计算机常用算法的设计与分析研究

JISUANJI CHANGYONG SUANFA DE SHEJI YU FENXI YANJIU

王文霞　潘玉霞　董改芳　编著

中国水利水电出版社
www.waterpub.com.cn

计算机常用算法的设计与分析研究

王文霞　潘玉霞　董改芳　编著

中国水利水电出版社
www.waterpub.com.cn
·北京·

内 容 提 要

计算机算法是计算机科学和计算机应用的核心。本书以算法设计策略为主线，系统介绍了算法的设计方法和分析技巧。书中既涉及传统算法的实例分析，更有算法领域热点研究课题的追踪，具有较高的实用价值。本书主要内容包括：递归与分治策略、动态规划算法、贪心算法、搜索算法、概率算法、NP 完全性理论、近似算法、现代计算智能算法简介等。

本书论述严谨，结构合理，条理清晰，内容丰富新颖，可供从事计算机算法设计与分析工作的研究人员参考。

图书在版编目（CIP）数据

计算机常用算法的设计与分析研究 / 王文霞，潘玉霞，董改芳编著. -- 北京 : 中国水利水电出版社，2017.4

ISBN 978-7-5170-5307-1

Ⅰ. ①计… Ⅱ. ①王… ②潘… ③董… Ⅲ. ①电子计算机－算法设计－研究②电子计算机－算法分析－研究 Ⅳ. ①TP301.6

中国版本图书馆CIP数据核字(2017)第074650号

书　　名	计算机常用算法的设计与分析研究 JISUANJI CHANGYONG SUANFA DE SHEJI YU FENXI YANJIU
作　　者	王文霞　潘玉霞　董改芳　编著
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路 1 号 D 座 100038） 网址：www. waterpub. com. cn E-mail：sales@waterpub. com. cn 电话：(010)68367658(营销中心)
经　　售	北京科水图书销售中心(零售) 电话：(010)88383994、63202643、68545874 全国各地新华书店和相关出版物销售网点
排　　版	北京亚吉飞数码科技有限公司
印　　刷	三河市佳星印装有限公司
规　　格	184mm×260mm　16 开本　17.5 印张　426 千字
版　　次	2017 年 5 月第 1 版　2017 年 5 月第 1 次印刷
印　　数	0001—2000 册
定　　价	62.00 元

前　言

计算机的普及使得人们的生活发生了很大的改变，它不断孕育着一代又一代的新技术、新概念。目前，计算机信息技术在很多不同的行业和领域都获得了广泛采用。计算机在各个领域的应用过程中，都会涉及数据的组织与程序的编排等问题，从而会用到各种各样的数据结构，这就更需要学会分析和研究计算机加工对象的特性、选择最合适的数据组织结构及其存储表示方法，以及编制相应实现算法的方法，这是计算机工作者必须具备的知识。可以说，算法是计算机的重要根基，对计算机行业的发展起着不可估量的作用。

计算机算法对人每天的生活都产生着影响。它运行在笔记本、服务器、智能手机、汽车、微波炉等各种设备上。可见，算法设计与分析是一门理论性与实践性相结合的学科，是计算机科学与计算机应用专业的核心。学习算法设计可以在分析解决问题的过程中，培养学习者抽象思维和缜密概括的能力，提高学习者的软件开发设计能力。

本书着重阐述典型算法的设计与分析。全书共 9 章，第 1 章算法引论，介绍计算机算法的基础知识；第 2～8 章分别介绍了递归与分治策略、动态规划算法、贪心算法、搜索算法、概率算法、NP 完全性理论、近似算法；第 9 章讨论了人工神经网络、遗传算法、蚁群优化算法、粒子群优化算法等一些现代计算智能算法。

本书是在计算机科学与技术专业规范和作者多年教学经验的基础上编撰而成的。在编撰方面力求突出以下特点：语言叙述通俗易懂，讲解由浅入深，算法可读性好，应用性强；在内容的深浅程度上，侧重实用的同时把握理论深度，通过一些例题、算法，有利于理论知识的掌握；取舍合理，体系完整、内容先进。由于计算机技术的发展是突飞猛进的，各类算法和技巧层出不穷，本书只能是管中窥豹，以求能够抛砖引玉。

全书由王文霞、潘玉霞、董改芳撰写，具体分工如下：

第 3 章、第 5 章、第 6 章：王文霞（运城学院）；

第 1 章、第 7 章、第 9 章：潘玉霞（三亚学院）；

第 2 章、第 4 章、第 8 章：董改芳（内蒙古农业大学）。

本书在编撰过程中得到了许多同行专家的支持和帮助，在此表示衷心的感谢；同时，还参考了很多的相关著作和文献资料，在此向有关作者表示由衷的感谢。

由于作者水平有限，书中难免存在错误或不妥之处，恳请读者批评指正。

作　者

2017 年 2 月

目　录

第 1 章　算法引论

1.1　算法在计算机科学体系中的地位

对于计算机科学而言，算法是其重要基础，同时也是计算机科学研究中的一项永恒主题。

有人曾将程序比作蓝色的诗。如此而言，算法应当称得上这首诗的灵魂了。因为计算机不能分析问题并产生问题的解决方案，必须由人来分析问题，确定问题的解决方案，采用计算机能够理解的指令描述问题的求解步骤，然后让计算机执行程序最终获得问题的解。可见，在各种计算机软件系统的实现中，算法设计往往处于核心地位。用计算机求解问题的一般过程如图 1-1 所示。

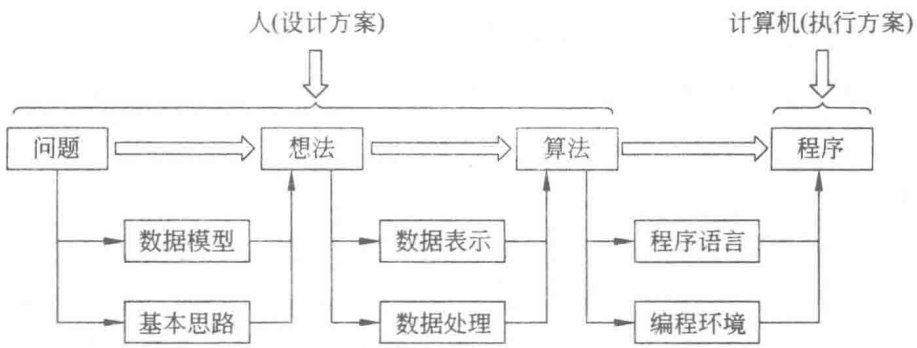

图 1-1　用计算机求解问题的一般过程

具体如下：由问题到想法需要分析问题，将具体的数据模型抽象出来，形成问题求解的基本思路；由想法到算法需要完成数据表示（将数据模型存储到计算机的内存中）和数据处理（将问题求解的基本思路形成算法）；由算法到程序需要将算法的操作步骤转换为某种程序设计语言对应的语句。

算法用来描述问题的解决方案，是形式化的、机械化的操作步骤。将人的想法描述成算法可以说是利用计算机解决问题的最关键的一步，也就是从计算机的角度设想计算机是如何一步一步完成这个任务的，告诉计算机需要做哪些事，按什么步骤去做。一般来说，对不同解决方案的抽象描述产生了相应的不同算法，而不同的算法将设计出相应的不同程序，这些程序的解题思路不同，复杂程度不同，解题效率也不相同。

总之，算法研究是推动计算机技术发展的关键。时间（速度）问题是算法研究的核心。人们可能认为，既然计算机硬件技术的发展使得计算机的性能不断提高，算法的研究也就没有必

要了。实际上并非如此。计算机的功能越强大，人们就越想去尝试更复杂的问题，与此相应的，计算量也会相应地扩大。现代计算技术在计算能力和存储容量上的革命仅仅提供了计算更复杂问题的有效工具，无论硬件性能如何提高，算法研究始终是推动计算机技术发展的关键。实际上，我们不仅需要算法，而且需要“好”算法。可以肯定的是，发明（或发现）算法是一个非常有创造性和值得付出的过程。下面看几个例子。

1. 检索技术

20 世纪 50—60 年代，检索仅仅是在规模比较小的数据集合中发生。例如，编译系统中的标识符表，表中的记录个数一般在几十至数百这样的数量级。

20 世纪 70—80 年代，数据管理采用数据库技术，数据库的规模在 K 级或 M 级，检索算法的研究在这个时期取得了巨大的进展。

20 世纪 90 年代以来，Internet 引起计算机应用的急速发展，研究的热点也转向了海量数据的处理技术，而且数据驻留的存储介质、数据的存储方法以及数据的传输技术也发生了许多变化，这些变化使得检索算法的研究更为复杂也更为重要了。

近年来，智能检索技术成为基于 Web 信息检索的研究热点。使用搜索引擎进行 Web 信息检索时，一些搜索引擎前 50 个搜索结果中几乎有一半来自同一个站点的不同页面，这是检索系统缺乏智能化的一种表现。另外，在传统的 Web 信息检索服务中，信息的传输是按 Pull 模式进行的，即用户找信息。而采用 Push 方式，是信息找用户，用户想要获得自己感兴趣的信息无须进行任何信息检索，这就是智能信息推送技术。这些新技术的每一项重要进步都与算法研究的突破有关。

2. 压缩与解压缩

计算机的处理对象随着多媒体技术的发展也在发生改变，从原来的字符发展到图像、图形、音频、视频等多媒体数字化信息，这些信息数字化后，其特点就是数据量非常庞大。例如，音乐 CD 的采样频率是 44kHz，假定它是双声道，每声道占用 2 字节存储采样值，则 1 秒钟的音乐就需要 $44000\times2\times2\approx160$KB，存储一首 4 分钟长的歌曲，总计需要 $4\times60\times160\approx36$MB。而且，计算机总线无法承受处理多媒体数据所需的高速传输速度。因此，对多媒体数据的存储和传输都要求对数据进行压缩。MP3 压缩技术就是一个成功的压缩/解压缩算法，一个播放 3～4 分钟歌曲的 MP3 文件通常只需 3MB 左右的磁盘空间。

3. 信息安全与数据加密

计算机应用的发展也带来了许多隐患。一位酒店经理曾经描述了这样一种可能性：“如果我能破坏网络的安全性，想想你在网络上预订酒店房间所提供的信息吧！我可以得到你的名字、地址、电话号码和信用卡号码，我知道你现在的位置，将要去哪儿，何时去，我也知道你支付了多少钱，我已经得到足够的信息来盗用你的信用卡！”这是非常可怕的。足以看出，信息安全在电子商务中的重要性。而对需要保密的数据进行加密是保证信息安全的一个重要方法。因此，在这个领域，数据加密算法的研究是绝对必需的，其必要性与计算机性能的提高无关。

1.2 算法与程序

算法是对解题过程的描述，这种描述是建立在程序设计语言这个平台之上的。就算法的实现平台而言，可以抽象地对算法的定义如下：

算法＝控制结构＋原操作(对固有数据类型的操作)

算法是指令的有限序列，其中，每条指令表示一个或多个操作。算法有以下5个重要特征：

①有穷性。一个算法必须总是(对任何合法的输入值)在执行有穷步之后结束，且每一步都可在有穷时间内完成。

②确定性。算法中每一条指令必须有确切的含义，不会产生二义性。

③可行性。一个算法是可行的，即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。

④输入性。一个算法有零个或多个的输入。

⑤输出性。一个算法有一个或多个的输出。

所谓程序，就是一组计算机能识别与执行的指令。每一条指令使计算机执行特定的操作，用来完成一定的功能。计算机的一切操作都是由程序控制的，离开了程序，计算机将一事无成。从这个意义来说，计算机的本质是程序的机器，程序是计算机的灵魂。

那么，程序与算法之间存在怎样的关系呢？

算法是程序的核心。程序是某一算法用计算机程序设计语言的具体实现。事实上，当一个算法使用计算机程序设计语言描述时，就是程序。具体来说，一个算法使用C语言描述，就是C程序。程序设计的基本目标是应用算法对问题的原始数据进行处理，从而解决问题，获得所期望的结果。在能实现问题求解的前提下，要求算法运行的时间短，占用系统空间小。

初学者往往把程序设计简单地理解为编写一个程序，这是不全面的。一个程序应包括对数据的描述与对运算操作的描述两个方面的内容。著名计算机科学家尼克劳斯·沃思(Niklaus Wirth)曾提出这样一个公式：数据结构＋算法＝程序。其中，数据结构是对数据的描述，而算法是对运算操作的描述。实际上，一个程序除了数据结构与算法这两个要素之外，还应包括程序设计方法。一个完整的C程序除了应用C语言对算法的描述之外，还包括数据结构的定义以及调用头文件的指令。

由此可见，根据案例的具体情况确定并描述算法，并为实现该算法设置合适的数据结构，是求解实际案例时需要解决的问题。

1.3 算法的描述方式及设计方法

无论是面向对象程序设计语言，还是面向过程的程序设计语言，都是用3种基本结构(顺序结构、选择结构和循环结构)来控制算法流程的。每个结构都应该是单入口单出口的结构

体。结构化算法设计常采用自顶向下逐步求精的设计方法，因此，要描述算法首先需要有表示3个基本结构的构件，其次能方便支持自顶向下逐步求精的设计方法。

1.3.1 算法的描述方式

描述算法的方法很多，有的采用类 PASCAL，有的采用自然语言等。这里介绍常用的用于描述算法的 C 语言语句。

1. 输入语句和输出语句

```
scanf(格式控制字符串,输入项表);
printf(格式控制字符串,输出项表);
```

2. 赋值语句

```
变量名=表达式;
```

3. 条件语句

```
if<条件><语句>;
或者
if<条件><语句 1>else<语句 2>;
```

4. 循环语句

```
(1)while 循环语句
while 表达式
  循环体语句;
(2)do-while 循环语句
do
  循环体语句;
while 表达式;
(3)for 循环语句
for(赋初值表达式 1;条件表达式 2;步长表达式 3)
  循环体语句;
```

5. 返回语句

```
return(返回表达式);
```

6. 定义函数语句

```
函数返回值类型(类型名 形参 1,类型名 形参 2,…)
{
```

```
    说明部分;
    函数语句部分;
}
```

7. 调用函数语句

函数名(实参 1,实参 2,…)

在 C++语言中,在函数调用时实参和形参的参数传递分为传值和传引用(引用符号为"&.")两种方式。

①传值方式是单向的值传递。例如,有一个函数 $fun1(x,y)$(其中,x 和 y 为值形参),在调用 $fun1(a,b)$(其中,a 和 b 为实参)时,将 a 的值传给 x,b 的值传给 y,然后执行函数体语句,执行完函数后 x、y 的值不会回传给 a、b。

②传引用方式是双向的值传递。例如,有一个函数 $fun2(x,y)$(其中,x 和 y 为引用形参),在调用 $fun2(a,b)$(其中,a 和 b 为实参)时,将 a 的值传给 x,b 的值传给 y,然后执行函数体语句,执行完函数后 x、y 的值分别回传给 a、b。

例如,有如下程序:

```
#include<stdio.h>
void fun1(int x,int y)
{
  y=(x<0)? -x:x;
}
void fun2(int x,int &.y)
{
  y=(x<0)? -x:x;
}
void main()
{
  int a,b;
  a=-2;b=0;
  fun1(a,b);
  printf("fun1:b=%d\n",b);
  a=-2;b=0;
  fun2(a,b);
  printf("fun2:b=%d\n",b);
}
```

其中,$fun1(x,y)$、$fun2(x,y)$的功能都是计算 $y=|x|$,$fun1$ 中形参 y 使用传值方式,$fun2$ 中形参 y 使用传引用方式,也就是说,执行 $fun1(a,b)$时,b 实参的值不会改变,而执行 $fun2(a,b)$时,b 实参的值可能发生改变。程序执行结果如下:

```
fun1:b=0
fun2:b=2
```

为此，在设计算法（通常设计成C/C++函数）时，若某个形参需要将计算的值回传给对应的实参，则需将其设计为引用传递参数的方式，否则不必使用引用方式。

1.3.2 算法的设计方法

算法是能获得问题答案的指令序列。实际中存在着千奇百怪的问题，因而问题求解的方法也就各不相同，所以，算法的设计过程是一个充满智慧的灵活过程，它要求设计人员根据实际情况做出具体分析。在设计算法时，遵循图1-2所示的一般过程可以在一定程度上指导算法的设计。

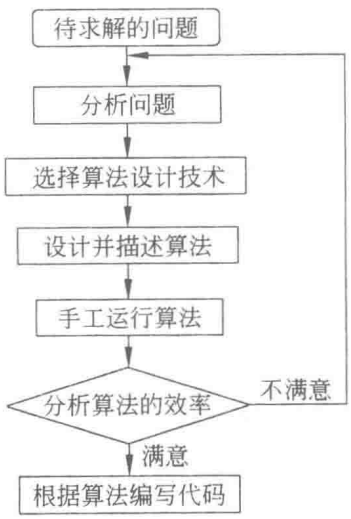

图1-2 算法设计的一般过程

1. 分析问题

对于待求解的问题，要弄清以下问题：求解的目标是什么？已经给出了哪些已知信息、显式条件或隐含条件？计算结果应当使用哪种形式的数据来对其进行表达？准确地理解算法的输入是什么，明确要求算法做的是什么，即明确算法的入口和出口，这是设计算法的切入点。若没有对问题进行全面、准确和认真的分析，就会导致事倍功半，造成不必要的反复，甚至留下严重隐患。

2. 选择算法设计技术

算法设计技术，也称为算法设计策略，是设计算法的一般性方法。使用它能够解决许多不同计算领域的多种问题。本书讨论的算法设计技术是已经被证明对算法设计非常有用的通用

技术，这些算法设计技术构成了一组强有力的工具，在为新问题（即没有令人满意的已知算法可以解决的问题）设计算法时，可以运用这些技术设计出新的算法。

3. 设计并描述算法

在构思和设计了一个算法之后，要对所设计的求解步骤进行清晰准确的记录，即描述算法。

4. 手工运行算法

因为计算机只会执行程序，而不会理解动机，所以由计算机是无法检测出逻辑错误的。经验和研究都表明，发现算法（或程序）中的逻辑错误的重要方法就是手工运行算法，即跟踪算法。跟踪者要像计算机一样，用一个具体的输入实例手工执行算法，并且这个输入实例要最大可能地暴露算法中的错误。即使有几十年经验的高级软件工程师，也经常利用此方法查找算法中的逻辑错误。

5. 分析算法的效率

算法效率体现在两个方面：其一为时间效率，它显示了算法运行得有多快；其二为空间效率，它显示了算法需要多少额外的存储空间。相比而言，算法的时间效率是我们关注的重点。事实上，计算机的所有应用问题，包括计算机自身的发展，都是围绕着“时间—速度”这样一个中心进行的。一般来说，一个好的算法首先应该是比同类算法的时间效率高，算法的时间效率用时间复杂性来度量。

6. 实现算法

现代计算机技术还不能将伪代码形式的算法直接“输入”进计算机中，而需要把算法转变为特定程序设计语言编写的程序。在把算法转变为程序的过程中，虽然现代编译器提供了代码优化功能，然而一些技巧还是会被用到的，例如，在循环之外计算循环中的不变式、合并公共子表达式、用开销低的操作代替开销高的操作等。一般来说，这样的优化对算法速度的影响是一个常数因子，程序可能会提高10%～50%的速度。

最后，需要强调的是，一个好算法是需要不断反复努力和重新修正才会得到的。也就是说，一个看上去再完美的算法，它还是有改进的空间的。其改进需要不断重复上述问题求解的一般过程，直到算法满足预定的目标要求。

1.4 算法的分析

算法分析主要包括两个方面的问题，即分析算法的时间复杂度和空间复杂度。其目的不是分析算法是否正确或是否容易阅读，主要是考察算法的时间和空间效率，以求改进算法或对不同的算法进行比较。一般情况下，运算空间（内存）较为充足，不需要过多考虑，重点是对算法的时间复杂度进行分析。

算法的执行时间主要与问题规模有关。问题规模是一个和输入有关的量，例如，数组的元素个数、矩阵的阶数等。所谓一个语句的频度，即指该语句在算法中被重复执行的次数。算法中所有语句的频度之和记作 $T(n)$，它是该算法所求解问题规模 n 的函数，当问题的规模 n 趋向无穷大时，$T(n)$ 的数量级称为渐近时间复杂度，简称为时间复杂度，记作 $T(n)=O(f(n))$。

上述表达式中“O”的含义是 $T(n)$ 的数量级，其严格的数学定义是：若 $T(n)$ 和 $f(n)$ 是定义在正整数集合上的两个函数，则存在正的常数 C 和 n_0，使得当 $n\geqslant n_0$ 时，总是满足 $0\leqslant T(n)\leqslant C\cdot f(n)$。但是应总是考虑在最坏情况下的时间复杂度，以保证算法的运行时间不会比它更长。

另外，由于算法的时间复杂度主要是分析 $T(n)$ 的数量级，而算法中基本运算的频度与 $T(n)$ 同数量级，所以通常采用算法中基本运算的频度来分析算法的时间复杂度，被视为算法基本运算的一般是最深层循环内的语句。

用数量级形式 $O(f(n))$ 表示算法执行时间 $T(n)$ 的时候，函数 $f(n)$ 通常取较简单的形式，如 1、$\log_2 n$、n、$n\log_2 n$、n^2、n^3、2^n 等。在 n 较大的情况下，常见的时间复杂度之间存在下列关系：

$$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<(n^3)<O(2^n)$$

1.5 最优算法

算法是问题的解决方法，针对一个问题可以设计出不同的算法，不同算法的时间复杂性也可能存在一定的差异。能否确定某个算法是求解该问题的最优算法？是否还存在更有效的算法？若我们能够知道一个问题的计算复杂性下界，也就是求解该问题的任何算法（包括尚未发现的算法）所需的时间下界，就可以较准确地评价解决该问题的各种算法的效率，进而确定已有的算法还有多少改进的余地。

1.5.1 问题的计算复杂性下界

求解一个问题所需的最少工作量就是该问题的计算复杂性下界，求解该问题的任何算法的时间复杂性都不会低于这个下界，通常采用大 Ω（读做大欧米伽）符号来分析某个问题或某类算法的时间下界。例如，已经证明基于比较的排序算法的时间下界为 $\Omega(n\log_2 n)$，那么，不存在基于比较的排序算法，其时间复杂性小于 $O(n\log_2 n)$。

若存在两个正的常数 c 和 n_0 的话，对于任意 $n\geqslant n_0$，都有 $T(n)\geqslant c\times g(n)$，则称 $T(n)=\Omega(g(n))$（或称算法在 $\Omega g(n)$ 中）。

大 Ω 符号用来描述增长率的下限，表示 $T(n)$ 的增长至少像 $f(n)$ 增长的那样快。与大 O 符号对称，这个下限的阶越高，相应的，结果就越有价值。大 Ω 符号的含义如图 1-3 所示。

对于任何待求解的问题，若能找到一个尽可能大的函数 $g(n)$（n 为输入规模），使得求解该问题的所有算法都可以在 $\Omega(g(n))$ 的时间内完成，则函数 $g(n)$ 就是该问题的计算复杂性下界。若已经知道一个和下界的效率类型相同的算法，则称该下界是紧密的。

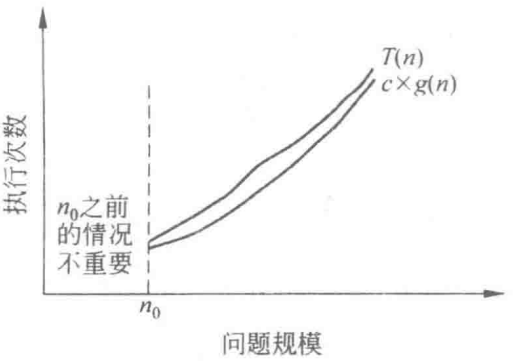

图 1-3　大 Ω 符号的含义

通常情况下，大 Ω 符号与大 O 符号配合以证明某问题的一个特定算法是该问题的最优算法，或是该问题中的某算法类中的最优算法。一般情况下，若能够证明某问题的时间下界是 $\Omega(g(n))$，则对以时间 $O(g(n))$ 来求解该问题的任何算法，都认为是求解该问题的最优算法。

例 1.1　如下算法实现在一个数组中求最小值元素，证明该算法是最优算法。

```
int ArrayMin(int a[],int n)
{
int min=a[0];
for(int i=1;i<n;i++)
  if(a[i]<min) min=a[i];
return min;
}
```

证明：在这个算法中，需要进行的比较操作共计 $n-1$ 次，其时间复杂性是 $O(n)$。下面证明对于任何 n 个整数，求最小值元素至少需要进行 $n-1$ 次比较，即该问题的时间下界是 $\Omega(n)$。

将 n 个整数划分为三个动态的集合 A、B 和 C，其中，A 为未知元素的集合，B 为已经确定不是最小元素的集合，C 是最小元素的集合，任何一个通过比较求最小值元素的算法都要从三个集合为 $(n,0,0)$（即 $|A|=n, |B|=0, |C|=0$）的初始状态开始，经过运行，最终到达 $(0, n-1, 1)$ 的完成状态，如图 1-4 所示。

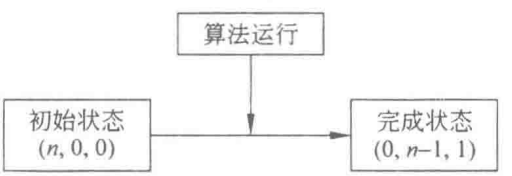

图 1-4　通过比较求最小值元素的算法

从本质上看来，这个过程是将元素从集合 A 向 B 和 C 移动的过程，但每次比较，至多能把一个较大的元素从集合 A 移向集合 B，因此，任何求最小值算法至少要进行 $n-1$ 次比较，其时间下界是 $\Omega(n)$。所以，算法 ArrayMin 是最优算法。

确定和证明某个问题的计算复杂性下界难度都相当的大，因为这涉及求解该问题的所有算法，而枚举所有可能的算法并加以分析，显然是不可能的。事实上，存在大量问题，它们的下界是不清楚的，大多数已知的下界要么是平凡的，要么是在忽略某些基本运算（如算术运算）的意义上，应用某种计算模型（如判定树模型）推导出来的。

1.5.2 平凡下界

确定一个问题的计算复杂性下界的简单方法如下：对问题的输入中必须要处理的元素进行计数，同时，对必须要输出的元素进行计数。因为任何算法至少要“读取”所有要处理的元素，并“写出”它的全部输出，这种计数方法产生的是一个平凡下界。例如，任何生成 n 个不同元素的所有排列对象的算法必定属于 $\Omega(n!)$，因为输出的规模是 $n!$；计算两个 n 阶矩阵乘积的算法必定属于 $\Omega(n^2)$，因为算法必须处理两个输入矩阵中的 n^2 个元素，并输出乘积中的 n^2 个元素。

无须借助任何计算模型或进行复杂的数学运算，平凡下界即可推导出来，但是平凡下界往往过小而意义不大。例如，TSP 问题的平凡下界是 $\Omega(n^2)$，因为对于 n 个城市的 TSP 问题，问题的输入是 $n(n-1)/2$ 个距离，问题的输出是构成最优回路的 $n+1$ 个城市的序列，而这个平凡下界是没有任何实际意义的，因为 TSP 问题至今还没有找到一个多项式时间算法。

1.5.3 判定树模型

许多算法的工作方式都是对输入元素进行比较，例如，排序和查找算法，因此可以用判定树来研究这些算法的时间性能。判定树是满足如下条件的二叉树：

①每一个内部结点都和一个形如 $x \leqslant y$ 的比较保持对应关系，若关系成立，则控制转移到该结点的左子树，否则，控制转移到该结点的右子树。

②每一个叶子结点表示问题的一个结果。在用判定树模型建立问题的时间下界时，通常求解问题的所有算术运算都会被忽略，只考虑执行分支的转移次数。

需要注意的是，判定树中叶子结点的个数可能大于问题的输出个数，因为对于某些算法，不同的比较路径可能得到的输出是相同的。但是，判定树中叶子结点的个数必须至少和可能的输出一样多。对于一个问题规模为 n 的输入，算法可以沿着判定树中一条从根结点到叶子结点的路径来完成，比较次数等于路径中经过的边的个数。

例 1.2 用判定树模型求解排序问题的时间下界。

解：基于比较的排序算法是通过对输入元素两两比较进行的，可以用判定树来描述完整的比较过程。例如，对三个元素进行排序的判定树如图 1-5 所示，判定树中每一个内部结点代表一次比较，每一个叶子结点表示算法的一个输出。显然，最坏情况下的时间复杂性不超过判定树的高度。

由判定树模型不难看出，可以把排序算法的输出解释为对一个待排序序列的下标求一种排列，这样一来序列中的元素就会按照升序进行排列。例如，待排序序列是 $\{a_1, a_2, a_3\}$，则下标的一个排列 321 使得输出满足 $a_3 < a_2 < a_1$，且该输出对应判定树中一个叶子结点。因此，