计算机算法设计与分析研究

柴宝杰 著

新 华 出 版 社

计算机算法设计与分析研究

柴宝杰 著

新 华 出 版 社

图书在版编目（CIP）数据

计算机算法设计与分析研究 / 柴宝杰 著.

北京：新华出版社，2015．7

ISBN 978-7-5166-1905-6

Ⅰ．①计… Ⅱ．①柴… Ⅲ．①计算机算法—研究 Ⅳ．① TP301.6

中国版本图书馆 CIP 数据核字（2015）第 171502 号

计算机算法设计与分析研究

作　　者：柴宝杰 著

出 版 人：张百新

责任编辑：鞠　景

出版发行：新华出版社

地　　址：北京市石景山区京原路 8 号　　**邮　　编：**100040

网　　址：http://www.xinhuapub.com　　http://press.xinhuanet.com

经　　销：新华书店

成品尺寸：185mm× 260mm　　**开　　本：**16

印　　张：8.625　　**字　　数：**170 千字

版　　次：2015 年 7 月第一版　　**印　　次：**2015 年 7 月第一次印刷

印　　刷：北京厚诚则铭印刷科技有限公司

书　　号：ISBN 978-7-5166-1905-6

定　　价：35.00 元

图书如有问题，请与出版社联系 010——63074590

前　言

计算机算法是计算机科学和计算机应用的核心。无论是计算机系统、系统软件的设计还是为解决计算机的各种应用课题做的设计都可归结为算法的设计。因此，学好计算机算法设计与分析，对从事计算机系统结构、系统软件及应用软件研究和开发人员来讲都是非常重要的和必不可少的。

目前对于计算机算法介绍的图书通常有两类：一类着重介绍的是数据结构本身的实现，即数据结构与算法；另一类着重介绍的是算法设计的原理，即算法设计与分析。作者撰写的这本《计算机算法设计与分析研究》则从算法实践的角度出发，着重介绍几种常用算法的基本思想、完整实现及其应用，使读者能用学过的理论知识解决具体、复杂的实际问题，从而达到应用型人才培养的目标。

本书以算法实践为知识单元，以期为读者提供坚实的计算机算法的应用知识。全书共分 8 章：第 1 章主要介绍了算法的基础知识；第 2 章主要介绍栈和队列的结构特性及基于这些结构的一些应用举例；第 3 章主要介绍树和图的结构特性及基于这些结构的常用算法；第 4 章主要介绍递归的概念和分治法的基本思想及基于递归或分治思想所解决的经典问题；第 5 章主要介绍贪心算法的思想及利用贪心算法解决实际问题；第 6 章主要介绍动态规划法的适用性和算法的设计要点及利用动态规划法所解决的经典问题；第 7 章对 NP– 完全问题与处理做了初步介绍；第 8 章主要介绍线性规划的概念及对二分图相关问题做了初步分析与研究。

本书在撰写过程中参考了大量优秀研究成果，得到广大专家学者及同仁的广泛帮助，在此，深表感谢！同时由于本人水平有限，研究还不够深入，书中若有不当之处，恳请广大专家读者批评指正。

作　者

2015 年 4 月

目　录

第 1 章　算法概述

1.1 算法与问题求解

1.1.1 算法的概念

算法 (Algorithm) 的概念在计算机科学与技术领域几乎无处不在，在各种计算机系统的实现中，算法的设计往往处于核心的位置。不过，计算机算法的研究受到人们如此的重视是在 20 世纪 70 年代以后。具体地说，D.E.Knuth 以及 A.V.Aho、J.E.Hopcroft、J.D.Ullman 等人的著述对算法的研究起到了奠基的作用。其中，Stanford 大学著名计算机科学家 D.E.Knuth 因他的这一著作，获得了计算机领域的最高奖——图灵奖。[1]

虽然我们每天都在与算法打交道，但要严格地指出什么是算法却不是一件容易的事。通过对众多学者关于算法研究的认识得出：算法是指解决问题的一种方法或者一个过程。如果将问题看作函数，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的问题。本书涉及了许多问题，其中有些问题给出了几种算法。知道一个问题的多种解法的好处在于，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，而另有一些算法则适合于变长字符串。

1.1.2 问题求解

就像生物学家把自然界的所有生物作为自己的研究对象一样，计算（Computation）学科或称计算机科学则把问题作为自己的研究对象，即用计算机来解决问题。在计算机专家的头脑中，世界是一个个要解决的问题的集合。进一步来说，每一个问题又都是该问题的所有实例（Instance）的集合。一般而言，一个问题的一个实例就是为计算该问题所需的一组输入。

例如：

（1）整数乘法问题。其全部可能输入集合：$\{\langle a, b\rangle \mid a, b \in Z\}$，其中 Z 表示整数集。整数乘法问题的一个实例：$\langle 2, 4\rangle$，求 $2\times4=?$。

（2）旅行商问题。其实例集为：$\{n, W_{n\cdot n}=[W_{i\cdot j}] \mid n \in Z, W_{i\cdot j} \in R, i, j \in [1\cdots n]\}$。旅行商问题的一个实例：n=4。

---

[1] 刘璟编著：《计算机算法引论：设计与分析技术》，科学出版社，2003，5。

$$W=\begin{bmatrix}0 & 6 & 7 & 8\\ 4 & 0 & 3 & 2\\ 2 & 4 & 0 & 9\\ 10 & 7 & 4 & 0\end{bmatrix}$$

旅行商问题的这个实例（n, W）就是求环游四个城市（1，2，3，4）的代价和最小的周游路线，其中权矩阵 W 给出了四个城市的距离（代价），$W_{i\text{-}j}$ 表示由城市 i 到城市 j 的距离（代价）。

（3）语言 L(G, ∑) 的识别问题。

$$L(G) \in \sum^{*}$$

其中，G 为文法，即语法规则集，∑为一个字符集。L(G) 为∑上由 G 生成的语言，$\sum^{*}$ 表示由∑中的字符组成的所有字符串集合。

语言 L 的识别问题 $\{G, \sum, \omega \mid \omega \in \sum^{*}\}$ 的一个实例 $\omega \in \sum^{*}$，识别算法判断 $\omega \in L(G)$ 是否成立。

一般来讲，对于问题 P，总有其相应的实例集 I，那么，一个算法 A 是问题 P 的算法，意味着把 P 的任一实例 input ∈ I 作为 A 的输入，都能得到问题 P 的正确输出。一个问题可以有许多个不同的算法。

1.1.3 算法的特性

作为一个算法，应该具备如下 5 个特性。

（1）输入性

一个算法要具有 0 个或多个外部量作为算法的输入。这些外部量通常体现为算法中的一组变量。有些输入量需要在算法执行过程中输入。从表面上看，有些算法好像没有输入量，实际上是输入量已被嵌入到算法之中。

（2）输出性

一个算法必须具有一个或多个输出，以反映算法对输入数据加工后的结果。没有输出的算法是毫无意义的。

（3）确定性

算法的每一个步骤必须具有确定的定义，即每一步要执行的动作是确定的，是无二义性的。在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入得出的输出结果也是相同的。

（4）有穷性

对于任何合法的输入值，算法必须在执行有限个步骤之后结束，并且每一步都可以在有限的时间内完成。

（5）可行性

算法中描述的操作都是可以通过已经实现的基本运算的有限次执行来实现，即算法的具体实现应该能够被计算机执行。

假如现在有一个问题 : 依据如下分段函数，要求对用户输入的一个自变量 x 的值，给出相应的函数值。

$$f(x)=\begin{cases} x+2, & x>0 \\ 2, & x=0 \\ x-2, & x<0 \end{cases}$$

实现分段函数求值功能的过程如下。

步骤 1 ：输入自变量 x 的值。

步骤 2 ：依据自变量 x 的值进行判断 ：

如果 x>0，执行 x+2 → f (x) 操作 ；

如果 x<0，执行 x−2 → f (x) 操作 ；

如果 x=0，执行 2 → f (x) 操作。

其中，A → B 表示将表达式 A 的值赋给变量 B。

步骤 3 ：输出步骤 2 计算的结果。

显然，上述描述满足算法的 5 个重要特征，因此，该描述就是一个算法。

在现实社会中，不同的人对于同一问题会有不同的看法或解决方法。同样，在计算机领域，对于同一问题存在多种算法也是很自然的事情。例如对于一批数据的排序问题，就存在多种排序方法。判断一个算法的好坏主要依据如下 4 个标准。

（1）正确性

正确性是设计一个算法的首要条件，如果一个算法不正确，其他方面就无从谈起。一个正确的算法是指在合理的数据输入下，能在有限的时间内得出正确的结果。

（2）可读性

算法主要是为了人的阅读与交流，其次才是让计算机执行，因此算法应该易于人的理解 ; 另一方面，晦涩难读的算法易于隐藏较多错误而使实现该算法的程序的调试工作变得更加困难。

（3）健壮性

算法应当具备检查错误和对错误进行适当处理的能力。一般而言，处理错误的方法不应是中断程序的执行，而应是返回一个表示错误或错误性质的值，以便在更高的抽象层次上进行处理。

（4）效率

效率是指算法执行时所需计算机资源的多少，包括运行时间和存储空间两方面的要求。运行时间和存储空间都与问题的规模有关。存储空间指的是算法执行过程中所需的最大存储空间。

在设计一个算法时，要从上述 4 个方面综合考虑。同时还要考虑到算法的使用频率及所使用机器的软硬件环境等因素，这样才能设计出一个好的算法。

1.2 算法的描述

算法的描述形式多种多样，不同的算法描述形式对算法的质量有一定的影响。描述同一个算法可以采用自然语言、流程图、盒图、伪代码、程序设计语言等。常用的描述算法方法有如下 3 种。

1.2.1 自然语言描述法

最简单的描述算法的方法是使用自然语言。用自然语言来描述算法的优点是简单且便于人们对算法的理解和阅读；缺点是不够严谨，易产生歧义。当算法比较复杂且包含很多转移分支时，用自然语言描述就不是那么直观清晰了。

1.2.2 算法框图法

使用程序流程图、盒图等算法描述工具来描述算法。其特点是简洁、明了，便于理解和交流。

1.2.3 伪码语言描述法

用上述两种方法描述的算法并不能够直接在计算机上执行。为了解决理解与执行之间的矛盾，人们常常使用一种称为伪码语言的描述方法来对算法进行描述。伪码语言介于高级程序设计语言和自然语言之间，它忽略高级程序设计语言中一些严格的语法规则与描述细节，因此它比程序设计语言更容易描述和被人理解，而比自然语言或算法框图更接近程序设计语言。

当然，大部分的算法最终是需要通过能够向计算机发送一系列命令的程序来实现的。所谓“程序”是指对所要解决问题的各个对象和处理规则的描述，或者说是数据结构和算法的描述，因此有人说，数据结构 + 算法 = 程序。

程序与算法不同。程序可以不满足算法的第 4 个特性。例如操作系统，它是在无限循环中执行的程序，因而不是算法。然而可把操作系统的各种任务看作一些单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法实现，该子程序得到输出结果后便终止。

1.3 算法分析

算法复杂性的度量主要是针对运行该算法所需要的计算机资源的多少。当算法所需要的资源越多，该算法的复杂性越高；反之，当算法所需要的资源越少，算法的复杂性越低。

对于任意给定的一个问题，设计出复杂性尽可能低的算法是在设计算法时追求的重要目标之一；而当给定的问题存在多种算法时，选择其中复杂性最低的算法是选用

算法时遵循的重要准则。因此，算法的复杂性分析对算法的设计或选用具有重要的指导意义和实用价值。

1.3.1 时间复杂度

通常，对于一个算法的复杂性分析主要是对算法效率的分析，包括衡量其运行速度的时间效率及衡量其运行时所需要占用空间大小的空间效率。

对于早期的计算机来说，时间与空间都是极其珍贵的资源。由于硬件技术的发展大大提高了计算机的存储容量，使得存储容量的局限性对于算法的影响大大降低。但是时间效率并没有得到相应程度的提高。因此，算法的时间效率或算法时间复杂度是算法分析中的关键所在。

对于算法的时间效率的计算，通常是抛开与计算机硬件、软件有关的因素，仅考虑实现该算法的高级语言程序。一般而言，对程序执行的时间复杂度的分析是分块进行的，先分析程序中的语句，再分析各程序段，最后分析整个程序的执行复杂度。通常以渐进式的大 0（希腊字母 Omicron，奥米克戎）形式来表示算法的时间复杂度。渐进式的大 0 形式表示时间复杂度。

1.3.2 空间复杂度

一般情况下，一个算法所占用的存储空间包括算法自身、算法的输入、算法的输出及实现算法的程序在运行时所占用空间的总和。

由于算法的输入和输出所占用的空间基本上是一个确定的值，它们不会随着算法的不同而不同。而算法自身所占用的空间与实现算法的语言和所使用的语句密切相关，例如程序越短，它所占用的空间就越少。一个算法在运行过程中所占用的空间，特别是算法临时开辟的存储空间单元则是由算法策略及该算法所处理的数据量决定的。因此，对于一个算法的空间复杂度的衡量主要考虑的是算法在运行过程中所需要的存储空间的大小。[1]

[1] 徐保民，陈旭东，李春艳编著，《计算机算法与实践教程》，北京交通大学出版社，2007，3-5。

第 2 章　线性数据结构与算法

由于算法是作用在数据上的，因此数据的组织形式即数据结构在很大程度上影响着算法的设计和实现。本章着重介绍几种常见的线性数据结构，包括线性表、栈与队列，并同时给出若干常用的基于上述线性数据结构的算法实现。

2.1 线性表

线性表（Linear List）是最常用且比较简单的一种数据结构，其主要特点是数据元素有序地存放在一起。

2.1.1 线性表的概念及特点

线性表是由有限个数据元素组成的有序集合，每个数据元素由一个或多个数据项组成。例如构成 26 个英文字母的字母表 (A，B，C，…，Z) 是一个线性表，表中每个元素由单个字符组成数据项。

对于非空的线性表而言，它具有如下 4 个特点：

（1）表中有且仅有一个开始结点；

（2）表中有且仅有一个终端结点；

（3）除了开始结点和终端结点外，其他每个元素前面均有且仅有一个称为直接前趋的数据元素，它的后面均有且仅有一个称为直接后继的数据元素；

（4）虽然不同线性表的数据元素可以是各种各样的，但是同一线性表中的数据元素必须具有相同的数据类型。

线性表上的操作主要有插入、删除和查找三种。

常见的线性表的存储结构是顺序表 (Sequential List) 和链表 (Linked List)。

2.1.2 顺序表

在计算机内存中存放线性表的最简单相最常用的方式是用一组地址上连续的存储单元依次存储线性表中的每个元素，即将线性表中的数据元素按其逻辑顺序依次存放到一个连续的内存区域，使线性表中相邻的元素存放在地址相邻的物理存储单元中，这种存储结构称为线性表的顺序存储结构，或简称为顺序表。

假设线性表的每个元素占用 L 个存储单元，并以其所占的第一个单元的存储地址 LOC（a_1）作为数据元素位置，则线性表中第 i+1 个数据元素的存储位置 LOC（a_{i+1}）与第 i 个数据元素的存储位置 LOC（a_i）之间满足如下关系：

$$LOC(a_{i+1}) = LOC(a_i) + L$$

线性表中任意一个数据元素 的存储位置与线性表中第一个数据元素 的存储位置之间的关系可以表示为：

$$LOC(a_i)=LOC(a_1)+(i-1)\times L$$

由于每一个数据元素的存储位置都和线性表中第一个数据元素的存储位置相差一个与数据元素在线性表中的位序成正比的常数。因此，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以随机存取，所以线性表的顺序存储结构是一种随机存取的存储结构。

目前，几乎所有的高级程序设计语言都把数组定义为一种最基本的数据类型，而且编译程序为数组所分配的空间通常是一片连续的存储区域。因此，利用高级程序设计语言所提供的数组类型来实现顺序表是很自然的事情了。

一个数组是由一组相同类型的数据元素的固定集合。它们在物理内存上连续存放。对数组元素的访问是通过数组的下标进行的。所谓下标就是指该元素相对于数组第一个元素的偏移量，通常是 0 与 n−1 之间的一个整数。通过下标，对数组中任何一个位量的元素的访问时间都是一样的。

在 Java 语言中，线性表的顺序存储结构可以描述为：

```
Class ArrayLinearList{
    int size; // 数组中已有的元素个数
    Object element[]; // 数据元素
    ……// 各种方法的定义
}
```

对于顺序表可以执行的操作很多，下面主要介绍对于用数组描述的顺序表所进行的空间分配、元素插入、元素删除和元素查找操作。

1．空间分配

在 Java 语言中，数组属于复合数据类型，在使用它之前，必须为其分配空间。

空间分配算法的描述如下。

```
// 参数 initialSize 为用户所申请的空间大小
public ArrayLinearList(int initialSize){
    if (initialSize<1)// 抛出异常
        throw new IllegalArgumentException(" 容量必须大于 1");
    element=new Object[initialSize]; // 分配空间
}
```

2．插入操作

顺序表的插入操作是指在具有 size 个元素的线性表的第 i 个元素前插入一个新的元素 x，使线性表的长度增加 1。

插入算法的基本思路是：当 1 ≤ i ≤ size 时，将第 size 个到第 i 个元素依次后移，

然后进行插入。如成功插入返回 true，否则返回 false。

插入算法的描述如下。

```
public boolean insert_SqList(int i, Object x){
        int j;
        int n=element.length; // 获取表的容量
        if ((i<1) ‖ (i>size)){
            System.out.println(" 插入元素位置不对 ");
            return false;
    }
    if (n=size){
        System.out.println(" 顺序表已满，无法插入 ");
        return false;
    }
    for(j=size−1; j ≥ i; j−−)
        element[j+1]=element[j]; // 元素依次后移
    element[i]=x; // 插入元素
    size++;
    return true;
}
```

如果顺序表未满，而且插入位置正确，算法将从最后一个元素开始后移，直到第 i 个元素，然后进行插入。如果顺序表已满，该算法只是简单地退出。在实际应用中，算法应该能动态调整顺序表的容量，这样就能减少算法出现异常的概率。

动态调整顺序表容量的基本思路是：重新创建一个容量较大的数组，将原数组中已有内容复制到新数组中。

调整顺序表容量算法的描述如下。

```
//n 表示原数组 a 的长度，newLength 为数组的新容量值
publicObject[]changeArrayLength(Object[]a, int newLength){
    int n=a.length;
    // 确保新的容量值合理
    If (n>newLength)
        Throw new IllegalArgumentException(" 新的容量值太小 ");
        // 创建一个新数组，与原数组类型相同，但容量不同
        Object[]newArray=( Object[])
        Array.newInstance(a.getClass().getComponentType(), newlength);
        // 将原数组 a 中从 0 开始的内容复制到新数组 newArray 中的
        // 从 0 开始的位置，n 为要复制的数组元素的数量
```

```
        System.arrayCopy(a, 0, newArray, 0, n);
        return newArray;
    }
```

3．删除操作

顺序表的删除操作是指在具有 size 个元素的线性表中，将第 i 个元素删除，使线性表的长度减 1。

删除算法的基本思路是：当 1 ≤ i ≤ size 时，将第 i 个元素到第 size 个元素依次前移。如成功删除返回 true，否则返回 false。

删除算法的描述如下。

```
public boolean delete_SqList(int i){
    int j;
    if ((i<1) || (i>size)){
        Systen.out.println(" 删除元素的位置不对 ");
        return false;
    }
    else{
        for(j=i; j<size; j++)
            element[j−1]=element[j]; // 元素依次前移
        element[−−size]=null; // 回收不用空间
        return true;
    }
}
```

4．查找操作

顺序表的查找操作是指在具有 size 个元素的线性表中，查找给定元素值为 x 的结点。

查找算法的基本思路是：从数组的第 1 个元素开始依次向后查找。如数组 v[i] 的值为 x 则返回它在线性表中的位置 i+1，否则返回 0。

查找算法的描述如下：

```
public int search_SqList(Object x){
    int i;
    for(i=0, i<size; i++){
        if (element[i].equal(x))// 查到给定元素
        return i+1;
        return 0
}
```

需要指出的是：与一般线性表不同，对于数组一般不作插入和删除操作，而只进

行通过给定的下标读取和修改相应位置的数组元素的值操作。

2.1.3 链表

如果想用一组地址位置任意的存储单元来存放线性表中的数据元素，则可以使用链表来组织这些数据元素。

链表是一组数据元素的集合，其中每个数据元素都是一个结点，结点的数据部分称为结点的数据域。为了能对整个链表进行遍历或访问，链表的每个结点至少还应该包含一个指向它的直接后继元素在物理内存中的位置信息 (通常称为指针域)。当然，一个结点也可以包含一个指向它的直接前趋元素在物理内存中的位置信息。一个链表的最后一个结点的指针域可以为空 (null)，表示整个链表的结束。也可以指向整个链表的第一个结点，构成一个回路，形成循环链表。

依据一个结点所具有的指针域的数目和最后一个结点的指针域的值不同，可以将链表分为单向链表、单向循环链表、双向链表、双向循环链表等。仅有前驱指针或后继指针的结点构成的链表称为单向链表，同时有前驱指针和后继指针的结点构成的链表称为双向链表。

在实际应用时，通常为整个链表增加一个头结点。此结点可以作为访问整个链表的入口也可以作为遍历整个链表的结束标记等。

1．单向链表及其运算

单向链表是指一个结点仅有一个指针域的链表。

（1）单向键表的结构描述

在 Java 语言中，线性表的单向链表存储结构可以描述为：

```
class ChainNode{
    Object element; // 数据域
    ChainNode next; // 指针域
    int size; // 结点个数
    ChainNode(){}
    ChainNode(Object element){
    this.element=element;
}
ChainNode(Object element, ChainNode next){
    this.element=element;
    this.next=next;
    ChainNode firstNode; // 链表头指针
    ……// 其他方法的定义
}
```

由于 Java 不支持指针，所以在上面所定义的链表结点结构中是通过保存在指针域

中的对象引用把不同结点联系起来构成一个链表。

（2）单向链表上的基本操作

下面主要介绍在单向链表存储结构上所进行的插入、删除和查找操作的实现。

①插入操作

在第 i 个位置后插入一个新结点 x 的操作过程为：首先产生新结点，然后修改相邻结点指针域的值。图 2-1 所示的是在单链表中插入指针 s 所指新结点的过程。

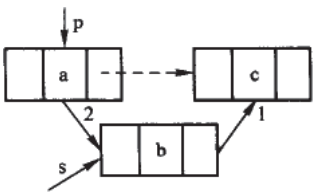

图 2-1　在 p 后插入结点

此时需要进行的指针修改操作是：

```
s.next=p.next;
p.next=s;
```

插入一个结点的算法描述如下。

```
public void insert(int i, Object newElement){
    if (i<0 ‖ i>size)
        System.out.println(" 插入元素位置不对 ") ;
        if (i==0)// 在空链表中插入第 1 个结点
        firstNode=new chainNode(newElement, firstNode);
    else{// 确定插入位置
        ChainNode p=firstNode;
        for(int j=0; j<i-1; j++)
        p=p.next;
        p.next=new ChainNode(newElement, p.next); // 在结点 p 后插入
    }
    size++;
}
```