Create account Log in

Article Talk Read Edit View history Search

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Español
Esperanto
Français
한국어
Italiano
ไทย
Edit links

Toom–Cook multiplication

From Wikipedia, the free encyclopedia

Toom–Cook, sometimes known as **Toom-3**, named after Andrei Toom, who introduced the new algorithm with its low complexity, and Stephen Cook, who cleaned the description of it, is a multiplication algorithm, a method of multiplying two large integers.

Given two large integers, *a* and *b*, Toom–Cook splits up *a* and *b* into *k* smaller parts each of length *l*, and performs operations on the parts. As *k* grows, one may combine many of the multiplication sub-operations, thus reducing the overall complexity of the algorithm. The multiplication sub-operations can then be computed recursively using Toom–Cook multiplication again, and so on. Although the terms "Toom-3" and "Toom–Cook" are sometimes incorrectly used interchangeably, Toom-3 is only a single instance of the Toom–Cook algorithm, where $k = 3$.

Toom-3 reduces 9 multiplications to 5, and runs in $\Theta(n^{\log(5)/\log(3)})$, about $\Theta(n^{1.465})$. In general, Toom-*k* runs in $\Theta(c(k)\ n^e)$, where $e = \log(2k - 1) / \log(k)$, n^e is the time spent on sub-multiplications, and *c* is the time spent on additions and multiplication by small constants.[1] The Karatsuba algorithm is a special case of Toom–Cook, where the number is split into two smaller ones. It reduces 4 multiplications to 3 and so operates at $\Theta(n^{\log(3)/\log(2)})$, which is about $\Theta(n^{1.585})$. Ordinary long multiplication is equivalent to Toom-1, with complexity $\Theta(n^2)$.

Although the exponent *e* can be set arbitrarily close to 1 by increasing *k*, the function *c* unfortunately grows very rapidly.[1][2] The growth rate for mixed-level Toom-Cook schemes was still an open research problem in 2005.[3] An implementation described by Donald Knuth achieves the time complexity $\Theta(n\ 2^{\sqrt{(2 \log n)}} \log n)$.[4]

Due to its overhead, Toom–Cook is slower than long multiplication with small numbers, and it is therefore typically used for intermediate-size multiplications, before the asymptotically faster Schönhage–Strassen algorithm (with complexity $\Theta(n \log n \log \log n)$) becomes practical.

Toom first described this algorithm in 1963, and Cook published an improved (asymptotically equivalent) algorithm in his PhD thesis in 1966.[5]

Contents [hide]

1 Details
1.1 Splitting
1.2 Evaluation
1.2.1 Faster evaluation
1.3 Pointwise multiplication
1.4 Interpolation
1.5 Recomposition
2 Interpolation matrices for various *k*
2.1 Toom-1
2.2 Toom-1.5
2.3 Toom-2
2.4 Toom-2.5
3 Notes
4 References
5 External links

Details [edit]

This section discusses exactly how to perform Toom-*k* for any given value of *k*, and is a simplification of a description of Toom–Cook polynomial multiplication described by Marco Bodrato.[6] The algorithm has five main steps:

1. Splitting
2. Evaluation
3. Pointwise multiplication
4. Interpolation

5. Recomposition

In a typical large integer implementation, each integer is represented as a sequence of digits in positional notation, with the base or radix set to some (typically large) value *b*; for this example we use *b* = 10000, so that each digit corresponds to a group of four decimal digits (in a computer implementation, *b* would typically be a power of 2 instead). Say the two integers being multiplied are:

m = 12 3456 7890 1234 5678 9012
n =   9 8765 4321 9876 5432 1098.

These are much smaller than would normally be processed with Toom–Cook (grade-school multiplication would be faster) but they will serve to illustrate the algorithm.

Splitting [edit]

The first step is to select the base $B = b^i$, such that the number of digits of both *m* and *n* in base *B* is at most *k* (e.g., 3 in Toom-3). A typical choice for *i* is given by:

$$i = \max\{\lfloor\lfloor \log_b m\rfloor/k\rfloor, \lfloor\lfloor \log_b n\rfloor/k\rfloor\} + 1.$$

In our example we'll be doing Toom-3, so we choose $B = b^2 = 10^8$. We then separate *m* and *n* into their base *B* digits m_i, n_i:

$$\begin{aligned} m_2 &= 123456 \\ m_1 &= 78901234 \\ m_0 &= 56789012 \\ n_2 &= 98765 \\ n_1 &= 43219876 \\ n_0 &= 54321098 \end{aligned}$$

We then use these digits as coefficients in degree *k*−1 polynomials *p* and *q*, with the property that *p*(*B*) = *m* and *q*(*B*) = *n*:

$$\begin{aligned} p(x) &= m_2x^2 + m_1x + m_0 = 123456x^2 + 78901234x + 56789012 \\ q(x) &= n_2x^2 + n_1x + n_0 = 98765x^2 + 43219876x + 54321098 \end{aligned}$$

The purpose of defining these polynomials is that if we can compute their product *r*(*x*) = *p*(*x*)*q*(*x*), our answer will be *r*(*B*) = *m*×*n*.

In the case where the numbers being multiplied are of different sizes, it's useful to use different values of *k* for *m* and *n*, which we'll call k_m and k_n. For example, the algorithm "Toom-2.5" refers to Toom–Cook with $k_m = 3$ and $k_n = 2$. In this case the *i* in $B = b^i$ is typically chosen by:

$$i = \max\{\lfloor\lceil \log_b m\rceil/k_m\rfloor, \lfloor\lceil \log_b n\rceil/k_n\rfloor\}.$$

Evaluation [edit]

The Toom–Cook approach to computing the polynomial product *p*(*x*)*q*(*x*) is a commonly used one. Note that a polynomial of degree *d* is uniquely determined by *d* + 1 points (for example, a line - polynomial of degree one is specified by two points). The idea is to evaluate *p(·)* and *q(·)* at various points. Then multiply their values at these points to get points on the product polynomial. Finally interpolate to find its coefficients.

Since deg(*pq*) = deg(*p*) + deg(*q*), we will need deg(*p*) + deg(*q*) + 1 = $k_m + k_n - 1$ points to determine the final result. Call this *d*. In the case of Toom-3, *d* = 5. The algorithm will work no matter what points are chosen (with a few small exceptions, see matrix invertibility requirement in Interpolation), but in the interest of simplifying the algorithm it's better to choose small integer values like 0, 1, −1, and −2.

One unusual point value that is frequently used is infinity, written ∞ or 1/0. To "evaluate" a polynomial *p* at infinity actually means to take the limit of $p(x)/x^{\deg p}$ as *x* goes to infinity. Consequently, *p*(∞) is always the value of its highest-degree coefficient (in the example above coefficient m_2).

In our Toom-3 example, we will use the points 0, 1, −1, −2, and ∞. These choices simplify evaluation, producing the formulas:

$$\begin{aligned} p(0) &= m_0 + m_1(0) + m_2(0)^2 = m_0 \\ p(1) &= m_0 + m_1(1) + m_2(1)^2 = m_0 + m_1 + m_2 \\ p(-1) &= m_0 + m_1(-1) + m_2(-1)^2 = m_0 - m_1 + m_2 \\ p(-2) &= m_0 + m_1(-2) + m_2(-2)^2 = m_0 - 2m_1 + 4m_2 \\ p(\infty) &= m_2 \end{aligned}$$

and analogously for *q*. In our example, the values we get are:

$p(0) = m_0$	= 56789012	= 56789012
$p(1) = m_0 + m_1 + m_2$	= 56789012 + 78901234 + 123456	= 135813702
$p(-1) = m_0 - m_1 + m_2$	= 56789012 − 78901234 + 123456	= −21988766
$p(-2) = m_0 - 2m_1 + 4m_2$	= 56789012 − 2×78901234 + 4×123456	= −100519632
$p(\infty) = m_2$	= 123456	= 123456
$q(0) = n_0$	= 54321098	= 54321098
$q(1) = n_0 + n_1 + n_2$	= 54321098 + 43219876 + 98765	= 97639739
$q(-1) = n_0 - n_1 + n_2$	= 54321098 − 43219876 + 98765	= 11199987
$q(-2) = n_0 - 2n_1 + 4n_2$	= 54321098 − 2×43219876 + 4×98765	= −31723594
$q(\infty) = n_2$	= 98765	= 98765.

As shown, these values may be negative.

For the purpose of later explanation, it will be useful to view this evaluation process as a matrix-vector multiplication, where each row of the matrix contains powers of one of the evaluation points, and the vector contains the coefficients of the polynomial:

$$\begin{pmatrix} p(0) \\ p(1) \\ p(-1) \\ p(-2) \\ p(\infty) \end{pmatrix} = \begin{pmatrix} 0^0 & 0^1 & 0^2 \\ 1^0 & 1^1 & 1^2 \\ (-1)^0 & (-1)^1 & (-1)^2 \\ (-2)^0 & (-2)^1 & (-2)^2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \end{pmatrix}.$$

The dimensions of the matrix are *d* by k_m for *p* and *d* by k_n for *q*. The row for infinity is always all zero except for a 1 in the last column.

Faster evaluation [edit]

Multipoint evaluation can be obtained faster than with the above formulas. The number of elementary operations (addition/subtraction) can be reduced. The sequence given by Bodrato[6] for Toom-3, executed here over the first operand (polynomial *p*) of the running example is the following:

$p_0 \leftarrow m_0 + m_2$	= 56789012 + 123456	= 56912468
$p(0) = m_0$	= 56789012	= 56789012
$p(1) = p_0 + m_1$	= 56912468 + 78901234	= 135813702
$p(-1) = p_0 - m_1$	= 56912468 − 78901234	= −21988766
$p(-2) = (p(-1) + m_2)\times 2 - m_0$	= (− 21988766 + 123456)×2 − 56789012	= − 100519632
$p(\infty) = m_2$	= 123456	= 123456.

This sequence requires five addition/subtraction operations, one less than the straightforward evaluation. Moreover the multiplication by 4 in the calculation of *p*(−2) was saved.

Pointwise multiplication [edit]

Unlike multiplying the polynomials *p*(·) and *q*(·), multiplying the evaluated values *p*(*a*) and *q*(*a*) just involves multiplying integers — a smaller instance of the original problem. We recursively invoke our multiplication procedure to multiply each pair of evaluated points. In practical implementations, as the operands become smaller, the algorithm will switch to the Schoolbook long multiplication. Letting *r* be the product polynomial, in our example we have:

$r(0) = p(0)q(0)$	= 56789012 × 54321098	= 3084841486175176
$r(1) = p(1)q(1)$	= 135813702 × 97639739	= 13260814415903778
$r(-1) = p(-1)q(-1)$	= −21988766 × 11199987	= −246273893346042
$r(-2) = p(-2)q(-2)$	= −100519632 × −31723594	= 3188843994597408

$r(\infty) = p(\infty)q(\infty) = 123456 \times 98765 = 12193131840.$

As shown, these can also be negative. For large enough numbers, this is the most expensive step, the only step that is not linear in the sizes of *m* and *n*.

Interpolation [edit]

This is the most complex step, the reverse of the evaluation step: given our *d* points on the product polynomial *r*(·), we need to determine its coefficients. In other words, we want to solve this matrix equation for the vector on the right-hand side:

$$\begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix} = \begin{pmatrix} 1^0 & 0^1 & 0^2 & 0^3 & 0^4 \\ 1^0 & 1^1 & 1^2 & 1^3 & 1^4 \\ (-1)^0 & (-1)^1 & (-1)^2 & (-1)^3 & (-1)^4 \\ (-2)^0 & (-2)^1 & (-2)^2 & (-2)^3 & (-2)^4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix}.$$

This matrix is constructed the same way as the one in the evaluation step, except that it's *d*×*d*. We could solve this equation with a technique like Gaussian elimination, but this is too expensive. Instead, we use the fact that, provided the evaluation points were chosen suitably, this matrix is invertible, and so:

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1/2 & 1/3 & -1 & 1/6 & -2 \\ -1 & 1/2 & 1/2 & 0 & -1 \\ -1/2 & 1/6 & 1/2 & -1/6 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix}.$$

All that remains is to compute this matrix-vector product. Although the matrix contains fractions, the resulting coefficients will be integers — so this can all be done with integer arithmetic, just additions, subtractions, and multiplication/division by small constants. A difficult design challenge in Toom–Cook is to find an efficient sequence of operations to compute this product; one sequence given by Bodrato[6] for Toom-3 is the following, executed here over the running example:

$r_0 \leftarrow r(0)$	= 3084841486175176
$r_4 \leftarrow r(\infty)$	= 12193131840
$r_3 \leftarrow (r(-2) - r(1))/3$	= (3188843994597408 − 13260814415903778)/3
	= −3357323473768790
$r_1 \leftarrow (r(1) - r(-1))/2$	= (13260814415903778 − (−246273893346042))/2
	= 6753544154624910
$r_2 \leftarrow r(-1) - r(0)$	= −246273893346042 − 3084841486175176
	= −3331115379521218
$r_3 \leftarrow (r_2 - r_3)/2 + 2r(\infty)$	= (−3331115379521218 − (−3357323473768790))/2 + 2×12193131840
	= 13128433387466
$r_2 \leftarrow r_2 + r_1 - r_4$	= −3331115379521218 + 6753544154624910 − 12193131840
	= 3422416581971852
$r_1 \leftarrow r_1 - r_3$	= 6753544154624910 − 13128433387466
	= 6740415721237444.

We now know our product polynomial *r*:

$$\begin{aligned} r(x) = {} & 3084841486175176 \\ & + 6740415721237444x \\ & + 3422416581971852x^2 \\ & + 13128433387466x^3 \\ & + 12193131840x^4. \end{aligned}$$

If we were using different k_m, k_n, or evaluation points, the matrix and so our interpolation strategy would change; but it does not depend on the inputs and so can be hard-coded for any given set of parameters.

Recomposition [edit]

Finally, we evaluate r(B) to obtain our final answer. This is straightforward since B is a power of *b* and so the multiplications by powers of B are all shifts by a whole number of digits in base *b*. In the running example b = 10^4 and B = b^2 = 10^8.

```
                                    3084 8414 8617 5176
                          6740 4157 2123 7444
                3422 4165 8197 1852
        13 1284 3338 7466
+ 121 9313 1840
---------------------------------------------------------
  121 9326 3124 6761 1632 4937 6009 5208 5858 8617 5176
```

And this is in fact the product of 1234567890123456789012 and 987654321987654321098.

Interpolation matrices for various k [edit]

Here we give common interpolation matrices for a few different common small values of k_m and k_n.

Toom-1 [edit]

Toom-1 (k_m = k_n = 1) requires 1 evaluation point, here chosen to be 0. It degenerates to long multiplication, with an interpolation matrix of the identity matrix:

$$\begin{pmatrix}1\end{pmatrix}^{-1} = \begin{pmatrix}1\end{pmatrix}.$$

Toom-1.5 [edit]

Toom-1.5 (k_m = 2, k_n = 1) requires 2 evaluation points, here chosen to be 0 and ∞. Its interpolation matrix is then the identity matrix:

$$\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}^{-1} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}.$$

Toom-2 [edit]

Toom-2 (k_m = 2, k_n = 2) requires 3 evaluation points, here chosen to be 0, 1, and ∞. It is the same as Karatsuba multiplication, with an interpolation matrix of:

$$\begin{pmatrix}1 & 0 & 0\\ 1 & 1 & 1\\ 0 & 0 & 1\end{pmatrix}^{-1} = \begin{pmatrix}1 & 0 & 0\\ -1 & 1 & -1\\ 0 & 0 & 1\end{pmatrix}.$$

Toom-2.5 [edit]

Toom-2.5 (k_m = 3, k_n = 2) requires 4 evaluation points, here chosen to be 0, 1, -1, and ∞. It then has an interpolation matrix of:

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 0 & 0 & 0 & 1\end{pmatrix}^{-1} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1/2 & -1/2 & -1\\ -1 & 1/2 & 1/2 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}.$$

Notes [edit]

1. ^ [a] [b] Knuth, p. 296
2. ^ Crandall & Pomerance, p. 474
3. ^ Crandall & Pomerance, p. 536
4. ^ Knuth, p. 302
5. ^ Positive Results, chapter III of Stephen A. Cook: *On the Minimum Computation Time of Functions*.
6. ^ [a] [b] [c] Marco Bodrato. Towards Optimal Toom–Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0. In *WAIFI'07 proceedings*, volume 4547 of LNCS, pages 116–133. June 21–22, 2007. author website

References [edit]

- D. Knuth. *The Art of Computer Programming*, Volume 2. Third Edition, Addison-Wesley, 1997. Section 4.3.3.A: Digital methods, pg.294.
- R. Crandall & C. Pomerance. *Prime Numbers – A Computational Perspective*. Second Edition, Springer, 2005. Section 9.5.1: Karatsuba and Toom–Cook methods, pg.473.
- M. Bodrato. *Toward Optimal Toom–Cook Multiplication...*. In WAIFI'07, Springer, 2007.

External links [edit]

- Toom-Cook 3-Way Multiplication from GMP Documentations

V · T · E	Number-theoretic algorithms [hide]
Primality tests	AKS TEST · APR TEST · Baillie–PSW · ECPP TEST · Elliptic curve · Pocklington · Fermat · Lucas · *LUCAS–LEHMER* · *LUCAS–LEHMER–RIESEL* · *PROTH'S THEOREM* · *PÉPIN'S* · Quadratic Frobenius test · Solovay–Strassen · Miller–Rabin
Prime-generating	Sieve of Atkin · Sieve of Eratosthenes · Sieve of Sundaram · Wheel factorization
Integer factorization	Continued fraction (CFRAC) · Dixon's · Lenstra elliptic curve (ECM) · Euler's · Pollard's rho · $p-1$ · $p+1$ · Quadratic sieve (QS) · General number field sieve (GNFS) · *Special number field sieve (SNFS)* · Rational sieve · Fermat's · Shanks' square forms · Trial division · Shor's
Multiplication	Ancient Egyptian · Long · Karatsuba · **Toom–Cook** · Schönhage–Strassen · Fürer's
Discrete logarithm	BABY-STEP GIANT-STEP · Pollard rho · Pollard kangaroo · POHLIG–HELLMAN · Index calculus · Function field sieve
Greatest common divisor	Binary · Euclidean · Extended Euclidean · Lehmer's
Modular square root	Cipolla · Pocklington's · Tonelli–Shanks
Other algorithms	Chakravala · Cornacchia · Integer relation · Integer square root · Modular exponentiation · Schoof's
	Italics indicate that algorithm is for numbers of special forms · SMALLCAPS indicate a deterministic algorithm

Categories: Computer arithmetic algorithms | Multiplication