算法与数据结构教程

李长荣 王一萍 赵硕 编著

哈尔滨地图出版社

算法与数据结构教程

SUANFA YU SHUJV JIEGOU JIAOCHENG

主编　李长荣　王一萍　赵　硕

哈尔滨地图出版社
·哈尔滨·

图书在版编目(CIP)数据

算法与数据结构教程/李长荣,王一萍,赵硕编著.
哈尔滨:哈尔滨地图出版社,2005.12
ISBN 7-80717-215-0

Ⅰ.数… Ⅱ.①李… ②王… ③赵… Ⅲ.①算法分析—教材②数据结构—教材 Ⅳ.①TP301.6②TP311.12

中国版本图书馆 CIP 数据核字(2005)第 151958 号

哈尔滨地图出版社出版、发行
(地址:哈尔滨市南岗区测绘路 2 号 邮政编码:150086)
哈尔滨太平洋彩印有限公司印刷
开本:787 mm×1 092 mm 1/16 印张:16.625 字数:426 千字
2005 年 12 月第 1 版 2005 年 12 月第 1 次印刷
印数:1～1 000 定价:32.00 元

前　言

算法与数据结构是一门计算机专业十分重要的基础课，计算机科学各领域及各种应用软件都要使用相关的数据结构和算法。

当面临一个新的问题时，设计者需要选择适当的数据结构，并设计出满足一定时间和空间限制的有效算法，本书的目标就是让读者根据问题的性质选择合理的数据结构，并对时间空间复杂度进行必要的控制，以提高学生分析问题、解决问题的能力。

在非数值计算中，处理的对象已从简单数值发展到具有一定结构的数据，这就需要讨论如何有效地组织计算机的存储，并在此基础上有效地实现对象间的运算，算法与数据结构就是研究与解决这些问题的重要基础。

在讲授数据结构课程时，深感算法部分是最难讲的，与其说学生不知如何写或设计算法，不如说学生没有解决问题的方法。如果学生在学习数据结构之前知道什么是分治策略、贪心策略或动态规划法，那么在讲解快速排序、kruskal 算法或 Floyd 算法时就好理解了。学生应该既学习数据结构知识、算法设计技术，更要培养解决问题的能力。

本书的特点是先让学生学会基本的算法设计技术，再学习数据结构，真正把这两者的知识应用于实际问题的解决中。

本书可以作为高等院校计算机专业和相近专业的教材，也可作为从事计算应用的工程技术人员的参考书。

由于作者水平有限，书中不足之处在所难免，敬请读者批评指正。

编者

2005 年 12 月

目　录

第1章 绪 论

为什么要学习数据结构？又为什么要学习算法？对于一个即将从事计算机专业的人士来说，无论从理论还是从实践的角度，学习数据结构和算法都是有必要的。从实践的角度来看，我们必须了解计算机领域中不同问题的一系列标准算法以及计算机程序加工的对象及对象之间的关系；此外，我们还要具备设计新算法和分析其效率的能力。算法是对输入数据的处理，以产生解决问题的输出的过程。我们要清楚地知道问题、算法和程序三者之间的关系及相应的技术。

1.1 本书讨论的范畴

信息的表示是计算机科学的基础。大多数计算机程序的主要目标与其说是完成运算，不如说是存储信息和尽快检索信息。因此，研究数据结构和算法就成为计算机科学的核心问题。本书的目的就是帮助读者理解怎样在计算机内组织信息，以便支持高效的数据处理。

众所周知，20 世纪 40 年代，电子数字计算机问世的直接原因是解决弹道学的计算问题。早期，电子计算机的应用范围，几乎只局限于科学和工程的计算，其处理的对象是纯数值性的信息，通常，人们把这类问题称为数值计算。

近 30 年来，电子计算机的发展异常迅猛，这不仅表现在计算机本身运算速度不断提高、信息存储量日益扩大、价格逐步下降，更重要的是计算机广泛地应用于情报检索、企业管理、系统工程等方面，已远远超出了科技计算的范围，而渗透到人类社会活动的一切领域。与此相应，计算机的处理对象也从简单的纯数值性信息发展到非数值性的和具有一定结构的信息。

因此，再把电子数字计算机简单地看作是进行数值计算的工具，把数据仅理解为纯数值性的信息，就显得太狭隘了。现代计算机科学的观点，是把计算机程序处理的一切数值的、非数值的信息，乃至程序统称为数据(Data)，而电子计算机则是加工处理数据(信息)的工具。

由于数据的表示方法和组织形式直接关系到程序对数据的处理效率，而系统程序和许多应用程序的规模很大，结构相当复杂，处理对象又多为非数值性数据。因此，单凭程序设计人员的经验和技巧已难以设计出效率高、可靠性强的程序。于是，就要求人们对计算机程序加工的对象进行系统的研究，即研究数据的特性以及数据之间的关系——数据结构(Data Structure)。

“数据结构”作为一门独立的课程在国外是从 1968 年才开始设立的。1968 年美国 Donald E. Knuth 教授开创了数据结构的最初体系，他所著的《计算机程序设计艺术》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构、存储结构及其操作(算法)的著作。

计算机解决一个具体问题时，大致需要经过下列几个步骤：首先要从具体问题中抽象出一个适当的数学模型，然后设计一个解此数学模型的算法(Algorithm)，最后编出程序、进行测试、调整直至得到最终解答。寻求数学模型的实质是分析问题，从中提取操作的对象，并找出这些操作对象之间含有的关系，然后用数学的语言加以描述。

计算机算法与数据的结构密切相关，算法无不依附于具体的数据结构，数据结构直接关系到算法的选择和效率。运算是由计算机来完成的，这就要设计相应的插入、删除和修改的算法。因此，算法与数据结构主要研究以下四个方面的内容：

- 确定所研究问题的数据对象及数据对象中数据元素之间的逻辑关系。
- 确定必须支持的基本操作，并度量每种操作所受的资源限制，资源限制包括可用来存储数据的全部空间（内存和外存）限制和允许执行每一个子任务所需要的时间。基本操作包括向数据对象中插入一个数据元素、从数据对象中删除一个数据元素和查找指定的数据元素。
- 确定物理结构。物理结构是对数据对象及数据元素之间逻辑关系的存储映像。
- 确定解决问题的算法，并对算法进行复杂度分析。

1.2 抽象数据类型和数据结构

1.2.1 基本概念和术语

集合(set)是若干具有共同可辨的事物的“聚合”，其中每个事物称为集合元素或成员。例如，图 1.1 中一个个人书库中所有的书将构成一个图书集合。

个人

登录号	书号	书名	作者	出版社	价格
000001	TP2233	Windows NT4.0中文版教程	赵健雅	电子工业	28.00
000002	TP1844	Authorware 5.1速成	孙强	人民邮电	40.00
000003	TP1684	Lotus Notes网络办公平台	赵丽萍	清华大学	16.00
000004	TP2143	Access 2000入门与提高	张堪	清华大学	22.00
000005	TP1110	PowerBuilder 6.5实用教程	樊金生	科技大学	29.00
000006	TP1397	Delphi数据库编程技术	刘前进	人民邮电	43.00
000007	TP2711	精通MS SQL Server 7.0	罗会涛	电子工业	35.00
000008	TP3239	Visual C++实用教程	郑阿奇	电子工业	30.00
000009	TP1787	电子商务万事通	赵乃真	人民邮电	26.00
000010	TP42	数据结构	江涛	中央电大	18.80

图 1.1 个人书库表

数据(data)是对客观信息的一种描述，它是由能被计算机识别与处理的数值、字符等符号构成的集合。在计算机科学中，数据的含义非常广泛，我们把一切能够输入到计算机中并被计算机程序处理的信息，包括文字、表格、图像等，都称为数据。例如，一个个人书库管理程序所要处理的数据可能是一张图 1.1 所示的表格。

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。数据元素可以是不可分割的“原子”，也可以由若干款项组成。数据元素有时也称为结点或记录。

例如，在图 1.1 所示的个人书库中，为了便于处理，把其中的每一行（代表一本书）作为一个基本单位来考虑，故该数据由 10 个数据元素组成。

一般情况下，一个数据元素中含有若干个字段（也叫**数据项**）。例如，在图 1.1 所示的表格数据中，每个数据元素都由登录号、书号、书名、作者、出版社和价格等六个字段构成。数据项是构成数据的最小单位。

关键码(key)指的是数据元素中能起标识作用的数据项。其中能起惟一标识作用的关键码称为"主关键字(简称主码)";反之称为"次关键码(简称次码)"。通常一个数据元素只有一个主码,但是可以有多个次码。例如,图 1.1 中书号可作为主码,而书名或作者可作为次码。

关系(Relation)指的是集合中元素之间的某种相关性。在图 1.1 所示的表格数据中,各数据元素之间在逻辑上有一种线性关系,它指出了 10 本书在表中的排列顺序。根据这种线性关系,可以看出表中第一本书是什么书,第二本书是什么书,等等。

数据处理(data process)是指对数据进行查找、插入、删除、合并、排序、统计以及简单计算等的操作过程。在早期,计算机主要用于科学和工程计算,进入 20 世纪 80 年代以后,计算机主要用于数据处理。据有关统计资料表明,现在计算机用于数据处理的时间比例达到 80%以上,随着时间的推移和计算机应用的进一步普及,计算机用于数据处理的时间比例必将进一步增大。

类型(type)是一组值的集合。例如,布尔类型由 ture 和 false 这两个值组成。整数也构成类型。整数是一种简单类型,因为它的值不含子结构。一个银行账户记录一般含有多项信息,如姓名、地址、账号和余额,该记录是复杂类型或组合类型的一个例子。

数据类型(data type)是指一种类型和定义在该类型上的一组操作。例如,一个整数变量是整数数据类型的一个成员,而加法是定义在整数数据类型上的一种操作。

数据类型的逻辑概念与其在计算机程序中的实现有很重要的区别。例如,线性表数据类型有两种传统的实现方法:链表和顺序表(基于数组的线性表)。因此,可以选择链表或者顺序表来实现线性表数据类型。但是,术语"数组"在计算机程序设计中常用于指一块连续的内存空间,每一个内存空间存储一个固定长度的数据元素。从这个意义上讲,数组是一个物理数据结构。然而,数组也能够表示一种由一组属性相同的数据元素组成的逻辑数据类型,每个数据元素由一特定的索引号(即数组中的下标)来标识。这样看来,可以采用多种不同的方法来实现数组。例如,第六章描述了用来实现一个稀疏矩阵的数据结构,其实现就与占用连续内存空间的传统数组大不相同。

1.2.2 数据结构(Data Structures)

若在特性相同的数据元素集合中的数据元素之间存在一种或多种特定的关系,则称该数据元素的集合为"数据结构"。数据结构常指存储在计算机内存中的数据。与其相关的术语"文件结构"常指外存储器(如磁盘、磁带)中数据的组织。

数据结构包括逻辑结构和物理结构(存储结构)两个层次。

按照数据元素之间存在的逻辑关系的不同数学特性,通常有下列 4 种数据结构:(1)**线性结构**;(2)**树形结构**;(3)**图状或网状结构**;(4)**纯集合结构**。如图 1.2 所示:

数据结构(逻辑结构)的形式定义为:数据结构是一个二元组

$$DS = (D,S) \tag{1-1}$$

其中:D 是相同属性的数据元素的有限集,也称作数据对象,S 是 D 上关系的有限集。

下面举例说明。

例 1-1 在计算机科学中,复数可取如下的定义:复数是一种数据结构

$$\text{Complex} = (C,R)$$

其中:C 是含两个实数的集合$\{c1,c2\}$;$R = \{P\}$,而 P 是定义在集合 C 上的一种关系$\{<c1,c2>\}$,其中有序偶 $<c1,c2>$ 表示 $c1$ 是复数的实数,$c2$ 是复数的虚部。

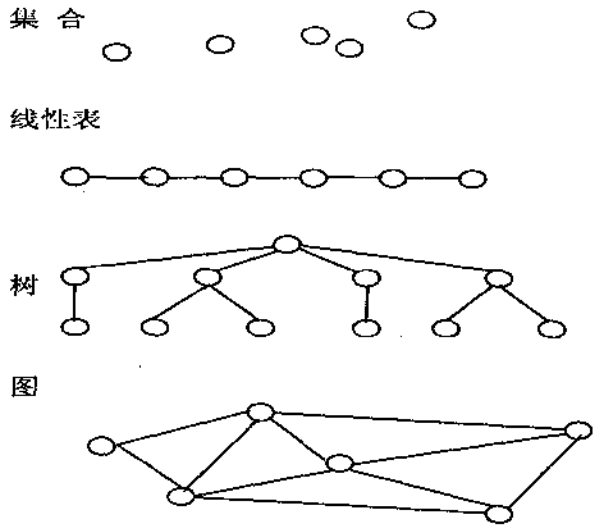

图 1.2　数据结构的逻辑结构图

例 1—2　为了用计算机管理学生的课外活动"小组"，首先要为小组设计一个数据结构。假设每个小组只有 7 名成员(为了便于陈述，分别赋予他们代号为：A,B,C,D,E,F,G)，其中 A 是组长，其余又分为两个小小组，每组各有一个召集人。假设 B,C,D 同属一个小小组，召集人是 D，另一个小小组的召集人是 G，则表示这个"小组"的逻辑结构可以用一个数据元素的集合 S 和定义在该集合上的一个关系 R 来表示，其中

$$S = \{A,B,C,D,E,F,G\}$$

$$R = \{<A,D>, <A,G>, <D,B>, <D,C>, <G,E>, <G,F>\}$$

数据的存储结构是逻辑结构在存储器中的映像，与逻辑结构相对应，存储结构包括数据元素的表示和关系的表示两个方面。

1.2.3　抽象数据类型(ADT—Abstract Data Type)

抽象数据类型是指数据结构作为一个软件组件的实现。ADT 的接口用一种类型和该类型上的一组操作来定义，每个操作由它的输入和输出定义。ADT 并不会指定数据类型如何实现，这些实现细节对于 ADT 的用户是隐藏的，并且通过封装(encapsula)来阻止外部对它的访问。

抽象数据类型是一些操作的集合。抽象数据类型是数学的抽象。在 ADT 的定义中根本没涉及如何实现操作的集合。

因此，一个数据结构加上定义在这个数据结构上的一组操作，即构成一个抽象数据类型。

对于诸如表、集合、图和它们的操作一起可以看作是抽象数据类型，就像整数、实数和布尔量是数据类型一样。整数、实数及布尔量有与它们相关的操作，而抽象数据类型也有与它们自己相关的操作。对于集合 ADT，我们可以有诸如并(union)、交(intesection)、测定大小(size)等操作。或者，我们也可以只要两种操作：并(union)和查找(find)，这两种操作又在该集合上定义了一种不同的 ADT。

运用抽象数据类型的概念实际上是将数据结构的讨论分成两部分：一部分是其概念所涵盖的数据逻辑结构的说明和运算的定义，突出的是在某种关系的数据对象上可以做什么；另一

部分是在计算机中如何存储这些数据并具体实现定义的运算，解决的是怎样做的问题。抽象数据类型面向使用层次，隐蔽了实现层次，主要的优点在于实现方法的改变，不会影响用户的使用，可以提高含有该抽象数据类型的软件模块的复用程度。抽象数据类型的概念与面向对象方法的思想是一致的。

1.3 问题、算法和程序

程序设计人员总是需要与问题、算法和计算机程序打交道。这是三种不同的概念。

问题(problem)：从直觉上讲，问题无非是一个需要完成的任务，即一组输入就有一组相应的输出。问题的定义不应包含有关怎样解决问题的限制。只有在准确定义并完全理解问题之后，才能研究问题的解决方法。然而，问题的定义应该包含对任何可行方案所需要资源的限制。对于计算机要解决的任何一个问题，总有一些直接或间接的资源限制。例如，任何计算机程序只能使用可用的主存储器和磁盘空间，而且必须在合理的时间内完成运行。

从数学角度讲，可以把问题看成函数。函数(function)是输入(即定义域，domain)和输出(即值域，range)之间的一种映射关系。函数的一组选定值称为问题的一个实例(instance)。例如，如果一个排序函数的输入参数是整数数组，那么一个长度给定并且数组每个位置也具体的整数数组，就是该排序问题的一个实例。不同的实例可能产生相同的输出，但是对于问题的任何一个实例，只要把它作为给定输入，每次函数计算得到的输出就必然相同。

算法(algorithm)：算法是指解决问题的一种方法或者一个过程。如果将问题看做函数，那么算法就是把输入转换为输出。一个问题可以用多种算法来解决，一种给定的算法解决一个特定问题。例如，本书的排序问题大概有 10 多种不同的算法。

知道一个问题的多种解法是有好处的。对于问题的一个特例或问题的一类特殊输入，解法 A 可能比解法 B 有效，而对于另一特例或问题的另一类特殊输入，解法 B 可能又比解法 A 更有效。例如，有些排序算法适合于数目较少的序列，有些排序算法则适合于数目较多的序列。

一种算法应该包含以下几条性质，只有具有以下所有性质的才能称为是一种解决特定问题的算法：

1. **正确性**(correct)。也就是说，它必须完成所期望的功能，把每一次输入转换成为正确的输出。

2. **具体步骤**(concrete steps)。一种算法应该由一系列具体步骤组成。“具体”意味着每一步所描述的行为对于必须完成算法的人或机器是可读的、可执行的。每一步必须在有限的时间内执行完毕。因此，算法好像给出了通过一系列步骤解决问题的“工序”，其中的每一步都是我们力所能及的，是否能够完成每一步依赖于谁或者什么来执行这个工序。

3. **确定性**(no ambiguity)。下一步应执行的步骤必须明确。选择语句是任何算法描述语言的组成部分，它允许对下一步执行的语句进行选择，但是选择过程必须确定。

4. **有限性**(finite)。一种算法必须由有限步骤组成。如果一种算法的描述是由无限步组成的，我们就不能将它们写出来，也不可能将它作为计算机程序来实现。

5. **可终止性**(terminable)。算法必须可以终止，即不能进入死循环。

程序(program)：一个计算机程序被认为是使用某种程序设计语言对一种算法的具体实现。当然，由于使用任何一种现代计算机程序设计语言都可以实现任何一种算法，所以可能有许多程序都是同一种算法的实现(尽管一些程序设计语言可能使编程人员的工作容易一些)。

定义一种算法时，必须提供足够多的细节，以便必要时转换为程序。

算法必须可终止意味着不是所有的计算机程序都是算法。操作系统是一个程序，而不是算法。然而，可以把操作系统的各种任务看成是一些单独的问题，每个问题由一部分操作系统程序通过某种算法来实现，得到输出结果后终止。

总之，问题(problem)是一个函数，或者是从输入到输出的一种映射。算法(algorithm)是一种能够解决问题的、有具体步骤的方法。算法步骤必须无二义性，算法必须正确，长度有限，必须对所有输入都能终止。程序(program)是算法在计算机程序设计语言中的实现。

1.4 用类C语言描述算法

在不同层次上讨论的算法有不同的描述方法。本书在高级程序设计语言的基础上讨论算法。同时为了使算法的描述和讨论简明清晰，容易被人理解，拟采用类C语言。它既不拘泥于某个具体的C语言，又能容易转换成可以上机调试的C程序或C++程序。

以下对其作简要说明。

(1)预定义常量和类型

常量说明采用C++语言的规范。

```
//函数结果主要状态代码
const TRUE=1;
const FALSE=0;
const OK=1;
const ERROR=0;
const INFEASIBLE=-1;
const OVERFLOW=-2;
//status是函数的返回值类型，其值是函数结果状态代码
typedef int status;
//布尔型类型
enum bool {TRUE,FALSE};
```

(2)数据结构的表示(存储结构)都用类型定义(typedef)的方式描述。基本数据元素类型约定为ElemType，由用户在使用该数据类型时再自行具体定义。

(3)基本操作的算法都用以下形式的函数描述：

```
函数类型  函数名(函数参数表)
{
    // 算法说明
        语句序列
}   // 函数名
```

除了函数的参数需要说明类型外，算法中使用的辅助变量可以不作变量说明，必要时对其作用给予注释。每个函数一般均返回一个状态码，向调用程序报告结果状态。当函数返回值为函数结果状态代码时，函数定义为status类型。

为了便于算法描述，在函数参数表中除了值调用方式外，增添了C++语言的引用调用的参数传递方式。在形参表中，以&打头的参数即为引用参数(reference parameter)。引用参

数能被函数本身更新参数值，可以此作为输出的管道。参数表中的某个参数允许预先用表达式的形式赋值，作为默认值使用，以简化参数表。

用值调用方式传递参数会产生参数的一个副本，这能防止函数调用时使用的实参被函数修改。在C语言中，函数若要改变用做实参的变量的值，就必须把参数定义为指针类型，考虑实现两个整数值交换的函数，若用C语言实现，虚参必须定义为指针类型。

```
/*C形式的指针类型作为参数*/
#include <iostream.h>
void swap(int *a,int *b)
{   //指针作为虚参，可以改变指针所指变量的值
    int t;
    t = *a;
    *a = *b;
    *b = t;
}
int main()
{//测试 swap 函数
   int i,j;
   i = 10;
   j = 20;
   cout<<i<<" "<<j<<endl;
   swap(&i,&j); //实参为整型变量的地址
   cout<<i<<" "<<j<<endl;
   return 0;
}
```

在C++语言中，可使用引用参数的引用调用方法，这种方法更清晰、更透明。如下的程序代码是使用引用调用的方法实现交换两个整数的值。

```
#include <iostream.h>
void swap(int &a,int &b)
{  //声明为引用参数
    int t;
    t=a;
    a=b;
    b=t;
}
int main()
{//测试 swap 函数
   int i,j;
   i = 10;
   j = 20;
   cout<<i<<" "<<j<<endl;
```

```
    swap(i,j); //实参是变量本身
    cout<<i<<" "<<j<<endl;
    return 0;
}
```

(4)内存的动态分配与释放

使用 **new** 和 **delete** 动态分配和释放内存空间。

分配空间:指针变量 = new 数据类型;

释放空间:delete 指针变量;

(5)赋值语句有

简单赋值　变量名=表达式;

串联赋值　变量名 1=变量名 2=…=变量名 k=表达式;

成组赋值　(变量名 1,变量名 2,…,变量名 k)=(表达式 1,表达式 2,…,表达式 k);

结构名 = 结构名;

结构名 =(值 1,值 2,…,值 k);

变量名[ ] = 表达式;

变量名[起始下标..终止下标] = 变量名[起始下标..终止下标];

条件赋值　变量名=条件表达式? 表达式 T:表达式 F;

(6)选择语句有

```
条件语句 1      if(条件表达式) 语句;
条件语句 2      if(条件表达式) 语句;
                else 语句;
开关语句 1      switch(表达式) {
                case 值 1:语句 1;break;
                …
                case 值 n:语句 n;break;
                default: 语句 n+1;
                }
开关语句 2      switch{
                case 条件 1:语句 1;break;
                …
                case 条件 n:语句 n;break;
                default: 语句 n+1;
                }
```

(7)循环语句有

```
for 语句    for(赋初值表达式序列;条件表达式;修改表达式序列) 语句;
while 语句  while (条件表达式)  语句;
do-while 语句  do {
                    语句序列;
               }while(条件表达式);
```

(8)结束语句有

函数结束语句　return　表达式;

　　return ;

case 结束语句　break;

(9)输入和输出语句使用流式输入输出的形式

输入语句　cin>>变量 1>>…>>变量 n;

输出语句　cout<<表达式 1<<…<<表达式 n;

(10)注释

单行注释　//文字序列

(11)基本函数有

求最大值　max(表达式 1,…,表达式 n)

求最小值　min(表达式 1,…,表达式 n)

求绝对值　abs(表达式)

退出程序　exit(表达式)

(12)逻辑运算约定

与运算 &&:对于 A && B,当 A 的值为 0 时,不再对 B 求值。

或运算||:对于 A || B,当 A 的值为非 0 时,不再对 B 求值。

1.5 习题

1. 简述下列术语:数据、数据元素、数据项、数据结构、数据类型、抽象数据类型。

2. 什么是算法? 算法的五个重要特性是什么?

3. 你是怎么理解问题、算法和程序三者之间关系的?

4. 写出对有 $n(n>100)$ 个整数的数组按值从小到大进行排序的算法(函数),至少写出 3 种不同的实现方法。

5. 用类 C 语言描述求两个整数最大公因子的算法。

6. 试根据以下信息:校友姓名、性别、出生年月、毕业时间、所学专业、现在工作单位、职称、职务、电话等,为校友录构造一种适当的数据结构(作图示意),并定义必要的运算和用文字叙述相应的算法思想。

7. 编译器和文本编辑器的一个普遍问题是判断一个字符串的圆括号(或其他括号)是否平衡并恰好匹配。例如,字符串"(((())())())"中的圆括号平衡且恰好匹配,但是字符串")()("中的圆括号不平衡,字符串"())"中的圆括号不匹配。

(1)给出一种算法,当字符串中的圆括号恰好平衡且匹配时返回 true,否则返回 false。提示:从左到右扫描一个合法的字符串,保证任何时候所遇到的右圆括号不会比左圆括号多。

(2)给出一种算法,如果字符串中圆括号不平衡或者不匹配,则返回字符串中第一个非法圆括号的位置。也就是说,如果发现一个多余的右圆括号,则返回它的位置;如果有多个左圆括号,则返回第一个多余左圆括号的位置;如果字符串平衡且恰好匹配,则返回-1。

8. 假设有一组记录,按照每个记录都包含的一些关键码字段排序。给出两种不同的方法搜索有特定关键码值的记录,你认为哪种更好,为什么?

第2章 算法分析

算法(algorithm)是求解一个问题需要遵循的、被清楚地指定的简单指令的集合。对于一个问题，一旦给定某种算法并且(以某种方式)确定其是正确的，那么重要的一步就是确定该算法将需要多少诸如时间或空间等资源量的问题。如果一个问题的求解算法需要长达一年时间，那么这种算法就很难能有什么用处。同样，一个需要1G字节内存的算法在当前的大多数机器上也是没法使用的。

“算法分析”这个术语常常仅用于狭义的技术层面，指的是对算法利用两种资源的效率作研究：运行时间和存储空间。

当把算法设计与分析提升为一种解决问题的通用方法时，读者一定会有很大收获，而且有可能终生受益。

算法与数据结构最关心的就是效率问题。本章将介绍算法分析的基本符号和基本技术。

2.1 渐近算法分析

算法分析涉及两种效率：时间效率和空间效率。时间效率指出正在讨论的算法运行得有多快；空间效率关心算法需要的额外空间。我们把主要精力集中在时间效率上，但这里介绍的分析方法对于分析空间效率也是适用的。

如何比较解决同一问题的两种算法的效率呢？一种方法是用源程序实现这两种算法，然后输入适当的数据运行，测算两个程序各自的开销。这种方法实际上并不可行：第一，编写两个程序来测算两种算法将花费较多的时间和精力；第二仅凭实验来比较两种算法，很可能因为一个程序比另一个“写得好”，而使算法的真正质量没有得到很好的体现；第三，测试数据的选择可能对其中一种算法有利；第四，你可能会发现即使那种较好的算法也超出了预算开销，这意味着你不得不寻找一种新的算法，再编写一个程序实现它。但是，你又怎么知道是否存在能够满足预算开销的算法呢？所以第一种方法没有实用价值。我们将用第二种方法来解决这些问题。这种方法就是**渐近算法分析**，它可以估算出当问题规模变大时，一种算法及实现它的程序的效率和开销。

2.1.1 输入规模的度量

“规模”一般指输入量的数目。几乎所有的算法，对于规模更大的输入都需要运行更长的时间。例如，待排序的数组元素个数越多，需要的排序时间越长。所以，使用一个以算法输入规模 n 为参数的函数，来研究算法效率是非常合乎逻辑的。在大多数情况下，选择这样一个参数是非常简单直接的。对于排序、查找、寻找列表的最小元素以及其他大多数和列表操作相关的问题来说，这个参数就是列表的长度。当然有些算法需要不止一个参数来表示它们的输入规模，例如，对于用邻接表来表示图的算法来说，这些参数就是顶点的数量和边的数量。

2.1.2 运行时间的度量单位

下面我们考虑算法运行时间的度量单位。当然，我们可以简单地使用时间的标准度量单

位来度量算法程序的运行时间，例如，秒、毫秒等。然而，这种方法有一些明显的缺陷：它依赖于特定计算机的运行速度，依赖于算法程序实现的质量，依赖于使用哪种编译器将程序转化成机器码，而且，对程序的实际运行时间进行计时也是困难的。既然我们寻求的是算法效率的度量，我们希望能够拥有一个不依赖于这些无关因素的量度。

当然，我们可以统计算法每一步操作的执行次数。这样做被证明是困难且没有必要的。我们要做的，是找出算法中最重要的操作，即所谓的**基本操作**，它们对总的运行时间贡献最大。然后计算它们的运行次数。

实际上，基本操作通常是算法最内层循环中最费时的操作。例如，大多数排序算法是通过比较待排序序列中元素的关键字来工作的，对于这种算法来说，基本操作就是比较操作。再举一个例子，矩阵的乘法算法和多项式的求值算法需要做两种算术运算：乘法和加法。在大多数计算机上，两个数的乘法运算比加法更耗时，所以，毫无疑问，我们应该选择乘法运算作为基本操作。

这样，**渐近算法分析**就是对输入规模为 n 的算法，通过统计它的基本操作执行次数来对其效率进行度量。

这里，我们约定，C_{op} 为特定计算机上一个算法基本操作的执行时间，而 $C(n)$ 是该算法需要执行基本操作的执行次数，那么，对运行在那台计算机上的某个算法程序的运行时间，我们就能用以下公式做估计了：

$$T(n) \approx C_{op} C(n)$$

当然，执行次数 $C(n)$ 并不包括非基本操作的任何信息，并且，实际上，它本身也常常是一个近似结果。另外，常量 C_{op} 也是一个近似值。但是，除非 n 非常大或者非常小，这个公式可以对算法的运行时间作一个合理的估计。

渐近算法分析可以估算出当问题规模变大时，一种算法及实现它的程序的效率和开销。如果两个程序中的一个总是比另一个“稍快一点”，它并不能判断那个“稍快一点”的程序相对优越。但在实际应用中，它被证明是很有效的，尤其是当科学家们确定某种算法是否值得实现的时候。

2.1.3 算法的增长率

算法的增长率(growth rate)是指当输入规模增长时，算法代价的增长速率。图 2.1 给出了 5 个运行时间函数的曲线，每个多项式反映一个程序或者一种算法的时间代价，图中显示了不同算法的增长率。标记为 $10n$ 和 $20n$ 的两个函数图像为直线，表达式为 cn（c 为任意正常数）的增长率称为线性增长率或者线性时间代价。这说明，当 n 增大时，算法的运行时间也以相同的比例增加。n 增大一倍，运行时间也增加一倍。如果算法的运行时间函数中有形如 n^2 高次项，则称为二次增长率。标有 2^n 的曲线属于指数增长率，由 n 出现在指数位置而得名。

为什么对于大规模的输入要强调算法的增长率呢？这是因为小规模输入在运行时间上的差别不足以将高效的算法和低效的算法区分开来。对 n 的较大值来说，有意义的是其函数的增长率，表 2—1 包含了一些对于算法分析来说，具有特别重要意义的函数值。

表 2—1 中给出了复杂性为 $f(n)$ 的程序在每秒执行 1 000 000 000 条指令的计算机上执行时所需要的时间。应该注意到，目前只有世界上最快的计算机才能每秒执行 1 000 000 000 条指令。从实际应用来看，对于相当大的 n（比如 $n>100$，仅那些复杂性比较小（如 n，$n\log n$，