第 2 章　整数的表示法和运算

整数的各种表示方法对于人们和计算机对这些整数进行有效运算有着重大的影响. 本章的目的是给出整数如何进行 b 进制展开，以及如何用这种展开式进行整数的基本算术运算. 特别地，我们要证明，对正整数 b，每个正整数有唯一的 b 进制展开式，例如当 b 为 10 时，我们有整数的十进制展开式. 当 b 为 2 时，我们有这个整数的二进制展开式. 而当 b 为 16 时，我们有十六进制展开式. 我们将给出整数进行 b 进制展开的一个程序和用 b 进制展开作整数算术运算的基本算法. 最后，在介绍大 O 符号之后，我们用位运算次数的大 O 估计来分析这些基本运算的计算复杂性.

2.1　整数的表示法

我们在日常生活中采用十进制表示整数. 用一些数字表示 10 的方幂来把整数写下来. 例如把一个整数写成 37 465，意思是

$$3 \cdot 10^4 + 7 \cdot 10^3 + 4 \cdot 10^2 + 6 \cdot 10 + 5.$$

十进制是计数制的一个例子，其中每个数字的位置决定它所代表的数值. 从古到今，人们还采用过许多其他表示整数的方法. 例如，三千年前巴比伦数学家采用十六进制表示整数. 罗马人采用的罗马数字在今天还用来表示年份. 古代玛雅人采用二十进制. 还有许多计数系统也被发明和使用过.

十进制成为一种固定下来的计数方法，很可能是因为人有十个手指. 我们还会看到，每个大于 1 的正整数都可作为进位制的基底. 随着计算机的发明和发展，十以外的进位制变得越来越重要. 特别是以 2，8 和 16 为基底的进位制在计算机各种功能中得到广泛的采用.

在本节中，我们将要说明无论把哪个整数 b 取为基底，每个正整数都可以唯一地表示为以 b 为基底的记号. 在 2.2 节中，我们将要说明如何应用这种表示来进行整数运算. （参考本节末的习题，学习计算机用来表示正负数的补 1 表示法和补 2 表示法.）

关于正整数系统的有趣历史的更多信息，我们给读者推荐[Or88]或[Kn97]，其中可以找到大量的综述和很多参考文献.

我们现在证明每个大于 1 的正整数都可以被取为基底.

定理 2.1　令 b 是正整数，$b>1$，则每个正整数 n 都可以被唯一地写为如下形式：

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0,$$

其中 k 为非负整数，a_j 为整数，$0 \leqslant a_j \leqslant b-1 (j=0, 1, \cdots, k)$，且首项系数 $a_k \neq 0$.

证明　我们按照下述方法通过连续应用带余除法来得到所描述类型的表示. 首先用 b 除 n 得到

$$n = bq_0 + a_0, \quad 0 \leqslant a_0 \leqslant b-1.$$

如果 $q_0 \neq 0$，则用 b 除 q_0 得到

$$q_0 = bq_1 + a_1, \quad 0 \leqslant a_1 \leqslant b-1.$$

继续这个过程得到

$$\begin{aligned} q_1 &= bq_2 + a_2, & 0 \leqslant a_2 \leqslant b-1,\\ q_2 &= bq_3 + a_3, & 0 \leqslant a_3 \leqslant b-1,\\ &\vdots\\ q_{k-2} &= bq_{k-1} + a_{k-1}, & 0 \leqslant a_{k-1} \leqslant b-1,\\ q_{k-1} &= b \cdot 0 + a_k, & 0 \leqslant a_k \leqslant b-1. \end{aligned}$$

当得到商 0 时这个过程就到了最后一步. 为了看清楚这一点，首先注意商序列满足

$$n > q_0 > q_1 > q_2 > \cdots \geqslant 0.$$

因为序列 q_0，q_1，q_2，…是一个递减的非负整数序列，且只要其中的项为正数就继续下去，因而在这个序列中至多存在 q_0 个项，且最后一项为 0.

从上面的第一个方程可以看出

$$n = bq_0 + a_0.$$

下面用第二个方程取代 q_0，得到

$$n = b(bq_1 + a_1) + a_0 = b^2 q_1 + a_1 b + a_0.$$

顺次取代 q_1，q_2，…，q_{k-1}，我们得到

$$\begin{aligned} n &= b^3 q_2 + a_2 b^2 + a_1 b + a_0\\ &\vdots\\ &= b^{k-1} q_{k-2} + a_{k-2} b^{k-2} + \cdots + a_1 b + a_0\\ &= b^k q_{k-1} + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0\\ &= a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0, \end{aligned}$$

其中 $0 \leqslant a_j \leqslant b-1$，$j=0$，1，…，$k$ 且 $a_k \neq 0$. 给定 $a_k = q_{k-1}$ 为最后的非零商. 这样，我们就找到了所述类型的展开式.

为了说明这个展开式的唯一性，假定有两个等于 n 的这种展开式，即

$$\begin{aligned} n &= a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0\\ &= c_k b^k + c_{k-1} b^{k-1} + \cdots + c_1 b + c_0, \end{aligned}$$

其中 $0 \leqslant a_k < b$，$0 \leqslant c_k < b$(并且如果必要，我们在其中的一个展开式中添加零系数的起始项以使得它们的项数相同). 从一个展开式中减去另外一个，我们得到

$$(a_k - c_k)b^k + (a_{k-1} - c_{k-1})b^{k-1} + \cdots + (a_1 - c_1)b + (a_0 - c_0) = 0.$$

如果这两个展开式不同，则存在一个最小的整数 j，$0 \leqslant j \leqslant k$，使得 $a_j \neq c_j$. 因此，

$$b^j((a_k - c_k)b^{k-j} + \cdots + (a_{j+1} - c_{j+1})b + (a_j - c_j)) = 0,$$

故

$$(a_k - c_k)b^{k-j} + \cdots + (a_{j+1} - c_{j+1})b + (a_j - c_j) = 0.$$

从中解出 $a_j - c_j$，得到

$$\begin{aligned} a_j - c_j &= (c_k - a_k)b^{k-j} + \cdots + (c_{j+1} - a_{j+1})b\\ &= b((c_k - a_k)b^{k-j-1} + \cdots + (c_{j+1} - a_{j+1})). \end{aligned}$$

因此，我们看到

$$b \mid (a_j - c_j).$$

但是因为 $0 \leqslant a_j < b$ 且 $0 \leqslant c_j < b$，故 $-b < a_j - c_j < b$. 因此 $b \mid (a_j - c_j)$意味着 $a_j = c_j$. 这与假

设两个展开式不同矛盾. 综上我们得到 n 关于基 b 的展开式是唯一的. ■

对于 $b=2$，由定理 2.1 可知下面的推论成立.

推论 2.1.1 每个正整数都可以被表示为 2 的不同幂次的和.

证明 设 n 为正整数. 在定理 2.1 中取 $b=2$，我们知道 $n=a_k2^k+a_{k-1}2^{k-1}+\cdots+a_1 2+a_0$，其中每个 a_j 或者为 0 或者为 1. 因此每个正整数都是 2 的不同幂次的和. ■

在定理 2.1 所描述的展开式中，b 被称为展开式的基(base)或根(radix). 我们称基为 10 的表示(即通常整数的写法)为十进制(decimal)表示. 基为 2 的表示被称为二进制(binary)表示，基为 8 的表示被称为八进制(octal)表示，基为 16 的表示被称为十六进制(hexadecimal)表示，或者简称为 hex. 系数 a_j 被称为展开式的位(digit). 在计算机术语中二进制数字被称为比特(bit，是英文 binary digit 的缩写).

为了区别整数关于不同基的表示，我们采用一种特别的记号，用 $(a_ka_{k-1}\cdots a_1a_0)_b$ 来表示数 $a_kb^k+a_{k-1}b^{k-1}+\cdots+a_1b+a_0$.

例 2.1 为了说明基为 b 的表示，注意到 $(236)_7=2\cdot 7^2+3\cdot 7+6=125$ 和 $(10010011)_2=1\cdot 2^7+1\cdot 2^4+1\cdot 2^1+1=147$. ◀

定理 2.1 的证明提供了一种求任意一个正整数 n 的 b 进制展开 $(a_ka_{k-1}\cdots a_1a_0)_b$ 的方法. 特别地，为了求 n 的 b 进制展开，我们首先要用 b 除 n，余数为数字 a_0. 然后用 b 除商 $[n/b]=q_0$，余数为数字 a_1. 继续这个过程，连续地用 b 除得到商，以获得 n 关于基 b 的展开式中的数字. 一旦得到的商为 0，这个过程就停止. 换句话说，为了求得 n 的 b 进制展开，我们重复地使用除法，每次用商取代被除数，当商为 0 时停止. 然后从下到上读取余数序列来得到 b 进制展开. 下面用例 2.2 来说明这个过程.

例 2.2 为了求出 1864 的二进制展开式，我们连续使用除法：

$$
\begin{aligned}
1864 &= 2\cdot 932+0,\\
932 &= 2\cdot 466+0,\\
466 &= 2\cdot 233+0,\\
233 &= 2\cdot 116+1,\\
116 &= 2\cdot 58+0,\\
58 &= 2\cdot 29+0,\\
29 &= 2\cdot 14+1,\\
14 &= 2\cdot 7+0,\\
7 &= 2\cdot 3+1,\\
3 &= 2\cdot 1+1,\\
1 &= 2\cdot 0+1.
\end{aligned}
$$

为了得到 1864 的二进制展开式，只需取这些除法中的余数即可，就是说 $(1864)_{10}=(11101001000)_2$. ◀

计算机内部是使用一系列状态为“开”或者“关”的“开关”来表示数的. (这可以使用磁头、电开关或者其他手段机械地实现.)因此，每个开关可以有两个可能的状态. 我们可以使用“开”来表示数字 1，而“关”表示数字 0；这就是为什么计算机内部使用二进制来表示整数.

为了实现不同的目的，计算机中也使用 8 或 16 为基. 在基于 16(十六进制)的表示中有 16 个数字，通常使用 0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F. 字母 A，B，C，D，E 和 F 被用来表示对应于 10，11，12，13，14 和 15(用十进制的写法)的数字. 下面的例子说明了从十六进制到十进制表示的转换.

例 2.3 把$(A35B0F)_{16}$从十六进制转换为十进制表示：

$$(A35B0F)_{16} = 10 \cdot 16^5 + 3 \cdot 16^4 + 5 \cdot 16^3 + 11 \cdot 16^2 + 0 \cdot 16 + 15$$
$$= (10705679)_{10}.$$

在二进制与十六进制表示之间可以有一个简单的转换. 我们可以把每个十六进制数字根据表 2.1 给出的对应关系写成一个由四位二进制数字组成的块.

表 2.1 从十六进制到二进制的转化

十六进制数	二进制数	十六进制数	二进制数
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

例 2.4 从十六进制到二进制的转换的一个例子是$(2FB3)_{16} = (10111110110011)_2$. 每个十六进制数字被转换为一个四位二进制数字块(与数字$(2)_{16}$相关的初始块$(0010)_2$的起始的零被省略了).

为了把二进制数转换为十六进制，考虑$(11110111101001)_2$. 我们从右端开始把这个数划分为四位的块. 这些块从右到左分别是 1001，1110，1101 和 0011(添加了两个起始的零). 把每个块转换为十六进制，我们得到$(3DE9)_{16}$.

我们注意到当两个基中一个是另一个的幂次时，它们之间的转化与二进制-十六进制的转化一样容易.

2.1 节习题

1. 把$(1999)_{10}$从十进制表示转换为七进制表示. 把$(6105)_7$从七进制表示转换为十进制表示.
2. 把$(89156)_{10}$从十进制表示转换为八进制表示. 把$(706113)_8$从八进制表示转换为十进制表示.
3. 把$(10101111)_2$从二进制表示转换为十进制表示，并把$(999)_{10}$从十进制表示转换为二进制表示.
4. 把$(101001000)_2$从二进制表示转换为十进制表示，并把$(1984)_{10}$从十进制表示转换为二进制表示.
5. 把$(100011110101)_2$和$(11101001110)_2$从二进制转换为十六进制.
6. 把$(ABCDEF)_{16}$，$(DEFACED)_{16}$和$(9A0B)_{16}$从十六进制转换为二进制.
7. 解释为何在实际中当我们把大的十进制整数分成三位的块并用空格隔开时是在使用基为 1000 的表示.

8. 证明：如果 b 是小于 -1 的负整数，则每个非零整数 n 可以被唯一地写成如下形式：

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0,$$

其中 $a_k \neq 0$ 且 $0 \leqslant a_j < |b|$，$j=0, 1, 2, \cdots, k$. 我们把它写成为 $n=(a_k a_{k-1} \cdots a_1 a_0)_b$，就像在基为正数那样.

9. 求 $(101001)_{-2}$ 和 $(12012)_{-3}$ 的十进制表示.

10. 求十进制数 -7，-17 和 61 的基于 -2 的表示.

11. 证明当所有的砝码都放在一个盘子中时，不超过 2^k-1 的重量可以使用重为 1，2，2^2，…，2^{k-1} 的砝码来测量.

12. 证明每个非零整数可以被唯一地表示为如下形式：

$$e_k 3^k + e_{k-1} 3^{k-1} + \cdots + e_1 3 + e_0,$$

其中 $e_j = -1$，0 或 $1(j=0, 1, 2, \cdots, k)$且 $e_k \neq 0$. 这个展开式被称为平衡三元展开式(balanced ternary expansion).

13. 应用习题 12 证明当砝码可以被放在任何一个盘子中时，不超过 $(3^k-1)/2$ 的重量可以使用重为 1，3，3^2，…，3^{k-1} 的砝码来测量.

14. 解释如何从三进制表示转换为九进制表示，以及如何从九进制表示转换为三进制.

15. 解释如何从基于 r 的表示转换为基于 r^n 的表示，以及如何从基于 r^n 的表示转换为基于 r 的表示，其中 $r>1$ 且 n 为正整数.

16. 证明：如果 $n=(a_k a_{k-1} \cdots a_1 a_0)_b$，则 n 被 b^j 除所得的商和余数分别是 $q=(a_k a_{k-1} \cdots a_j)_b$，$r=(a_{j-1} \cdots a_1, a_0)_b$.

17. 如果 n 的 b 进制展开为 $n=(a_k a_{k-1} \cdots a_1 a_0)_b$，那么 $b^m n$ 的 b 进制展开是什么？

整数的补 1 表示被用来简化计算机算法. 为了表示绝对值小于 2^n 的正、负整数，一共要用到 $n+1$ 字节.

最左边的字节被用来表示符号. 这个位置上的 0 用来表示正数，而 1 用来表示负数.

对于正整数，余下的字节和整数的二进制表示是一样的. 对于负整数，余下的字节如下确定：首先求这个整数的绝对值的二进制表示，然后对每个字节取其补. 这里 1 的补为 0，而 0 的补为 1.

18. 求下列整数的补 1 表示，使用长度为 6 的字节串.

a) 22　　b) 31　　c) -7　　d) -19

19. 下面长度为五的表示是哪个整数的补 1 表示？

a) 11001　　b) 01101　　c) 10001　　d) 11111

20. 当使用长度为 n 的字节串时，如何从 m 的补 1 表示得到 $-m$ 的补 1 表示？

21. 证明：如果整数 m 的补 1 表示为 $a_{n-1} a_{n-2} \cdots a_1 a_0$，那么 $m = -a_{n-1}(2^{n-1}-1) + \sum_{i=0}^{n-2} a_i 2^i$.

整数的补 2 表示也被用来简化计算机算法(事实上，它们比补 1 表示更常用). 为了表示满足 $-2^{n-1} \leqslant x \leqslant 2^{n-1}-1$ 的整数 x，需要用到 n 个字节.

最左边的字节用来表示符号，0 表示正数，而 1 表示负数.

对于正整数，余下的 $n-1$ 个字节和该整数的二进制表示相同. 对于负整数，余下的字节是 $2^{n-1}-|x|$ 的二进制展开.

22. 用长度为六的字节串求习题 18 中的整数的补 2 表示.

23. 如果习题 19 中的每个数都是一个整数的补 2 表示，那么它们分别对应哪些整数？

24. 证明：如果整数 m 的补 2 表示为 $a_{n-1} a_{n-2} \cdots a_1 a_0$，那么 $m = -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$.

25. 当使用长度为 n 的字节串时，如何从 m 的补 2 表示得到 $-m$ 的补 2 表示？

26. 如何从一个整数的补 1 表示得到它的补 2 表示？

27. 有时整数编码采用由四位的二进制展开来表示一个十进制数字的方法，这产生了整数的二进制编码的十进制(binary coded decimal)形式. 例如，791 用这种方法编码为 011110010001. 使用这种编码方法需要用多少个字节来表示一个 n 位的十进制数？

正整数 n 的**康托尔展开**(Cantor expansion)是一个和式

$$n = a_m m! + a_{m-1}(m-1)! + \cdots + a_2 2! + a_1 1!,$$

其中每个 a_j 都是一个满足 $0 \leqslant a_j \leqslant j$ 的整数，且 $a_m \neq 0$.

28. 求 14，56 和 384 的康托尔展开.

* 29. 证明每个正整数有唯一的康托尔展开. (**提示**：对每个正整数 n，存在正整数 m 使得 $m! \leqslant n < (m+1)!$. 对于 a_m，取 n 除以 $m!$ 的商，然后迭代.)

中国的**拿子**(nim)游戏是这样玩的. 有几堆火柴棍，在游戏的开始每一堆中都包含着任意数目的火柴棍. 每一步中一个玩家从任意一堆火柴棍中拿走一根或多根. 玩家轮流拿火柴，谁拿到最后一根火柴谁就赢得游戏.

取胜位置是每堆火柴数目的一种置法，使得如果一个玩家可以把火柴拿走后，剩下火柴堆具有那种置法，则(无论第二个玩家怎么做)第一个玩家有必赢的方法. 这种位置的一个例子是有两堆火柴，每一堆包含一根火柴；这就是取胜位置，因为第二个玩家必须拿走一根火柴，从而把拿走最后一根火柴的取胜机会留给第一个玩家.

30. 证明在拿子游戏中，有两堆火柴而每堆都包含两根火柴的位置是取胜位置.

31. 对于火柴堆中火柴数目的每种组合，把每堆的火柴数目用二进制表示，然后把这些数每行一个排起来对齐(如果有必要在首位补零). 证明一个位置是取胜位置当且仅当在每一列中 1 的数目是偶数. (例如：三堆分别为 3，4 和 7 的火柴可以写为

$$\begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}$$

其中每一列恰有两个 1.)(**提示**：证明从一个取胜位置开始的任意一步都将产生非取胜位置，并证明从任意一个非取胜位置开始都存在一种做法达到一个取胜位置.)

设 a 为一个四位的十进制整数，其中所有的数字不全相同. 设 a' 是通过把 a 的各位数字按照递减的顺序排列得到的十进制整数，a'' 为通过把 a 的各位数字按照递增的顺序排列得到的十进制整数. 定义 $T(a) = a' - a''$. 例如，$T(7318) = 8731 - 1378 = 7353$.

* 32. 证明唯一一个使得 $T(a) = a$ 的四位十进制整数(其中所有的数字不全相同)为 $a = 6174$. 整数 6174 被称为**卡普瑞卡常数**(Kaprekar's constant)，是以印度数学家 D. R. Kaprekar 的名字命名的，因为它是具有这个性质的唯一整数.

** 33. a) 证明：如果 a 是一个有着四位十进制展开的正整数，并且所有的数字不全相同，则通过迭代 T 得到的序列 a，$T(a)$，$T(T(a))$，…，最终达到整数 6174.

b) 确定在(a)中定义的序列达到 6174 所需的最大步数.

卡普瑞卡(D. R. Kaprekar，1905—1986)出生于印度的 Dahanu，从小就对数字感兴趣. 他在 Thana 接受了中学教育，并曾在 Poona 的 Ferguson 学院学习. 卡普瑞卡后来进入了庞拜大学并于 1929 年获得学士学位. 从 1930 年直到 1962 年退休，他一直在印度的 Devlali 作教师. 卡普瑞卡发现了趣味数论中许多有意思的性质. 他发表过许多诸如幻方数、循环数以及其他具有特殊性质的整数的作品.

设 b 为正整数，a 是具有 b 进制四位展开式的整数，并且所有的数字不全相同. 定义 $T_b(a)=a'-a''$，其中 a' 是通过把 a 的基于 b 进制展开的各位数字按照递减的顺序排列得到的基于 b 进制展开的整数，a'' 为通过把 a 的基于 b 进制展开的各位数字按照递减的顺序排列得到的基于 b 进制展开的整数.

** 34. 设 $b=5$. 求唯一的一个具有五进制展开的四位整数 a_0 使得 $T_5(a_0)=a_0$. 证明这个整数 a_0 是一个基于 5 的卡普瑞卡常数；换句话说，只要 a_0 是以 5 为基的四位展开整数，并且并非所有的数字都相同，则 a，$T(a)$，$T(T(a))$，$T(T(T(a)))$，…最终达到 a_0.

* 35. 证明不存在基为 6 的四位数的卡普瑞卡常数.

* 36. 确定是否存在基为 10 的三位数的卡普瑞卡常数. 证明你的答案的正确性.

37. 一个序列 $a_j(j=1, 2, \cdots)$ 被称为是**西唐序列**(以匈牙利数学家西蒙·西唐(Simon Sidon)的名字命名)，如果所有的两项和 $a_i+a_j(i\leqslant j)$ 互不相同. 用定理 2.1 来证明 $a_j(j=1, 2, \cdots)$是西唐序列，其中 $a_j=2^j$.

计算和研究

1. 求下列各个整数的二进制、八进制和十六进制展开.
   a) 9876543210　　b) 1111111111　　c) 10000000001
2. 求下列各个整数的十进制展开.
   a) $(1010101010101)_2$　　b) $(765432101234567)_8$　　c) $(ABBAFADACABA)_{16}$
3. 求下列和式的值，用各自表达式所使用的基来表示你的答案.
   a) $(11011011011011011)_2+(10010010010010010010001)_2$
   b) $(12345670123456)_8+(765432107654321)_8$
   c) $(123456789ABCD)_{16}+(BABACACADADA)_{16}$
4. 求整数 100 000，10 000 000 和 1 000 000 000 的康托尔展开. (康托尔展开的定义参看习题 28 前面的导言.)
5. 对于各位数字不全相同的几个不同的四位整数验证习题 33 中描述的结果.
6. 通过计算数据给出一个关于序列 a，$T(a)$，$T(T(a))$，…的猜测，其中 a 为基于 10 表示的五位整数且所有数字不全相同，$T(a)$如习题 32 前的导言中所定义.
7. 研究序列 a，$T(a)$，$T(T(a))$…，关于不同的基 b 的规律，其中 a 为基于 b 表示的三位整数，你可以做出什么样的猜测？使用基于 b 表示的四位整数和五位整数重复你的研究.

程序设计

1. 从一个整数的十进制展开式求其二进制展开式，反之亦然.
2. 把基为 b_1 的表示转换为基为 b_2 的表示，其中 b_1 和 b_2 是大于 1 的任意整数.
3. 把二进制表示转换为十六进制表示，反之亦然.
4. 从一个整数的十进制表示求其基为(−2)的表示(参看习题 8).
5. 从一个整数的十进制展开式求其平衡三元展开式(参看习题 12).
6. 从一个整数的十进制展开式求其康托尔展开式(参看习题 28 前面的导言).
7. 设计一个在拿子游戏中的取胜策略(参看习题 30 前面的导言).

* 8. 研究序列 a，$T(a)$，$T(T(a))$，…(习题 32 前的导言中定义)，其中 a 为正整数，找出达到 6174 所需的最少步骤.

2.2 整数的计算机运算

在计算机发明之前，数学家是用手或一些机械设备来进行计算的. 采用这两种方法中的任何一种都只能处理不是很大的整数. 很多数论问题，例如大数分解和素性检验，都需

要计算100位甚至200位的整数. 在本节中，我们将要学习用计算机运算的一些基本算法. 在下面一节中，将研究实现这些算法所需要的运算的次数.

我们已经提过，计算机本质上是使用字节或二进制数来表示数的. 计算机对于可以在机器算法中使用的整数大小是有内在限制的. 这个上限被称为字长(word size)，用 w 表示. 字长通常是2的幂次，例如在奔腾系列上是 2^{32} 或 2^{35}，而有时字长为10的幂次.

为了实现关于大于字长的整数的算法，我们必须把每个整数用多个字来表示. 为了存储整数 $n>w$，我们把 n 作基于 w 的表示，并且对每个数位用计算机的一个字表示. 例如，如果字长为 2^{35}，则由于小于 2^{350} 的整数在采用基为 2^{35} 的表示时不超过10个数位，因此使用10个计算机字就可以存储像 $2^{350}-1$ 那么大的整数. 还要注意为了找到一个整数基于 2^{35} 的展开表示，我们只需要将长为35比特的块合并在一起.

讨论大整数的计算机算法的第一步是刻画基本的算术运算是如何系统地实现的.

下面描述 r 进制表示的整数的基本算术运算实现的经典方法，其中 $r>1$ 为整数. 这些方法是算法(algorithm)的例子.

定义 **算法**是为了实现一个计算或者解决一个问题的精确指令的有限集合.

算法(Algorithm)一词的来历

"Algorithm"是单词"algorism"的讹误，最初来源于9世纪一本书《Kitab al-jabr w'al-muqabala》(复位与约简规则)作者的名字 Abu Ja'far Mohammed ibn Mûsâ al-Khwârizmî(请参看稍后他的小传). "algorism"一词最初是指用印度-阿拉伯数字进行运算的规则. 但18世纪演变为"algorithm". 随着对机器计算的兴趣日益剧增，算法的概念也被广泛地理解为解决问题的所有确定步骤，而不仅仅限于当初用阿拉伯记法对整数的算术运算了.

阿布·贾法·穆哈默德·伊本·穆萨·阿科瓦里茨米(Abu Ja'far Mohammed Ibn Mûsâ al-Khwârizmî，780—850)是天文学家和数学家. 他是智慧堂即巴格达科学院的成员. 阿科瓦里茨米(al-Khwârizmî)的原意是"来自花剌子模(Kowarzizm)"，即现在乌兹别克斯坦的希瓦(Khiva). 阿科瓦里茨米写了很多关于数学、天文学和地理方面的书. 西方人从他的书中第一次学习了代数. 他的书名是《Kitab al-jabr w'al muqabala》，单词"algebra"就是来自于 al-jabr，这本书被翻译成拉丁文，并且被广泛地作为教科书使用. 他的另外一本书讲述了用印度－阿拉伯数字来进行算术计算的过程.

我们将要描述两个 n 位整数 $a=(a_{n-1}a_{n-2}\cdots a_1a_0)_r$ 和 $b=(b_{n-1}b_{n-2}\cdots b_1b_0)_r$ 的加法、减法和乘法，如果有必要则在初始位补零以使得两个展开式具有相同的长度. 这里描述的算法既适用于小于计算机字长的二进制整数，也适用于大于字长 w 且以 w 为基的整数的高精度(multiple precision)算法.

加法 当把 a 和 b 加在一起时，得到和

$$a+b=\sum_{j=0}^{n-1}a_jr^j+\sum_{j=0}^{n-1}b_jr^j=\sum_{j=0}^{n-1}(a_j+b_j)r^j.$$

为了求得 $a+b$ 的 r 进制展开式，首先根据带余除法，存在整数 C_0 和 s_0，使得

$$a_0+b_0=C_0r+s_0,\quad 0\leqslant s_0<r.$$

由于 a_0 和 b_0 为不超过 r 的正整数，故 $0\leqslant a_0+b_0\leqslant 2r-2$，因此 $C_0=0$ 或 1；这里 C_0 是进位到下一个位置的数. 下面，我们求整数 C_1 和 s_1，使得

$$a_1+b_1+C_0=C_1r+s_1,\quad 0\leqslant s_1<r.$$

由于 $0\leqslant a_1+b_1+C_0\leqslant 2r-1$，故 $C_1=0$ 或 1. 这样进行归纳，我们对于 $1\leqslant i\leqslant n-1$ 求整数 C_i 和 s_i，

$$a_i+b_i+C_{i-1}=C_ir+s_i,\quad 0\leqslant s_i<r,$$

其中 $C_i=0$ 或 1. 最后，设 $s_n=C_{n-1}$，这是由于两个 n 位整数相加若在第 n 个位置有进位则它们的和为 $n+1$ 位. 我们总结得到这个和基于 r 的展开式为 $a+b=(s_ns_{n-1}\cdots s_1s_0)_r$.

当我们手算基于 r 的和时，可以使用类似于十进制加法的技巧.

例 2.5 把$(1101)_2$和$(1001)_2$加起来，写作

$$\begin{array}{ccccc} \textit{1} & & & \textit{1} & \\ & 1 & 1 & 0 & 1 \\ + & 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 1 & 1 & 0 \end{array}$$

这里用斜体的 *1* 在适当的列上表明进位. 我们通过如下等式得到和的二进制数字：$1+1=1\cdot 2+0$，$0+0+1=0\cdot 2+1$，$1+0+0=0\cdot 2+1$，$1+1+0=1\cdot 2+0$. ◀

减法 假定 $a>b$. 考虑

$$a-b=\sum_{j=0}^{n-1}a_jr^j-\sum_{j=0}^{n-1}b_jr^j=\sum_{j=0}^{n-1}(a_j-b_j)r^j.$$

注意由带余除法，存在整数 B_0 和 d_0 使得

$$a_0-b_0=B_0r+d_0,\quad 0\leqslant d_0<r,$$

且由于 a_0 和 b_0 是小于 r 的整数，因而有

$$-(r-1)\leqslant a_0-b_0\leqslant r-1.$$

当 $a_0-b_0\geqslant 0$ 时，我们得到 $B_0=0$. 否则，当 $a_0-b_0<0$ 时，我们得到 $B_0=-1$；B_0 是从 a 的 r 进制展开式的下一个位置的借位数. 再次使用带余除法求整数 B_1 和 d_1，使得

$$a_1-b_1+B_0=B_1r+d_1,\quad 0\leqslant d_1<r.$$

从这个方程可以看出只要 $a_1-b_1+B_0\geqslant 0$，则借位 $B_1=0$，否则 $B_1=-1$，这是因为 $-r\leqslant a_1-b_1+B_0\leqslant r-1$. 这样一步步归纳地进行下去，可求出整数 B_i 和 d_i，使得

$$a_i-b_i+B_{i-1}=B_ir+d_i,\quad 0\leqslant d_i<r$$

其中 $B_i=0$ 或 -1，$1\leqslant i\leqslant n-1$. 由于 $a>b$，故 $B_{n-1}=0$. 于是得到

$$a-b=(d_{n-1}d_{n-2}\cdots d_1d_0)_r.$$

当我们手算基于 r 的减法时，可以使用类似于十进制减法的技巧.

例 2.6 用$(11011)_2$减去$(10110)_2$，我们有

$$\begin{array}{ccccccc} & & & \textit{-1} & & & \\ & 1 & 1 & 0 & 1 & 1 \\ - & 1 & 0 & 1 & 1 & 0 \\ \hline & & & 1 & 0 & 1 \end{array}$$

其中一列上面的斜体−1 表示一个借位. 我们通过如下等式得到差的二进制数字：1−0=0·2+1，1−1+0=0·2+0，0−1+0=−1·2+1，1−0−1=0·2+0 且 1−1+0=0·2+0. ◀

乘法 在讨论乘法之前，我们先讨论移位(shifting). 用 r^m 乘$(a_{n-1}a_{n-2}\cdots a_1a_0)_r$，只需要把展开式左移 m 位，并附加 m 个 0 位即可.

例 2.7 用 2^5 乘$(101101)_2$，我们把所有的数字左移五位并在后面附加五个零，得到$(10110100000)_2$. ◀

首先讨论一个 n 位整数与一个一位整数的乘法. 为了用$(b)_r$ 乘$(a_{n-1}\cdots a_1a_0)_r$，我们首先注意到

$$a_0b = q_0r + p_0, \quad 0 \leqslant p_0 < r,$$

且 $0 \leqslant q_0 \leqslant r-2$，这是因为 $0 \leqslant a_0b \leqslant (r-1)^2$. 接着有

$$a_1b + q_0 = q_1r + p_1, \quad 0 \leqslant p_1 < r,$$

且 $0 \leqslant q_1 \leqslant r-1$. 一般地，我们有

$$a_ib + q_{i-1} = q_ir + p_i, \quad 0 \leqslant p_i < r,$$

且 $0 \leqslant q_i \leqslant r-1$. 进一步，我们有 $p_n = q_{n-1}$. 这样得到$(a_{n-1}\cdots a_1a_0)_r(b)_r = (p_np_{n-1}\cdots p_1p_0)_r$.

为了实现两个 n 位整数的乘法，我们将其写成

$$ab = a\Big(\sum_{j=0}^{n-1} b_jr^j\Big) = \sum_{j=0}^{n-1}(ab_j)r^j.$$

对每个 j，首先用 b_j 乘 a，然后左移 j 位，最后把这样得到的所有 n 个整数加起来得到乘积.

当我们手算两个具有 r 进制展开的整数的乘积时，可以使用类似于十进制乘法的技巧.

例 2.8 把$(1101)_2$ 和$(1110)_2$ 相乘，有如下算式：

$$\begin{array}{cccccccc}
 & & & 1 & 1 & 0 & 1 & \\
 & & \times & 1 & 1 & 1 & 0 & \\
\hline
 & & & 0 & 0 & 0 & 0 & \\
 & & 1 & 1 & 0 & 1 & & \\
 & 1 & 1 & 0 & 1 & & & \\
1 & 1 & 0 & 1 & & & & \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0
\end{array}$$

注意首先用$(1110)_2$ 的每个数字乘$(1101)_2$，每次做适当数目的移位，然后把适当的整数相加得到积. ◀

除法 我们希望求出带余除法中的商 q

$$a = bq + R, \quad 0 \leqslant R < b.$$

如果 q 的 r 进制展开为 $q=(q_{n-1}q_{n-2}\cdots q_1q_0)_r$，则

$$a = b\Big(\sum_{j=0}^{n-1} q_jr^j\Big) + R, \quad 0 \leqslant R < b.$$

为了确定 q 的第一个数字 q_{n-1}，我们注意到

$$a - bq_{n-1}r^{n-1} = b\Big(\sum_{j=0}^{n-2} q_jr^j\Big) + R.$$

这个方程的右边不仅仅是正的，而且小于 br^{n-1}，这是因为 $\sum_{j=0}^{n-2} q_j r^j \leqslant \sum_{j=0}^{n-2}(r-1)r^j = \sum_{j=0}^{n-1} r^j - \sum_{j=0}^{n-2} r^j = r^{n-1}-1$．因此

$$0 \leqslant a - bq_{n-1}r^{n-1} < br^{n-1}.$$

这告诉我们

$$q_{n-1} = \left[\frac{a}{br^{n-1}}\right].$$

我们通过从 a 中连续地减去 br^{n-1} 直到得到一个负的结果来求得 q_{n-1}．q_{n-1} 比减法的次数小 1.

为了得到 q 的其他位上的数字，我们定义部分余数(partial remainders)序列 R_i 如下：

$$R_0 = a,$$

且对于 $i=1, 2, \cdots, n$，

$$R_i = R_{i-1} - bq_{n-i}r^{n-i}.$$

利用数学归纳法，我们来证明

$$R_i = \left(\sum_{j=0}^{n-i-1} q_j r^j\right)b + R. \tag{2.1}$$

对于 $i=0$，显然这是正确的，因为 $R_0=a=qb+R$．现在，假定

$$R_k = \left(\sum_{j=0}^{n-k-1} q_j r^j\right)b + R.$$

则

$$\begin{aligned} R_{k+1} &= R_k - bq_{n-k-1}r^{n-k-1} \\ &= \left(\sum_{j=0}^{n-k-1} q_j r^j\right)b + R - bq_{n-k-1}r^{n-k-1} \\ &= \left(\sum_{j=0}^{n-(k+1)-1} q_j r^j\right)b + R, \end{aligned}$$

这样就得到(2.1).

由(2.1)可知对于 $i=1, 2, \cdots, n$，由于 $\sum_{j=0}^{n-i-1} q_j r^j \leqslant r_{n-i}-1$，故 $0 \leqslant R_i < r^{n-i}b$．从而，由 $R_i = R_{i-1} - bq_{n-i}r^{n-i}$ 和 $0 \leqslant R_i < r^{n-1}b$ 可知数字 q_{n-i} 由 $[R_{i-1}/(br^{n-i})]$ 给出，并通过从 R_{i-1} 中连续地减去 br^{n-i} 直到得到一个负的结果而得到，q_{n-i} 比减法的次数小 1．这就说明了如何求得 q 中的数字.

例 2.9 用 $(111)_2$ 除 $(11101)_2$，设 $q=(q_2q_1q_0)_2$．我们从 $(11101)_2$ 中减去一次 $2^2(111)_2=(11100)_2$ 得到 $(1)_2$，再减一次得到一个负数，因此 $q_2=1$．现在，$R_1=(11101)_2-(11100)_2=(1)_2$．可以求得 $q_1=0$，因为 $R_1-2(111)_2$ 小于零，类似地，$q_0=0$．因此，除法得到的商为 $(100)_2$，余数为 $(1)_2$． ◀

2.2 节习题

1. 求 $(101111011)_2$ 加上 $(1100111011)_2$ 的和.

2. 求$(100010001111101)_2$加上$(11111101011111)_2$的和.
3. 求$(1111000011)_2$减去$(11010111)_2$的差.
4. 求$(1101101100)_2$减去$(101110101)_2$的差.
5. 求$(11101)_2$乘以$(110001)_2$的积.
6. 求$(1110111)_2$乘以$(10011011)_2$的积.
7. 求$(110011111)_2$除以$(1101)_2$的商和余数.
8. 求$(110100111)_2$除以$(11101)_2$的商和余数.
9. 求$(1234321)_5$加上$(2030104)_5$的和.
10. 求$(4434201)_5$减去$(434421)_5$的差.
11. 求$(1234)_5$乘以$(3002)_5$的积.
12. 求$(14321)_5$除以$(334)_5$的商和余数.
13. 求$(ABAB)_{16}$加上$(BABA)_{16}$的和.
14. 求$(FEED)_{16}$减去$(CAFE)_{16}$的差.
15. 求$(FACE)_{16}$乘以$(BAD)_{16}$的积.
16. 求$(BEADED)_{16}$除以$(ABBA)_{16}$的商和余数.
17. 解释如何在字长为 1000 的计算机上实现整数 18 235 187 和 22 135 674 的加法、减法和乘法.
18. 写出基于(-2)表示的整数的基本运算的算法(见 2.1 节的习题 8).
19. 如何从两个整数的补 1 表示得到它们的和的补 1 表示?
20. 如何从两个整数的补 1 表示得到它们的差的补 1 表示?
21. 给出康托尔展开的加法算法和减法算法(见 2.1 节习题 28 前面的导言).
22. 一打(dozen)等于 12，一罗(gross)等于 12^2. 用 12 为基或十二进制(duodecimal)算法，回答下列问题.
    a) 如果从 11 罗 3 打鸡蛋中取出 3 罗 7 打零 4 个鸡蛋，还剩下多少鸡蛋?
    b) 如果每卡车有 2 罗 3 打零 7 个鸡蛋，共往超市运送 5 卡车，那么一共运了多少鸡蛋?
    c) 如果 11 罗 10 打零 6 个鸡蛋被分成等数量的 3 堆，那么每堆有多少鸡蛋?
23. 对于十进制展开为$(a_n a_{n-1}\cdots a_1 a_0)$且末位数字 $a_0=5$ 的整数，求其平方的一个众所周知的规则是求乘积$(a_n a_{n-1}\cdots a_1)_{10}[(a_n a_{n-1}\cdots a_1)_{10}+1]$并在最后添上数字$(25)_{10}$. 例如，$(165)^2$ 的十进制展开由 $16\cdot 17=272$ 开始，所以$(165)^2=27\,225$. 证明这个规则是有效的.
24. 在这个习题中，我们推广习题 23 中给出的规则，来求 $2B$ 进制展开且末位数字为 B 的整数的平方，这里 B 为正整数. 证明整数$(a_n a_{n-1}\cdots a_1 a_0)_{2B}^2$的 $2B$ 进制展开式中前面的数字为$(a_n a_{n-1}\cdots a_1)_{2B}[(a_n a_{n-1}\cdots a_1)_{2B}+1]$，当 B 为偶数时，后面的数字为 $B/2$ 和 0；当 B 为奇数时，后面的数字为$(B-1)/2$ 和 B.

计算和研究

1. 用你自己选定的例子验证习题 23 和 24 给出的规则.

程序设计

1. 实现任意大整数的加法.
2. 实现任意大整数的减法.
3. 用传统算法计算两个任意大的整数的乘积.
4. 计算任意大的整数的除法，求商和余数.

2.3 整数运算的复杂度

一旦给定一种运算的算法，就可以考虑这个算法在计算机上实现所需的时间量. 我们以位运算(bit operations)为单位来衡量时间量. 这里位运算是指两个二进制数字的加、减、乘以及一

个二位整数除以一个一位整数(得到一个商和一个余数)，或者把一个二进制整数移位一位.(在一台计算机上进行一次位运算所需的实际时间依赖于计算机的结构和容量.)当描述实现一个算法所需的位运算的次数时，就是在描述这个算法的计算复杂度(computational complexity).

在描述实现计算所需的位运算时，我们将使用大 O 记号. 当变量很大时，大 O 记号用一个众所熟知的参考函数给出函数大小的一个上界，而这个大的参考函数的值是容易理解的.

为了引出这个记号的定义，考虑下面的情况. 假定为了实现关于整数 n 的指定运算需要至多 $n^3+8n^2\log n$ 次位运算. 由于对每个整数 n 有 $8n^2\log n<8n^3$，因此这个运算需要少于 $9n^3$ 次位运算. 由于所需的位运算的次数总是小于一个常数乘以 n^3，即 $9n^3$，因此我们称需要的位运算为$O(n^3)$.一般说来，我们有下面的定义.

定义 *S 是一个指定的实数集合，如果 f 和 g 为取正值的函数，且对所有的 $x\in S$ 有定义，则如果存在正常数 K 使得对所有充分大的 $x\in S$ 均有 $f(x)<Kg(x)$，那么 f 在 S 上是 $O(g)$的.(通常我们取 S 为正整数集合，这时便不再另提集合 S.)*

大 O 记号在数论和算法分析中被广泛使用. 保罗·巴赫曼(Paul Bachmann)在 1892 年引入大 O 记号([Ba94]). 大 O 记号有时被称为兰道符号，是根据埃德蒙·兰道(Edmund Landau)的名字命名的，他在数论的很多函数的估计中使用了这个符号. 在算法分析中大 O 记号是由著名的计算机科学家高纳德·克努特(Donald Knuth)所推广使用的.

保罗·古斯塔夫·海因里希·巴赫曼(Paul Gustav Heinrich Bachmann，1837—1920)是牧师的儿子，继承了他父亲虔诚的生活方式和对音乐的热爱. 小时候他的老师就发现了他的数学天赋，在他的结核病痊愈后，先就读于柏林大学后来转入哥廷根大学，在那里他参加了狄利克雷(Dirichlet)教授的课程. 1862 年，在数论学家库默尔(Kummer)的指导下获得了博士学位. 巴赫曼首先受聘为布雷斯劳大学(Breslau)的教授，之后转到了明斯特(Münster)大学. 退休之后，他继续从事数学研究、弹奏钢琴和发表专栏音乐评论. 他的著作包括 5 卷本的数论概要、2 卷本的初等数论、一本关于无理数的书和一本关于费马大定理的书(这个定理将在第 13 章讨论). 巴赫曼 1892 年引入了大 O 记号.

艾德蒙·兰道(Edmund Landau，1877—1938)是一个柏林妇科医生的儿子，曾在柏林就读高中. 1899 年在弗罗贝尼乌斯(Frobenius)的指导下获得博士学位. 兰道先在柏林大学教书后来转到哥廷根大学，在那里他一直担任全职教授直到纳粹强迫他离开教学岗位. 兰道对数学的主要贡献在解析数论；他给出了若干有关素数分布的重要结果. 兰道写过一部 3 卷本的数论著作和很多关于数学分析以及解析数论的书.

我们用几个例子来解释大 O 记号的概念.

例 2.10 我们可以在正整数集合上证明 n^4+2n^3+5 是 $O(n^4)$的. 为了证明这个结果，注意对所有正整数都有 $n^4+2n^3+5\leqslant n^4+2n^4+5n^4=8n^4$. (我们在定义中取 $K=8$.)读者应该也注意到 n^4 是 $O(n^4+2n^3+5)$的. ◀

例 2.11 我们可以容易地给出 $\sum_{j=1}^{n} j$ 的一个大O估计. 注意被加数均小于n,于是$\sum_{j=1}^{n} j \leqslant \sum_{j=1}^{n} n = n \cdot n = n^2$. 易从公式$\sum_{j=1}^{n} j = n(n+1)/2$ 导出这个估计 . ◀

现在我们要给出一些对函数组合运算的大 O 估计有用的结果.

定理 2.2 如果 f 是$O(g)$的，c 是正常数，则 cf 是$O(g)$的.

证明 如果 f 是$O(g)$的，则存在常数 K，使得对我们考虑的所有 x，有 $f(x)<Kg(x)$，因此 $cf(x)<(cK)g(x)$，所以 cf 是$O(g)$的. ■

定理 2.3 如果 f_1 是$O(g_1)$的，f_2 是$O(g_2)$的，则 f_1+f_2 是$O(g_1+g_2)$的，且 $f_1 f_2$ 是$O(g_1 g_2)$的.

证明 如果 f_1 是$O(g_1)$的，f_2 是$O(g_2)$的，则存在常数 K_1 和 K_2，使得对我们考虑的所有 x，有 $f_1(x)<K_1 g_1(x)$，$f_2(x)<K_2 g_2(x)$. 因此

$$\begin{aligned} f_1(x)+f_2(x) &< K_1 g_1(x)+K_2 g_2(x) \\ &\leqslant K(g_1(x)+g_2(x)), \end{aligned}$$

其中 K 是 K_1 和 K_2 的最大值，从而 f_1+f_2 是$O(g_1+g_2)$. ■

另外，

$$\begin{aligned} f_1(x)f_2(x) &< K_1 g_1(x) K_2 g_2(x) \\ &= (K_1 K_2)(g_1(x)g_2(x)), \end{aligned}$$

因此 $f_1 f_2$ 是$O(g_1 g_2)$的. ■

高纳德·克努特(Donald Knuth，1938—)在密尔沃基市(Milwaukee)长大，他的父亲经营一个小印刷工厂，同时教授记账课程. 他是个非常优秀的学生，同时也将他的聪明用在了一些异乎寻常的地方，比如从“Ziegler's Giant Bar”这些字母中组出了超过 4500 个的单词，这为他的学校赢得了一台电视机，并为班上的每位同学赢得了一根棒棒糖.

1960 年克努特毕业于凯斯理工学院(Case Institute of Technology)，因为他的杰出成绩，学校破例同时授予他学士和数学硕士学位. 在凯斯理工学院，他把自己的数学天赋用在管理篮球队上，用他改进的方程评估每个球员(这曾被 CBS 电视台和 Newsweek 报纸报道过). 克努特于 1963 年在加州理工学院(California Institute of Technology)获得博士学位.

克努特先后在加州理工学院和斯坦福大学执教，为了集中精力写书，他于 1992 年退休. 他特别喜欢更新续写他的著名系列《计算机程序设计的艺术》(the Art of Computer Programming). 这一系列著作对计算机科学产生了深远的影响. 克努特是研究计算复杂度的奠基人，他对程序编译也有奠基性的贡献. 克努特发明了用于数学(和普通)排版的 TeX 和 Metafont 系统. TeX 在 HTML 和浏览器的发展过程中扮演了重要的角色. 他在有关算法分析的作品中普及了大 O 记号.

克努特在许多专业的计算机和数学杂志上发表过文章. 但他的处女作却是 1957 年大一时发表在《疯狂》杂志上的一篇《普茨比度量衡体系》(The Potrzebie System of Weights and Measures)，这是一篇关于度量体系的打油诗.

推论 2.3.1 如果 f_1 和 f_2 是$O(g)$的，则 f_1+f_2 是$O(g)$的.

证明 定理 2.3 告诉我们 f_1+f_2 是$O(2g)$的. 但是如果 $f_1+f_2<K(2g)$，则 $f_1+f_2<$

$(2K)g$，因此 f_1+f_2 是 $O(g)$的. ■

使用大 O 估计的目的是使用最简单的参考函数来得到最好的大 O 估计. 在大 O 估计中常用的参考函数包括 1，$\log n$，n，$n\log n$，$n\log n \cdot \log\log n$，n^2 和 2^n，以及其他一些重要函数. 可以通过计算说明在这列函数中每个函数都比下一个函数小，因为随着 n 无限增大，相邻两个函数的比趋于 0. 注意在大 O 估计中会出现更复杂的函数，这将在后面的章节中涉及.

我们用下面的例子解释如何利用前面的定理进行大 O 估计.

例 2.12 为了给出$(n+8\log n)(10n\log n+17n^2)$的大 O 估计，首先注意到根据定理 2.2、2.3 和推论 2.3.1，$n+8\log n$ 是 $O(n)$ 的. 且 $10n\log n+17n^2$ 是 $O(n^2)$的(因为 $\log n$ 是 $O(n)$的，而 $n\log n$ 是 $O(n^2)$的). 再由定理 2.3 可知$(n+8\log n)(10n\log n+17n^2)$是 $O(n^3)$的. ◀

使用大 O 记号，我们可以看到加或减两个 n 位整数都需要$O(n)$次位运算，然而用通常的方法来将两个 n 位整数相乘则需要$O(n^2)$次位运算(参见本节末的习题 12 和 13). 令人吃惊的是存在计算大整数乘法的快速算法. 为了介绍这一算法，我们首先考虑两个 $2n$ 位整数的乘法，比如 $a=(a_{2n-1}a_{2n-2}\cdots a_1a_0)_2$ 和 $b=(b_{2n-1}b_{2n-2}\cdots b_1b_0)_2$. 我们将其写为

$$a = 2^nA_1 + A_0 \qquad b = 2^nB_1 + B_0,$$

其中

$$A_1 = (a_{2n-1}a_{2n-2}\cdots a_{n+1}a_n)_2 \qquad A_0 = (a_{n-1}a_{n-2}\cdots a_1a_0)_2$$
$$B_1 = (b_{2n-1}b_{2n-2}\cdots b_{n+1}b_n)_2 \qquad B_0 = (b_{n-1}b_{n-2}\cdots b_1b_0)_2.$$

我们将要使用恒等式

$$ab = (2^{2n}+2^n)A_1B_1 + 2^n(A_1-A_0)(B_0-B_1) + (2^n+1)A_0B_0. \tag{2.2}$$

为了应用(2.2)求 a 和 b 的乘积，需要进行三个 n 位整数的乘法(分别为 A_1B_1，$(A_1-A_0)(B_0-B_1)$和 A_0B_0)以及一些加法和移位. 这可用下面的例子说明.

例 2.13 可以使用(2.2)来计算$(1101)_2$ 和$(1011)_2$ 的积. 我们有$(1101)_2=2^2(11)_2+(01)_2$ 和$(1011)_2=2^2(10)_2+(11)_2$. 应用(2.2)，可得

$$\begin{aligned}(1101)_2(1011)_2 &= (2^4+2^2)(11)_2(10)_2 + 2^2((11)_2-(01)_2)\cdot\\ &\quad ((11)_2-(10)_2) + (2^2+1)(01)_2(11)_2\\ &= (2^4+2^2)(110)_2 + 2^2(10)_2(01)_2 + (2^2+1)(11)_2\\ &= (1100000)_2 + (11000)_2 + (1000)_2 + (1100)_2 + (11)_2\\ &= (10001111)_2.\end{aligned}$$

◀

我们现在来估计反复使用(2.2)将两个 n 位整数相乘所需的位运算的次数. 如果令 $M(n)$表示两个 n 位整数相乘所需的位运算的次数，从(2.2)中可得

$$M(2n) \leqslant 3M(n) + Cn, \tag{2.3}$$

这里 C 为一个常数，因为三个 n 位整数乘法中的每一个都需要 $M(n)$次位运算，而用(2.2)计算 ab 所需的加法和移位的次数不依赖于 n，这些操作中的每一个都需要 $O(n)$次的位运算.

从(2.3)中，利用数学归纳法，可以证明

$$M(2^k) \leqslant c(3^k - 2^k), \tag{2.4}$$

其中 c 是 $M(2)$和 C((2.3)中的常数)中的最大值. 为了进行归纳，我们首先注意到当 $k=1$ 时，由于 c 是 $M(2)$和 C 的最大值，故 $M(2)\leqslant c(3^1-2^1)=c$.

作为归纳假设，我们假定

$$M(2^k) \leqslant c(3^k - 2^k).$$

所以，应用(2.3)有

$$\begin{aligned} M(2^{k+1}) &\leqslant 3M(2^k) + C2^k \\ &\leqslant 3c(3^k - 2^k) + C2^k \\ &\leqslant c3^{k+1} - c \cdot 3 \cdot 2^k + c2^k \\ &\leqslant c(3^{k+1} - 2^{k+1}). \end{aligned}$$

这就说明对所有正整数 k，(2.4)是正确的.

应用不等式(2.4)可以证明下面的定理.

定理 2.4 两个 n 位整数的乘法可以用 $O(n^{\log_2 3})$ 次位运算实现. (注意：$\log_2 3$ 近似为 1.585，小于在传统乘法算法所需的位运算次数的估计中的次数 2.)

证明 从(2.4)中，我们有

$$\begin{aligned} M(n) = M(2^{\log_2 n}) &\leqslant M(2^{\lceil \log_2 n \rceil + 1}) \\ &\leqslant c(3^{\lceil \log_2 n \rceil + 1} - 2^{\lceil \log_2 n \rceil + 1}) \\ &\leqslant 3c \cdot 3^{\lceil \log_2 n \rceil} \leqslant 3c \cdot 3^{\log_2 n} = 3cn^{\log_2 3} (\text{因为 } 3^{\log_2 n} = n^{\log_2 3}). \end{aligned}$$

因此，$M(n)$是 $O(n^{\log_2 3})$的. ■

我们现在不加证明地陈述两个相关的定理. 证明可以在[Kn97]和[Kr79]中找到.

定理 2.5 给定一个正数 $\varepsilon > 0$，存在计算两个 n 位整数的乘积的算法，只需要 $O(n^{1+\varepsilon})$ 次位运算.

注意定理 2.4 是定理 2.5 在 $\varepsilon = \log_2 3 - 1$ 时的特殊情况，此时 ε 近似等于 0.585.

定理 2.6 存在计算两个 n 位整数乘积的算法，该算法只使用

$$O(n \log_2 n \log_2 \log_2 n)$$

次位运算.

对于大数 n，由于 $\log_2 n$ 和 $\log_2 \log_2 n$ 比 n^{ε} 小得多，因此定理 2.6 是定理 2.5 的改进. 尽管我们知道 $M(n)$是 $O(n\log_2 n \log_2 \log_2 n)$ 的，但为了简单起见，我们将要在下面的讨论中使用一个显然的事实：$M(n)$是 $O(n^2)$的.

在 2.2 节中给出的传统算法用 $O(n^2)$次位运算实现了一个 $2n$ 位整数被一个 n 位整数除的算法. 然而，整数除法所需的位运算的次数与整数乘法所需的位运算的次数相关. 我们基于[Kn97]中讨论的算法给出下面的定理.

定理 2.7 当 $2n$ 位整数 a 被整数 b(不超过 n 位)除时，有使用 $O(M(n))$次位运算求商 $q = \lfloor a/b \rfloor$的算法，其中 $M(n)$是求两个 n 位整数乘积所需的位运算次数.

2.3 节习题

1. 在正整数集合上确定下列函数是否是 $O(n)$的.

   a) $2n+7$　　b) $n^2/3$　　c) 10　　d) $\log(n^2+1)$　　e) $\sqrt{n^2+1}$　　f) $(n^2+1)/(n+1)$

2. 在正整数集合上证明 $2n^4 + 3n^3 + 17$ 是 $O(n^4)$的.
3. 证明 $(n^3 + 4n^2 \log n + 101n^2)(14n \log n + 8n)$是 $O(n^4 \log n)$的.
4. 在正整数集合上证明 $n!$ 是 $O(n^n)$的.
5. 证明 $(n! + 1)(n + \log n) + (n^3 + n^n)((\log n)^3 + n + 7)$是 $O(n^{n+1})$的.
6. 若 m 是正实数，证明 $\sum_{j=1}^{n} j^m$ 是 $O(n^{m+1})$的.

* 7. 在正整数集合上证明 $n\log n$ 是 $O(\log n!)$的.

8. 证明：如果 f_1 和 f_2 分别是 $O(g_1)$和 $O(g_2)$的，且 c_1 和 c_2 为常数，则 $c_1 f_1+c_2 f_2$ 是 $O(g_1+g_2)$的.

9. 证明：如果 f 是$O(g)$的，则对所有正整数 k，f^k 是$O(g^k)$的.

10. 设 r 是大于 1 的正实数，证明函数 f 是 $O(\log_2 n)$的当且仅当 f 是 $O(\log_r n)$的.（提示：回顾 $\log_a n/\log_b n=\log_a b$.）

11. 证明正整数 n 的 b 进制展开有$[\log_b n]+1$ 位.

12. 分析传统的加法和减法算法，证明 n 位整数的这些运算需要 $O(n)$次位运算.

13. 证明用传统方法求一个 n 位整数和一个 m 位整数乘积需要 $O(nm)$次位运算.

14. 估计计算 $1+2+\cdots+n$ 所需的位运算的次数.

a) 通过逐项相加；

b) 通过使用恒等式 $1+2+\cdots+n=n(n+1)/2$ 和乘法以及移位.

15. 给出计算下面式子所需的位运算次数的估计.

a) $n!$　　b) $\binom{n}{k}$

16. 给出把一个整数从十进制转为二进制所需的位运算次数的估计.

17. 用 $n=2$ 的恒等式(2.2)来计算$(1001)_2$ 和$(1011)_2$ 的乘积.

18. 先利用 $n=4$、再利用 $n=2$ 的恒等式(2.2)计算$(10010011)_2$ 和$(11001001)_2$ 的乘积.

19. a) 证明对于十进制展开存在一个类似于(2.2)的恒等式.

b)应用(a)的结果，只用三个一位整数乘法以及移位和加法来计算 73 和 87 的乘积.

c)应用(a)的结果，把 4216 和 2733 的乘法简化到三个二位整数乘法以及一些移位和加法；然后再次应用(a) 部分，把每个二位数乘法简化到三个一位数乘法和一些移位与加法，最终只使用九个一位整数乘法和一些移位、加法完成这个乘法运算.

20. 如果 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 是 $n\times n$ 矩阵，其元素分别为 a_{ij} 和 b_{ij}，$1\leqslant i\leqslant n$，$1\leqslant j\leqslant n$，则 $\boldsymbol{AB}$ 是 $n\times n$ 矩阵，其元素为 $c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}$. 证明直接根据定义计算 $\boldsymbol{AB}$ 需要用到 n^3 个整数乘法.

21. 证明通过下面的等式，只用七个整数乘法实现两个 2×2 矩阵的乘法是可能的.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}=\begin{bmatrix} a_{11}b_{11}+a_{12}b_{21} & x+(a_{21}+a_{22})(b_{12}-b_{11}) \\ & +(a_{11}+a_{12}-a_{21}-a_{22})b_{22} \\ x+(a_{11}-a_{21})(b_{22}-b_{12}) & x+(a_{11}-a_{21})(b_{22}-b_{12}) \\ -a_{22}(b_{11}-b_{21}-b_{12}+b_{22}) & +(a_{21}+a_{22})(b_{12}-b_{11}) \end{bmatrix}$$

其中 $x=a_{11}b_{11}-(a_{11}-a_{21}-a_{22})(b_{11}-b_{12}+b_{22})$.

* 22. 使用归纳的方法，把$(2n)\times(2n)$矩阵分成四个 $n\times n$ 矩阵，应用习题 21 证明只使用 7^k 个乘法和少于 7^{k+1}个加法实现两个 $2^k\times 2^k$ 矩阵的乘法是可能的.

23. 从习题 22 中推出如果两个 $n\times n$ 矩阵中的所有元素都是少于 c 位的数，则只需 $O(n^{\log_2 7})$次位运算即可实现它们的乘法，其中 c 是一个常数.

计算和研究

1. 计算 81 873 569 和 41 458 892 的乘积，对这两个八位整数使用等式(2.2)，归结为四位整数相乘，再次应用(2.2)，再归结为二位整数相乘.

2. 用习题 21 中矩阵的恒等式计算你自己选择的两个 8×8 矩阵的乘法，然后对得到的 4×4 矩阵再次应用习题 21.

程序设计

* 1. 用恒等式(2.2)乘两个任意大的整数.

** 2. 用习题 21～23 中讨论的算法计算两个 $n\times n$ 矩阵的乘积.