高等学校计算机基础课程教材

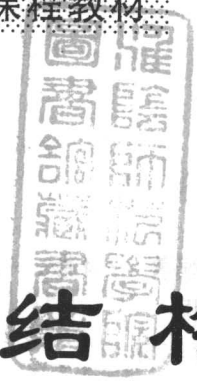
数据结构基础（第四版）

曹桂琴／编著

大连理工大学出版社

1014996

高等学校计算机基础课程教材



数据结构基础

（第四版）

浩强创作室组织编写

曹桂琴　编著

大连理工大学出版社

© 曹桂琴 2002

图书在版编目(CIP)数据

数据结构基础 / 曹桂琴编著 .—4 版. — 大连:大连理工大学出版社, 2002.3(2003.1 重印)
高等学校计算机基础课程教材
ISBN 7-5611-0966-0

Ⅰ. 数… Ⅱ. 曹… Ⅲ. 数据结构—高等学校—教材
Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字(2002)第 015250 号

大连理工大学出版社出版
地址:大连市凌水河 邮政编码:116024
电话:0411-4708842 传真:0411-4701466 邮购:0411-4707955
E-mail:dutp@mail.dlptt.ln.cn URL:http://www.dutp.com.cn
大连理工印刷有限公司印刷 大连理工大学出版社发行

幅面尺寸:185mm × 260mm 印张:13.25 字数:314 千字
印数:41 001 ~ 45 000
1994 年 12 月第 1 版 2002 年 3 月第 4 版
2003 年 1 月第 9 次印刷

责任编辑:刘新彦 刘晓晶 责任校对:郭 芳
封面设计:孙宝福

定 价:15.00 元

序

我们即将进入21世纪，大家都在讨论：21世纪需要什么样的人才？大学生应当具有什么样的素质？应当有什么样的知识结构和能力结构？怎样才能实现这些要求？

有一点是肯定的：21世纪是信息时代。人们的工作、学习和思想都应当适应信息时代的要求。大学生的知识结构是随着社会的发展而不断变化的，决不是一成不变的。计算机是一门新兴的而又十分重要的学科。计算机的产生和发展，对所有领域的科学技术的发展起着巨大的推动作用。新世纪的知识分子应当站在新技术发展的前沿，掌握计算机应用技术，并且把它和自己从事的专业结合起来，有力地推动各领域科学技术及其应用的发展。

显然，目前许多高校所进行的计算机教育与日益增长的要求相比，差距是大的，难以满足新世纪的要求。近年来，许多高校进行了计算机基础教育的改革，在原有的基础上，提高了要求，增加了课程，更新了教学内容，改革了教学方法。许多学校按照以下三个层次设置计算机课程：

一、计算机公共基础；

二、计算机技术基础；

三、计算机应用课程。

不同学校不同专业根据不同需要开设了不同的课程。全国高校计算机基础教育研究会还提出了需要正确处理好的十个关系，即：

1.理论与应用的关系；

2.深度与广度的关系；

3.当前与发展的关系；

4.硬件与软件的关系；

5.跟踪先进水平与教学相对稳定的关系；

6.课内与课外的关系；

7.课程设置与统一考试的关系；

8.计算机课程与其他课程的关系；

9.要求学生动手能力强与当前设备不足的关系；

10.计算机技术迅速发展与师资现状的关系。

许多学校在以上这些方面摸索出了许多宝贵的经验。其中应当引起注意的是：计算机应用课程的教学，应该由教师传授为主逐步改变为学生自学为主，由课堂讲授为主逐步改变为上机实践为主。此外，除了规定的必修课以外，还应设若干选修课，以满足不同专业不同学生的需要。

为了推进高校计算机基础教学改革，浩强创作室组织了这套“高等学校计算机基础课程教材”，邀请高校中具有丰富教学经验的教师参加编写工作。在编写这些教材的过程中，我们力求符合非计算机专业的特点，而不是简单地将计算机专业的教材浓缩。非计算机专业的培养目标、学生特点、将来工作性质、教学内容、教学方法等各方面都和计算机专业有很大的不同。二十年的经验表明：照搬计算机专业的一套是不成功的。非计算机专业的计算机教材必须有自己的体系，根据需要精选内容，叙述方法必须做到由浅入深通俗易懂。

编写这套丛书遵循的方针是：内容新颖、概念清晰、实用性强、通俗易懂、层次配套。我们在确定丛书中各教材的选题时，充分考虑到大多数高校当前和今后一个时期所需要开设的课程。各校情况不同，有的可作为必修课，有的可作为选修课，也有的可以暂时不学。随着计算机科学技术的发展和教学改革的深入，今后还将陆续增加新的教材。各本书的内容也将不断修改补充，以跟上计算机技术的发展。

本丛书由浩强创作室策划并组织编写。参加本丛书策划、讨论和编写的有：谭浩强、李盘林、刘晓晶、朱桂兰、薛淑斌等。

希望本丛书的出版能推动高校计算机基础教育的发展。

全国高校计算机基础教育研究会理事长

谭浩强

1999.1

此为试读，需要完整PDF请访问：www.ertongbook.com

前　言

“数据结构”是计算机学科的核心课程，这门课程主要研究如何合理地组织数据；怎样在计算机中有效地表示数据和处理数据。通过对这门课程的学习可增强选择合适的数据结构与编写高效的程序的能力，因而这门课程还被不同层次的非计算机类专业学生作为选修课和辅修专业的学生必修课。

书中第一章综述了数据结构的基本概念及算法分析初步；第二章至第七章分别讨论了线性表、栈、队列、数组、广义表、树、二叉树、图、串和集合等常用的数据结构，包括数据的逻辑结构、存储结构及有关运算。第八章和第九章讨论了在数据处理中常用的查找和排序的各种方法和算法；第十章介绍了常用的文件组织方法。

本书的第四版在对全书内容进行修订的基础上，增加了有关算法分析和设计方面的内容。

全书的选材注重于实际应用，略去一些理论推导和证明；采用通俗易懂的语言描述各种数据结构的定义；采用类 C 语言来描述数据结构和算法，尽量考虑 C 语言的特点。其中的算法只要稍加修改就可变成能上机执行的 C 语言程序，这样不仅使算法清晰，而且还能给学生提供数据结构在计算机中的表示方法及运算的具体实现方法。读者只需掌握 C 语言程序设计方法就可以学习本书。该书可作为非计算机类各专业选修课的教材，讲授时间可为 36 学时至 60 学时。

大连理工大学计算机系的许宏和张华两位老师调试过书中部分算法。大连轻工业学院的郭芳老师参加了本书的修订工作。

由于作者水平有限，必有许多不足之处，欢迎读者批评指正。

曹桂琴

2002 年 3 月

目　录

第一章 绪论

随着计算机软、硬件技术飞速发展，计算机应用领域越来越广泛，计算机处理的对象由简单的数和字符发展到具有不同结构关系的数据。为了更好地进行程序设计，就需要深入地研究如何合理地组织被处理的数据，这就是“数据结构”这门学科形成和发展的背景。

1.1 基本概念和术语

为了以后学习方便，我们首先介绍一些基本概念和术语。

数据是计算机化的信息。它是对现实世界的事物采用计算机能够识别、存储和处理方式进行的描述。例如：整数、字符、声音、图像等都是数据。因为它们都可以输入机器并加以处理。

数据元素是数据的基本单位，即数据集合中的个体。有些情况下也把数据元素称做结点或记录等。一个数据元素可由一个或多个数据项组成。数据项是有独立含义的数据最小可命名单位。有时也把数据项称做域、字段等。例如：在学生档案管理系统中，可以把一个与学生有关的信息作为一个数据元素，它由学号、姓名、年龄等数据项组成。

数据结构是相互之间存在一种或多种特定关系的数据元素的集合。数据元素之间的相互关系称为结构。例如：全校学生档案的组织方式如图 1-1 所示。这种形式实际上是一棵自顶向下生长的树，像这样按分支和层次组织的数据称为“树”结构。

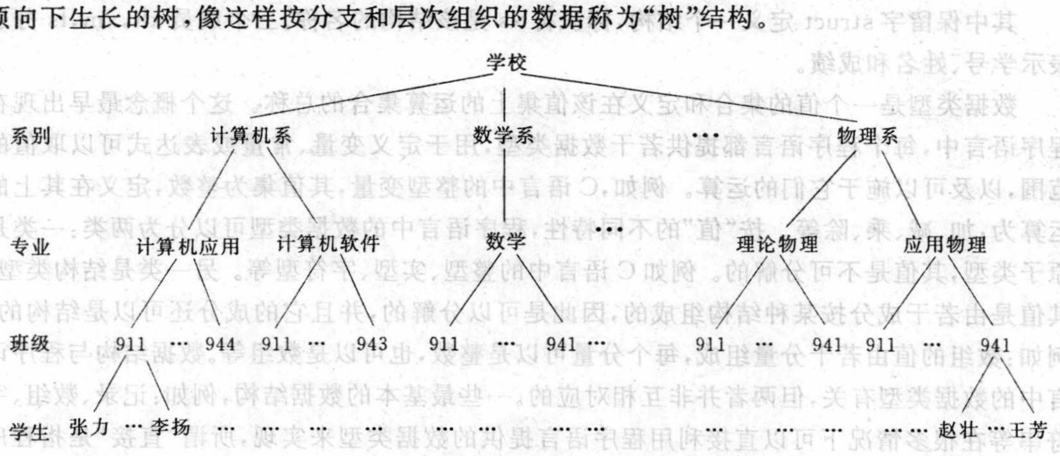


图 1-1 全校学生档案管理的组织方式

数据的逻辑结构只抽象地描述数据元素间的逻辑关系，而不管其在计算机中的存储表示方式。数据的逻辑结构分为线性结构和非线性结构。若各数据元素之间的逻辑关系可以用一个线性序列简单地表示出来，则称之为线性结构，否则称为非线性结构。

此为试读，需要完整PDF请访问：www.ertongbook.com

线性结构的特点是：在一个数据元素的非空有限集中，仅有一个被称为“第一个”的数据元素；仅有一个被称为“最后一个”的数据元素；除第一个数据元素外，其余所有的数据元素有且仅有一个直接前驱元素；除最后一个数据元素外，其余所有的数据元素有且仅有一个直接后继元素。后面章节要讲的线性表、栈和队等是线性结构。

非线性结构则不具备上述特点。如后面章节要讲的树、图等均是非线性结构。

数据的逻辑结构形式上用一个二元组表示 $S=(D,R)$，其中：D 是数据元素的有限集，R 是 D 上的关系的有限集合。例如：$D=\{a,b,c,d\}$，$R=\{\langle a,b\rangle,\langle b,c\rangle,\langle c,d\rangle\}$。关系 R 中的有序对 $\langle x,y\rangle$ 表示 x 是 y 的直接前驱，y 是 x 的直接后继。在这个结构中，a 是第一个数据元素（无直接前驱），除 a 外每一个结点有且仅有一个直接前驱，d 是最后一个数据元素（无直接后继），除 d 外每一个数据元素有且仅有一个直接后继，因此该例是具有 4 个数据元素的线性表。

数据的物理结构是数据的逻辑结构在计算机存储器里的实现。数据的物理结构也称为存储结构。为全面地表示一个逻辑结构，它在存储器中的实现应包括数据元素自身值的表示和数据元素之间的关系的表示两个方面。因此，存储在计算机中的数据结构，其结点的各域按性质可分成两大类：一类是存放自身值的域，通常称为数据域；另一类是存放该结点与其他结点的关系的域，通常称为链域。一种数据的逻辑结构可以选用不同的存储结构。本书是在高级程序语言层次上来讨论存储结构，需用高级程序语言中的数据类型来描述这种实现。例如：可用 C 语言中的结构类型描述具有多于一个域的结点。如在学生单科成绩表中，结点含：学号、姓名、分数三个数据域。可用如下的结构说明。

```
struct xs {
        int xh;
        char xm[10];
        int fs;
        };
```

其中保留字 struct 定义一个结构，标识符 xs 是该结构的名称。三个成员 xh，xm，fs 分别表示学号、姓名和成绩。

数据类型是一个值的集合和定义在该值集上的运算集合的总称。这个概念最早出现在程序语言中，每个程序语言都提供若干数据类型，用于定义变量、常量或表达式可以取值的范围，以及可以施于它们的运算。例如，C 语言中的整型变量，其值集为整数，定义在其上的运算为：加、减、乘、除等。按“值”的不同特性，程序语言中的数据类型可以分为两类：一类是原子类型，其值是不可分解的。例如 C 语言中的整型、实型、字符型等。另一类是结构类型，其值是由若干成分按某种结构组成的，因此是可以分解的，并且它的成分还可以是结构的。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。数据结构与程序语言中的数据类型有关，但两者并非互相对应的。一些最基本的数据结构，例如：记录、数组、字符串等在很多情况下可以直接利用程序语言提供的数据类型来实现，所谓“直接”是指程序语言本身提供了对这些结构的描述手段和对它们的操作。例如：从一个记录值中取一个成分值，但有许多常用的数据结构，例如：栈、队列、链表等，在很多程序语言中并没有相应的数据类型，需采用程序语言中提供的基本数据类型和供程序员构造结构化数据类型方法作为工具实现相应的数据结构。

抽象数据类型是指抽象数据的组织和与之相关的操作。它可以看作是数据的逻辑结构及其定义在逻辑结构上的操作。抽象数据类型也可以看成是描述问题的模型，一个数学模型及定义在该模型上的一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关，即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

算法是解决某一特定类型问题的有具体步骤的方法。一个算法应该具有下列特性：

(1)有穷性。一个算法必须是在执行有限步之后结束。

(2)确定性。算法的每一步必须是确切地定义的，无二义性。对于每种情况，有待执行的运算必须被严格地和清楚地规定。

(3)可行性。算法应该是可行的，这意味着算法中描述的运算都是相当基本的，它们都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 0 个或多个输入。它们是在算法开始前对算法给定的量。这些输入取自于特定的对象的集合。

(5)输出。一个算法有一个或多个输出。它们是同输入有某种特定关系的量。

算法的设计可以避开具体的计算机程序设计语言，但算法的实现必须借助程序语言中提供的数据类型及其运算。数据结构与算法是相辅相成的，它们是利用计算机解决实际问题时不可缺少的两个方面。

数据的运算是定义在数据的逻辑结构上的，但运算的具体实现要在存储结构上进行。每一种数据的逻辑结构均有相应的一组运算。常用的运算有查找、插入、删除、更新和排序等。数据的运算是数据结构的一个重要方面。本书在研究每一种数据结构时，除了研究数据的逻辑结构和存储结构外，还讨论在该结构上的数据运算及主要运算的实现算法。

1.2 算法的描述

本书采用类 C 语言描述算法，为了简明易懂且具有实用价值，在描述算法时做如下约定：

(1)每一个算法以函数形式给出，一般形式是

```
函数类型  函数名(参数表);
{算法说明
语句序列}
```

其中，参数表中列出的参数需要说明类型。例如：

```
int max (int x,int y)
{...}
```

由于 C 语言规定，函数调用时实参为形参设置初值，当函数的参数为非数组型变量时，形参值的变化不影响实参的值，所以若想使被调用函数直接改变非数组型实参变量值，需把变量的地址作为实参，在被调用的函数中相应的形参用指针类型。例如：设形参 p 是指针变量，把变量 a 的地址(&a)作为实参，在调用开始时，实参为形参设初值，形参 p 的值为 &a，如图 1-2 所示，然后在被调用函数中改变 p 所指向的变量(*p)值时，调用程序中变量 a 的值也就改变了。

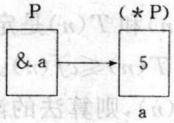


图 1-2

(2)数据结构的表示(存储结构)用类型定义描述,该类型定义、全局变量的说明等均在算法之前进行说明。

(3)为了算法描述清晰,本书约定算法中 NULL 表示"空指针",用 M 表示预定的数组元素的个数。若无预先定义就用 M 时,约定 M 值定义为:#define M 1000。

(4)本书的主要算法都在 Turbo C 2.0 版本下调试通过。一般情况下可作为函数在程序中调用。为了便于初学者阅读,加上作者水平有限,书中的算法可能不是最优的。

一种数据结构的优劣是由实现其各种运算的算法体现的。对数据结构的分析实质上也就是对实现其各种运算的算法的分析。

1.3 算法分析基础知识

1.3.1 概述

一种数据结构的优劣是由实现其各种运算的算法来体现的,对数据结构的分析实质是对实现其各种运算的算法的性能的分析。判断一个算法的好坏,主要有以下几个标准:

(1)正确性:算法应当实现具体问题需求的功能和性能。

(2)可读性:算法应当便于阅读,以利于理解与修改。如:算法的结构清晰并加入注释,简要说明主要参数的使用规则、各程序段完成的功能等。

(3)健壮性:要求算法具有查错和处理功能。如对输入非法数据或执行过程中出现的异常状态进行检测、报错和纠正错误。

(4)效率:算法的效率主要指算法运行时所需要的计算机资源的多少,包括运行时间和存储空间的消耗,称为算法的时间代价和空间代价。

算法分析是对一种算法所消耗的计算机资源的估算,算法设计者可以据此对解决同一个问题的多种算法的代价进行比较,还可以判断一种算法在实现时是否会遇到资源限制的问题。

通常,确定算法效率的一种方法是:先把算法用某种高级程序设计语言编写成程序,并在特定的计算机上输入选定的数据运行,测量实际运行时间。但是实测的运行时间依赖于运行时的机型、操作环境和编译程序的质量等。换句话说,实测的运行时间反映运行该算法的计算机系统的综合效率,而不单纯反映算法的效率。

另一种方法是在不实际运行该算法的前提下,估算当问题规模扩大时,算法的时间代价和空间代价的增长率。实用结果表明,该方法是很有效的,可用于确定某种算法是否值得实现。

设 $T(n)$是问题规模 n 的时间函数,采用"大 O 表示法"表示算法代价增长率的上界,设 $f(n)$和 $T(n)$是定义在正数集合上,当且仅当存在正整数 c 和 n_0,使得对所有的 $n \geqslant n_0$,都满足 $T(n) \leqslant cf(n)$。就是说,当 n 充分大时,随 n 的增加,如果函数 $T(n)$存在一个增长的上界 $cf(n)$,则算法的渐近时间复杂度为 $O(f(n))$,本书简称为时间复杂度,算法时间复杂度的渐近阶为 $f(n)$,记为 $T(n)=O(f(n))$。

设 $S(n)$是问题规模 n 的空间函数,类似于算法的时间复杂度,可以定义算法的渐近空间复杂度为 $O(f(n))$,记为 $S(n)=O(f(n))$。

例 1-1 设 $T(n)=3n^3+n^2$，证明 $T(n)=O(n^3)$。

因为存在 $n_0=1, c=4$，当 $n\geqslant 1$ 时，有 $3n^3+n^2\leqslant 4n^3$，所以 $T(n)=O(n^3)$。由此可见，在例 1-1 中，当 n 充分大时，n^2 的数值常常不起决定影响，可以忽略不计，因此，用大 O 表示法，对于多项式，我们只保留最高次幂的项，常数系数和低阶项可以省略。

算法复杂度包括时间复杂度和空间复杂度，较常见的算法复杂度有：$O(1)$（常量阶）；$O(n)$，$O(n^2)$，$O(n^k)$（多项式阶）；$O(\log_2 n)$，$O(n\log_2 n)$（对数阶）；$O(2^n)$，$O(e^n)$（指数阶）。所有的非 0 正常数都属于常量阶。当我们设计算法时，应设计算法复杂度尽可能低的算法；当解决同一个问题已有多种算法时，应尽量选择算法复杂度最低者。因此，算法分析对算法设计或选用有一定的参考价值。

1.3.2 算法时间复杂度的计算

撇开具体的计算机系统，注重分析算法执行时间的增长率，可以从描述算法的高级语言程序来分析算法时间复杂度。通常，分析程序的时间复杂度是分块进行的，先按某个时间单位求出各语句、程序段的运行时间，再计算整个程序的运行时间，最后用大 O 表示法的运算规则分析算法的时间复杂度。

大 O 表示法的主要运算规则有：

(1) $O(f(n))+O(g(n))=O(\max(f(n),g(n)))$

(2) $O(f(n))*O(g(n))=O(f(n)*g(n))$

(3) $O(cf(n))=O(f(n))$，其中 c 是任一正的常数。

由于本书是用类 C 语言来描述算法，C 程序是由一系列的语句组成，该算法执行所需的时间等于每一个语句执行所需的时间之和，如果每一种语句所需要的时间都给出计量规则单位，则程序执行所需时间的计量便只是一个求代数和问题。

我们把计算不含函数调用的表达式值的时间约定为一个时间单位，E 表示〈表达式〉，S 表示〈语句〉，$T(P)$ 表示执行 P 所需时间，则几种基本可执行语句 S 的执行时间计量规则如下：

(1) 赋值语句所需要的时间是：$T(S)=T(E)$，$T(E)$为计算表达式值的时间，如果表达式中含有函数调用，还需要加上函数调用过程消耗的时间。

(2) 条件语句：

if (E) S_1;

else S_2;

需要的时间是：$T(S)=T(E)+\max(T(S_1),T(S_2))$

其中：$T(E)$为测算条件表达式值的时间；

$T(S_1)$ 和 $T(S_2)$分别为计算语句 S_1 和 S_2 的时间。

(3) switch 语句：

设 S 是如下语句：

```
switch(E){
  case E1:S1;
  case E2:S2;
    ...
```

case E_m:S_m;

default:S_{m+1}}

则 $T(S)=T(E)+\sum_{i=1}^{m}E_i+\max(T(S_1),T(S_2),\ldots,T(S_m),T(S_{m+1}))$

(4) 循环语句:

设 n 为重复执行循环体的次数,则循环语句所需时间等于 n 次执行循环体所消耗的时间的总和。每次重复所消耗的时间包括:循环控制时间和循环体执行一次所需时间。不同类型的循环语句的循环控制时间计算方法略有不同。当循环中没有明显给出循环次数时,可以在循环体中找出与循环终止条件相关的变量,通过计算求出循环次数。遇到多层循环时,要由内层向外层逐层分析。当分析外层循环时,作为循环体的内层循环所需时间应该是已知的。

C 语言的循环语句有三种形式:

●for 语句

for(S_E;E_1;E_2) S;

通常,for 语句控制部分第一次执行时,循环控制时间为初始化赋值语句的时间加上测试循环终止条件的时间($T(S_E)+T(E_1)$),即 2 个单位时间;后续执行时,循环控制时间为计算循环参数(E_2)的时间加上测试循环终止条件(E_1)的时间($T(E_2)+T(E_1)$),也是 2 个单位时间。由于具体计算时,每一次重复执行循环体的所需时间可能不同,所以需逐次求 n 次执行循环体所消耗的时间的总和。这里 $T(S_i)$表示第 i 次重复执行循环体的所需时间。

for 语句所需时间是:

$$T(S)=2n+\sum_{i=1}^{n}T(S_i)$$

●while 与 do 语句

while (E) S;

do S while E;

两个语句的循环控制时间均为 T(E),语句所需时间是:

$$T(S)=T(E)+\sum_{i=1}^{n}T(S_i)$$

(5) 转移语句:这类语句包括 continue,break,goto,return E。它们均需要一个单位时间。其中 E 不包含函数调用。

(6) 动态存储管理语句:这类语句包括 malloc 和 free 它们均需要一个单位时间。

(7) 函数调用语句:该语句所需时间包括参数传递时间和执行函数本身所需时间。其中:一个参数传递时间需要一个单位时间。

如果算法中只有非递归调用,则可以根据函数调用的层次,从内向外计算,直到计算出最外层的时间便为所求。

若算法中含有递归调用,则可以将递归函数的执行时间设定为相应规模的待定函数,再根据函数的内涵建立这些待定函数间的递归关系,得到递归方程并解之。递归方程的种类很多,这里仅举一例说明。

例 1-2　计算 n! 的递归算法 fact(n)的执行时间。

递归算法	单位时间数
int fact(int n)	
{if(n<=1)	1
return 1;	1
else return n * fact(n-1);	1+Tfact(n-1)
}	

由算法中各语句所需要的单位时间数可计算出当 n<2 时，需要 2 个单位时间；n>1 时，需要 2+Tfact(n-1)个单位时间，建立递归方程如下：

$$\text{Tfact}(n)=\begin{cases}2 & (n=0,1)\\ 2+\text{Tfact}(n-1) & (n>1)\end{cases} \tag{1-1}$$

这是一个递推公式，通过重复代入来实现递推计算 Tfact(n)：

Tfact(n)= 2+Tfact(n-1)

Tfact(n)=2+2+Tfact(n-2)=2 * 2+Tfact(n-2)

Tfact(n)=2+2+2+Tfact(n-3)=2 * 3+Tfact(n-3)

=...

Tfact(n)=2 * (n-1)+Tfact(1)= 2 * (n-1)+2=2n

所以时间复杂度为 O(n)

下面给出几个含有循环语句的算法分析例子。

例 1-3 在含 n 个正整数的一维数组中查找最大值的算法如下：

```
语句行   int max(int *a,int n)
1        {int s=0;
2         for(i=0;i<n;i++)
3          if(a[i]>s)
4           s=a[i];
5         return s;}
```

这个问题的规模是 n，算法分析如下：

(1) 赋值语句：行 1 语句需要 1 个单位时间，时间复杂度为 O(1)。

(2) 条件语句：行 3 到行 4 语句需要 2 个单位时间。

(3) 循环语句：行 2 到行 4 语句，其中一次循环控制和执行循环体各需要 2 个单位时间，循环次数为 n，循环语句需要 $4n$ 个单位时间，根据“大 O 表示法”的运算规则(3)可得：$O(4n)=O(n)$，时间复杂度为 $O(n)$

(4) 转移语句：行 5 的 return 语句需要 1 个单位时间，时间复杂度为 O(1)。

根据加法规则，整个算法的运行时间为

$$O(1)+O(n)+O(1)=O(\max(1,n,1))=O(n)$$

例 1-4 求矩阵 a 中所有元素之和的算法如下：

```
语句行   float matrix(float **a,int n)
1        { int i,j;
2          float s=0.0;
3          for(i=0;i<n;i++)
```

此为试读，需要完整PDF请访问：www.ertongbook.com