Chapter 8 Fast multiplication

M1. Classical multiplication algorithm ($O(n^2)$)

M2. Karatsuba multiplication algorithm ($O(n^{\log 3})$)

M3. Asymptotically fast algorithms based on FFT ($O(n \log n)$)

M4. Schönhage and Strassen's multiplication algorithm ($O(n \log n \log \log n)$)

1. crossover points

2. complexity

3. experiments, benchmark problem

8.1 Karatsuba's multiplication algorithm

For instance, multiplying

$$(ax + b)(cx + d) = acx^2 + (ad + bc)x + bd$$

a better way

$$(ax + b)(cx + d) = acx^2 + ((a + b)(c + d) - ac - bd)x + bd$$

Given $f, g \in R[x]$ of degree less than n, assume $n = 2^k, m = n/2$,

$$f = F_1 x^m + F_0, \ \ g = G_1 x^m + G_0$$

$$f \cdot g = F_1 G_1 x^n + ((F_0 + F_1)(G_0 + G_1) - F_0 G_0 - F_1 G_1) x^m + F_0 G_0$$

We have

$$T(2n) \leq 3T(n) + cn.$$

Algorithm 8.1 Karatsuba's polynomial multiplication algorithm

Input: $f, g \in R[x]$ of degree less than n, assume $n = 2^k$.

Output: $f \cdot g \in R[x]$.

1. if $n = 1$ then return $f \cdot g \in R$

2. let $f = F_1 x^{n/2} + F_0$ and $g = G_1 x^{n/2} + G_0$

3. compute $F_0 G_0, F_1 G_1$, and $(F_0 + F_1)(G_0 + G_1)$ by a recursive call

4. return $F_1 G_1 x^n + ((F_0 + F_1)(G_0 + G_1) - F_0 G_0 - F_1 G_1) x^m + F_0 G_0$

Q 1: If n is not a power of 2 , how to proceed? (two ways)

Theorem 1. *Karatsuba's algorithm for multiplying polynomials of degree less than a power* n *of 2 over a ring can be done with at most* $9n^{\log 3}$ *or* $O(n^{1.59})$ *ring operations.*

Theorem 2. *Multiplication of two integers of length at most* n *words can be done with* $O(n^{\log 3})$ *or* $O(n^{1.59})$ *word operations using Karatsuba's algorithm.*

8.2 Discrete Fourier Transform and Fast Fourier Transform

An element $a \in R$ is a zero divisor if there exist nonzero $b \in R$ with $ab = 0$.

Let R be a ring, $n \in \mathbb{N}_{\geq 1}$, and $\omega \in R$.

1. ω is an nth root of unity if $\omega^n = 1$.

2. ω is a primitive nth root of unity if it is an nth root of unity, $n \in R$ is a unit in R, and $\omega^{n/t} - 1$ is not a zero divisor for any prime divisor t of n.

For examples

1. Z_8 has no primitive square root of unity, despite $3^2 \equiv 1 \mod 8$, since 2 is not a unit in Z_8.

2. $\omega = e^{2\pi i/8} \in \mathbb{C}$ is a primitive 8th root of unity

3. 3 is a primitive 16th root of unity in Z_{17}, but 2 is not since $2^8 - 1 \equiv 0 \mod 17$.

Lemma 1. *Let R be a ring, $l, n \in \mathbb{N}_{\geq 1}$ such that $1 < l < n$, and $\omega \in R$ a primitive nth root of unity. Then*

1. *$\omega^l - 1$ is not a zero divisor in R.*

2. *$\sum_{0 \leq j < n} \omega^{lj} = 0$.*

Lemma 2. *For a prime power q and $n \in \mathbb{N}$ with $\gcd(q, n) = 1$, F_q contains a primitive nth root of unity if and only if n divides $q - 1$.*

(For the above example, Z_8, $2 \nmid 7$; $Z_{17}, 16|(17-1)$)

The R-linear map which evaluate a polynomial at the powers of ω is called the Discrete Fourier Transform(DFT),

$$DFT_\omega : \begin{array}{ccc} R^n & \longrightarrow & R^n \\ f & \longmapsto & \big(f(1), f(\omega), \ldots, f(\omega^{n-1})\big) \end{array}$$

The convolution of two polynomials $f = \sum_{0\le i<n} f_i x^i$ and $g = \sum_{0\le j<n} g_j x^j$ in $R[x]$ is the polynomial

$$h = f *_n g = \sum_{0\le k<n} h_k x^k \in R[x],$$

where

$$h_k = \sum_{i+j\equiv k \mod n} f_i g_j = \sum_{0\le i<n} f_i g_{k-i} \text{ for } 0 \le k < n,$$

It is equivalent to polynomial multiplication in the ring $R[x]/ < x^n - 1 >$

$$f *_n g \equiv f \cdot g \mod x^n - 1$$

For example $f = x^3 + 1, g = 2x^3 + 3x^2 + x + 1$ in Z_5,

$$f *_4 g \equiv f \cdot g = 2x^6 + 3x^5 + x^4 + 3x^3 + 3x^2 + x + 1 \equiv 3x^3 + 4x + 2 \mod x^4 - 1$$

Lemma 3. *For polynomials $f, g \in R[x]$ of degree less than n, we have*

$$\mathit{DFT}_\omega(f *_n g) = \mathit{DFT}_\omega(f) \cdot \mathit{DFT}_\omega(g)$$

Proof: It comes from $f *_n g = f \cdot g + q \cdot (x^n - 1)$ for some $q \in R[x]$, so that

$$(f * g)(\omega^i) = f(\omega^i)g(\omega^i) + q(\omega^i)(\omega^{in} - 1) = f(\omega^i)g(\omega^i),\ 0 \le i < n$$

Fast polynomial multiplication: evaluate the two input polynomials, multiply the results pointwise, and finally interpolate to get the product of polynomials.

Theorem 3. *Let R be a ring(commutative, with 1), $n \in \mathbb{N}_{\geq 1}$, and $\omega \in R$ be a primitive nth root of unity and $V_\omega V_\omega^{-1} = nI$ where I is the $n \times n$ identity matrix.*

Proof:

$$(V_\omega \cdot V_{\omega^{-1}})_{ik} = \sum_{0 \leq j < n} \omega^{ij} \omega^{-jk} = \sum_{0 \leq j < n} (w^{i-k})^j.$$

Algorithm 8.14 Fast Fourier Transform (FFT)

Input: $n = 2^k \in \mathbb{N}$ with $k \in \mathbb{N}$, $f = \sum_{0 \le i < n} f_i x^i \in R[x]$, and a primitive nth root of unity $\omega \in R$.

Output: $\mathrm{DFT}_\omega(f) = (f(1), f(\omega), \ldots, f(\omega^{n-1})) \in R^n$.

1. if $n = 1$ then return (f_0)
2. $r_0 \longleftarrow \sum\limits_{0 \le j < n/2} (f_j + f_{j+n/2}) x^j$, $r_1^* \longleftarrow \sum\limits_{0 \le j < n/2} (f_j - f_{j+n/2}) \omega^j x^j$
3. call the algorithm recursively to evaluate r_0 and r_1^* at the powers of ω^2.
4. return $\big(r_0(1), r_1^*(1), r_0(\omega^2), r_1^*(\omega^2), \ldots, r_0(\omega^{2n-2}), r_1^*(\omega^{2n-2})\big)$

It bases on $f(\omega^{2i}) = r_0(\omega^{2i})$, $f(\omega^{2i+1}) = r_1^*(\omega^{2i})$ and $r_1^* = r_1(\omega x)$

Theorem 4. *Let n be a power of 2 and $\omega \in R$ be a primitive nth root of unity. Then the algorithm correctly computes DFT$_\omega$ using $n \log n$ additions in R and $(n/2) \log n$ multiplication by powers of ω, in total $\frac{3}{2} n \log n$ ring operations.*

Algorithm 8.16 Fast convolution

Input: $f, g \in R[x]$ of degree less than $n = 2^k \in \mathbb{N}$ with $k \in \mathbb{N}$, and a primitive nth root of unity $\omega \in R$.

Output: $f *_n g \in R[x]$.

1. compute $\omega^2, \ldots, \omega^{n-1}$
2. $\alpha \longleftarrow \mathrm{DFT}_\omega(f)$, $\beta \longleftarrow \mathrm{DFT}_\omega(g)$
3. $\gamma \longleftarrow \alpha \cdot \beta$ (pointwise product)
4. return $\mathrm{DFT}_\omega{}^{-1}(\gamma) = \frac{1}{n}\mathrm{DFT}_{\omega^{-1}}(\gamma)$

Theorem 5. *Let R be a ring that supports the FFT(R has a primitive 2^kth root of unity for any $k \in \mathbb{N}$), and $n = 2^k$ for some $k \in \mathbb{N}$. Then convolution in $R[x]/ < x^n - 1 >$ and multiplication of polynomials $f, g \in R[x]$ with $\deg(f \cdot g) < n$ can be performed using $3n \log n$ additions in R, $\frac{3}{2} n \log n + n - 2$ multiplications by powers of ω, n multiplications in R, and n divisions by n, in total $\frac{9}{2} n \log n + O(n)$ arithmetic operations.*

1. $\mathsf{DFT}_\omega(f *_n g) = \mathsf{DFT}_\omega(f) \cdot \mathsf{DFT}_\omega(g)$.

2. $f *_n g \equiv f \cdot g \mod x^n - 1 \Longrightarrow f *_n g = f \cdot g$ if $\deg f \cdot g < n$.

8.3 Schönhage and Strassen's multiplication algorithm

FFT-based polynomial multiplication over rings containing certain primitive roots of unity. We will show how to make it work if R is arbitrary ring by adjoining "virtual" roots of unity.

Assume $\deg(f \cdot g) < n = 2^k, m = 2^{\lfloor k/2 \rfloor}, t = n/m,$

$$\xi = x \mod (x^{2m} + 1)$$

$$\eta = \xi \text{ if } t = 2m, \ \eta = \xi^2 \text{ if } t = m$$

Algorithm 8.20 Fast negative wrapped convolution

Input: $f, g \in R[x]$ of degree less than $n = 2^k \in \mathbb{N}$ with $k \in \mathbb{N}$, where R is a (commutative) ring such that 2 is a unit in R.

Output: $h \in R[x]$ such that $f \cdot g \equiv h \mod (x^n + 1)$ (negative wrapped convolution of f and g) and $\deg h < n$.

1. if $k \leq 2$ then call classical algorithm

2. $m \longleftarrow 2^{\lfloor k/2 \rfloor}, t \longleftarrow n/m$, $f = \sum_{0 \leq i < t} f_i x^{mi}$, $g = \sum_{0 \leq i < t} g_i x^{mi}$,
   let $f' = \sum_{0 \leq i < t} f_i y^i$, $g' = \sum_{0 \leq i < t} g_i y^i$.

3. Let $D = R[x]/ < x^{2m} + 1 >$ compute $2t$th primitive root of unity
   if $t = 2m$ then $\eta \longleftarrow x \mod (x^{2m} + 1)$ else $\eta \longleftarrow x^2 \mod (x^{2m} + 1)$

$f^* \longleftarrow f' \mod (x^{2m}+1), g^* \longleftarrow g' \mod (x^{2m}+1)$
call the fast convolution algorithm with $\omega = \eta^2$ to compute

$$f^*(\eta y) g^*(\eta y) \equiv h^*(\eta y) \mod (y^t - 1)$$

applying algorithm recursively for the multiplications in D

4. let $h' \in R[x, y]$ with $\deg_x h' < 2m$ such that $h^* = h' \mod (x^{2m}+1)$
   $h \longleftarrow h'(x, x^m)rem(x^n + 1)$
   return h

For example $R = F_5, f = x^4 + 2x + 3$ and $g = 2x^3 + x^2 + 4x + 2$ in $F_5[x]$.

1. $n = 8 = 2^k, k = 3$ and $\deg_x f \cdot g < 2^3 = 8$

2. $m = 2^{\lfloor k/2 \rfloor} = 2, t = n/m = 4,\ f' = y^2 + 2x + 3, g' = (2x + 1)y + 4x + 2.$

3. since $t = 4 = 2{*}2$, $\eta = x \mod (x^4{+}1), f^* = y^2{+}2\eta{+}3, g^* = (2\eta{+}1)y{+}4\eta{+}2$
   $f * (\eta y) = \eta^2 y^2 + 2\eta + 3, g^*(\eta y) = (2\eta^2 + \eta)y + 4\eta + 2.$
   $h^*(\eta y) = f^*(\eta y)g^*(\eta y) = (2\eta^4 + \eta^3)y^3 + (4\eta^3 + 2\eta^2)y^2 + (4\eta^3 + 3\eta^2 + 3\eta)y + 3\eta^2 + \eta + 1$
   $h^* = h^*(\eta(\eta^{-1}y)) = (2\eta + 1)y^3 + (4\eta + 2)y^2 + (4\eta^2 + 3\eta + 3)y + 3\eta^2 + \eta + 1.$

4. $h' = (2x + 1)y^3 + (4x + 2)y^2 + (4x^2 + 3x + 3)y + 3x^2 + x + 1.$
   $h'(x, x^2) = 2x^7 + x^6 + 4x^5 + x^4 + 3x^3 + x^2 + x + 1 \equiv fg \mod (x^8 + 1).$

Theorem 6. *Algorithm 8.20 works correctly and uses* $\frac{9}{2} n \log n \log \log n + O(n \log n)$ *operations.*

Theorem 7. *Multiplication of integers of length* n *can be performed with* $O(n \log n \log \log n)$ *word operations.*

Algorithm 8.25 Three primes FFT integer multiplication

Input: Two integers $a, b \in \mathbb{N}$ in 64-adic representation such that $a, b < 2^{64\cdot 2^{s-1}}$, $s \leq 62$

Output: $a \cdot b \in \mathbb{N}$.

1. Let $A, B \in \mathbb{Z}[x]$ with all coefficients nonnegative and less than 2^{64}, such that $a = A(2^{64}), b = B(2^{64}), t \longleftarrow \lceil \log_2(1 + \deg(AB)) \rceil$

2. for $i = 1, 2, 3$ call the fast convolution algorithm over F_{p_i} with $\omega \equiv \omega_i^{2^{s-t}} \mod p_i$ to compute $C_i = AB$remp_i.

3. call the CRA to compute $C \in \mathbb{N}$ such that $C \equiv C_i \mod p_i$

4. return $C(2^{64})$

8.4 Multiplication in $\mathbb{Z}[x]$ and $\mathbb{R}[x,y]$

Kronecker substitution

$f = \sum_{0\le i<n} f_i x^i, g = \sum_{0\le i<n} g_i x^i, f_i, g_i \in \mathbb{Z}$ of length at most l in the 64-adic representation. Choose $t \in \mathbb{N}$ so that $n2^{128l} < 2^{64t}$,

$$h(2^{64t}) \equiv f(2^{64t})g(2^{64t}) \mod x - 2^{64t}$$

$f = \sum_{0\le i<n} f_i x^i, g = \sum_{0\le i<n} g_i x^i \in R[y][x]$, $\deg_y f_i < d, \deg_y g_i < d$, substituting $x = y^{2d-1}$, we obtain

$$\begin{aligned} f \cdot g &\equiv f(y^{2d-1}, y) \cdot g(y^{2d-1}, y) = h(y^{2d-1}, y) \\ &= \sum_{0\le i\le 2n-2} h_i y^{2d-1} i \equiv h \mod x - y^{2d-1} \end{aligned}$$