电子计算机软件

算法设计与分析

国防科学技术大学
曹新谱编著

湖南科学技术出版社

电 子 计 算 机 软 件

算 法 设 计 与 分 析

国 防 科 学 技 术 大 学
曹 新 谱 编 著

湖 南 科 学 技 术 出 版 社

《算法设计与分析》内容简介

如何设计各种高效率的算法及评价一类算法的时空复杂性，是计算机科学工作者必须研究的核心课题，具有极大的应用价值和理论价值。

算法的研究范围极广。本书主要介绍计算机科学领域及应用领域常见的有代表性的非数值算法及算法设计的若干重要方法，同时，介绍算法分析的基本知识。全书共分十章。第一章阐述计算模型和时空复杂性的定义。第二章讨论递归技术和算法分析的基本手段。第三至九章分别介绍算法设计的几类方法，如分治法、回溯法、贪心法、动态规划法、分枝限界法等，并结合某些有实用意义的经典算法来加深设计方法的探讨，由浅入深地进行算法效率分析，使读者在掌握各种算法设计方法的同时，掌握算法分析的基本技术和某些技巧。第十章介绍NP-完全问题，为进一步研究算法理论奠定基础。

全书内容丰富，理论与实际并重。适合于用作大专院校计算机科学与工程专业学生、研究生教材，亦适合于计算机设计和应用单位的广大科技人员参考。

电子计算机软件
算法设计与分析
国防科学技术大学　曹新谱编著
责任编辑：周翰宗
*
湖南科学技术出版社出版
（长沙市展览馆路14号）
湖南省新华书店发行　湖南省新华印刷二厂印刷
*
1984年11月第1版第1次印刷
开本：787×1092毫米 1/16　印张：13.75　字数：341,000
印数：1——14,200
统一书号：15204·128　定价：2.20元

前　言

算法研究是计算机科学的核心问题之一。《算法设计与分析》是计算机系软、硬件专业的基础课程。通过对计算机领域中许多常见而有代表性算法的研究，理解和掌握算法设计的主要方法，培养对算法复杂性进行正确分析的能力，为独立地设计算法和对给定算法进行复杂性分析奠定坚实的知识基础。对从事计算机系统结构、系统软件和应用软件的同志是非常重要和必不可少的。

计算机算法包括数值算法和非数值算法两大部分。由于数值算法通常在《数值分析》中介绍得较多，故本书侧重讨论非数值算法。如果按处理问题的方式来分，算法又可分为串行算法和并行算法两大类。当然，现在的许多算法实际上可能既有串行处理，又有并行处理。因为串行处理不仅是并行处理的基础，而且便于进行算法分析，所以本书只介绍串行算法。

这是一本供计算机工作者，尤其是软件工作者阅读的专业参考书。实践证明，也适合作为计算机专业的教材。全书共分十章。讲授全书约需一百个学时左右。但对不同专业、不同程度的学生，可以抽出一个适当的子集以满足不同的教学要求。

第一章介绍计算机模型和算法复杂性的测度，是算法分析的理论基础。

第二章介绍算法设计和算法分析中常用的一些知识，数据结构是先导课程。对较简单的数据结构，如表、队列、堆栈等，读者已较为熟悉，不再赘述。但递归技术、递归方程与生成函数等内容，是较重要的基本知识，不可忽视，应加深理解与掌握。

从第三章开始介绍各种算法设计方法。其中分治法是设计有效算法的重要方法之一，也是必须掌握的方法。

第四章介绍一类应用范围很广的算法——各种排序算法。重点是较系统的算法复杂性分析，与数据结构中的排序内容无多大重复。

第五章是动态规划算法。介绍了有代表性的六个专题的动态规划及一些具体应用。

第六章介绍贪心法，这也是一种重要的算法设计方法。按贪心法设计的许多算法，往往能导出问题的最佳解。其中也有许多典型的问题和典型的算法可供学习和使用。

七、八两章介绍的算法设计方法对于处理某些难解问题，是各具特色的。其基本思想是值得学习和掌握的。

第九章讲述了几个最基本的图的算法。如果读者已在其它课程中学习过这方面的知识，可以忽略。但§9.4和§9.5的内容有一定特色，其设计思想有独到之处，值得一读。

第十章介绍NP-完全问题。首先介绍了确定型图灵机和非确定型图灵机，然后证明了COOK定理。理论性较强，难度较大。对高年级学生或研究生可能比较适宜。

关于算法描述语言，考虑到众多读者已有的语言知识基础和习惯，认为采用一种类ALGOL-PASCAL语言较为适宜。这样在学习时易读易懂，比使用任何特定的高级语言都会要好些。

为了加深对知识的理解，各章配备有难易适当的习题，以适应不同程度读者练习的需要。教学实践证明，这种练习对开发智能很有好处。

因笔者的知识和写作水平有限，书稿虽几经修改，仍难免差错。热切地希望诸位学者和读者批评指正，如蒙赐教，则不胜感激！

本书的出版要衷心感谢陈火旺教授，他在繁重的工作之余不辞辛劳，主审了全书。上海科技大学顾训穰同志曾经协助审校过大部分初稿。吴明霞同志也协助审阅了部分章节并提出了一些有益建议。余祥宣、郑若忠、王鸿武、王广芳、徐绪松、李庆、陈增武等同志对本书的出版提出了建设性意见并给予有力的支持。此外，还有许多同志对本书的修改也提出过许多宝贵意见，恕未一一列举，在此一并致以谢忱。

曹新谱

一九八三年十月于长沙

目　录

第一章　计算模型和计算复杂性的测度

首先，我们要介绍算法的概念。其次，做为算法设计和算法分析的基础，要讨论一个简单而适用的计算模型"随机存取机器"。它与实际的计算机有许多共同之处，但比实际机器更简单、更理想。它对于算法设计，特别是对于算法分析，是十分有用的。一个算法的计算复杂性可以在这一模型的基础上严格地、定量化地建立起来。此外，还要介绍一种算法描述语言——类ALGOL-PASCAL语言。它是我们以后各章中用于描述算法的工具语言。

§1.1　引　　言

算法(*algorithms*)的研究是计算机科学的核心课题之一。早在电子计算机问世以前，就有人开创了算法的研究，并创造了许多有效的算法。如欧几里德算法(求两数的最大公因子)和孙子算法(求若干个数的最小公倍数)等等。特别是1946年以后，由于快速计算工具电子计算机的出现和迅猛发展，使算法的研究取得了空前的进展。诸如六十年代后半期产生的快速富里叶变换(FFT)等高效算法。到了七十年代，随着大规模集成电路的出现和计算机的换代更新，算法的研究又发生了一次飞跃，产生了与并行处理机相适应的并行算法，从而使很多原来难于处理的问题得以迎刃而解。据不完全统计，从三十年代到七十年代中期，有关算法研究的资料和文献，产生于五十年代以前的，约占文献总量的10%左右；产生于六十年代的约占文献总量的30%左右；其余约60%则产生于七十年代前半期(至1976年为止)。目前，算法的研究仍方兴未艾。这个领域之所以令人感兴趣，是因为它所涉及的范围十分广泛。不论是从事计算机硬件设计(如计算机部件设计，系统设计或网络设计等)，还是从事计算机软件设计（如设计系统软件和各种应用软件)，都需要认真研究算法。从理论研究到各种实际应用，无一处不与算法有密切关系。由于有关算法研究的材料如此丰富，要想把所有有价值的算法及算法分析的内容都收集起来，在一本书中详细阐述几乎是不可能的，所以只能讲述最基本的方法和一些应用广泛且有代表性的结果。目的在于使读者能学习算法设计的一般原理和算法分析的主要手段，为今后开展工作和进行更深入的研究打下良好的基础。

关于算法的严格定义，如果建立了图灵可计算的概念之后，是可以用图灵机来形式地描述的。在此，只能先给出"算法"概念的一个非形式的描述。

简而言之，一个算法是一个有穷规则的有序集合。这些规则确定了解决某一类问题的一个运算序列。对于某一类问题的任何初始输入，它能机械地一步一步地计算，通过有限步之后，计算终止，并产生一个输出。

算法有如下五大特征：

1.有穷性　一个算法在执行有穷个计算步后必须终止。

2.确定性　一个算法中给出的每一个计算步，必须是精确地定义的、无二义性的。

3.能行性　算法中要执行的每一个计算步都是可以在有限时间内做完的。能行性与有穷性和确定性是相容的。

4.输入　一个算法一般都要求一个或多个输入信息（有的算法可能不要求输入信息)，这

些输入量是算法所需的初始数据，它们取自某一特定的集合。

5.输出 一个算法一般有一个或多个输出信息。它们通常可以被解释为“对输入的计算结果”。

例如，给定两个正整数m和n，考虑求它们的最大公因子(m，n)的问题。

这个问题通常用所谓辗转相除法求解。这一方法在西方称做欧几里得(Euclid)算法。我国数学家秦九韶在《数书九章》(1247年)中记载了这个方法。现用下面三个计算步来描述这个算法：

1.〔**求余数**〕 以n除m，令r是所得余数，$0 \leqslant r < n$，执行步骤2。

2.〔**判余数是否等于零**〕 如果r＝0，输出n的当前值，算法结束；否则执行第3步。

3.〔**更新被除数和除数**〕 作m←n，n←r，转入第一步。

图1.1是这个算法的计算流程图。过程E uclid 是用高级语言PASCAL编写的实现这个算法的程序。

辗转相除法的PASCAL程序

```
procedure  EUCLID
  var
  M, N, R, integer
  begin
    read (M, N);
    while  N≠0  do
      begin
        R←—M  mod  N;
        M←—N;
        N←—R
      end;
    write (M)
  end
```

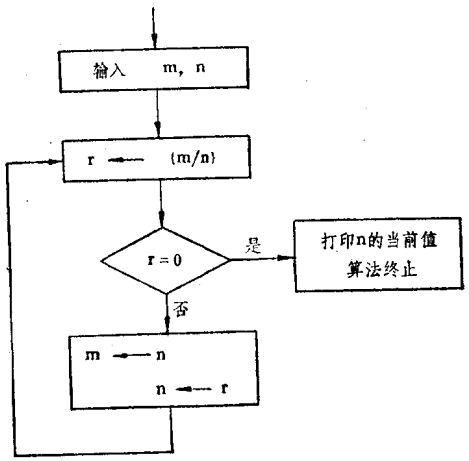

图1.1 Euclicl算法流程图

上述计算过程规定了三个计算步骤，而且每个计算步骤的意义很明确，也是能行的。虽然第三步要返回到第一步，出现一个循环，因为m和n都是给定的有限数，每次相除后得到的余数r若不为0，总有$r < min(m, n)$。这就保证了经过有限次循环后，总有r＝0的情况发生。这时循环就会终止，而且产生一个输出。所以上述计算过程是一个算法。

当然，一个算法并不等同于一个计算机程序或一个过程，只不过是类似于过程、方法、规程、程序等。算法与过程的区别在于：对任何合法输入，算法要在有限的时间内通过有穷步计算后终止；而过程则可以是有穷的，也可以是无穷的。无穷过程永远不会终止。而算法与程序的区别则在于：一个计算机程序往往是指用某种机器语言所书写的一个计算过程，但一个特定算法不一定表现为一个计算机程序，二者不一定有某种必然联系。一个算法可以采用多种方式描述，如图解描述、语言描述等。如果用语言描述一个算法，既可以用各种计算机语言描述（这时就表现为一个程序），也可以用人类的自然语言描述。此外，一个计算机程序一般只能在某些机器上执行；而一个算法，既可以编成程序在各种计算机上执行，也可以用纸和笔手工执行，甚至可以用算盘，算筹或其它工具执行。因此，算法具有描述的多样性和多种方式实现。当然，我们最感兴趣的还是用计算机语言来描述和在计算机上执行各种算法。

算法设计和算法分析的步骤可概括为：

1.问题的陈述　为了设计求解某一问题的算法，首先必须了解问题的实质。即：已知条件是什么？要求回答什么？一个问题的正确描述应当使用科学的语言把所有已知条件和需要的答案陈述清楚。

2.模型的选择或拟制　当问题陈述清楚后，选择或拟制描述问题的数学模型是非常重要的工作。在某些情况下，人们可以根据自身的经验，从已知的数学模型中选择一些能方便地描述需要求解的问题的数学模型。模型适当与否影响算法设计的速度和算法的效率。模型选择无公式可循，取决于设计者的知识结构、工作经验……。设计者应当十分重视模型的选择，宁可多下些功夫来选择或拟制一个适合当前问题的数学模型，才能为后续工作的顺利开展铺平道路。

3.算法设计和正确性证明　数学模型一经选定，就可进行算法设计。虽然现在已经形成了一系列较为有效的算法设计方法，但永远也不会有一种适用于解决一切算法设计课题的万能的算法设计方法，算法设计始终是一种复杂的、艰苦的创造性劳动，要求设计者充分发挥主观能动性，充分运用已有知识和抽象思维，从千头万绪之中，逐渐形成算法的基本思想，勾勒出一个算法的各个具体步骤，此过程不仅与模型的选择，而且与算法正确性证明紧密相关。因为在设计算法时，最重要的问题莫过于算法的正确性。当然，多学习一些典型的算法设计方法，掌握一些基本算法的思想，是设计新算法的知识基础。

算法的正确性证明与程序的正确性证明一样，是算法理论研究中很引人注目的一部分。一个算法的正确性证明，就是要证明对于一切合法输入，算法都能在有限次计算后产生正确的输出。这实际上是一种穷举证明法。当一个算法的输入很庞大时，实际上是很难做到的。特别是当算法的合法输入是一个无穷集时，穷举证明是不可能的。另一种证明方法是将一个算法的输入和输出都表示成“输入断言”和“输出断言”，一个算法的各计算步可以表示成一组“谓词演算”规则。论证通过这一组谓词演算，能由输入断言导出输出断言。这种形式演绎过程是很冗长繁复的，它们往往比一个算法（或程序）本身还要长得多。由于篇幅的限制，我们对算法的正确性不采用严格的形式论证，而只给出非形式的直观解释。

4.算法的程序实现　将一个算法正确地编写成一个机器程序，叫做算法的程序实现。将给定的算法正确地转换成一个程序，并不是一件简单的工作。这一转换过程要求程序设计者有很好的程序设计基础，掌握多种程序设计方法和技巧。这样，在实现由算法到程序的转换时，才可以避免算法走样，使程序有较高的运行效率。怎样判定一个程序正确地反映了某一算法（没有走样）呢？这是“程序证明论”中要讨论的问题。一个算法的程序实现，也许还应包括程序的测试和调试、文件的编制等方面，但这些都不是本课程所要解决的问题。

5.算法分析　算法设计和算法分析虽然有必然联系，但毕竟是两个不同的任务。算法设计的中心任务是，对于现实生活中提出的某些问题，如何设计出一个求解的算法。至于算法的优劣，在某种意义上则不必深究。而算法分析恰恰是要研究各种算法的特性和好坏。对于解决同一类问题，或许可以设计出若干个算法，人们自然会问：如何判定这些算法的优劣，标准是什么？这正是算法分析要解决的问题。

§1.2　计算复杂性的测度

算法的**计算复杂性**(*computational complexity*)是衡量算法计算难度的尺度，人们可以从不同的角度来衡量和评价。使用最普遍的评价标准是一个算法需要耗费的时间和空间。如

果一个算法所需要的时间或空间，比另一个求解同一问题的算法所需要的时间或空间少，就可以说这一个算法比另一个算法好。无论是从时间方面考虑还是从经济效益等方面考虑，人们总是希望使用效率高（消耗的时间或空间较少）的算法。

当我们涉及一个具体问题时，不可避免地要谈到这个问题的所谓**体积**（*size*），或者称做问题的**"大小"**。一个问题的体积或大小可以用一个整数来表示，它是对问题的输入数据（或初始数据）的多少的一种量度（有时是初始数据的大小的量度）。例如，矩阵乘法问题的大小可以用矩阵的阶数来定义，也可以用矩阵的元素个数来度量。一个图论问题的体积可以定义为这个图的边数等等。

定义 如果一个问题的体积是n，解这一问题的某一算法所需的时间为T(n)，它是n的某一函数。T(n)称做这一算法的**"时间复杂性(*time complexity*)"**。当输入量n逐渐增大时，时间复杂性的极限情形称做算法的**"渐近时间复杂性(*asymptotic time complexity*)"**，类似地可以定义一个算法的**"空间复杂性**（*space complexity*）**"**和**"渐近空间复杂性**（*asymptotic space complexity*）**"**。不过，对空间复杂性的讨论不如对时间复杂性的讨论那么广泛。有许多问题，人们主要是研究其算法的时间复杂性而很少讨论它们的空间耗费。

关于时间复杂性，人们最感兴趣的问题是算法的**"渐近时间复杂性"**。因为要确定一个算法所能解决的问题的体积，主要依赖于对算法的渐近时间复杂性的分析。如果对于某个常数$c>0$，一个算法能在cn^2的时间内处理完大小为n的输入，就说这个算法的时间复杂性是$O(n^2)$，读做"n平方级"的。严格地说：一个函数g(n)是$O(f(n))$，当且仅当存在一个常数$c>0$和一个$n_0\geqslant 0$，对一切$n\geqslant n_0$，有$g(n)\leqslant cf(n)$成立。有时亦称函数g(n)以函数f(n)为界或者说g(n)囿于f(n)。

一个算法的复杂性函数的量级是反映算法性能的重要标准。对于某一问题而言，如果算法复杂性函数的量级越低，说明算法的效率越高。也许有人会提出这样的看法：现代数字计算机的发展是如此之快，计算速度的增长如此惊人，算法效率的高低无足轻重。事实正好相反：在信息爆炸的时代，人们需要处理的数据量越来越大。算法效率的高低对所能处理的数据量的多少有决定性作用。当输入量急剧地增大时，如果没有高效率的算法，单纯依靠提高计算机的速度，有时是很不理想的。请看下面的实例。

设有五个算法A_1，A_2，A_3，A_4，A_5，它们的时间复杂性函数如右表所示：

算法	时间复杂性函数
A_1	$T_1(n)=n$
A_2	$T_2(n)=n\log_2 n$
A_3	$T_3(n)=n^2$
A_4	$T_4(n)=n^3$
A_5	$T_5(n)=2^n$

表中，一个算法的时间复杂性是它处理完一个大小为n的输入所需要的单位时间数。当取单位时间为一毫秒时，显然，在一秒钟里，算法A_1可以处理完一个大小为1000的输入，而算法A_5却做不到这一点。因为A_5的复杂性量级高，它只能处理大小为9的输入。表1.1给出了在一秒钟、一分钟或一小时的时间内，这五个算法所能解决问题的输入量的上界。从这里我们不难得到一个直观的概念：由于算法时间复杂性的量级不同，它们在相同的时间里所能解决的问题大小相差是极其悬殊的。

当计算机的速度成倍提高时，比如假定下一代计算机的速度比当代计算机的速度快十倍或快一万倍，来分析一下上面这五个算法所能处理的输入量的大小有何变化？表1.2说明了由于计算机速度的提高给每个算法所带来的处理能力的改变情况。算法A_1和A_2在计算机的速度提高十倍或一万倍后，同一时间里所能处理的输入量几乎也增加十倍或一万倍；而算法A_3

表1.1　　在某一时间里不同算法所能处理的输入量上界

算　法	时间复杂性	一秒钟内所能处理的最大输入量	一分钟内所能处理的最大输入量	一小时内所能处理的最大输入量
A_1	n	1000	6×10^4	3.6×10^6
A_2	$n\log_2 n$	140	4893	2.0×10^5
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

表1.2　　计算机速度提高十倍和一万倍的效果

算　法	时间复杂性	速度提高前单位时间里所能处理的数据量	速度提高十倍后单位时间里所能处理的数据量	速度提高一万倍后单位时间里所能处理的数据量
A_1	n	S_1	$10S_1$	$10000S_1$
A_2	$n\log_2 n$	S_2	对大的S_2，接近$10S_2$	当$\log_2 S_2 \geqslant 9\log_2 9000$时超过$9000S_2$
A_3	n^2	S_3	$3.16S_3$	$100S_3$
A_4	n^3	S_4	$2.15S_4$	$21.54S_4$
A_5	2^n	S_5	$S_5+3.32$	$S_5+13.32$

和A_4就差些，最令人沮丧的是算法A_5，即使是计算机的速度提高一万倍，算法A_5在某一时间内所能处理的输入量不过比原来增加了13个左右，真是寥寥无几。对照表1.1，不难推断出这样的结果：对算法A_6而言，一台高速计算机（它的速度是某台低速计算机的一万倍），在一分钟内所能处理完的输入量不过是一台低速计算机（它的速度是那台高速计算机的万分之一）一分钟内所能处理完的输入量的两倍（这里假定取一毫秒做时间单位）。

现在不考虑计算机速度的增长而考察使用更有效的算法的效果（仍以一分钟为基础来讨论）。从表1.1不难看出，如果用算法A_3来代替算法A_4，可以解比原来大6倍的问题；如果用A_2来代替A_4，可以解比原来大125倍的问题。这些结果比计算机的速度提高十倍百倍所得的收效更吸引人。如果以一小时为基础来讨论，效果将更显著。由此可见，算法的复杂性函数是衡量算法效率高低的重要尺度。改进算法的复杂性能大大提高计算机处理问题的能力。可以预见，随着计算机速度的不断提高和要求处理的信息量的急剧增长，高效率算法的设计变得更为重要。尽管提高计算机的速度也能提高处理问题的能力，如果不能设计出更高效率的算法，计算机的处理能力将会受到严重影响。

对于算法A_5，为什么计算机速度提高一万倍仍收效甚微呢？这是因为算法A_5的时间复杂性函数是输入量n的指数函数，而算法A_1、A_2、A_3、A_4的渐近时间复杂性都是以多项式函数为界的。特别是算法A_1，它的渐近时间复杂性是输入量n的线性函数。在这种情况下，随着计算机速度的提高，机器在同一时间里所能处理的输入量可以成倍地增长。计算机科学家们普遍认为：如果一个算法的时间复杂性是以多项式为界的，则认为这个算法是一个有效算法。也就是说，对于一个有效算法，哪怕输入量很大，现代的计算机也许还可以处理得了。如果一个算法的时间复杂性是输入量n的指数函数（且底大于1），则认为这是一个低效率的算法。对于一个时间复杂性是指数函数的算法，只要问题的输入量稍微大一些，现代计算机就处理不了。图1.2清楚地表明了这种趋势。

一个问题的计算复杂性的下界，是这个问题本身所固有的。例如，用比较判别法从n个整数中找出最大的数，其计算复杂性下界为$(n-1)$次比较。如果某一算法的复杂性已达到了该

问题的计算复杂性的下界，算法不可能进一步改进。达到下界的算法称做**最佳算法**。但是，一个问题的固有计算难度或复杂性的下界是很难确定的。

现实生活中要“计算”的问题是纷繁复杂的。是否一切要求“计算”的问题都可以设计出算法呢？回答是否定的。按算法论和计算复杂性的观点，现有的问题大致可分为三大类型：

1.无法写出算法的问题　如“哥德巴赫猜想”、“费马猜想”等问题都属于这一类。

2.有以多项式为界的算法存在的问题(P类问题)这一类问题是大量存在的，如前例所述求m和n的最大公因数、求两个矩阵的乘积等问题都属于这一类。

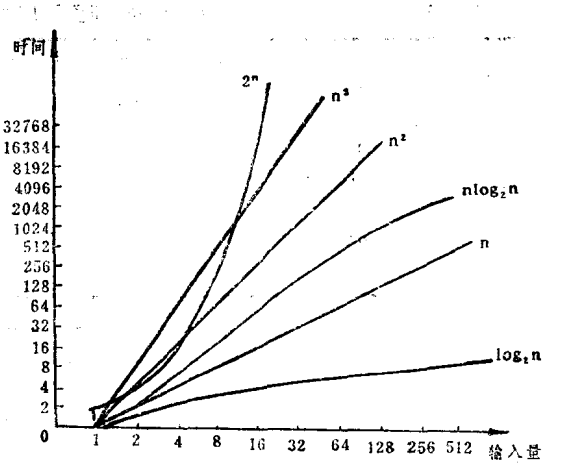

图1.2　常见函数的增长率

3.介于前两类问题之间的问题　这些问题一般可以写出算法（原则上可解），但算法的时间复杂性往往是输入量n的指数函数（底大于1）。当输入量n较大时，现代计算机实际上计算不了，因为它们所消耗的时间（或空间）太多。有一类被人们称做“NP——完全”的问题就属于这一类。这一类问题中包含了许多很有研究价值的实际问题。例如货郎担问题（*traveling salesman problem*）、整数分划问题和顶点覆盖问题等。到目前为止，对这些问题还没有设计出多项式时间为界的算法。因此，人们有理由怀疑它们不存在多项式时间为界的算法。

在研究算法复杂性时，虽然算法的渐近增长率的数量级是判定算法好坏的一个重要标准，但也不应忽视：一个计算复杂性量级较高的算法可能比一个计算复杂性量级较低的算法带有一个更小的常数因子。在这种情况下，当问题的输入量较小时，计算复杂性量级高的算法或许反而比计算复杂性量级低的算法要好。因此，研究时不仅要注意复杂性函数的量级，而且要重视所包含的常数因子。例如，设算法 A_1、A_2、A_3、A_4和A_5 的时间复杂性函数实际上是：

$$T_1(n) = 1000n;$$
$$T_2(n) = 100n\log_2 n;$$
$$T_3(n) = 10n^2;$$
$$T_4(n) = n^3;$$
$$T_5(n) = 2^n。$$

则当：　$2 \leqslant n \leqslant 9$时，$A_5$比$A_1$、$A_2$、$A_3$、$A_4$都好；

$10 \leqslant n \leqslant 58$时，$A_3$比$A_1$、$A_2$、$A_4$、$A_5$都好；

$59 \leqslant n \leqslant 1024$时，$A_2$比其它算法好；

$n \geqslant 1024$时，算法A_1最好。

§1.3　随机存取模型

为了进一步讨论算法的复杂性，必须选择执行算法的计算模型。一个计算模型应该对每一个基本的计算步有严格的定义。遗憾的是在现实生活中，任何计算机的字长总是很有限的，

人们不能不考虑计算或存储一个充分大的数值的困难。目前还没有一台计算机的字长能容纳下一个任意大小的整数，否则问题就好办多了。因此，必须选择一个能正确反映计算机的计算能力和计算速度的模型，才有现实意义和典型意义。

下面介绍**随机存取机**(*random access machine*)，简称为 RAM。这是一种很有用的计算模型。

1.3.1 RAM的构造

随机存取机是不允许自行修改其指令的单累加器计算机模型。一台随机存取机RAM由以下部件构成：

(1) **程序存储部件** 一片特殊的存储器件，供存放程序(指令)用。

(2) **累加器**R_0 其功能与一般计算机的累加器相同。

(3) **内存贮器** 其存贮单元依次命名为 R_1，R_2，R_3，…。每个单元能存放一个任意大小的整数。同时，任何程序可以使用任意多个内存单元，即不限定内存单元的上界。这两条假设只适用于这样的场合：(i) 问题的体积不超过一台计算机的存储容量；(ii) 计算过程中所出现的任何数值的大小不会超过计算机的字长。

(4) **只读输入带** 它由一系列方格和一个带头组成，每格可以存放一个任意大小的整数。每当机器从输入带上读出一个数时，带头（读出磁头）就自动向右移动一格。

(5) **只写输出带** 它也是由一系列方格和一个带头组成。每格可以记录一个任意大小的整数。开始计算以前，所有方格全部为空白。一旦在输出带头下面的方格中打印了一个整数后，输出带头自动右移一格，且印出的符号不能再修改。

(6) **程序控制部件** 它主要包括指令计数器和操作码译码器等控制部件。它能控制一个程序按程序的走向和每条指令的严格定义一步步执行。

图1.3是随机存取机RAM的总体结构示意图。

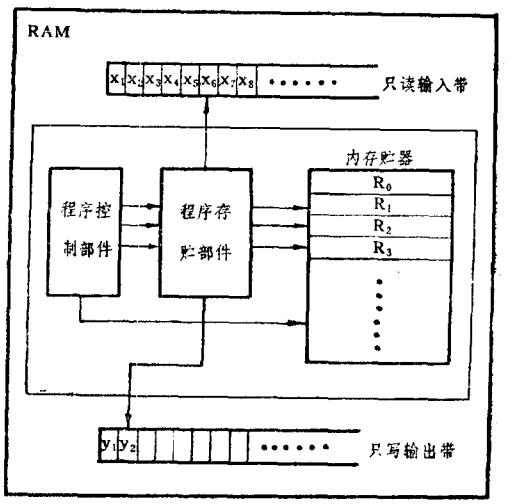

图1.3 随机存取机结构示意图

1.3.2 RAM 的 指 令 系 统

在这个指令系统中，各条指令的严格性质虽未给出，不妨假定有存取数指令，算术运算指令，转移指令和输入输出指令等几种与实际计算机中相同类型的指令。寻址方式主要有直接和间接两种基本寻址方式。同通常的机器一样，所有的操作都在累加器 R_0 中进行。一台随机存取机的指令系统如表1.3所示，表中的指令结构和通常的单累加器单地址机一样，每条指令都由"操作码"和"操作地址"两部分组成。原则上，可以根据实际需要而增设其它指令。比如可以增加逻辑运算指令、字符操作指令或位操作指令等等，以扩充指令系统的功能。

这个指令系统的操作码是众所周知的。操作地址则有以下三种形式：

表1.3　　RAM 的 指 令 系 统

操　作　码	操 作 地 址	说　　明
LOAD	=i/i/*i	取一操作数至累加器中
STORE	i/*i	累加器中的数送入内存
ADD	=i/i/*i	加法运算
SUB	=i/i/*i	减法运算
MULT	=i/i/*i	乘法运算
DIV	=i/i/*i	除法运算
READ	i/*i	输入
WRITE	=i/i/*i	输出
JUMP	标　号	无条件转移
JGTZ	标　号	正转移
JZERO	标　号	零转移
HALT		停机

(1) =i,称做直接数型。这是一条无地址指令，表示操作数是整数i 本身。

(2) i，称做直接地址型。i是一非负整数。表示操作数是内存单元Ri中的内容C(i)。

(3) *i，称做间接地址型。i是一非负整数。表示操作数是内存单元j 中的内容 C(j)，而j则是内存单元i中所存贮的那个整数。即是j = C(i)，操作数是C(j) = C(C(i))。当C(i)<0时，C(j) = C(C(i)) 无定义。RAM对所有无定义的指令作停机处理。

如果记操作数a的值为V(a)，则有

$$V(=i)=i$$

$$V(i)=C(i)$$

$$V(*i)=C(C(i))$$

显然，C是从非负整数到整数的单射函数。

1.3.3 RAM 的 工 作 方 式

在一台RAM机器中，一个程序P怎样工作呢？开始，指令计数器指向程序 P 的第一条指令，输入带上已有一串输入数据（当这个程序要求输入某些数据时），输入带头对准第一个输入字x_1所在的方格；输出带上全是空白，输出带头指向左端第一方格。当机器执行第k 条指令时，如果被执行的指令不是转移指令和停机指令，机器就完成操作码定义的有关操作，然后指令计数器自动加1，指向第$k+1$ 条指令。当执行的指令是转移指令时，或者是无条件转向这条指令的操作地址部分的标号所标定的那条指令（设程序中的指令都是可以加标号的），或者当条件不满足时（对条件转移指令）顺序执行下一条指令。当执行的指令是 HALT时就停机。

当机器执行输入指令时，就从只读输入带上读入一个数据。当机器执行输出指令时，就在只写输出带上打印结果。当然，一个程序P 也可能没有输入指令（或者它不需要输入数据；或者它需要的数据在程序中用直接数型指令给出）。一般说来，一个程序P至少应有一条输出指令。

总之，RAM在执行一个程序时的工作过程与实际计算机的工作过程基本相同。表1.4列出了RAM每条指令的严格定义。凡是没有定义的指令都是非法的。例如

STORE =i

READ =i

等指令，RAM 都无法执行。除数为零时也非法。RAM认为一切非法指令同HALT指令等价。

表1.4 RAM 指令的涵义，操作数 a 为 =i/i/ *i

指令		指令的涵义
LOAD	a	C(o)←—V(a)
STORE	i	C(i)←—C(o)
STORE	*i	C(C(i))←—C(o)
ADD	a	C(o)←—C(o)+V(a)
SUB	a	C(o)←—C(o)−V(a)
MULT	a	C(o)←—C(o)×V(a)
DIV	a	C(o)←—⌊C(o)/V(a)⌋*
READ	i	C(i)←—当前输入字；带头右移一格
READ	*i	C(C(i))←—当前输入字；带头右移一格
WRITE	a	在只写输出带的当前方格（即带头所指的方格）上打印V(a)后，带头右移一格
JUMP	b	指令计数器←—标号b的值
JGTZ	b	当C(o)>0时，指令计数器←—标号b的值；否则，指令计数器←—指令器数器的当前值+1
JZERO	b	当C(o)=0时，指令计数器←—标号b的值；否则，指令计数器←—指令计数器的当前值+1
HALT		停机

* 记号⌊x⌋表示取小于或等于x的最大整数，⌈x⌉表示取大于或等于x的最小整数。

1.3.4 RAM程序的两种解释

一般说来，一个 RAM 程序定义了一个从输入信息到输出信息的映射。由于程序并非对一切输入信息都在产生输出后停止，故它是一个部分映射。人们虽可以对这种映射关系做不同的解释，但最重要的两种解释是：一个RAM 程序或者计算一个函数，或者识别一个语言。

假定一个程序 P 总是从输入带上读入n 个整数，并且在输出带上至多写出一个整数。如果从输入带上读入的n个整数是x_1，x_2，…，x_n，而且在输出带的第一格印出了一个整数y后程序终止，我们就说程序P计算了函数$f(x_1,x_2,\cdots x_n)$，且得到函数值$f(x_1,x_2,\cdots,x_n)=y$。

一个RAM程序也可以解释为一个语言接受器。一个语言L是一个有穷字母表 Σ 上的字符串的集合。设字母表的长度为k，可以用整数1，2，3，…，k分别表示字母表Σ中的各符号。n个整数$x_1,x_2,\cdots,x_n$放于输入带上（$1\leqslant x_i\leqslant k$，$1\leqslant i\leqslant n$），它可以表示一个输入字符串$S=a_1a_2\cdots a_n$。可以在输入带的第n+1格放上数字 0 表示输入串的结束。这样就建立了字母表 Σ 上长度为n的字符串与整数序列之间的对应关系。如果一个RAM程序P在读入整数串S和结束标志 0 后，在输出带的第一格印出一个数字 1 并停机，就说程序P 接受输入字符串 $a_1a_2\cdots a_n$，或者说$a_1a_2\cdots a_n$是可由P识别的。P 接受的（或P 识别的）语言 L(P) 是P可接受的所有输入串的集合（这个集合中字符串的个数可能是有穷的，也可能是无穷的）。对于不属于P接受的语言 L(P) 中的输入串，P 或者在输出带上印出一个异于数字 1 的符号并终止，或者P永远不终止。

现列举两个RAM程序的实例。其中例1.1计算一个函数；例1.2识别一个语言。

例1.1 考虑函数

$$f(n)=\begin{cases}n^n, & \text{对一切}n\geqslant 1\text{的整数;}\\ 0, & \text{其它情形。}\end{cases}$$

下面给出了计算这个函数的PASCAL程序及与之相对应的RAM程序。其中变量r_1，r_2和r_3分别存放在RAM的内存储器R_1，R_2和R_3中。

计算 f(n) 的 PASCAL 程序

```
procedure FUNCTION
 var
 R1, R2, R3: integer;
 begin
  read (R1);
  if R1≤0 then write (0)
  else
   begin
    R2←—R1;
    R3←—R1 - 1;
    while R3>0 do
     begin
      R2←—R2*R1;
      R3←—R3 - 1
     end;
    write (R2)
   end
 end
```

计算 n^n 的 RAM 程序

标 号	RAM 指 令	相应的 PASCAL 语句
	READ 1	read (R1)
	LOAD 1	
	JGTZ pos	if R1≤0 then write(0)
	WRITE=0	
	JUMP endif	
pos,	LOAD 1	R2:=R1
	STORE 2	
	LOAD 1	
	SUB =1	R3:=R1-1
	STORE 3	
while,	LOAD 3	
	JGTZ continue	while R3>0 do
	JUMP endwhile	
continue,	LOAD 2	
	MULT 1	R2:=R2*R1
	STORE 2	

```
                LOAD    3
                SUB     =1          R3:=R3-1
                STORE   3
                JUMP while
endwhile:       WRITE   2           write (R2)
endif:          HALT
```

例1.2 考察这样一个语言，它是字母表 $\Sigma=\{1,2\}$上所有由同样多个 1 和 2 组成的符号串构成的语言L_{12}。下面分别给出了接受语言L_{12}中的任何输入串的 PASCAL 程序及相对应的RAM程序。这个程序的核心思想是，依次从输入带上读入符号x到寄存器R_1中，而且在寄存器R_2中记着到目前为止所输入的数字 1 和 2 的个数之差 d。当输入带头读到结束标志符 0 时，程序检查d是否为0。如果d = 0，则在输出带上打印数字 1 并停机；否则机器不输出任何结果。

识别有相同个数的 1 和 2 的符号串的PASCAL程序

```
procedure ACCEPTE
 var
 D, X: integer;
 begin
  D←0;
  read (X);
  while X≠0 do
   begin
    if X≠1 then D←D-1
           else D←D+1;
    read (X)
   end
  if D=0 then write (1)
 end
```

与 PASCAL 程序所对应的 RAM 程序

标　号	RAM 指 令	相应的 PASCAL 语句
	LOAD =0 STORE 2	D:=0
	READ 1	read (X)
while:	LOAD 1 JZERO endwhile	while X≠0 do
	LOAD 1 SUB =1 JZERO one	if X≠1
	LOAD 2 SUB =1 STORE 2 JUMP endif	then D:=D-1
one:	LOAD 2 ADD =1 STORE 2	else D:=D+1