关于整数乘法算法的报告

数学科学学院　胡苏麟

【摘要】 整数乘法能算多快？这是一个历史悠久的问题。2019 年，David Harveys 和 Joris van der Hoeven 宣布，他们发现了一个时间复杂度为 $n\log n$ 的整数乘法算法，创造了历史。本文回顾了整数乘法算法的研究历史，介绍了其中容易理解的一些方法，讨论了该算法的理论和实际价值，并且发表了笔者对该问题研究历程的感想。

【关键词】 整数乘法　算法

一、整数乘法算法的研究历史

2019 年，David Harveys 和 Joris van der Hoeven 宣布，他们发现了一个复杂度为 $n\log n$ 的整数乘法算法。人们已经在整数乘法上探索了上千年。在这一问题上的研究历史是令人心潮澎湃的。

数千年前，人们已经掌握了乘法，并初步产生了进行乘法运算的算法，尽管当时还没有所谓“算法”的概念。其中古埃及人计算乘法的方式是将其中一个乘数展开为二的次幂的和，然后计算另一个数不断加倍所得的结果，最后将所需的结果求和。而古巴比伦人发明了六十进制。在进制的意义下，两个 n 位的整数相乘，可以转化为 n^2 个个位数乘法，以及最后的求和。这样的算法，时间复杂度在 $O(n^2)$ 的级别 [2]。

然而，一直到 1960 年之前，这种传统的竖式计算算法仍然在量级上是最高效的乘法算法之一。1960 年，一名叫 Anatoly Karatsuba 的 23 岁俄罗斯数学家提出了一种新的算法 [6]。直观而言，对两个 n 位的整数，可以写成 $a\times 10^{n/2}+b$ 和 $c\times 10^{n/2}+d$ 的形式，其中 a,b,c,d 都至多是 $n/2$ 位的整数。分别计算 $ab, cd, (a+c)(b+d)$ 的值，注意到所求乘积就是

$$((a+c)(b+d)-ab-cd)\times 10^{n/2}+ab\times 10^{n}+cd.$$

这样的方法只需要求三次 $n/2$ 位的乘法，因此复杂度达到 $O(n^{\log 3/\log 2})$, 约为 $O(n^{1.58})$。

Karatsuba 实质上使用了如下的方法 [2]：将输入的两个整数分拆成更小的部分，并将它们作为两个整多项式的系数。通过求出多项式函数在某些点处的取值（比如上述方法为 0，1 和 ∞ 这三个点），可以递归地算出多项式乘积在这些点的值。最后，使用简单的差值和拼接就可以从这些点的值还原乘积多项式的系数，进而得到所求乘积的结果。

不久以后,Toom 等人在 1963 年将 Karatsuba 的方法推广到一般的 r 个点的赋值，让 r 取到最佳的情况（与 n 相关，假设事先能知道输入的位数）。这样，他们将 Karatsuba 的结果加强到了 $O(n2^{5\sqrt{\log n/\log 2}})$. 在 1969 年，Kruth 用类似的方法进一步将复杂度上界继续推进到 $O(n2^{2\sqrt{\log n/\log 2}}\log n)$[2].

Toom 的方法是如何推广的？在文 [7] 中对此有简要的介绍：

以 $r = 3$ 为例，Karatsuba 将每个 n 位数划分为 2 个部分，但我们可以考虑划分为 3 部分，例如

$$A = A_2 2^{2k} + A_1 2^k + A_0$$
$$B = B_2 2^{2k} + B_1 2^k + B_0$$

其中 $A_i, B_i (i = 0, 1, 2)$ 都是 k 为位的二进制数。而乘积

$$\begin{aligned} C &= AB \\ &= C_4 2^{4k} + C_3 2^{3k} + C_2 2^k + C_1 2^k + C_0 \end{aligned}$$

其中

$$\begin{aligned} C_4 &= A_2 B_2 \\ C_3 &= A_1 B_2 + A_2 B_1 \\ C_2 &= A_0 B_2 + A_1 B_1 + A_2 B_0 \\ C_1 &= A_0 B_1 + A_1 B_0 \\ C_0 &= A_0 B_0 \end{aligned}$$

为了确定 $C_4 \sim C_0$ 的值，采用传统的算法需要 9 次乘法。但是仔细观察，会发现我们只需要计算 $A_0 B_0, (A_2 \pm A_1 + A_0)(B_2 \pm B_1 + B_0), (4A_2 \pm 2A_1 + A_0)(4B_2 \pm 2B_1 + B_0)$ 这 5 次乘法就可以了：

定义 $P_A(x) = A_2 x^2 + A_1 x + A_0$, 同理 $P_B(x)$, 以及

$$P_C(x) = C_4 x^4 + C_3 x^3 + C_2 x^2 + C_1 x + C_0 = P_A(x) P_B(x)$$

上述 5 次乘法其实是算出了 $P_C(x)$ 在 $-2,-1,0,1,2$ 这 5 个点处的取值。通过拉格朗日差值即可还原 $P_C(x)$ 的系数。这样的方法递归至 5 次 $n/3$ 位的乘法，复杂度是 $O(n^{\log 5/\log 3}) \approx O(n^{1.465})$ 明显优于 Karatsuba 的二分法。

下一个重大进展由 Schönhage 和 Strassen 取得。1965 年，快速傅里叶变换问世，引起了轰然大波。作为算法界中历史悠久的问题，整数乘法的计算也受到了这种算法的“增幅”。基于快速傅里叶变换，上文中利用多项式的方法可以在复数域 $\mathbb{C}$ 中取值。Schönhage 和 Strassen 提出了“Schönhage-Strassen 第一算法”，将 n 位的乘法递归到 $\log n$ 级别的乘法，并将算法的复杂度降低到了由如下递推式确定的数量级 [1]：

$$\mathrm{M}(n) = O(n\mathrm{M}(n')) + O(n\log n), \quad n' = O(\log n)$$

其中，我们假设 $\mathrm{M}(n)$ 表示计算 2 个 n 位整数乘积的时间复杂度。

很快，在 1971 年，Schönhage 和 Strassen 又发现了“Schönhage-Strassen 第二算法”。这次，他们将复数域 $\mathbb{C}$(可以看作 $\mathbb{Z}[x]/(x^2+1)$) 进行了改进，虽然只将 n 位递归到 $O(\sqrt{n})$ 位，但成功将算法降低到了 $O(n\log n\log\log n)$ 级别 [3]。

与此同时，他们在发表的文章中，他们猜想整数乘法最优解的复杂度达到 $O(n\log n)$，开启了整数乘法优化新的征程。

Fürer 在 2007 年，再次对已经领跑 36 年的算法进行了改进。简而言之，他综合了 Schönhage 和 Strassen 两种算法中的优点，将算法的复杂度进一步降低为 [2]

$$\mathrm{M}(n) = O(n\log nK^{\log^* n}),$$

其中 $\log^* n$ 定义如下：

$$\log^* x := min\{k \in \mathbb{N} : \log^{\circ k} x \le 1\},$$
$$\log^{\circ k} := \log\underbrace{\circ\cdots\circ}_{k\times}\log.$$

而 K 是一个常数，并且在 2014 年文 [2] 中被减少至 8。至此，距离 Schönhage 和 Strassen 猜想的数量级只有一步之遥了。

在 2019 年，David Harveys 和 Joris van der Hoeven 在文 [1] 中得到了 $O(n\log n)$ 的算法。根据 [4] 的解释，他们将前人讨论的一元多项式环拓展到了多元多项式环，最终解决了 Schönhage-Strassen 猜想的其中一半。

至于整数乘法的下界，至今仍然是一个困难的问题，尚未得到任何超线性的下界。

二、对整数乘法算法的认识

乘法在我们一般的编程中是相当基础的一次运算，甚至在大部分的语言中都只用一个小小的“$*$”号就可以完成，不需要考虑计算机是如何计算乘法的，更不需要考虑乘法的算法。但事实上，就像我们笔算加法会十分轻松，笔算乘法却要费一番功夫一样，即使是功能强大的计算器，在计算超大位数的乘法时，也要花费很长的时间。计算机的乘法不是能立即得到答案，而是要使用递归等方式，将大位数转化为更小的数。整数乘法的算法是十分基础的一个问题，它的研究对其他更加复杂的算法都会有一定价值。从古时代的 $O(n^2)$ 级别，到现在的 $O(n\log n)$ 级别，在大位数的乘法上能节省计算机一大笔时间。

从算法的研究过程中可以看出，虽然整数乘法本身是一个相当简单的概念，但是探究计算复杂度的道路却是十分艰辛的。从 1960 年的第一次突破开始，每次优化似乎都会运用许多数论，群论等方面的数学知识作为基础，让算法理解更加困难。比如，Karatsuba 的方法相当容易理解，甚至在介绍竖式计算的时候一并提及，也能够被听众接受。但是，这种方法的背景却涉及到多项式的插值理论，因此在推广的时候解释如 $r = 3$ 时用 5 次乘法就足以完成原先 9 次乘法的原理，就没有那么容易了。而引进快速傅里叶变换这一大“利器”之后，算法的优化被扯进了更高深的数学领域，笔者也没有看懂其原理。好在每次突破的主要成果都可以用比较简单的式子表达出来，让人能够直观地感受到算法一步步的优化。看似简单的一个相乘，却能牵涉到多元多项式环等理论，这无不令人感叹算法设计之精妙。

从理论的角度上看，乘法运算的速度最终达到 $O(n\log n)$，是一次不小的进步。这次优化摆脱了复杂度中比较奇怪的添项，得出了一个相当的简单的结论。同时，乘法的优化会一并带动许多更复杂方面的最优算法 [1]。例如，实数的除法和求 k 次根能够在 $O(n\log n)$ 的时间内达到 n 位的精度；计算机能够在 $O(n\log^2 n)$ 的时间内计算诸如 e^x,π 的超越函数或常数至 n 位的精度。此外，这一成果在计算离散傅里叶变换的领域中也可以得到运用。同时，这样的结论实现了约半个世纪前的前人提出的一个设想，达到的新的高度。

在实际的运算中，之前的结论和最新的算法也不会差距很多。文 [5] 中提到，1971 年的 $O(n\log n\log\log n)$ 时间算法已经相当有效。之后的改进只

是在理论上更加优化，只有在计算不实际的天文数字时，才可能超越原有的算法。现在的新算法相比于文 [5] 时期，优化的部分在实际运算中，几乎可以被当作一个常数。所以新的算法在实际用途中，比如工业界，影响不一定会很大 [8]。随着科技的发展，人们或许能够建造出更强大的计算机，计算更庞大的数字。这个时候，新算法的优势就会逐渐体现出来。

正如上文所述，对这一问题的探究还没有抵达终点。究竟 $O(n \log n)$ 的复杂度是最优的可能，还是存在更加快速的计算方法，现在还不为人所知。在信息时代，算法的重要性日益增加。虽然大部分人都不能比肩算法研究领域中的领跑者，但是了解算法，运用算法，却能够为我们更加适应时代的发展产生帮助。

参考文献

[1] David Harveys, Joris van der Hoeven. *Integer multiplication in time O(n logn).*[J] Archive Ouverte HAL, hal-02070778,2019.

[2] David Harveys, Joris van der Hoeven, Grégoire Lecerf. *Even faster integer multiplication.*[J] J.Complexity36(2016),1-30. MR3530637

[3] M.Fürer. *Faster integer multiplication*[C] In Proceedings of the Thirty-Ninth ACM Symposium on Theory of Computing, STOC 2007, pages 57-66, New York, NY, USA, 2007 ACM Press

[4] KWRegan. *Integer Multiplication in NlogN Time*[N] Godel's Lost Letter and P=NP https://rjlipton.wordpress.com/2019/03/29/integer-multiplication-in-nlogn-time/

[5] M.Fürer *How Fast Can We Multiply Large Integers on an Actual Computer?*[R], http://arxiv.org/abs/1402.1811, 2014

[6] DeepTech 深科技，2 名数学家或发现史上最快超大乘法运算法，欲破解困扰人类近半个世纪的问题。[R], http://zhuanlan.zhihu.com/p/63291183

[7] ————,comp9101 lecture 3 快速大数乘法 [R], http://zhuanlan.zhihu.com/p/58355718

[8] ————，如何看待 $O(n \log n)$ 时间的整数乘法算法, https://www.zhihu.com/question/317392704