算法设计与分析

宋　文　吴　晟　杜亚军　编著

重庆大学出版社

内 容 简 介

本书介绍串行算法设计与分析。全书共分12章，主要内容包括：算法及算法的复杂性、贪婪法、递归、回溯法、动态规划、分治法、探索法、分枝—限界法、内存分类法、图的算法、NP完备理论、现代优化计算方法简介等，每章后附有习题。

本书可作为高等理工科院校计算机专业或相关专业本科生、研究生作为算法设计与分析的教材，也可以供从事计算机科学与应用的科技人员参考。

图书在版编目(CIP)数据

算法设计与分析/宋文，吴晟，杜亚军编著. —重庆：重庆大学出版社，2001.12
计算机科学与技术专业本科系列教材
ISBN 7-5624-2348-2

Ⅰ.算… Ⅱ.①宋… ②吴… ③杜… Ⅲ.①电子计算机—算法设计—高等学校—教材 ②电子计算机—算法分析—高等学校—教材 Ⅳ.TP301.6

中国版本图书馆CIP数据核字(2001)第072485号

算法设计与分析

宋 文 吴 晟 杜亚军 编著
责任编辑 曾显跃

*

重庆大学出版社出版发行
新 华 书 店 经 销
重庆大学建大印刷厂印刷

*

开本：787×1092 1/16 印张：15.5 字数：387千
2001年12月第1版 2001年12月第1次印刷
印数：1—5000
ISBN 7-5624-2348-2/TP·304 定价：23.00元

前言

算法设计与分析是计算机科学的核心问题之一。这门功课是计算机各专业以及相关专业的一门重要的课程。它包括计算机研究与应用领域的一些常用算法。按处理的数据分类，可分为数值计算和非数值计算，本书的重点在后者；按处理问题的方式分类，又分为串行算法与并行算法，本书涉及的内容是前者。对于需要介绍的内容，我们参照了 Fundamental of Computer Algorithms（E Horowitz, S Sahni）和算法设计与分析（周培德）的编写风格，以介绍常用的算法设计方法（第 2 章到第 8 章）为全书的第一部分主要内容；第 9,10 两章是对数据结构的表、图的扩充，作为本书的第二部分内容；第 11 章介绍 NP 完全理论，它对于深入认识算法的本质，回答什么是难解问题，具有重要意义。它是用数学方法处理计算机问题，也是从事计算机科学研究必备的知识；第 12 章是现代优化计算方法简介。遗传算法是一类借鉴生物界自然选择和自然遗传机制的随机化搜索算法；神经网络系统理论是人工智能的一个前沿研究领域，它们在近几年得以很快的发展，在本书的最后一个章节介绍给读者。

我们希望读者通过学习，掌握算法设计的常用方法，在解决实际问题时，对于较复杂的问题能设计出有效的算法。通过学习算法分析，掌握估计算法的时空复杂度的方法，从而能够正确地评价一个算法，设计出真正好的算法。

作为计算机专业的本科生，第一、二部分是必学内容，第三部分是选学内容，教师可视情况选用。

本书框架由宋文、吴晟讨论制订。宋文完成第 1,3,8,9,10 章的编写，以及全书的统稿工作。吴晟完成第 2,4,5,11,12 章的编写。杜亚军完成第 6,7 章的编写。牟行军同志在本书的编写过程中作了大量的工作，本书作者所在院校的院系领导对本书的编写，给予了大力的支持，在此表示衷心的感谢。虽然我们尽了努力，但是由于水平有限，除这项任务之外工作繁忙，加之时间仓促，奉献给读者的这本书缺点和错误一定不少，敬请同行与读者批评指正。

编　者

2001 年 9 月

目录

第 1 章
算法及算法的复杂性

本章是全书的基础。首先介绍算法的定义，继而介绍分析算法的工具：时间复杂性和空间复杂性，主要介绍时间复杂性。第三节介绍递归方程的解法，如果读者具有组合数学方面的知识，则可略去这一节内容。最后是通过对一个例子的讲解，给读者示范如何分析算法的时间复杂性和空间复杂性。

1.1 算法的定义

我们相信，凡是有过程序设计经历的人都会对 N. Wirth 提出的公式：

算法 + 数据结构 = 程序

有着深刻的领悟：算法是程序的灵魂。

程序设计主要包括两个方面：行为设计与结构设计。行为设计是对要解决的问题，提出达到目的需要实施的一些步骤，并对这些步骤加以必要的细化，给予定义，在此基础上用某种方式完整地描述出来，就是算法设计，其结果就是算法；结构设计是针对所要解决的问题，对数据定义数据结构（包括物理结构和逻辑结构）。有了好的算法、合适的数据结构，再使用某种程序设计语言加以具体实现，即可得到程序。因此，算法是程序设计的灵魂，它在产生程序的过程中，占有重要的地位。

对于算法，D. E. Knuth 给出了一个非形式化的定义：

一个算法，就是一个有穷规则（指令）的集合。它为某个特定类型问题提供了解决问题的运算序列。

从这一定义可以引申出算法具有的五个特征：

（1）有穷性

一个算法必须在执行有穷步之后终止，即必须在有限的时间内完成。如何理解有限的时间呢？如果一个算法需要在计算机上运行千万年，那么这样的算法，它所需要的时间是有限的，但是这样的算法没有实用价值。所谓需要有限时间的算法，应该理解成人们可以接受的时间内完成的算法。

例如：计算行列式的值。

依线性代数中对 n 阶行列式的定义：n 阶行列式的值等于所有取自于不同行、不同列的 n 个元素的乘积 $a_{1j_1},a_{2j_2},\cdots,a_{nj_n}$ 的代数和，这里 $j_1,j_2,\cdots,j_n$ 是 $1,2,\cdots,n$ 的一个排列，每一项按下列规则带有符号：当 $j_1,j_2,\cdots,j_n$ 是偶排列时带正号；当 $j_1,j_2,\cdots,j_n$ 是奇排列时带负号。n 阶行列式一共有 $n!$ 项，计算它需做 $n!(n-1)$ 个乘法。按这个定义设计的求值过程，只能是一种理论上可行的计算方法，不是一种实用的计算方法，更不能称做为算法。

再如：货郎担问题（售货员路线问题）：

设货郎在一天内要到 n 个城市去推销货物，已知从一个城市到其他城市的费用，求总费用最少的路线。

用穷举法可以求得最少费用的路线，花费的时间按 $n!$ 增长，其时间复杂性是 $O(n!)$（与前一例有同样的时间复杂度）。这个解法实际上是不可行的。当 $n=20$ 时，$20!=2\times10^{18}$，假设计算每条路线需要 CPU 时间为 10^{-7}s，则总共需要 7 千年才能得到结果。这样的运行时间对于人们来说是不可容忍的。因此，算法的有穷性实际上应包含合理的执行时间的含义。

(2)确定性

算法的每一步必须是确定的。即：让计算机执行的每一步，不允许有模棱两可的解释，不允许有多义性。比如，让计算机下一步执行“将 m 或 n 与 y 相乘”之类的运算就存在二义性。因为按此指令，究竟是把 m 与 y 相乘，还是把 n 与 y 相乘不确定。

(3)可行性

算法的可行性包括两个方面：一是算法所描述的每一步必须是基本的、有意义的。例如：在有限时间内完成的算法中，不允许做除法时，除式为 0；在实数范围内不能求一个负数的平方根等。二是算法执行的结果要能达到预期的目的。比如使 $\sin x$ 的近似值的绝对值大于 1 的求解过程，不具有可行性，它不是算法。

(4)输入

一个算法有 0 个或多个输入。算法处理的数据来源于两种方式：一种是从外部设备获得数据的输入，另一种是由算法中自己产生被处理的数据。算法的 0 个输入指不需要从外部设备获取数据，数据来源于第二种方式。

(5)输出

一个算法必须有一个或多个输出。输出的数据是算法对数据加工的结果。既然算法是为解决问题而设计的具体实现的若干步骤，那么算法实现的最终目的就是要获得问题的解。没有输出的算法是无意义的。

综上所述，所谓算法是一组定义严谨的运算顺序规则，并且每一个规则都是有效的，且是明确的。这组规则在有限的时间内终止。

解决问题的目的不只是获得算法。算法有“好”有“歹”，那么好歹的标准是什么？也就是说，用什么样的尺度去评价一个算法的好歹。下面一节介绍这一内容。

1.2 算法的复杂度与评价

算法分析是对一个算法所消耗的资源进行估算。在串行算法中（本书讨论的算法设计与分析是串行的，并行算法的设计与分析在该系列教材中专门作为一门课程介绍），资源消耗指

时间、空间的耗费。时空是一对矛盾。算法分析的目的，就是通过对同一问题的多个不同算法进行时空耗费这两方面的分析。在时空资源兼受限制时，找出时空代价的平衡点，即时间相对少，空间耗费也相对少的算法。当两种资源之一受限时，选出适合这一限制的算法。算法分析是一项很考智力的，而且是很有实际意义的工作。它力求完美，而这种追求正是优秀程序员所具有的品质。

如何计算算法的时空消耗？

一种办法是事后分析。具体地说，对于问题 P 的算法 A，在计算机上分别运行 A 对应的程序，输入适当的数据，测试程序 A 的开销，这一方法似乎很合理，细想一下其实不尽人意。

第一，如此测试的结果与程序的运行环境有关。如计算机主频、总线和外部设备等，都会影响测试结果。

第二，语言的编译系统对生成的机器代码的质量会产生很大的影响，从而影响运行的速度。

第三，测试结果与选取的样本数据有关。算法的时空耗费是 n 的函数，它的结果应是在 n 足够大时才有意义。对于不同的 n 个数据，运行的结果是不一样的。欲要获得真实可靠的结果，必须科学地选取样本数据，而要做到这一点，并不是一件简单的事情。

总之，这种事后分析是面向机器，面向程序员，面向语言的。从理论上讲，只有在标准环境下进行算法测试，得到的资源耗费才是可信的。

这些因素与问题 P 的两种算法的差异无关。为了公平起见，同一问题的两种算法所对应的两个程序，应该在同样的条件下用同一个编译器编译，在同一台机器上运行，并且两次编程时所花费的精力也应尽可能地相等，使算法的实现“等效”。如果做到这几点，上面提到的那些因素就不会对结果产生影响，因为它对于每一个算法都是公平的。

但是，仅凭实验来比较算法，很有可能因为一个程序比另一个程序“写得好”，而使算法的真正质量没有得到很好的体现。再者，可能两种算法，通过艰苦的编程，测试后都超过了预算的耗费，那么只有重新设计适合预算耗费的新算法。

另一种方法是事前分析的方法，称为渐进算法分析（asymptotic algorithm analysis），简称为算法分析，这种分析方法是求得算法耗费的限界函数。它可以估算出当问题规模变大时，一个算法以及实现它的程序的效率和耗费。

在介绍这种方法之前，首先对算法实现的机器类型作出假定，因为它对求得算法的耗费影响很大。从算法的本质特征出发，应选一种确定的机械装置作为前提进行耗费计算。这种机械装置最理想的形式模型是图灵机（Tuning machine）和 RAM 机（Random access machine，即随机存取机器），但是读者会对这些严谨的、形式化的定义费解。有关内容在第 11 章作简要介绍。现假定关于算法的讨论在一台通用机的基础上进行。这台通用机就是平时使用的顺序处理机。它每一次执行算法中的一条指令，有足够大的随机存储器，对每一个单元中数据的访问时间是固定的，在这样的前提下，进行事前分析的讨论。

对于时间耗费的计算来说，在这种理想的通用机模型上，用尺度来代替运行时间。这个尺度是处理一定“规模”的输入时，算法所需要执行的“基本操作”数。

例 1.1　下面是查找一维 n 元整数数组中最大元的算法。

算法 1.1

```
procedure  findmax(a,n)
```

```
/ 输入:长度为 n 的数组 a 。输出:a 中元素的最大者在 a 中的位置。/
begin
  curmax ← a[1]
  for  i ← 2  to  n  do
       if a[i] > curmax  then
        curmax ← a[i];
  return ( i )
endp;
```

在这个算法中,规模指输入量的数目,即数组的长度。基本操作的选取应满足该操作具有这样的特征:完成该操作所需时间与具体的输入无关。在 findmax 中,基本操作可选择元素间的比较。因为完成比较操作所需时间与输入无关,可记作常量 C。当然在 findmax 中,也可选择其他操作作为基本操作,比如选择 i 是否有效的判断作为基本操作也可以,只要这个选择是遵循了"基本操作"的定义亦可。事实上可以看到,元素间的比较与判断 i 值是否有效在 findmax 中的地位是平等的。

由于时间耗费是规模的函数,这个函数可记为 $T(n)$。算法 1.1 的时间耗费:$T(n) = Cn$,其中 C 为常数。它反映了时间耗费对 n 的增长率。

例 1.2 再看下面程序段。

```
sum ← 0 ;
for i ← 1  to  n  do
    for  j ← 1  to  n  do
        sum ← sum + a[i,j]
```

其中 a 是行列数为 n 的方阵。

该算法中规模为 n^2,基本操作是做加法,时间耗费是:$T(n) = Cn^2$,其中 C 为常数。把时间耗费为 Cn 称为线性增长率,Cn^2 称为二次增长率,$C2^n$ 称为指数增长率。显然,当 n 很大时,$2^n > n^2 > n\log_2 n > n > \log_2 n > C$,其中 C 为常数。

1.2.1 最佳、最差、平均情况的时间耗费分析

在前面例 1.1、例 1.2 的算法分析过程中,所选择的基本操作与具体输入无关,所求得的时间耗费也与具体输入无关。但是,对于某些算法,计算它的时间耗费所得结果与具体输入有关。

比如,在一维数组结构上的查找,一旦查找成功算法结束。输入 n 个不同的数据,要查找的数据为 k。在这个问题中,规模是数组长度 n,基本操作若是元素的比较,顺序查找法依次在数组中取出每个元素与 k 比较,若某次比较相等,则查找成功,算法结束;若依次取出每个元素与 k 比较后都不相等,则查找失败,算法也结束。对于成功的查找,有可能第一个元素就是要查找的 k,在这种情况下,元素的比较次数为 1 次,这是算法的最佳情况。如果第 n 个元素才是要查找的 k,那么,元素间的比较要进行 n 次,这是算法的最差情况。一般地,对于成功的查找,假定 k 出现在第 i 个位置上是等概率的,$0 < i < n + 1$。元素的比较平均要进行 $(n + 1)/2$ 次,这是算法的时间耗费的平均情况。

在这个算法中,n 值并不决定执行元素间的比较操作的次数。如果对于 n 个数据的输入,

要查找 k 所有出现的位置，基本操作仍选取元素的比较，则不存在最佳、最差和平均情况，此时，时间耗费为：$T(n) = Cn$，C 为常数，$T(n)$ 依赖于 n。

可以看到，最佳情况不具有普遍性，最差情况至少会告诉人们时间耗费的上界是多少，而平均情况更具有普遍性。对于时间耗费与输入有关的算法来说，要求得它的平均性能并不那么容易。有些算法早就有人提出，但是，至今在数学上也没能求得它的平均性能的值。在实时系统中最为关注的是最差情况的算法分析。

在计算时间耗费中，考虑的是完成基本操作的次数。对于 n 的增长率，有时并不需要知道精确的时间耗费，只要知道时间耗费大体在什么范围内即可。在后面的章节中，除非专门说明时间分析是指平均情况的时间耗费，一般情况下将最坏情况下的时间耗费的极限作为算法的时间耗费，称为时间复杂性。求解过程称为时间渐进分析。

1.2.2　时间渐进分析

在讨论算法的时间复杂性中，需要引入算法复杂性阶的概念，这些概念不仅适合于时间复杂性的讨论，也适合于空间复杂性的讨论。

定义 1.1　如果存在着正数 C 和 n_0，当 $n \geq n_0$ 时，有 $T(n) \leq Cf(n)$，则称 $T(n)$ 是 $O(f(n))$，记作 $T(n) = O(f(n))$，此时，$f(n)$ 是 $T(n)$ 的增长率的一个上界。

定义 1.2　如果 $T(n) = O(f(n))$，并且 $f(n) = O(T(n))$ 同时成立，则称 $T(n)$ 是 $\theta(f(n))$，记作 $T(n) = \theta(f(n))$，此时称 $T(n)$ 和 $f(n)$ 的增长率是同阶的。

定义 1.3　如果存在着正数 C，使得对于无穷多个 n，关系式 $T(n) \geq Cf(n)$ 成立，则称 $T(n)$ 是 $\Omega(f(n))$，记作 $T(n) = \Omega(f(n))$，此时 $f(n)$ 是 $T(n)$ 的增长率的一个下界。

这里，并不要求存在 $n_0 > 0$，$n > 0$，当 $n_0 > n$ 时，所有的 n 都使 $T(n) \geq Cf(n)$，而只要求这种情况发生得足够频繁。

$$T(n) = \begin{cases} n & n \text{ 为奇数} \\ n^2/100 & n \text{ 为偶数} \end{cases}$$

比如：按定义 $T(n) = \Omega(n^2)$

显然，如果 $T(n) = O(f(n))$ 和 $T(n) = \Omega(f(n))$ 同时成立，$T(n) = \theta(f(n))$。

下面分析中最常用的是定义 1.1。

例 1.3　若 $T(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$ 是关于 n 的一个 m 次多项式，则 $T(n) = O(n^m)$。

解　根据定义 1.1，取 $n_0 = 1$，当 $n \geq n_0$ 时

$$|T(n)| \leq |a_m| n^m + \cdots + |a_1| n + |a_0| \leq$$

$$(|a_m| + |a_{m-1}|/n + \cdots + |a_0|/n^m)\, n^m \leq$$

$$(|a_m| + |a_{m-1}| + \cdots + |a_0|)\, n^m$$

取 $C = |a_m| + |a_{m-1}| + \cdots + |a_0|$，则 $T(n) = O(n^m)$。

根据上述定义，下面的结论对于后面的分析是有用的。

①若 $f(n) = O(g(n))$ 且 $g(n) = O(h(n))$，则 $f(n) = O(h(n))$。

②若 $f(n) = O(Kg(n))$，其中 K 为任意大于 0 的常数，则 $f(n) = O(g(n))$。

③若 $f_1(n) = O(g_1(n))$ 且 $f_2(n) = O(g_2(n))$，则

$f_1(n)+f_2(n)=O(\max(g_1(n),g_2(n)))$

④若$f_1(n)=O(g_1(n))$且$f_2(n)=O(g_2(n))$，则

$f_1(n)\cdot f_2(n)=O(g_1(n)\cdot g_2(n))$

在分析渐进时间中，如果$f(n)=O(\log_2 n)$，按定义 1.1，存在着 C 和 n_0，当 $n\geqslant n_0$时，

$$f(n)<C\log_2 n=C_2\frac{1}{\log_a 2}\log_a n$$

令 $C_2\frac{1}{\log_a 2}=C'$

则 $f(n)\leqslant C'\log_a n$

再按定义 1.1，有$f(n)=O(\log_a n)$，于是，a 在式子中并不重要，只要 a 使对数有意义即可，所以，在以后的叙述中经常略去 a，上式记为$f(n)=O(\log n)$。

当符号“O”出现在表达式 $T(n)=2^{O(n)}$中的指数位置上时，按定义 $2^{O(n)}$是 2^{Cn}的一个上界，其中 C 为某个常数。而 $n^{O(1)}$代表 n^C的一个上界。在 $2^{O(\log n)}$中，由恒等式 $n=2^{\log_2 n}$及 $n^C=2^{C\log_2 n}$，说明 $2^{O(\log n)}$是 n^C的一个上界。把渐进时间为 n^C（C 是大于 0 的常数），称作多项式界；把形如 2^{n^δ}的界，当 δ 是大于 0 的实数时，称为指数界。

显然，当 n 较大时，$2^n<n!<n^n$

把 $O(n!)$和 $O(n^n)$也称为指数界，它们的关系是 $O(2^n)<O(n!)<O(n^n)$

常见的渐进时间以及之间的关系是：

$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(2^n)$

当 n 取得很大时，指数时间和多项式时间悬殊很大，因为对于任意的 $m\geqslant 0$，总可以找到 n_0，当 $n\geqslant n_0$时，有 $2^n>n^m$。在算法优化中，如在空间耗费增加不大的情况下，使算法的时间复杂度降低一个数量级是一件极其有意义的劳动。

1.2.3 空间复杂性

算法在运行时所需的存储空间大小 $S(n)$称为算法的空间耗费。

对于一个算法，空间耗费的度量方法通常定义为算法在运行时所占用内存单元的总数，即存放数据的变量单元、程序代码、工作变量、常数以及运行时的引用型变量所占用空间和递归栈空间的总和。它与输入规模 n 有关，S 是 n 的函数。

在最坏情况下的空间耗费，对于 n 的渐进结果作为算法的空间复杂性。渐进表示中的“O”，“Ω”，“θ”对于空间代价同样适合。

在分析算法时，总希望找到 $T(n)$和 $S(n)$都小的算法，这种矛盾性在前面已介绍了。在设计算法时，常采用牺牲空间以减少时间代价。

1.2.4 分析的实际意义

对于一个算法，如果有两个方案可以选择：一是换一台更快的计算机，二是换一种更快的算法，究竟哪种方案好，两个方案都可选择，表 1.1 列出的数据，对你的选择提供了依据。

假设新的计算机的运行速度是原来旧计算机的 10 倍，不妨设原来计算机 1 小时可以处理的问题的规模为 n_1，那么是否在时间仍为 1 小时的条件下，新的计算机 1 小时可处理的规模 $n_2=10n_1$呢？表中列出了时间一定的情况下，5 个时间耗费函数可解决的问题的规模。

表1.1　1小时内速度相差10倍的新旧两种计算机能处理问题尺寸的增长情况

$T(n)$	n_1	n_2	n_1和n_2的关系	n_1和n_2的比值
$10n$	1 000	10 000	$n_2 = 10n_1$	10
$20n$	500	5 000	$n_2 = 10n_1$	10
$5n\log n$	250	1 842	$\sqrt{10}n_1 < n_2 < 10n_1$	7.37
$2n^2$	70	223	$n_2 = \sqrt{10}n_1$	3.16
2^n	13	16	$n_2 = n_1 + \log n_1$	—

从表1.1中可以看出：

①对于线性的算法，问题规模的增长为10，等于新机器与旧机器运行速度的倍数。

②$5n\log n$和$2n^2$这两个多项式算法，问题的规模的增长不等于新旧机器运行速度的倍数。对于$T(n)=2n^2$的算法，将现有机器更换为新机器，且新机器运行速度是旧机器运行速度的10倍的话，在给定时间内，新旧机器处理问题的规模的比是3.16，而线性算法的规模比是10。可见，多项式算法其规模增长的倍数比线性算法增长的倍数小，时间耗费高的算法从机器升级中得益小，能解决的问题的规模小。

③特别是时间耗费为2^n的算法，新旧机器运行速度比为10时，对于同一问题的同一算法，旧机器处理问题的规模为n，则新机器处理问题的规模的尺寸为$n+\log_2 n \approx n+3$，其尺寸仅增加了一个常数$\log_2 n \approx 3$。如果再换一台更新的机器，比旧机器运行速度快100倍，它的处理问题的规模仅增加了常数9，即旧机器1小时能解决的规模为1 000的问题，而速度快100倍的机器，1小时解决这一问题的规模为1 009。因此对于解决时间耗费较大的问题，当问题的规模较大时，最好是考虑用一个时间耗费较低的算法，而不是去换一台新机器，换机器只能对于时间耗费较小的算法获益较大。

④从表中还可以看出，前两行都是线性的，仅是系数的差异而已。随着机器的换代，问题规模的增长都是一样的。因此，系数影响给定时间内解决问题的规模，但不影响问题规模的增长。

$T(n)=10n$，　$n_1=1\ 000$　　$n_2=10\ 000$

$T(n)=20n$，　$n_1=500$　　$n_2=5\ 000$

二者的规模比都是10。事实上，无论时间耗费是多少，常系数不改变机器提速后，问题规模的增长倍数。因此，算法分析最关注的是时间渐进分析，因为它揭示了问题的本质。

1.3　解递归方程

本节中提出的求解方法，其正确性证明略去，有兴趣的读者可以阅读有关组合数学方面的书籍。另外，本书也仅介绍解递归方程的基本方法。对于一些特殊解法，有必要时实时、实地补充。

1.3.1　递推法

对于某些递归关系，可以通过逐次递推求得递归方程的解。

例 1.4 Hanoi 塔问题

Hanoi 塔问题的递归算法的时间复杂性，由下面递归方程给出：

$$\begin{cases} T(n) = 2T(n-1) + 1 & n \geqslant 2 \\ T(1) = 1 \end{cases}$$

解 $T(n) = 2T(n-1) + 1 =$

$2(2T(n-2) + 1) + 1 =$

$2^2T(n-2) + 2 + 1 =$

$2^2(2T(n-3) + 1) + 2 + 1 =$

$2^3T(n-3) + 2^2 + 2 + 1 =$

……

$2^{n-1}T(1) + 2^{n-2} + \cdots + 2^2 + 2 + 1 =$

$2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1 =$

$2^n - 1$

所以，Hanoi 塔问题的递归算法的时间复杂性为 $T(n) = O(2^n)$。

这是一个指数时间的算法。

例 1.5 第 6 章介绍分治法。这里举一个分治法的例子。设 n 表示问题的尺寸，n/b 表示将这个问题分成 a 个子问题后，每个子问题的尺寸，其中 a, b 为常数。$d(n)$ 表示在分解或合成子问题而得到整个问题解决时的时间耗费。则整个问题的时间耗费由下面递归方程给出，求解递归方程。

$$\begin{cases} T(n) = aT\left(\dfrac{n}{b}\right) + d(n) & n > 1 \\ T(1) = 1 \end{cases}$$

解 用递推法

$$\begin{aligned} T(n) &= a(aT(n/b^2) + d(n/b)) + d(n) = \\ &\quad a^2T(n/b^2) + ad(n/b)) + d(n) = \\ &\quad a^2(aT(n/b^3) + d(n/b^2)) + ad(n/b) + d(n) = \\ &\quad a^3T(n/b^3) + a^2d(n/b^2) + ad(n/b) + d(n) = \\ &\quad \cdots\cdots \\ &\quad a^kT(n/b^k) + \sum_{i=0}^{k-1} a^id(n/b^i) \end{aligned}$$

设 $n = b^k$，则 $k = \log_b n$。于是

$$T(n) = a^kT(1) + \sum_{i=0}^{k-1} a^id(n/b^i)$$

当 $d(n)$ 为常数时，有

$$T(n) = a^k + c\sum_{i=0}^{k-1} a^i$$

这是一个关于 a 的多项式，由前面渐进分析可知：

$T(n) = O(a^k)$，而 $k = \log_b n$，$a^k = a^{\log_b n}$

由对数等式 $a^{\log_b n} = n^{\log_b a}$ 得

$$T(n)=\begin{cases}O(n^{\log_b a}) & a\neq 1\\ O(\log_b n) & a=1\end{cases}$$

当 $d(n)=cn$，c 为常数时，则

$$\sum_{i=0}^{k-1}a^i d(n/b^i)=\sum_{i=0}^{k-1}a^i(cn/b^i)=cn\sum_{i=0}^{k-1}\left(\frac{a}{b}\right)^i$$

若 $a<b$，则　$cn\sum_{i=0}^{k-1}\left(\frac{a}{b}\right)^i=O(n)$

那么有

$$T(n)=n^{\log_b a}+O(n)=O(n)$$

若 $a=b$，则 $cn\sum_{i=0}^{k-1}\left(\frac{a}{b}\right)^i=cnk=cn\log_b n$

那么有

$$T(n)=n^{\log_b a}+cn\log_b n=O(n\log_b n)$$

若 $a>b$，因 $n=b^k$，$cn\sum_{i=0}^{k-1}\left(\frac{a}{b}\right)^i=cn\dfrac{\left(\frac{a}{b}\right)^k-1}{\frac{a}{b}-1}=c\dfrac{a^k-b^k}{\frac{a}{b}-1}=O(a^k)=O(n^{\log_b a})$

于是 $T(n)=n^{\log_b a}+O(n^{\log_b a})=O(n^{\log_b a})$

综上所述，$T(n)=\begin{cases}O(n) & a<b\\ O(n\log n) & a=b\\ O(n^{\log_b a}) & a>b\end{cases}$

1.3.2　公式解法

对于某些递归方程，一般项并不是由一个次低项，而是由多个次低项表示时，用递推法就难于求解。

下面介绍递归方程的公式解法：

$T(n)-a_1T(n-1)-a_2T(n-2)-\cdots-a_kT(n-k)=0$　　$n\geqslant k$，a_i，$i=1,\cdots,k$ 是常数，$a_k\neq 0$，称作 k 阶常系数齐次递推方程，简称齐次方程。

方程 $x^k-a_1x^{k-1}-a_2x^{k-2}-\cdots-a_{k-1}x-a_k=0$ 称为齐次方程的特征方程，它的 k 个根 q_1，$q_2,\cdots,q_k$ 称为齐次方程的特征根。

①如果 k 个根互不相同，则齐次方程的通解为：

$$T(n)=c_1q_1^n+c_2q_2^n+\cdots+c_kq_k^n$$

其中，c_i，$i=1,\cdots,k$ 为待定常数，由初始条件可以确定这 k 个常数，从而得到齐次方程的解。

例 1.6　求解齐次递归方程

$$\begin{cases}T(n)=2T(n-1)+2T(n-2) & n\geqslant 3\\ T(1)=3,T(2)=8\end{cases}$$

解　该方程的特征方程是　$x^2-2x-2=0$，特征根为：

$$x_1=1+\sqrt{3},x_2=1-\sqrt{3}$$

所以通解为：$T(n)=c_1(1+\sqrt{3})^n+c_2(1-\sqrt{3})^n$

由 $T(1)=3, T(2)=8$

$$\begin{cases} c_1(1+\sqrt{3})+c_2(1-\sqrt{3})=3 \\ c_1(1+\sqrt{3})^2+c_2(1-\sqrt{3})^2=8 \end{cases}$$

解得 $c_1=\dfrac{2+\sqrt{3}}{2\sqrt{3}}, c_2=\dfrac{-2+\sqrt{3}}{2\sqrt{3}}$

因此 $T(n)=\dfrac{2+\sqrt{3}}{2\sqrt{3}}(1+\sqrt{3})^n+\dfrac{-2+\sqrt{3}}{2\sqrt{3}}(1-\sqrt{3})^n, n=1,2,\cdots$

$T(n)=O(2^n)$

②如果特征方程有 r 个重根，设 $q_1, q_2, \cdots, q_k$ 中，$q_i=q_{i+1}=\cdots=q_{i+r-1}$，则通解的形式是：

$T(n)=c_1q_1^n+\cdots+c_{i-1}q_{i-1}^n+(c_i+c_{i+1}n+\cdots+c_{i+r-1}n^{r-1})q_i^n+c_{i+r}q_{i+r}^n+\cdots+c_kq_k^n$

例 1.7　求解递归方程

$$\begin{cases} T(n)+T(n-1)-3T(n-2)-5T(n-3)-2T(n-4)=0 \quad n\geqslant 4 \\ T(0)=1, T(1)=0, T(2)=1, T(3)=2 \end{cases}$$

解　递归方程的特征方程是：

$$x^4+x^3-3x^2-5x-2=0$$

它的特征根是 $-1, -1, -1, 2$

通解为：$T(n)=C_1(-1)^n+C_2n(-1)^n+C_3n^2(-1)^n+C_4 2^n$

由初始条件可得：

$$\begin{cases} c_1+c_4=1 \\ -c_1-c_2-c_3+2c_4=0 \\ c_1+2c_2+4c_3+4c_4=1 \\ -c_1-3c_2-9c_3+8c_4=2 \end{cases}$$

解得 $c_1=\dfrac{7}{9}, c_2=-\dfrac{1}{3}, c_3=0, c_4=\dfrac{2}{9}$

所以原方程的解是：

$$T(n)=\frac{7}{9}(-1)^n-\frac{1}{3}n(-1)^n+\frac{2}{9}\cdot 2^n, T(n)=O(2^n)$$

常系数线性非齐次递归方程的一般形式是：

$$T(n)-a_1T(n-1)-\cdots-a_kT(n-k)=f(n), \qquad n\geqslant k$$

$a_k\neq 0, f(n)\neq 0, f(n)$ 与 $T(n)$ 线性无关。

它的通解是：

$$T(n)=\overline{T}(n)+T^*(n)$$

其中，$\overline{T}(n)$ 是它对应的齐次方程的通解，$T^*(n)$ 是它的一个特解。

如果能找到非齐次方程的一个特解，问题就迎刃而解了。但是，对于一般的 $f(n)$，并不存在寻找特解的一般方法，只能用观察法去猜测特解的形式，然后用待定系数的方法去确定系数。

当 $f(n)$ 是 n 的 t 次多项式时，一般可设特解 $T^*(n)$ 也是 n 的多项式。即：

$$T^*(n)=P_1n^t+P_2n^{t-1}+\cdots+P_tn+P_{t+1}$$

其中，$P_i,i=1,t+1$ 是待定系数。

例 1.8　求解递归方程

$T(n)+5T(n-1)+6T(n-2)=3n^2$ 的一个特解。

解　设特解为　$T^*(n)=P_1n^2+P_2n+P_3$ 带入原递归方程

$$P_1n^2+P_2n+P_3+5(P_1(n-1)^2+P_2(n-1)+P_3)+6(P_1(n-2)^2+P_2(n-2)+P_3)=3n^2$$

化简得：$12P_1n^2+(-34P_1+12P_2)n+(29P_1-17P_2+12P_3)=3n^2$

左边是含有待定系数的关于 n 的多项式，右边是一个确定的关于 n 的多项式，比较两边系数有：

$$\begin{cases}12P_1=3\\-34P_1+12P_2=0\\29P_1-17P_2+12P_3=0\end{cases}$$

解得 $P_1=\frac{1}{4},P_2=\frac{17}{24},P_3=\frac{115}{288}$，于是所求特解是

$$T^*(n)=\frac{1}{4}n^2+\frac{17}{24}n+\frac{115}{288}$$

如果 $f(n)$ 是指数函数时，若 $f(n)=\alpha\cdot\beta^n$，其中，α,β 为给定常数，此时特解可设定如下：

若 β 不是与之对应的齐次方程的特征根，则设

$$T^*(n)=P\cdot\beta^n,P\text{ 待定。}$$

若 β 是与之对应的齐次方程的 e 重特征根，则设

$$T^*(n)=Pn^e\beta^n,P\text{ 待定。}$$

例 1.9　求解递归方程

$$\begin{cases}T(n)+5T(n-1)+6T(n-2)=42\cdot4^n\\T(0)=0,T(1)=0\end{cases}$$

解　可以解得原方程对应的齐次方程的通解是

$$\overline{T}(n)=C_1(-2)^n+C_2(-3)^n,C_1,C_2\text{ 待定}$$

设特解 $T^*(n)=P\cdot4^n,P$ 待定，将特解带入原方程

$$P\cdot4^n+5P\cdot4^{n-1}+6P\cdot4^{n-2}=42\cdot4^n$$

解得　　$P=16$

从而通解为：

$$T(n)=C_1(-2)^n+C_2(-3)^n+16\cdot4^n$$

由初始条件

$$\begin{cases}C_1+C_2+16=0\\C_1(-2)+C_2(-3)+64=0\end{cases}$$

解得　$C_1=-112,C_2=96$，因此，原方程的解是

$$T(n)=-112(-2)^n+96(-3)^n+16\cdot4^n$$

这里需要提起注意的是，在 $\overline{T}(n)$ 求得之后，不能根据初始条件 $T(0)=0,T(1)=1$，马上去确定 C_1,C_2 的值。这是错误的，因为初始条件 $T(0)=0,T(1)=1$ 是针对非齐次方程的，而不是针对对应的齐次方程的。应在解出特解之后，得到通解，再由初始条件去确定待定系数

C_1, C_2。

1.3.3 生成函数法

生成函数也叫母函数或发生函数。利用生成函数可以解递归方程。

回顾本节已介绍的内容：利用递推解递归方程的方法很自然，但是，如果一般项由多个低次项表示时，用递推是不方便的。利用公式解法，条件是常系数递归方程时，对于常系数齐次方程，需解对应的特征方程，而特征方程是一个一元 n 次方程，求根的过程并不都是容易的。如果是常系数非齐次的递归方程，前面只介绍了常用的两种，多项式和指数型，许多递归方程至今还没有找到精确解。生成函数法不仅能解递推法、公式解法能解的递归方程，而且能解更一般的递归方程。

下面首先定义：

设 $a_0, a_1, \cdots, a_n, \cdots$ 是一个数列，作形式幂级数：

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots$$

称 $A(x)$ 是数列 $a_0, a_1, \cdots$ 的生成函数。

上式只是一个形式幂级数，它是否收敛？收敛域是什么？均不得而知。这样做是为了从 $A(x)$ 的函数特征来得到 $\{a_n\}$ 的有用信息，特别是当 $\{a_n\}$ 的通项由递归方程的形式给出时，利用生成函数可以求解出 a_n 的结果。

具体方法是：

首先由 $\{a_n\}$ 定义生成函数 $A(x)$，设对 x 的某些值 $A(x)$ 收敛，根据递归方程对作变换，求出 $A(x)$ 的解析表达式，然后将 $A(x)$ 重新展开成 x 的幂级数，则这个展开式中的 x^n 项的系数就是原序列的通项 a_n 的解析表达式。

下面举一个简单的例子：

例 1.10 汉诺塔(Hanoi)问题。

设 a_n 表示移动 n 个盘子所需要的移动次数。根据递归算法，递归方程为：

$$\begin{cases} a_n = 2a_{n-1} + 1 \\ a_1 = 1 \end{cases}$$

解 这个递归方程可以用递推法，也可用公式解法，这里用生成函数法解。

以 $\{a_n\}$ 为系数构造生成函数：

$$A(x) = \sum_{i=1}^{\infty} a_i x^i \tag{1}$$

设法求出 $A(x)$ 的解析表达式，然后把它展开成幂级数，幂级数中 x^n 项的系数即为 a_n 的值。根据递归方程，对 $A(x)$ 作初等运算。

$$\begin{aligned} A(x) &= a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n + \cdots \\ +) -2xA(x) &= \quad -2a_1 x^2 - 2a_2 x^3 + \cdots + 2a_{n-1} x^n + \cdots \end{aligned}$$

$$(1-2x)A(x) = a_1 x + (a_2 - 2a_1)x^2 + (a_3 - 2a_2)x^3 + \cdots + (a_n - 2a_{n-1})x^n + \cdots =$$

$$a_1 x + \sum_{i=2}^{\infty} (a_i - 2a_{i-1}) x^i$$