Computer Algebra 2

October 5, 2018

1 Notations and conventions

Unless otherwise mentioned, we use the following notations:

- $k, K, \mathbb{K}$ are (commutative) fields
- R is a (commutative, with 1) ring

Given a ring R, R^* is the group of its invertible elements.

We assume that algebraic computations (sum, inverse, test of 0, test of 1, inverse where applicable) can be performed.

For a vector v in a vector space V of dimension n, we denote its coordinates by $(v_1, \ldots, v_{n-1})$. If f is a polynomial of degree $\deg(f) = d$, its coefficients are denoted $f_0, \ldots, f_d$, such that

$$f(X) = f_0 + f_1 X + \cdots + f_d X^d = \sum_{i=0}^{d} f_i X^i.$$

In order to simplify notations, we may at times use the convention that $f_i = 0$ if $i < 0$ or $i > \deg(f)$, so that

$$f = \sum_{i \in \mathbb{Z}} f_i X^i.$$

By convention, the degree of the 0 polynomial is $-\infty$.

The logarithm log, without a base, is in base 2.

Definition 1. Given two functions $f, g : \mathbb{N} \to \mathbb{R}_{>0}$

$$\begin{aligned} f = O(g) &\iff \frac{f(n)}{g(n)} \text{ is bounded when } n \to \infty \\ &\iff \exists c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N}, \forall n \geq n_0, f(n) \leq c g(n); \end{aligned}$$

$$f = \tilde{O}(g) \iff \exists l \in \mathbb{N}, f = O(g \log(g)^l).$$

1.1 Exercises

Exercise 1.1. Show that the "when $n \to \infty$" clause in the definition of O can be left out. In

other words, given $f, g : \mathbb{N} \to \mathbb{R}_{>0}$, show that

$$\begin{aligned} f = O(g) &\iff \frac{f(n)}{g(n)} \text{ is bounded} \\ &\iff \exists c \in \mathbb{R}_{>0}, \forall n \in \mathbb{N}, f(n) \leq cg(n) \end{aligned}$$

2 Semi-fast multiplication

In this chapter, let R be an infinite ring.

Given $f, g \in R[X]$ with degree less than n, we want to compute the coefficients of $h = f \cdot g$.

The complexity of the algorithm will be evaluated in number of multiplications and additions in R. Typically, multiplications are more expensive!

2.1 Naive algorithm

Each coefficient h_k $(0 \leq k < 2n)$ can be computed with

$$h_k = \sum_{i=0}^{n} f_i g_{k-i},$$

each costing $O(n)$ multiplications and additions.

The total complexity of the naive algorithm is $O(n^2)$ multiplications and $O(n^2)$ additions.

2.2 Karatsuba's algorithm

Remark 2. Linear polynomials can be multiplied using 3 multiplications instead of 4 :

$$(a + bX)(c + dX) = ac + (ad + bc)X + bdX^2$$

with

$$ad + bc = ad + bc + ac + bd - ac - bd = (a + b)(c + d) - ac - bd.$$

This can be used recursively to compute polynomial multiplication faster.

Algorithm 1 Karatsuba

Input: $f = f_0 + \cdots + f_{n-1}X^{n-1}$, $g = g_0 + \cdots + g_{n-1}X^{n-1}$

Output: $h = h_0 + \cdots + h_{2n-1}X^{2n-1}$ such that $h = fg$

1. If $n = 1$, then return $f_0 g_0$
2. Write $f = A + BX^{\lceil n/2 \rceil}$, $g = C + DX^{\lceil n/2 \rceil}$ where all of A, B, C, D have degree $< \lceil \frac{n}{2} \rceil$.
3. Compute recursively:
   - $P = AC$
   - $Q = BD$
   - $R = (A + B)(C + D)$
4. Return $P + (R - P - Q)X^{\lceil n/2 \rceil} + RX^{2\lceil n/2 \rceil}$

Theorem 3. *Karatsuba's algorithm multiplies polynomials with $O(n^{\log_2(3)}) = O(n^{1.585})$ multiplications and additions.*

Proof. Let $M(n)$ (resp. $A(n)$) be the number of multiplications (resp. additions) in a run of Algo. 1 on an input with size n. Then:

$$M(n) = 3M(n/2)$$

and

$$A(n) = 3A(n/2) + O(n)$$

so $M(n) = O(n^{\log_2(3)})$ and $A(n) = O(n^{\log_2(3)})$. □

Remark 4. Karatsuba's algorithm hides an evaluation/interpolation mechanism:

$$\begin{aligned} a &= (a + bX)_{X=0} \\ a + b &= (a + bX)_{X=1} \\ b &= \left(\frac{a + bX}{X}\right)_{X=\infty} \end{aligned}$$

and for two linear polynomials f, g, if $fg = h = h_0 + h_1X + h_2$, we have

$$\begin{aligned} &f(0)g(0) = h(0) = h_0 \\ &f(1)g(1) = h(X = 1) = h_0 + h_1 + h_2 \\ &\left(\frac{f}{X}\right)_{X=\infty}\left(\frac{g}{X}\right)_{X=\infty} = \left(\frac{h}{X^2}\right)_{X=\infty} = h_2 \end{aligned}$$

2.3 Toom-k algorithm

For the remainder of this section, assume that the ring R contains $\mathbb{Z}$, so that $\{0, \ldots, k\}$ contains k distinct elements for all values of k. In general the coefficients of h can be obtained as a linear combination of $f(i)g(i)$ for $i \in \{0, \ldots, 2n-1\}$ via

$$\begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \ldots \\ 1 & 1 & 1 & \ldots \\ 1 & 2 & 4 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}^{-1} \left[\begin{pmatrix} 1 & 0 & 0 & \ldots \\ 1 & 1 & 1 & \ldots \\ 1 & 2 & 4 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \end{pmatrix} \odot \begin{pmatrix} 1 & 0 & 0 & \ldots \\ 1 & 1 & 1 & \ldots \\ 1 & 2 & 4 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \end{pmatrix} \right]$$

where $\odot$ is the component-wise multiplication of two vectors.

This suggests the following generalization of Algo. 1 for any fixed $k \geq 2$. First, let $V = \left(i^j\right)_{i,j=0}^{2k-1}$ (Vandermonde matrix), and precompute V^{-1}.

Algorithm 2 Toom-k

Input: $f = f_0 + \cdots + f_{n-1}X^{n-1}$, $g = g_0 + \cdots + g_{n-1}X^{n-1}$

Output: $h = h_0 + \cdots + h_{2n-1}X^{2n-1}$ such that $h = fg$

1. If $n < \max(k, 16)$, compute h naively and stop # Forget the "16" until Sec. 2.4
2. Write $f = F_0 + F_1X^{\lceil n/k \rceil} + \cdots + F_{k-1}X^{(k-1)\lceil n/k \rceil}$ and $g = G_0 + G_1X^{\lceil n/k \rceil} + \cdots + G_{k-1}X^{(k-1)\lceil n/k \rceil}$ where $\deg(F_i)$ and $\deg(G_i) < n/k$
3. Compute $\bar{f} = V\begin{pmatrix} F_0 \\ F_1 \\ \vdots \end{pmatrix}$ and $\bar{g} = V\begin{pmatrix} G_0 \\ G_1 \\ \vdots \end{pmatrix}$
4. Compute $\bar{h} = \bar{f} \odot \bar{g}$ recursively
5. Return $V^{-1}\bar{h}$

Theorem 5. *For every fixed $\epsilon > 0$ there exists a multiplication algorithm for $R[X]$ which requires $O(n^{1+\epsilon})$ operations in R.*

Proof. See Exercise 2.1. □

Remark 6. For fixed k, the cost of precomputing V and V^{-1} can be neglected, since it is a fixed cost of $O(k^2)$ and $O(k^3)$ respectively.

2.4 Toom-Cook algorithm

Theorem 7 (Toom-Cook)**.** *There exists a multiplication algorithm for $R[X]$ that requires $O(n^{1+2/\sqrt{\log(n)}})$ operations in R. This algorithm is obtained by adapting Algo. 2 to choose at each recursion level $k = \left\lfloor 2^{2\sqrt{\log(n)}} \right\rfloor$.*

Proof. See Exercise 2.2. □

Remark 8. This complexity is better than that of Toom-k, since it is better than $O(2^{1+\epsilon})$ for *all* $\epsilon > 0$.

Remark 9. Strassen's algorithm for matrix multiplication is based on the same idea as Karatsuba's algorithm, and runs in time $O(n^{\log_2(7)}) \leq O(n^{2.82})$. Is there a Toom-Cook style algorithm for matrix multiplication, with complexity better than $O(2^{2+\epsilon})$ for all $\epsilon > 0$?

For even k, we can multiply $k \times k$ matrices with $\frac{1}{3}k^3 + 6k^2 - \frac{4}{3}k$ operations, so there are matrix multiplication algorithms with complexity $O(n^{\log_k(\frac{1}{3}k^3+6k^2-\frac{4}{3}k)})$. But $\log_k(\frac{1}{3}k^3 + 6k^2 - \frac{4}{3}k)$ tends to 3 when k tends to ∞. Its minimum (over $2\mathbb{N}$) is reached at $k = 70$, leading to a complexity $O(n^{2.796})$ (Pan's algorithm).

The current record is $O(n^{2.372\,863\,9})$ (Le Gall 2014), and yes, that many decimal points are necessary! It is conjectured that a complexity of $O(2^{1+\epsilon})$ for all ϵ is realizable.

Remark 10. It is conjectured that polynomial multiplication in $O(n)$ operations is not possible.

2.5 Exercises

Exercise 2.1. Prove that for any fixed $\epsilon > 0$, there exists $k \in \mathbb{N}$ such that Algo. 2 (Toom-k) requires $O(n^{1+\epsilon})$ operations in R.

Exercise 2.2. Prove Theorem 7.

Exercise 2.3. Show that there is no algorithm which can multiply two linear polynomials (over any ring) in 2 multiplications.