计算机算法基础

陈业纲 著

西南交通大学出版社

计算机算法基础

陈业纲 著

西南交通大学出版社
·成都·

内容简介

计算机算法作为程序设计的灵魂，在大型程序设计中越来越受到人们的重视，掌握常见计算机算法是计算机软件开发人员应具备的基本素质。本书将经典问题和算法设计技术进行了巧妙地结合，系统地论述了算法设计技术及其在经典问题中的应用。全书共 11 章。第 1 章介绍了算法的基本概念和与算法分析相关的数学问题，第 2 ~ 11 章分别介绍了分治法、动态规划法、回溯法、贪心法、分支限界法、图、数论、组合数学、背包、博弈等算法及设计技术。书中所有程序均在 VC 6.0 环境下调试通过。

全书内容丰富，深入浅出，结合应用，图例丰富，可作为程序设计大赛、计算机专业本科高年级学生和研究生学习算法的教材，也可供工程技术人员、软件设计师培训使用和读者自学参考。

图书在版编目（CIP）数据

计算机算法基础 / 陈业纲著. —成都：西南交通大学出版社，2015.2
ISBN 978-7-5643-3684-4

Ⅰ. ①计… Ⅱ. ①陈… Ⅲ. ①电子计算机 – 算法理论
Ⅳ. ①TP301.6

中国版本图书馆 CIP 数据核字（2015）第 005819 号

计算机算法基础
陈业纲　著

责任编辑	李芳芳
助理编辑	赵雄亮
特邀编辑	黄庆斌
封面设计	米迦设计工作室
出版发行	西南交通大学出版社 （四川省成都市金牛区交大路 146 号）
发行部电话	028-87600564　028-87600533
邮政编码	610031
网址	http://www.xnjdcbs.com
印刷	成都蓉军广告印务有限责任公司
成品尺寸	185 mm × 260 mm
印张	15.25
字数	368 千
版次	2015 年 2 月第 1 版
印次	2015 年 2 月第 1 次
书号	ISBN 978-7-5643-3684-4
定价	60.00 元

前　言

以最快的速度、最低的成本开发出低复杂度的软件是软件工程师追求的目标。而要开发此种软件，既要遵循软件工程的原理，又要合理的数据组织和高效简洁的算法。掌握常见的计算机算法是计算机软件开发人员应具备的基本素质，在软件设计的各个领域，算法都扮演着重要角色，被认为是计算机科学的灵魂。没有算法，计算机程序将不复存在。理解经典的算法是软件开发中的基本要求，而发明算法并证明该算法的正确性则是最高境界。

没有算法，计算机就成了一个冰冷的机器。许多人认为，算法是数学的内容，学起来特别麻烦。软件作为一种复合技术，如果只知道算法，不能用编程语言来实现，即使再优秀的算法也发挥不了作用。只有具备了基本的计算机知识和相关的数学知识，并且通过不断的学习才能进步。不管算法多么简单，都要动手实现，只有不断发现错误并改正错误，才能提高编程能力。

在理解算法的基础上最终实现算法是本书所追求的目标，通过对计算机算法系统的学习与研究，理解和掌握算法设计的主要方法，培养对算法的计算复杂性进行正确分析的能力，为独立设计算法并分析算法的复杂性奠定坚实的基础。因此，无论是否涉及计算机，特定的算法设计技术都可以看做是问题求解的有效策略。

本书是作者在多年的软件开发和计算机程序设计大赛辅导中总结出来的，将经典问题和算法设计技术进行了巧妙结合，系统地论述了常见的算法及其在经典问题中的应用。通过对同一问题用不同算法实现并进行比较，使读者更容易体会到算法设计技术的思想，达到融会贯通的效果。

全书共 11 章。第 1 章介绍了算法的基本概念和与算法分析相关的数学问题；第 2～11 章分别介绍了分治法、动态规划法、回溯法、贪心法、分支限界法、图、数论、组合数学、背包、博弈等算法及设计技术。书中所有程序均在 VC 6.0 环境下调试通过。

本书有配套的源程序代码，所有代码在实际工程中作简单修改就可直接运用于工程实际。可以让算法学习者由理解算法到亲自实现算法的能力得到迅速提高。

本书在编写过程中得到了软件开发人员和同行的大力支持，他们提出了许多宝贵的意见，同时参考了其他院校的一些算法实验和相关文献。还要感谢我院的任大飞老师部分输入，最后，向关心和支持本书编写的每一位人士表示衷心感谢。

由于作者水平有限，书中难免存在谬误之处，敬请读者批评指正。为了方便读者的学习，本书有配套的源程序代码，读者可通过两种方式获得：一是直接与 baogxm@163.com 联系；二是通过出版社网站下载。

目 录

第 1 章　算法与数学

凡编写过程序的人，也许都有这种体会，学会编程容易，但要想编出好程序难。虽然人们对算法一词非常熟悉，可到目前为止，对于算法尚没有统一而精确的定义。有人说：算法就是一组有穷的规则，它们规定了解决某一特定类型问题的一系列运算。而权威的描述为：算法是任何定义好了的计算程式，它取某些值或值的集合作为输入，并产生某些值或值的集合作为输出。因此，算法是将输入转化为输出的一系列计算步骤。我们也可以把算法看作解决特定问题的工具。一个算法必须具备的性质：

（1）算法首先必须是正确的，即对于任意的一组输入，包括合理的输入与不合理的输入，总能得到预期的输出。如果一个算法只是对合理的输入才能得到预期的输出，而在异常情况下却无法预料输出结果，那么它就不是正确的。

（2）算法必须是由一系列具体步骤组成的，并且每一步都能够被计算机所理解和执行，而不是抽象和模糊的概念。

（3）每个步骤都有确定的执行顺序，即上一步在哪里，下一步是什么，都必须明确，无二义性。

（4）无论算法有多么复杂，都必须在有限步之后结束并终止运行，即算法的步骤必须是有限的。在任何情况下，算法都不能陷入死循环中。

同时每个算法都有如下特点：

（1）有穷性。一个算法应包含有限的操作步骤，而不能是无限的。事实上“有穷性”往往是指“在合理的范围之内”。

（2）确定性。算法中的每一个步骤都应当是确定的，而不应当是含糊的、模棱两可的。算法中的每一个步骤应当不致被解释成不同的含义，而应是十分明确的。

（3）有零个或多个输入。所谓输入是指在执行算法时需要从外界取得必要的信息。

（4）有一个或多个输出。算法的目的是为了求解，没有输出的算法是没有意义的。

（5）有效性。算法中的每一个步骤都应当能有效的执行，并得到确定的结果。

程序是算法的计算机高级语言描述，算法是程序的灵魂。一般来说，一个算法要经过设计、确认、分析、编码、检查、调试、维护等阶段。据此，算法的学习可分为设计算法、表示算法、确认算法、分析算法、测试算法。求解同一计算问题可能有许多不同的算法，究竟如何来评价这些算法的好坏以便从中选出较好的算法呢？选用的算法首先应该是“正确”的。此外，还应考虑如下三点：

（1）执行算法所耗费的时间；

（2）执行算法所耗费的存储空间，其中主要考虑辅助存储空间；

（3）算法应易于理解，易于编码，易于调试等。

一个占存储空间小、运行时间短、其他性能也好的算法是很难做到的。因为上述要求有时相互抵触，要节约算法的执行时间往往要以牺牲更多的空间为代价，而为了节省

空间可能要耗费更多的计算时间。因此，我们只能根据具体情况有所侧重：若该程序使用次数较少，则力求算法简明易懂；对于反复多次使用的程序，应尽可能选用快速的算法；若待解决的问题数据量极大，机器的存储空间较小，则相应算法主要考虑如何节省算法运行时间。

一个算法所耗费的时间等于算法中每条语句的执行时间之和。每条语句的执行时间等于语句的执行次数（频度（Frequency Count））×语句执行一次所需时间。

算法转换为程序后，每条语句执行一次所需的时间取决于机器的指令性能、速度以及编译所产生的代码质量等因素。

若要独立于机器的软、硬件系统来分析算法的时间耗费，则设每条语句执行一次所需的时间均是单位时间，一个算法的时间耗费就是该算法中所有语句的频度之和。

1.1 复杂性的计量

算法的复杂性是指算法运行所需要的计算机资源的量。需要的时间资源的量称作时间复杂性，需要的空间（即存储器）资源的量称作空间复杂性。这个量应该集中反映算法中所采用的方法的效率，而从运行该算法的实际计算机中抽象出来。换句话说，这个量应该是只依赖于算法要解决的问题的规模、算法的输入和算法本身的函数。如果分别用 N、I 和 A 来表示算法要解问题的规模、算法的输入和算法本身，用 C 表示算法的复杂性，那么应该有：

$$C=F(N, I, A)$$

其中 $F(N, I, A)$ 是 N，I，A 的一个确定的三元函数。如果把时间复杂性和空间复杂性分开，并分别用 T 和 S 来表示：

$$T=T(N, I, A)$$
$$S=S(N, I, A)$$

通常简写为：

$$T=T(N, I)$$
$$S=S(N, I)$$

由于时间复杂性和空间复杂性概念类同，计算方法相似，且空间复杂性分析相对简单些，所以将主要讨论时间复杂性。

$T(N, I)$ 是算法在一台抽象计算机上运行所需的时间。设此抽象计算机所提供的元运算有 k 种，它们分别记为 O_1，O_2，…，O_k；再设这些元运算每执行一次所需要的时间分别为 t_1，t_2，…，t_k。对于给定的算法 A，设经过统计，用到元运算 O_i 的次数为 e_i，$i=1$，2，…，k，很明显，对于每一个 i，$1\leqslant i\leqslant k$，e_i 是 N 和 I 的函数，即 $e_i=e_i(N, I)$。那么有

$$T(N,I)=\sum_{i=1}^{k}t_ie_i(N,I)$$

其中 t_i（$i=1$，2，…，k）是与 N，I 无关的常数。

但是，我们不可能对规模 N 的每一种合法的输入 I 都去统计 e_i（N，I），$i=1$，2，…，k。因此 T（N，I）的表达式还得进一步简化。实际中只考虑最坏情况、最好情况和平均情况下三种情况的时间复杂性，并分别记为 $T_{\max}$（N）、$T_{\min}$（N）和 T_{avg}（N）。

$$T_{\max}(N)=\max_{I\in D_N}T(N,I)=\max_{I\in D_N}\sum_{i=1}^{k}t_ie_i(N,I)=\sum_{i=1}^{k}t_ie_i(N,I^*)=T(N,I^*)$$

$$T_{\min}(N)=\min_{I\in D_N}T(N,I)=\min_{I\in D_N}\sum_{i=1}^{k}t_ie_i(N,I)=\sum_{i=1}^{k}t_ie_i(N,\tilde{I})=T(N,\tilde{I})$$

$$T_{\text{avg}}(N)=\max_{I\in D_N}P(I)T(N,I)=\sum_{I\in D_N}P(I)\sum_{i=1}^{k}t_ie_i(N,I)$$

其中，D_N 是规模为 N 的合法输入的集合；I^* 是 D_N 中一个使 T（N，I）达到 $T_{\max}$（N）的合法输入，$\tilde{I}$ 是 D_N 中一个使 T（N，I）达到 $T_{\min}$（N）的合法输入；而 P（I）是在算法的应用中出现输入 I 的概率。如：

几乎每个人都玩过扑克牌的游戏。一般的扑克玩法都是一边摸牌，一边理牌。假如我们拿到了这样一手牌，好像是同花顺，别急，我们得理一理顺序才知道是否真的是同花顺。请问，应该如何理牌呢？应该说，哪怕你是第一次玩扑克牌，只要认识这些数字，理牌的方法都是不用教的。将 4 和 5 移动到 6 的左侧，再将 2 移动到最左侧，顺序就算是理好了。这就是理牌的方法，也是所谓的直接插入排序法。

下面对插入排序的每条语句进行数学分析。

Insertion_sort(A)	代价	次数
1. *for j←2 to length[A]*	C_1	n
2. *do key←A[j]*	C_2	n-1
3. 将 *A[j]*插入排好序的序列	0	n-1
4. *i←j*-1	C_4	n-1
5. *while i*>0 *and A[i]>key*	C_5	$\sum_{j=2}^{n}t_j$
6. *do A[i* + 1*]←A[i]*	C_6	$\sum_{j=2}^{n}(t_j-1)$
7. *i←i*-1	C_7	$\sum_{j=2}^{n}(t_j-1)$
8. *A[i* + 1*]←key*	C_8	n-1

由上面的分析，可以求得其时间复杂度为

$$T_{(n)}=C_1n+C_2(n-1)+C_4(n-1)+$$
$$C_5\sum_{j=2}^{n}t_j+C_6\sum_{j=2}^{n}(t_j-1)+C_7\sum_{j=2}^{n}(t_j-1)+C_8(n-1)$$

最好的情况为顺序排列，因此最好情况下运行时间可表示为：

$$T_{\min}(N)=C_1n+C_2(n-1)+C_4(n-1)+C_5(n-1)+C_8(n-1)$$
$$=(C_1+C_2+C_4+C_5+C_8)n-(C_2+C_4+C_5+C_8)$$
$$=an+b$$

逆序为最坏情况。要将每个 A[j]与已排序的子数组 A[1，…，j－1]中的每一个作比较，就有 $t_j=j,\ j=2,3,\cdots,n$

$$\sum_{j=2}^{n}j=\frac{n(n+1)}{2}-1 \text{ 和 } \sum_{j=2}^{n}(j-1)=\frac{n(n-1)}{2}$$

$$T_{\max}(N)=C_1n+C_2(n-1)+C_4(n-1)+C_5\left[\frac{n(n+1)}{2}-1\right]+C_6\left[\frac{n(n-1)}{2}\right]+$$
$$C_7\left[\frac{n(n-1)}{2}\right]+C_8(n-1)$$
$$=\left(\frac{C_5}{2}+\frac{C_6}{2}+\frac{C_7}{2}\right)n^2+\left(C_1+C_2+C_4+\frac{C_5}{2}-\frac{C_6}{2}-\frac{C_7}{2}+C_8\right)n-$$
$$(C_2+C_4+C_5+C_8)$$

考虑到分析算法复杂性的目的在于比较求解同一问题的两个不同算法的效率，而当要比较的两个算法的渐近复杂性的阶不相同时，只要能确定出各自的阶，就可以判定哪一个算法的效率高。换句话说，这时的渐近复杂性分析只要关心 $T(N)$ 的阶就够了，不必关心包含在 $T(N)$ 中的常数因子。所以，我们常常又对 $T(N)$ 的分析进一步简化，即假设算法中用到的所有不同的元运算各执行一次，所需要的时间都是一个单位时间。

综上所述，我们已经给出了简化算法复杂性分析的方法，当问题的规模充分大时只要考察算法复杂性在渐近意义下的阶。与此简化的复杂性分析方法相适应，特引入五个渐近意义下的记号。这些记号是用定义在自然数 $\mathbf{N}=\{0，1，2\cdots\}$ 上的函数来定义的。因为 $T(n)$ 一般仅定义在整数的输入规模上。记号如下：

1. Θ—渐近确界（上下限：lower bound and upper bound)

$\Theta(g(n))=\{f(n)$：存在正常数 c_1, c_2 和 n_0，使对所有的 $n\geqslant n_0$，有 $0\leqslant c_1.g(n)\leqslant f(n)\leqslant c_2.g(n)\}$。即对于任意一个函数 $f(n)$，若存在正常数 c_1，c_2，使当 n 充分大时，$f(n)$能被夹在 $c_1\cdot g(n)$和 $c_2\cdot g(n)$中间，则 $f(n)$属于集合 $\Theta(g(n))$。

由于时间和空间需求都是非负值，因此当 n 足够大时 $f(n)$是非负值（渐近正函数），$g(n)$也必须是渐近非负的。

例 1　证明 $1/2n^2-3n=\Theta(n^2)$。

要证明存在常数 c_1，c_2 和 n_0，使得对所有的 $n\geqslant n_0$，有 $c_1n^2\leqslant 1/2n^2-3n\leqslant c_2n^2$ 成立

用 n^2 除以不等式得：$c_1\leqslant 1/2-3/n\leqslant c_2$

于是，$c_2\geqslant 1/2-3/n$ 得知：$c_2\geqslant 1/2$ 时对所有 $n\geqslant 1$ 成立

同样可知，$c_1\leqslant 1/2-3/n$ 在 $c_1\leqslant 1/14$ 时对所有 $n\geqslant 7$ 成立

因此，取 $c_1=1/14$，$c_2=1/2$，$n_0\geqslant 7$ 有 $c_1n^2\leqslant 1/2n^2-3n\leqslant c_2n^2$ 成立！

2. O—渐近上界（upper bound）

O 记号一般是指最小上界。

$O(g(n)) = \{f(n)$：存在正常数 c 和 n_0，使对所有的 $n \geqslant n_0$，有 $0 \leqslant f(n) \leqslant c \cdot g(n)\}$。即对于任意一个函数 $f(n)$，若存在正常数 c，使当 n 充分大时，$f(n)$的值总在 $c \cdot g(n)$之下，则 $f(n)$属于集合 $O(g(n))$。

为了表示一个函数 $f(n)$是集合 $O(g(n))$的一个元素，记：

$$f(n) = O(g(n))$$

按照 O 的定义，容易证明它有如下运算规则：

$O(f) + O(g) = O(\max(f,g))$；

$O(f) + O(g) = O(f + g)$；

$O(f) \cdot O(g) = O(f \cdot g)$；

如果 $g(n) = O(f(n))$，则 $O(f) + O(g) = O(f)$；

$O(Cf(n)) = O(f(n))$，其中 C 是一个正的常数；

$f = O(f)$。

3. Ω—渐近下界（lower bound）

Ω 记号一般是指最大下界。

$\Omega(g(n)) = \{f(n)$: 存在正常数 c 和 n_0，使对所有的 $n \geqslant n_0$，有 $0 \leqslant c \cdot g(n) \leqslant f(n)\}$

即对于任意一个函数 $f(n)$，若存在正常数 c，使当 n 充分大时，$f(n)$的值等于或大于 $c \cdot g(n)$，则 $f(n)$属于集合 $\Omega(g(n))$。

4. o 记号

o 记号所提供的渐近上界可能是渐近紧确的也可能不是渐近紧确的。

例如，$2n^2 = O(n^2)$是渐进紧确的，$2n = o(n^2)$就不是渐近紧确的。使用 o 符号来表示非渐近紧确的上界。即：

$f(n) = o(g(n)) = \{$对任意正常数 c，存在常数 $n_0 > 0$，使得所有的 $n \geqslant n_0$，有 $0 \leqslant f(n) \leqslant c \cdot g(n)\}$

5. ω 记号

ω 记号与 Ω 记号的关系就像 o 记号和 O 记号的关系一样。

$f(n) = \omega(g(n)) = \{f(n)$：对任意正常数 c，存在常数 $n_0 > 0$，使得所有的 $n \geqslant n_0$，有 $0 \leqslant c \cdot g(n) < f(n)\}$

1.2 生成函数

生成函数即母函数，生成函数有普通型生成函数和指数型生成函数两种，其中普通型用得较多。简单来说，生成函数的应用在于研究未知（通项）数列规律，用这种方法在给出递推式的情况下求出数列的通项，生成函数是推导 Fibonacci 数列的通项公式方法之一。另外生成函数也广泛应用于编程与算法设计、分析上，运用这种数学方法往往对程序效率与速度有很大改进。

幂级数 $P(x)=a_0+a_1x+a_2x^2+\cdots$ 是我们所熟悉的多项式，我们定义 $P(x)$ 为数列 $\{a_0,a_1,a_2,\cdots\}$ 的生成函数，通常记为 $G\{a_n\}$。

生成函数的中心思想是：首先使用多项式或幂级数把需要研究的数列合为一个整体，通过研究多项式或幂级数的性质以及使用合并同类项的方法，来研究数列的性质，从而得到相关的结论。

例如：数列 $2^0,2^1,2^2,\cdots,2^n,\cdots$ 的生成函数是：

$$f(x)=2^0+2^1x+2^2x^2+\cdots$$

这个生成函数的值为

$$f(x)=\frac{1}{1-2x}$$

用了非常简洁紧凑的方式显示了上述数列的序列信息。

常见的生成函数有：

（1）$\dfrac{1}{1-x}=1+x+x^2+\cdots$

（2）$\dfrac{1}{(1-x)^n}=1+nx+\dfrac{n(n+1)}{2!}x^2+\dfrac{n(n+1)(n+2)}{3!}x^3+\cdots$

（3）$\dfrac{1}{1-2x}=2^0+2^1x+2^2x^2+\cdots$

（4）$\dfrac{1}{1-ax}=a^0+a^1x+a^2x^2+\cdots$

（5）$\dfrac{x}{(1-x)^2}=1+x+2x^2+\cdots+kx^k+\cdots$

（6）$\dfrac{x(1+x)}{(1-x)^3}=1^2x+2^2x^2+3^2x^3+\cdots+k^2x^k+\cdots$

（7）$\dfrac{2x}{(1-x)^3}=(1\times2)x+(2\times3)x^2+(3\times4)x^3+\cdots+k\times(k+1)x^k+\cdots$

（8）$\dfrac{6x}{(1-x)^4}=(1\times2\times3)x+(2\times3\times4)x^2+(3\times4\times5)x^3+\cdots+k\times(k+1)\times(k+2)x^k+\cdots$

（9）$e^x=1+\dfrac{1}{1!}x+\dfrac{1}{2!}x^2+\cdots+\dfrac{1}{k!}x^k+\cdots$

1.2.1 基本性质

首先假定，序列 $\{a_i\}$，$\{b_j\}$ 的生成函数分别为

$$P(x)=a_0+a_1x+a_2x^2+\cdots$$
$$Q(x)=b_0+b_1x+b_2x^2+\cdots$$

因为生成函数与数列之间是一一对应的关系，所以研究两个数列之间的关系可以转化为研究其生成函数之间的关系，这样就给解题带来了许多便利。

1. 线性性质

（1）若 $b_n = ca_n$，则 $Q(x) = cP(x)$

（2）若 $c_n = a_n + b_n$，则 $C(x) = Q(x) + P(x)$

2. 乘积性质

若 $c_n = \sum_{i=1}^{n} a_i b_{(n-i)}$，则 $C(x) = Q(x)P(x)$

3. 移位性质

（1）若 $b_k = \begin{cases} 0 & (k < i) \\ a_{k-i} & (k \geqslant i) \end{cases}$，则 $Q(x) = x^i P(x)$

（2）若 $b_n = a_{k+i}$，则 $Q(x) = \frac{1}{x^j}\left(P(x) - \sum_{k=0}^{i-1} a_k x^k \right)$

（3）若 $b_n = \sum_{i=0}^{k} a_i$，则 $Q(x) = \frac{P(x)}{1\text{-}x}$

（4）若 $b_n = \sum_{i=k}^{\infty} a_i$，则 $Q(x) = \frac{P(1) - xP(x)}{1\text{-}x}$，其中 $\sum_{i=k}^{\infty} a_i$ 是收敛的。

4. 换元性质

若 $b_n = c^n a_n$，则 $Q(x) = P(cx)$

5. 求导与积分性质

（1）若 $b_n = na_n$，则 $Q(x) = xP'(x)$

（2）若 $b_n = \frac{a_n}{n+1}$，则 $Q(x) = \frac{1}{x}\int_0^x P(t)\mathrm{d}t$

1.2.2 生成函数的计算

计算生成函数系数的方法是把比较复杂的生成函数化简为简单的二项式类型或若干个二项式类型的生成函数的积，这样就比较容易得出所需的 x^k 的系数。我们需要用到牛顿二项式定理及其生成函数的性质。

牛顿二项式定理：

设实数 a，对一切 x, y 有：

$$(x+y)^a = \sum_{i=0}^{a} \binom{a}{i} x^i y^{a-i}$$

其中 $\binom{a}{i} = \frac{a(a-1)\cdots(a-i+1)}{i!}$。

当 $a = n$ 时，变成我们所熟悉的二项式定理

$$(1+x)^n = 1 + C(n,1)x + C(n,2)x^2 + \cdots + C(n,n)x^n$$

当 $a=-n$ 时，

$$(1+x)^{-n} = \sum_{k=0}^{\infty}(-1)^k \binom{n+k-1}{k} x^k \quad |x|<1$$

$$(1-x)^{-n} = \sum_{k=0}^{n} \binom{n+k-1}{k} x^k \quad |x|<1$$

例 2 求解 $(1+x+x^2+\cdots+x^{10})(1+x+x^2+\cdots)^5$。

解：

$$\begin{aligned}
&(1+x+x^2+\cdots+x^{10})(1+x+x^2+\cdots)^5 \\
&= \left(\frac{1-x^{11}}{1-x}\right)\left(\frac{1}{1-x}\right)^5 \\
&= (1-x^{11})\left(\frac{1}{1-x}\right)^6 \\
&= (1-x^{11})\left\{1+\binom{1+6-1}{1}x+\binom{2+6-1}{2}x^2+\cdots+\binom{k+6-1}{k}x^k+\cdots\right\}
\end{aligned}$$

利用牛顿二项式定理求得生成函数的系数。

例 3 已知 $A(x)=G\{k^3\}$，求解 $A(x)$ 的值。

解：

$$\begin{aligned}
G\{k^3\} &= G\{k(k+1)(k+2)\} - 3G\{k^2\} - 2G\{k\} \\
&= \frac{6x}{(1-x)^4} - \frac{3x(1+x)}{(1-x)^3} - \frac{2x}{(1-x)^2}
\end{aligned}$$

例 4 x^{12} 在 $(x^2+x^3+x^4+\cdots)^5$ 中的系数？

解：

$$\begin{aligned}
&(x^2+x^3+x^4+\cdots)^5 \\
&= x^{10}(1+x+x^2+\cdots)^5 \\
&= x^{10}\frac{1}{(1-x)^5}
\end{aligned}$$

$\frac{1}{(1-x)^5}$ 中 x^2 的系数是 $\binom{5-1+2}{2}$ 即 15，所以 x^{12} 的系数是 15。

同理可得：

$$x^k = \begin{cases} 0 & k<10 \\ \binom{5-1+k-10}{k-10} & k \geqslant 10 \end{cases}$$

1.3 递归方程求解

递归算法在最坏情况下的时间复杂性渐近阶的分析，都转化为求相应的一个递归方程的

解的渐近阶。因此，求递归方程的解的渐近阶是对递归算法进行分析的关键步骤。递归方程的形式多种多样，求其解的渐近阶的方法也多种多样。

1.3.1 递推法

例 5 Hanoi 塔问题递归算法的时间复杂性，由以下递归方程给出：

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

递推求解如下：

$$\begin{aligned} 2T(n) &= 2T(n-1)+1 \\ &= 2(2T(n-2)+1)+1 \\ &= 2^2T(n-2)+2+1 \\ &= 2^2(2T(n-3)+1)+2+1 \\ &= 2^3T(n-3)+2^2+2+1 \\ &\cdots\cdots \\ &= 2^{n-1}T(1)+2^{n-2}+\cdots+2^2+2+1 \\ &= 2^{n-1}+2^{n-2}+\cdots+2^2+2+1 \\ &= 2^n-1 \end{aligned}$$

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

例 6 分治法实例。设 n 表示问题的尺寸，n/b 表示将问题分成 a 个子问题后的每个子问题的尺寸，其中 a，b 为常数。$d(n)$表示在分解或合成子问题而使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的递归方程给出：

分治法中时间复杂度一般为如下递归方程给出：

$$T(n)=\begin{cases} 1 & n=1 \\ \mathrm{a}T\left(\dfrac{n}{\mathrm{b}}\right)+d(n) & n\geqslant 2 \end{cases}$$

其中，a 是子问题的个数，b 是递进的步长，$d(n)$是合成子问题的开销。

$$\begin{aligned} T(n) &= \mathrm{a}T\left(\frac{n}{\mathrm{b}}\right)+d(n) \\ &= \mathrm{a}\left[\mathrm{a}T\left(\frac{n}{\mathrm{b}^2}\right)+d\left(\frac{n}{\mathrm{b}}\right)\right]+d(n) \\ &= \mathrm{a}^2T\left(\frac{n}{\mathrm{b}^2}\right)+\mathrm{a}d\left(\frac{n}{\mathrm{b}}\right)+d(n) \\ &= \mathrm{a}^3T\left(\frac{n}{\mathrm{b}^3}\right)+\mathrm{a}^2d\left(\frac{n}{\mathrm{b}^2}\right)+\mathrm{a}d\left(\frac{n}{\mathrm{b}}\right)+d(n) \\ &= \mathrm{a}^kT\left(\frac{n}{\mathrm{b}^k}\right)+\mathrm{a}^{k-1}d\left(\frac{n}{\mathrm{b}^{k-1}}\right)+\cdots+\mathrm{a}d\left(\frac{n}{\mathrm{b}}\right)+d(n) \end{aligned}$$

令 $\frac{n}{b^k}=1$，则 $k=\log_b n$

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

$a^k=a^{\log_b n}=n^{\log_b a}=n^p$（令 $\log_b a=p$）

此方程中 n^p 为齐次解，第二项为特解，特解很难估计，下面对 $d(n)$ 的三种情况进行分析：

（1）当 $d(n)$ 为常数 c 时：

$$\sum_{i=0}^{k-1}a^i c=c\sum_{i=0}^{k-1}a^i$$

$$a^k=n^p$$

$$T(n)=a^k+c_1a^{k-1}+ca^{k-2}+\cdots+ca+a\leqslant ca^k=O(a^k)=O(n^p)=O(n^{\log_b a})$$

（2）当 $d(n)$ 为线性函数 cn 时：

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

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

$$T(n)=n^p+cnk=n+cnk=n+cn\log_b n=O(n\log_b a)$$

② 当 $a\neq b$ 时

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

又∵

$$\left(\frac{a}{b}\right)^k=\frac{a^k}{b^k}=\frac{a^{\log_b n}}{b^{\log_b n}}=\frac{a^k}{n}=\frac{n^p}{n}\quad(p=\log_b a)$$

$$=\frac{cn}{\left(1-\frac{a}{b}\right)}\cdot\left(\frac{np}{n}-1\right)=c'np-c'n$$

∴ $T(n)=n^p+c'n'-cn$

∴ $T(n)=c_1n^p+c_2n\qquad c_1=1+c'\qquad c_2=-c'$

当 $a\neq b$ 时，则 $T(n)=O(n^p+n)$

若 $a<b$ 时，$\log_b a<1$，则 $T(n)=O(n)$

若 $a>b$ 时，$T(n)=O(n')$

故当 $d(n)$ 为 cn 时，其复杂度如下：