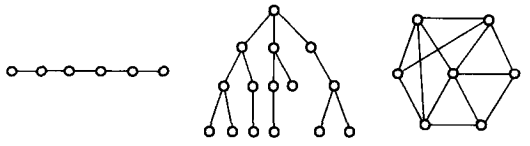


图 1-1　常见的数据结构

表结构用于描述结点之间的先后次序关系，比如学生成绩单。

树结构用于描述结点之间的层次关系、分支关系和嵌套关系，比如某部门的组织机构。

图结构用于描述结点之间复杂的多对多关系，比如城市交通图。

数据以及数据之间的关系在计算机内的存储形式叫做数据存储结构，或物理结构。相应地，上面说的数据结构是逻辑结构。

对数据结构和结点所作的处理操作统称为对这个数据结构进行的运算。例如，表结构的查找、插入、删除、排序，以及二叉树的遍历等都是常见的运算。

学习数据结构和算法的目的，就在于了解各种数据结构的特点，通常进行哪些运算，如何设计完成运算的算法，从而提高程序运行效率。

1.2　算法的描述和实现

算法是人们求解问题时所采用的方法和步骤。这里所说的“求解问题”是指为解此问题而设计出计算机程序。因为有了程序，就能让计算机“计算出”问题的答案，所以从这个意义上来说，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（比如用 C 语言编写的程序）又称为算法的实现形式，是算法的最终形式。

算法的描述形式是向人们介绍算法时使用的一种形式，或者说是算法的记述形式。有多种描述形式，常用的有文字叙述形式，框图（即流程图）形式，以及类语言（又称伪代码）形式。

文字叙述形式简单直观，易于理解。

下面通过例子，简略地介绍算法各种描述形式的用法和特点。

【例 1-1】　排序问题。将数组 a[n]的元素重新排列，使之呈由小到大的排列形式。

求解排序问题的方法很多，其中有一种简单的选择排序算法。用文字叙述如下：

先从 a 的 n 个元素中选择一个最小元素,将它换到 a[0]位置;
再从剩下的 n－1 个元素中选择一个最小元素,将它换到 a[1]位置;
再从剩下的 n－2 个元素中选择一个最小元素,将它换到 a[2]位置;
……
反复进行上述选择和交换工作,直到选出最后一个元素,就完成了排序。

这样的描述形式简单易懂,即使"外行"也能看懂。但是这种描述形式与算法最终形式(程序)相差太远,文字叙述又不太"规范",用来描述复杂算法就显得不够精确。

将上述算法,写成不精确的 C 语言程序段(伪代码):

```
for (i=0;i<n-1;i++)   //注意,只要循环 n-1 遍就行了
  {
   从 a[i]到 a[n-1]选出最小元素 a[j];//语句 1
   交换 a[i]与 a[j]的值;  //语句 2
  }
```

显然,这种伪代码形式与最终的程序更接近。懂得 C 语言的读者更喜欢这种形式。

如果需要进一步细化,可将语句 1 和语句 2 改写成 C 语言程序段:

```
for(j=i,k=i+1;k<n;k++) if(a[k]<a[j])  j=k;  //选出最小元素 a[j]
t=a[i];  a[i]=a[j];  a[j]=t; //将最小元素换到 a[i]处
```

实现该算法的 C 语言程序段为:

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

只要按 C 语言要求,加上变量定义和说明,配上输入输出语句,就是一个完整的程序。

```
#include <stdio.h>
#define N 12
void main( )
{ int i,j,k,t,a[N];
  for(i=0;i<N;i++) scanf("%d",&a[i]);//输入数据
  for(i=0;i<N-1;i++)
   {
      for(j=i,k=i+1;k<N;k++) if(a[k]<a[j])j=k;
      t=a[i]; a[i]=a[j]; a[j]=t;
   }
  for(i=0;i<N;i++) printf("%5d",a[i]); //输出排序结果
   printf("\n");
 }
```

不过,完整的程序太长,里面有很多与算法本身关系不大,却又必不可少的内容,读起来不

方便。

为了突出算法的主体，将实现算法的程序写成下面的函数形式。

算法 1-1 简单选择排序算法。

```
void selection_sort(int a[ ],int n )
// a[ ]是待排序数组，n 是数组长度
  {int i,j,k,t; //函数的局部量
   for(i=0;i<n-1;i++)
    {
      for(j=i,k=i+1;k<n;k++) if(a[k]<a[j]) j=k;
      t=a[i]; a[i]=a[j]; a[j]=t;
    }
  }
```

主调语句形如：

```
selection_sort(a, n );
```

上机实验时，配齐必要的说明和主函数，就可以了。

按照这种做法，上面的例子对应的完整程序如下，其中，主函数完成数据的输入输出、调用选择排序函数等工作。

```
#include <stdio.h>
#define N 12
void selection_sort(int a[ ],int n); //函数声明
void main( ) //主函数
{ int i,a[N];
  for(i=0;i<N;i++) scanf("%d",&a[i]);
  selection_sort(a,N);  //函数调用
  for(i=0;i<N;i++) printf("%5d",a[i]);
   printf("\n");
}
void selection_sort(int a[ ],int n) //选择排序函数
//a 是待排序数组，n 是数组长度
  {int i,j,k,t;  //函数的局部量
    for(i=0;i<n-1;i++)
      {
        for(j=i,k=i+1;k<n;k++) if(a[k]<a[j]) j=k;
        t=a[i]; a[i]=a[j]; a[j]=t;
      }
  }
```

关于本书算法的描述形式作如下说明。

采用函数形式描述算法，突出了算法的主体，也符合程序设计的常规要求。

本书将陆续介绍很多算法，除少数算法只给出文字叙述形式外，其余的都在文字叙述的基础上，用 C/C++ 语言的函数形式给出，而不给出完整的程序，这样，可以消除冗长的程序干扰

读者对算法主体的理解。

这里“C/C++语言”指的是以C语言为主，并借鉴了C++的两种“好用的”语法成分：

(1) 使用了“行注释”，C语言只允许使用段注释，不允许使用行注释。

行注释是以双斜杠“//”开头，到本行末的一段内容，全部是注释内容；而段注释，则是以“/*”开头，“*/”结尾的中间一段内容作为注释内容。

采用行注释完全是为了录入方便，并不增加程序的功能。

(2) 有的C/C++函数(主要是要求返回多个计算结果值的函数)中使用了“引用型”的形式参数，而C语言却不允许使用这样的形式参数。

采用引用型的形式参数，可以简化语句结构。

至于“引用”的含义、引用型的形式参数用法，请参考附录B。

由于上面的程序没有出现引用型的形式参数，所以，只需将行注释都改为段注释，它就是一个标准的C语言程序(当然，也可以去掉所有的注释)，例如，将(上面程序)其中的三行内容

```
void selection_sort(int a[ ],int n) //选择排序函数
//a是待排序数组,n是数组长度
  {int i,j,k,t;   //函数的局部量
```

改为

```
void selection_sort(int a[ ],int n) /*选择排序函数*/
/* a是待排序数组,n是数组长度*/
 {int i,j,k,t; /*函数的局部量*/
```

至于如何修改含有引用型的形式参数的程序，请参考后面的章节，或附录B和附录D的有关部分。

为了使算法更加完整，也使初学者进一步加深对算法的理解，帮助读者掌握上机调试算法的基本方法，本书为每个算法都配备了完整的、可直接运行的配套程序，并分别放在各章正文内容的后面，供查阅参考。

算法的另一种描述形式是流程图(框图)。读者也许对流程图的画法并不陌生，本书不再介绍流程图所用符号的含义和基本画法。下面给出一个用流程图描述算法的例子。

【例1-2】 查找问题。在已排序(升序)的数组a[n]中，查找指定元素x。

问题进一步叙述为：判断数组a中是否有一个元素值等于x，如果有，则输出该元素的下标i；如果没有，则输出一个无效下标“NOTFOUND”。这里NOTFOUND是程序前面定义的值为-1的常数。即定义为：

```
#define NOTFOUND -1
```

求解查找问题有多种算法，其中对有序数组的查找有一个速度非常快的二分查找(binary search)算法。

图1-2是二分查找算法的流程图，相信读者能够看懂这个图。

用流程图形式描述算法，具有结构清晰的特点，但是画起来比较麻烦。此外，为了进一步理解算法，通常还要加文字说明。

二分查找算法的文字叙述如下：

开始查找时，执行：

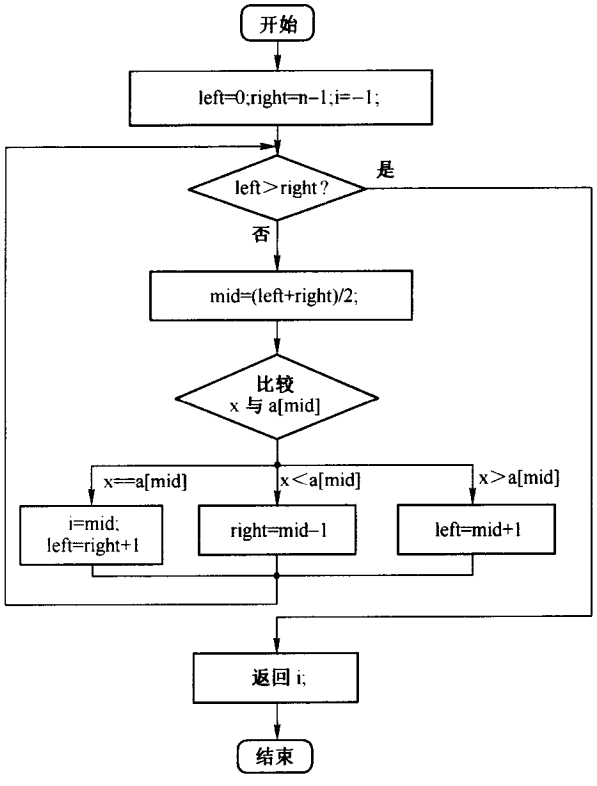


图 1-2　二分查找算法的流程图

1）置 left 和 right 的初值：left = 0；right = n - 1；

2）如果 left > right，则查找不成功（即数组 a 中没有 x），输出“ - 1”，算法终止。否则继续执行；

3）计算中值下标 mid = (left + right)/2；

4）比较 x 与 a[mid]的值；

5）如果 x == a[mid]，则找到 x，输出下标 mid，算法终止；否则继续执行；

6）如果 x < a[mid]，则调整指针 right，使 right = mid - 1，返回第 2 步，继续查找；

7）如果 x > a[mid]，则调整指针 left，使 left = mid + 1，返回第 2 步，继续查找。

不难写出实现二分查找算法的程序（函数形式）。

算法 1-2　有序表的二分查找算法。

```
int binary_search(int a[ ], int x, int n)
//参数 a[ ]是长度为 n 的有序表(升序)数组,x 是待查元素值
//若 x 在数组 a 中,则返回值 x 所在的下标;
//若 x 不在数组 a 中,则返回无效值 NOTFOUND(已定义的常数)
  { int left , right ,mid;
1.    left = 0; right = n - 1; //left 和 right 分别是待查段的左右端下标
2.    while(left <= right) // left≤right 表示待查段不空
3.      { mid = (left + right)/2; //计算中值下标 mid
```

```
4.          if(x==a[mid]) return(mid); // 已找到,返回下标 mid
5.          if(x<a[mid])right=mid-1; // 准备查前半段
6.           else left=mid+1; //准备查后半段
        }
7.      return(NOTFOUND); //没找到 x,返回无效下标
    }
```

主调语句形如:

```
binary_search(a,x,n);
```

说明:算法中,语句左侧的 1,2,……等,不是程序的一部分,而是为了便于讲解算法,给语句加的编号(后文中的算法,也作如此处理),上机运行时,不能录入这些编号。

下面,结合前面给出的流程图和文字叙述,详细讲解函数 binary_search 各条语句的含义和功能,然后再给出具体的查找示例。

为了讲述方便,这里(包括后文)把用于记录数组下标的变量 left、right 和 mid 等称为指针。虽然它们的类型是 int(不是指针类型),但是起着指针的作用。

数组中,需要查找的范围称为待查段,用一对指针(下标)指示待查段。left 是待查段的左指针,right 是待查段的右指针。mid 是中值指针,指示本次查找的测试点。

待查段的长度(元素个数)等于 right-left+1。

如果 left>right,表明待查段长度为 0(待查段为空段)。

语句 1 对应文字叙述的第 1 步,给左右指针 left 和 right 定初值。开始时,左指针 left=0,右指针 right=n-1,表示待查段就是整个数组。

语句 2(循环语句)和语句 7 对应文字叙述的第 2 步(循环控制)。

语句 3 对应文字叙述的第 2 步,计算中值下标。

语句 4 对应文字叙述的第 4 步和 5 步。如果中值元素 a[mid]与 x 相等,表明已经查到 x,返回 mid,函数调用结束。

语句 5 对应文字叙述的第 6 步,如果 x<a[mid],表明 x 不可能在 mid 至 right 的范围内,所以,要将待查段的右指针 right,调整到中值指针 mid 的左侧(right=mid-1),从而缩小待查段的长度。

语句 6 对应文字叙述的第 7 步。因为语句 4 和语句 5 两个条件都不成立(x 既不等于 a[mid],也不小于 a[mid]),必然是 x>a[mid],这表明 x 不可能在 left 至 mid 的范围内,所以,要将待查段的左指针 left 调整到中值指针 mid 的左侧(left=mid+1),缩小待查段的长度。

这样,语句 2 的循环体每执行一次,或者在语句 4 处找到 x(结束调用,返回),或者在语句 5(或语句 6)处,缩小待查段的长度,进入下一轮循环。

当 left>right 时,待查段变为空,循环终止。因为语句 2 的循环体中没有执行到返回语句,表明 x 不在数组中。

接着执行语句 7,返回无效下标 NOTFOUND。

例如,图 1-3 所示的有序数组 a[n],这里 n 等于 12。

假如现在要查找 x=21,查找步骤如下:

开始时,left=0,right=11,计算中值下标:mid=(0+11)/2=5,如图 1-3a。

第一次比较:x<a[5](21<43),置 right=mid-1=4,如图 1-3b,计算中值下标:mid=(0