

Asymptotically fast algorithms for modern computer algebra

Tutorial at ISSAC 2010, München

Jürgen Gerhard, Maplesoft

25 July 2010

1

Outline

- Introduction
- Basic polynomial arithmetic
- Modular algorithms
- B r e a k
- Advanced polynomial arithmetic
- Optimizations
- Applications

2

Introduction

3

Contents

- Motivation
- Implementations
- Agenda

4

Why fast arithmetic?

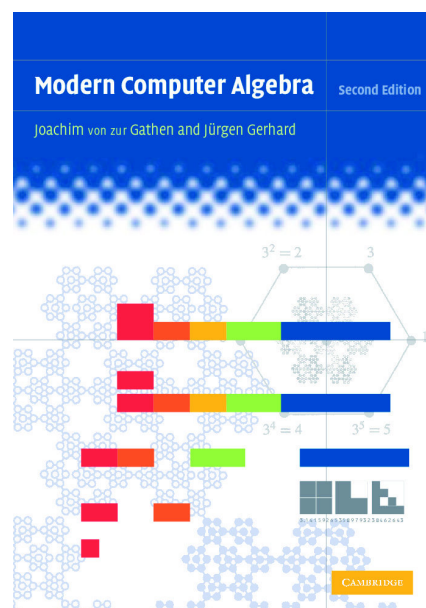
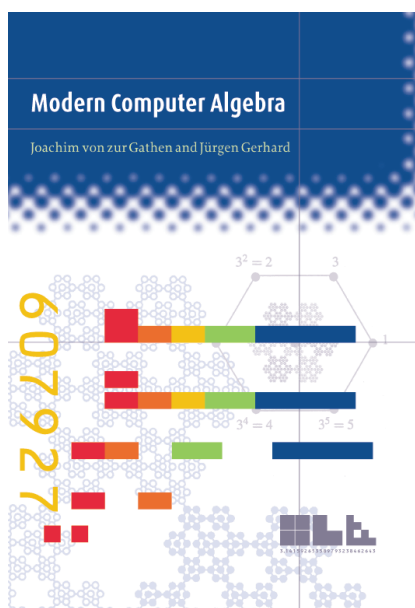
- Performance required by industrial size problems
- Dense uni-/bivariate arithmetic at the root of sparse/multivariate arithmetic
- It's fun!

5

(The Art Of) Modern Computer Algebra

1999

2003



6

Implementations

- NTL (Shoup)
- TPS (Schönhage)
- Magma (Cannon, Steele)
- GMP (Granlund, Zimmermann)
- Modpn (Moreno Maza et al)

7

Agenda

- Core topic: dense univariate polynomials
- Periphery: integers
- Out of scope:
 - sparse / multivariate polynomials
 - differential and recurrence equations
 - linear algebra
 - parallel / space efficient algorithms
- Light on references

8

Basic polynomial arithmetic

9

Contents

- Representation of polynomials
- Addition and scalar multiplication
- Classical multiplication
- Karatsuba's algorithm
- Toom-Cook's algorithm
- The Fast Fourier Transform
- Classical division with remainder
- Reducing division to multiplication
- Newton inversion
- Pseudo-division

10

Representation of polynomials

Dense univariate polynomials over a ring R

$$f = f_n x^n + f_{n-1} x^{n-1} + \dots + f_1 x + f_0,$$

$$f_i \in R, \quad R = \mathbb{Z}, \mathbb{F}_q, \dots$$

- x is just a placeholder and $+$ is just a separator
- n is the *degree* of f if $f_n \neq 0$
- only arithmetic operations $+$, $-$, \times , \div on coefficients are counted
- We try to give explicit constants for the dominant terms

11

Representation of integers

Multiprecision integers

$$a = a_n 2^{64n} + a_{n-1} 2^{64(n-1)} + \dots + a_1 2^{64} + a_0,$$

$$a_i \in \mathbb{N}, \quad 0 \leq a_i < 2^{64}$$

- n is the *word length* of a if $a_n \neq 0$
- only arithmetic operations $+$, $-$, \times , \div on words are counted (*word operations*)

12

Addition and scalar multiplication

Addition: $\deg f < n, \deg g < n$

for $0 \leq i < n$ **do**

$$h_i \leftarrow f_i + g_i$$

Scalar multiplication: $\deg f < n, a \in R$

for $0 \leq i < n$ **do**

$$h_i \leftarrow a \cdot f_i$$

n operations in R

13

Exercise 1: Evaluation

$\deg f = n, a \in R$

Compute $f(a) = f_n a^n + \cdots + f_1 a + f_0 \in R$

$$b_0 \leftarrow 1$$

for $1 \leq i \leq n$ **do**

$$b_i \leftarrow a \cdot b_{i-1}$$

$$h_0 \leftarrow f_0$$

for $1 \leq i \leq n$ **do**

$$h_i \leftarrow h_{i-1} + f_i \cdot b_i$$

$3n$ operations in R . Can you do it with $2n$?

14

Classical multiplication I

$\deg f < n, \deg g < m$

$$h_k = \sum_{i+j=k} f_i g_j$$

for $0 \leq k \leq m + n - 2$ **do**

$h_k \leftarrow 0$

for $\max(0, k + 1 - m) \leq i \leq \min(n - 1, k)$ **do**

$h_k \leftarrow h_k + f_i \cdot g_{k-i}$

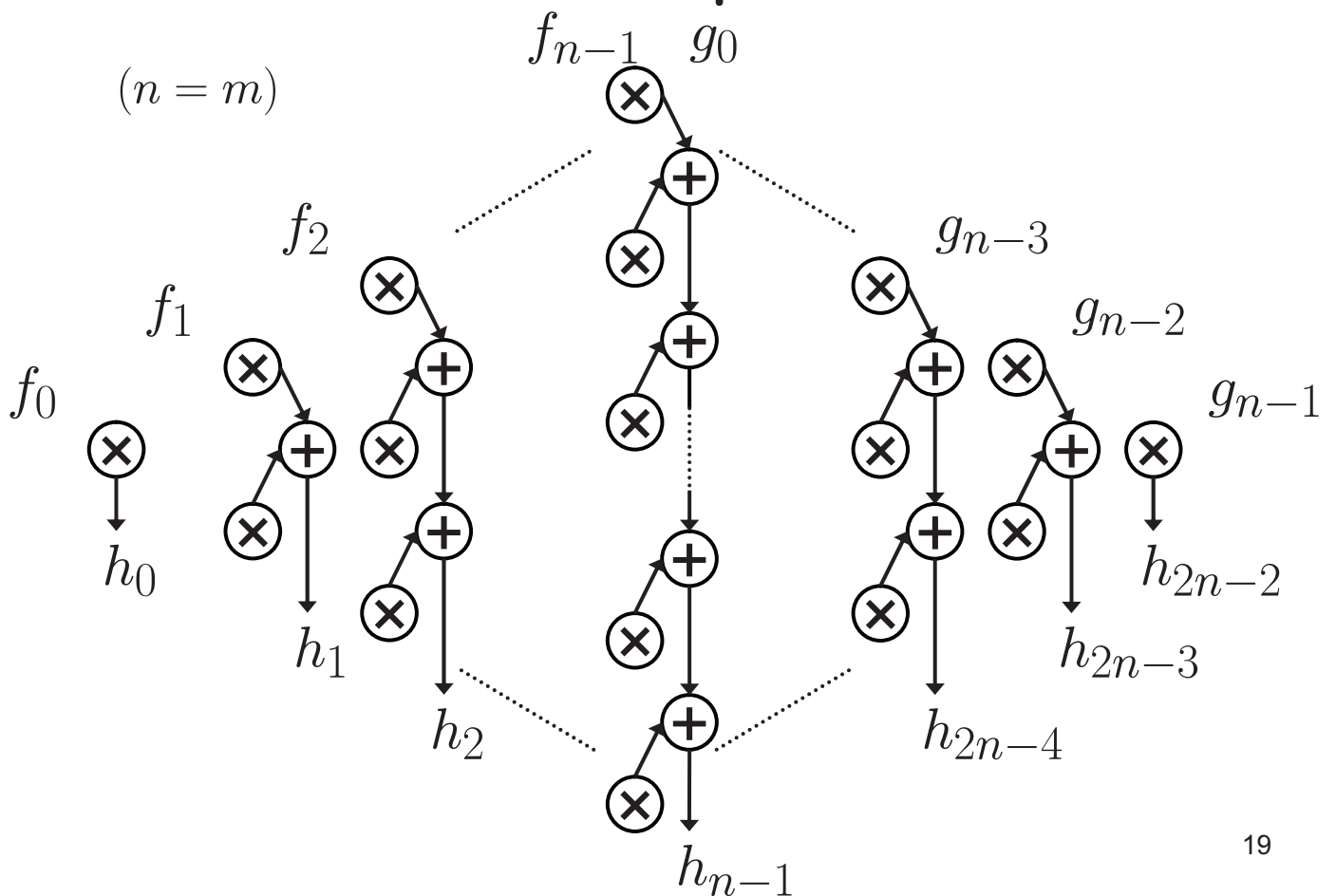
nm multiplications and $(n - 1)(m - 1)$ additions in R

Exercise 2

What is the arithmetic cost for multiplying two *monic* polynomials?

17

Classical multiplication II



19

Karatsuba's algorithm I

$$\begin{aligned} & (f_3x^3 + f_2x^2 + f_1x + f_0) \cdot (g_3x^3 + g_2x^2 + g_1x + g_0) \\ = & ((f_3x + f_2)x^2 + (f_1x + f_0)) \cdot ((g_3x + g_2)x^2 + (g_1x + g_0)) \\ = & (F_1x^2 + F_0) \cdot (G_1x^2 + G_0) \\ = & F_1G_1x^4 + (F_1G_0 + F_0G_1)x^2 + F_0G_0 \\ = & F_1G_1x^4 + ((F_1 + F_0)(G_1 + G_0) - F_1G_1 - F_0G_0)x^2 + F_0G_0 \end{aligned}$$

For $\deg f < n$, $\deg g < n$, and n even:

Classical: 4 recursive calls + $O(n)$ additions

Karatsuba: 3 recursive calls + $O(n)$ additions

Exercise 3: Determine $O(n)$ for Karatsuba exactly.

Note: The coefficients of $F_1G_1x^4$ and of F_0G_0 overlap with the middle term.

Karatsuba's algorithm II

$$K(n) = 3K\left(\frac{1}{2}n\right) + cn + d$$

$$n = 2^k:$$

$$\begin{aligned} K(n) &= K(2^k) = 3K(2^{k-1}) + c2^k + d \\ &= 9K(2^{k-2}) + c(32^{k-1} + 2^k) + d(3 + 1) \\ &= 3^3 K(2^{k-3}) + c(3^2 2^{k-2} + 32^{k-1} + 2^k) + d(3^2 + 3 + 1) \\ &= \dots \\ &= 3^k K(1) + c(3^{k-1} \cdot 2 + \dots + 3^0 2^k) + d(3^{k-1} + \dots + 3^0) \\ &= 3^k + 2c(3^k - 2^k) + \frac{1}{2}d(3^k - 1) \\ &= \left(1 + 2c + \frac{1}{2}d\right)n^{\log_2 3} - 2cn - \frac{1}{2}d = O(n^{1.59}) \end{aligned}$$

23

Karatsuba's algorithm III

$$Y = x^{n/2}: (F_1 Y + F_0)(G_1 Y + G_0) = F_1 G_1 Y^2 + ((F_1 + F_0)(G_1 + G_0) - F_1 G_1 - F_0 G_0)Y + F_0 G_0$$

Observe:

$$F_0 = F(0), G_0 = G(0), F_0 G_0 = (FG)(0),$$

$$F_1 + F_0 = F(1), G_1 + G_0 = G(1),$$

$$F_1 = F(\infty), G_1 = G(\infty)$$

Alternative: 2 instead of ∞

$$F(2) = 2F_1 + F_0, G(2) = 2G_1 + G_0$$

$$F_1 G_1 = \frac{1}{2}((FG)(2) - 2(FG)(1) + (FG)(0))$$

$$\begin{aligned} F_1 G_0 + F_0 G_1 &= (FG)(1) - (FG)(0) - F_1 G_1 \\ &= -\frac{1}{2}(FG)(2) + 2(FG)(1) - \frac{3}{2}(FG)(0) \end{aligned}$$

24

Generalization

Split f and g into k blocks instead of 2 ($Y = x^{n/k}$):

$$f = F_{k-1}Y^{k-1} + \dots + F_1Y + F_0$$

$$g = G_{k-1}Y^{k-1} + \dots + G_1Y + G_0$$

Compute $F(0), F(1), \dots, F(2k-2)$

and $G(0), G(1), \dots, G(2k-2)$, using $O(kn)$ additions

Compute $(FG)(0), (FG)(1), \dots, (FG)(2k-2)$ recursively

Interpolate $\text{coeff}_Y(FG)$ using $O(kn)$ additions

Back-substitute $Y = x^{n/k}$ using $O(kn)$ additions

$$T(n) = (2k-1)T\left(\frac{n}{k}\right) + O(kn)$$

25

Toom-Cook's method

$$T(n) = (2k-1)T\left(\frac{n}{k}\right) + O(kn)$$

$$T(n) = n^{\log_k(2k-1)} + O(kn)$$

$$\log_k(2k-1) \leq 1 + \frac{\ln 2}{\ln k}$$

$$k \rightarrow \infty: T(n) = O(n^{1+\varepsilon})$$

Using evaluation and interpolation, we can multiply in almost linear time!

26

Special evaluation points

Suppose $\omega \in R$, $\text{char} R \nmid n$,

$\omega^n = 1$ and $\omega^\ell - 1$ not a zero divisor for $1 \leq \ell < n$.

ω is a **primitive n th root of unity**.

Examples:

1) $\omega = e^{2\pi i/n} \in \mathbb{C}$

2) $\omega = 2 \in \mathbb{Z}_{2^{n/2}+1}$ ($n = 2^k$)

3) $\omega = y \in R[y]/(y^{n/2} + 1)$ ($n = 2^k$)

Idea: multiplication via evaluation and interpolation at the powers of ω

27

The discrete Fourier transform

$\deg f + \deg g < n$

Compute $f(1), f(\omega), \dots, f(\omega^{n-1})$ and

$g(1), g(\omega), \dots, g(\omega^{n-1})$

Compute $(fg)(1), (fg)(\omega), \dots, (fg)(\omega^{n-1})$

Interpolate fg

The map $f \in R[x] \mapsto (f(1), f(\omega), \dots, f(\omega^{n-1})) \in R^n$ is called the **discrete Fourier transform** DFT_ω .

28

The fast Fourier transform I

$\deg f < n$ even, $i < n$: $f(x) = F_1(x^2) \cdot x + F_0(x^2)$

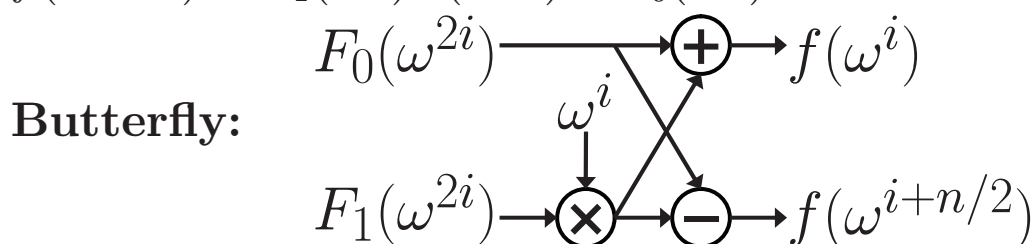
$\deg F_0(x), \deg F_1(x) < \frac{1}{2}n$

$f(\omega^i) = F_1(\omega^{2i}) \cdot \omega^i + F_0(\omega^{2i})$

$\eta = \omega^2$ primitive $\frac{n}{2}$ th root of unity

Note $\omega^{i+n/2} = \omega^i \omega^{n/2} = -\omega^i$ (Exercise: why?) and

$f(\omega^{i+n/2}) = F_1(\omega^{2i}) \cdot (-\omega^i) + F_0(\omega^{2i})$



(COOLEY & TUCKEY, STOCKHAM)

29

The fast Fourier transform II

$F_1(\omega^{2i})$ and $F_0(\omega^{2i})$ ($0 \leq i < \frac{1}{2}n$)

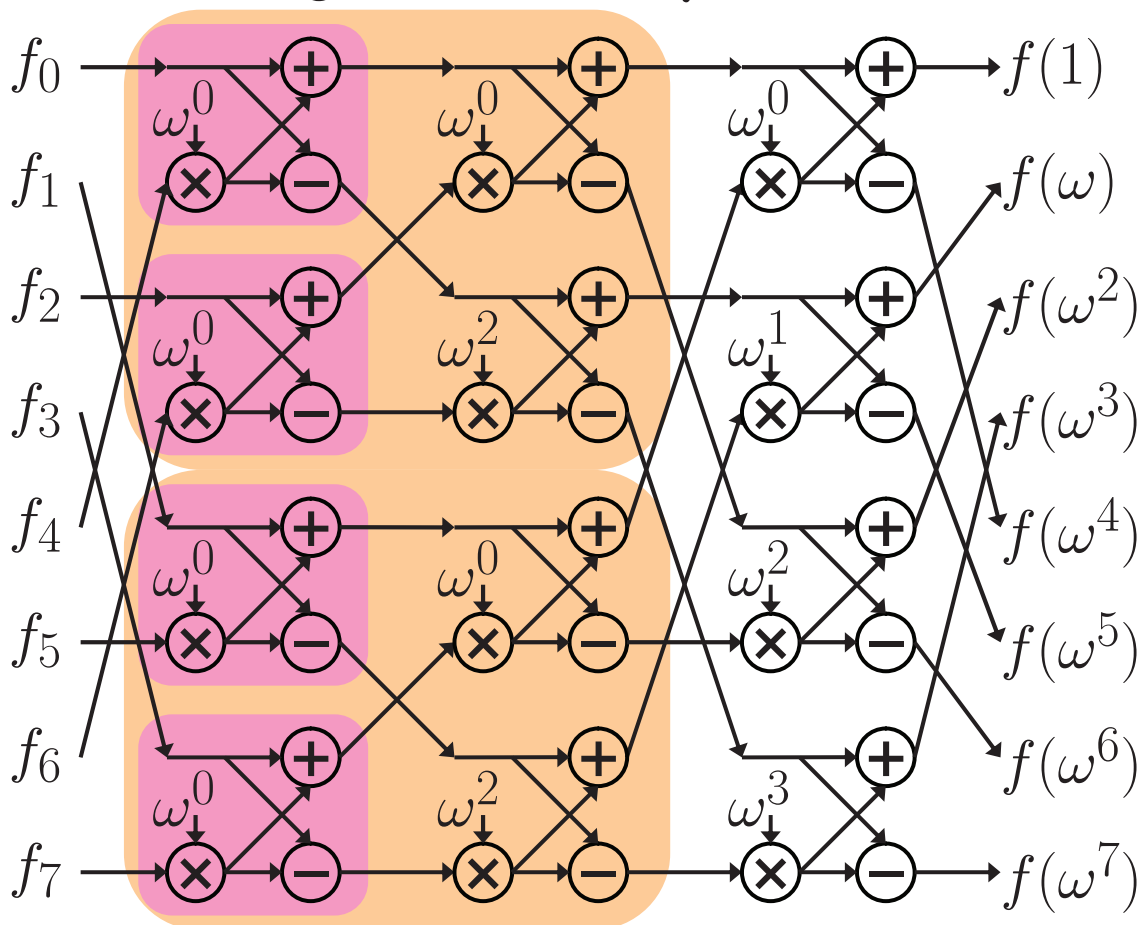
can be computed using 2 DFT $_{\omega^2}$

$$\begin{aligned}
 n = 2^k : \quad F(n) &= F(2^k) = F(2^{k-1}) + \frac{3}{2} \cdot 2^k \\
 &= 2(2F(2^{k-2}) + \frac{3}{2}2^{k-1}) + \frac{3}{2}2^k \\
 &= \dots \\
 &= 2^k F(1) + \frac{3}{2}k2^k = \frac{3}{2}n \log_2 n
 \end{aligned}$$

$+\frac{1}{2}n - 2$ for precomputing $\omega^2, \dots, \omega^{n/2-1}$

30

FFT₈ butterfly network



31

Inverting the DFT

$$\text{Butterfly: } \begin{pmatrix} f(\omega^i) \\ f(\omega^{i+n/2}) \end{pmatrix} = \begin{pmatrix} \omega^i & 1 \\ -\omega^i & 1 \end{pmatrix} \cdot \begin{pmatrix} F_1(\omega^{2i}) \\ F_0(\omega^{2i}) \end{pmatrix}$$

$$\begin{pmatrix} \omega^i & 1 \\ -\omega^i & 1 \end{pmatrix}^{-1} = \frac{1}{2\omega^i} \begin{pmatrix} 1 & \omega^i \\ -1 & \omega^i \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \omega^{-i} & 1 \\ -\omega^{-i} & 1 \end{pmatrix}$$

ω^{-1} is a primitive n th root of unity and

$$\text{DFT}_{\omega}^{-1} = \frac{1}{n} \text{DFT}_{\omega^{-1}}$$

32

Fast multiplication

$\deg f + \deg g < n$

- 1) Evaluate f and g at $1, \omega, \dots, \omega^{n-1}$
- 2) Pointwise multiplication $(fg)(1), (fg)(\omega), \dots$
- 3) Interpolate fg

Cost:

$$\frac{1}{2}n + 3n \log_2 n + n + \frac{3}{2}n \log_2 n + n = \frac{9}{2}n \log_2 n + \frac{5}{2}n$$

Over a ring R containing a primitive $2n$ th root of unity, polynomials of degree $< n$ can be multiplied using $M(n) = O(n \log n)$ arithmetic operations in R .

33

Fast multiplication II

What if there is no suitable ω (e.g., $R = \mathbb{Z}$)?

Enlarge R to $S = R[\omega]/(\omega^{\sqrt{n}} + 1)$.

This reduces multiplication of polynomials of degree $< n$ to multiplication of polynomials of degree $< \sqrt{n}$.

Over an arbitrary ring R , polynomials of degree $< n$ can be multiplied using $M(n) = O(n \log n \log \log n)$ arithmetic operations in R .

(SCHÖNHAGE, STRASSEN, CANTOR, KALTOFEN)

34

Division with remainder

Classical:

$f = qg + r$, $\deg f = m + n$, $\deg g = n$, $\deg r < n$,
 g_n invertible in R

$r \leftarrow f$

for $i = m, m - 1, \dots, 0$ **do**

$$q_i \leftarrow \frac{r_{i+n}}{g_n}$$

$$r \leftarrow r - q_i x^i \cdot g \quad \{ f = (q_m x^m + \dots + q_i x^i)g + r \}$$

$(m + 1)(2n + 1)$ operations in R

35

Exercise 4

What is the arithmetic cost for dividing by a *monic* polynomial?

36

Exercise 5: Fast exact division

Suppose $r = 0$, i.e., $f = qg$, $\deg q = m < n = \deg g$, and R contains a primitive n th root of unity ω . Give an algorithm for computing q using $\frac{9}{2}n \log_2 n + 3n + 1$ operations in R .

38

Reducing division to multiplication

$$f = qg + r$$

Idea: revert the coefficients

$$g^* = x^n g(x^{-1}), \text{ similarly } f^* \text{ and } q^*$$

$$\text{Example: } g = x^2 + 2x + 3 \implies g^* = 3x^2 + 2x + 1$$

Then $f^* = q^* g^* + x^{n+m} r(x^{-1})$, and $\deg r < n$ implies $f^* \equiv q^* g^* \pmod{x^{m+1}}$ and $q^* = f^* \cdot (g^*)^{-1} \pmod{x^{m+1}}$

40

Newton inversion I

$u^{-1} \bmod x^n, n = 2^k$

$v_0 \leftarrow u_0^{-1}$

for $i = 0, \dots, k - 1$ **do**

$v_{i+1} \leftarrow 2v_i - uv_i^2 \bmod x^{2^{i+1}}$

Cost (assuming *superlinearity* $M(\frac{1}{2}t) \leq \frac{1}{2}M(t)$):

$$\begin{aligned} N(n) &= 1 + \sum_{0 \leq i < k} (M(2^i) + M(2^{i+1}) + 2^i) \\ &\leq 1 + M(n) \sum_{0 \leq i < k} (2^{i-k} + 2^{i+1-k}) + \sum_{0 \leq i < k} 2^i \\ &= 3M(n) \left(1 - \frac{1}{n}\right) + n \end{aligned}$$

41

Newton inversion II

Why does it work?

Invariant: $uv_i \equiv 1 \bmod x^{2^i}$

$uv_{i+1} \equiv 2uv_i - u^2v_i^2 = 1 - (uv_i - 1)^2 \equiv 1 \bmod x^{2^{i+1}}$

Newton iteration from numerical analysis:

Want to find v with $f(v) = 0$.

Given v_i , iterate $v_{i+1} = v_i - \frac{f(v_i)}{f'(v_i)}$

Here $f(v) = uv - 1$ and

$v_{i+1} = v_i - \frac{uv_i - 1}{u} \equiv v_i - (uv_i - 1) \cdot v_i \bmod x^{2^{i+1}}$

42

Fast division

$$f = qg + r, \deg g = n, \deg q, \deg r < n$$

1) Compute $(g^*)^{-1} \bmod x^n$

2) Compute $q^* \equiv f^* \cdot (g^*)^{-1} \bmod x^n$

3) $r \leftarrow f - qg$

$5M(n) + 2n$ operations in R

(COOK, SIEVEKING, STRASSEN,
KUNG, BORODIN & MOENCK)

43

Pseudo-division

What if g_n is not invertible?

$$g_n^{m+1} f = qg + r, \deg f = m + n, \deg g = n, \deg r < n$$

$$r \leftarrow f, q \leftarrow 0$$

for $i = m, m - 1, \dots, 0$ **do**

$$q \leftarrow g_n q + r_{i+n} x^i$$

$$r \leftarrow g_n r - r_{i+n} x^i \cdot g \quad \{ g_n^{m-i+1} f = qg + r \}$$

$(m + 1)(m + 3n)$ operations in R

44

Fast pseudo-division I

$$uv \equiv u_0^n \pmod{x^n}, \quad n = 2^k$$

$$v_0 \leftarrow 1$$

for $i = 0, \dots, k - 1$ **do**

$$\{ uv_i \equiv u_0^{2^i} \pmod{x^{2^i}} \}$$

$$v_{i+1} \leftarrow 2u_0^{2^i} v_i - uv_i^2 \pmod{x^{2^{i+1}}}$$

$$\text{Cost: } 3M(n) + O(n)$$

45

Fast pseudo-division II

$$g_n^n f = qg + r, \quad \deg g = n, \quad \deg q, \deg r < n$$

- 1) Compute v with $g^* v \equiv g_n^n \pmod{x^n}$
- 2) Compute $q^* \equiv f^* v \pmod{x^n}$
- 3) $r \leftarrow g_n^n f - qg$

$$5M(n) + O(n) \text{ operations in } R$$

46

Modular algorithms

47

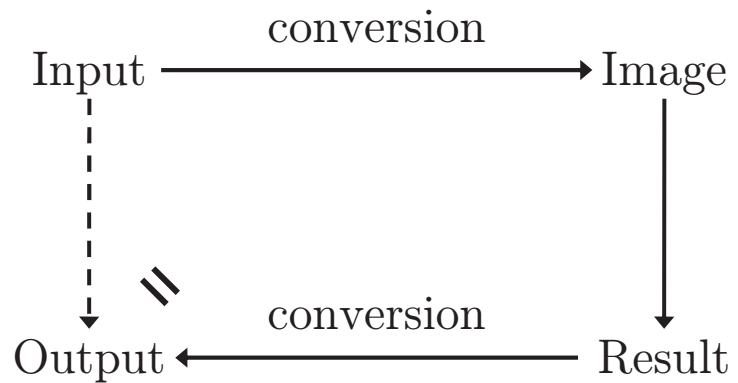
Contents

- Change of representation
- The Euclidean algorithm
- Big prime modular algorithm
- Evaluation and interpolation
- Small primes modular algorithms
- The extended Euclidean algorithm
- The Chinese remainder algorithm
- Modular gcd algorithm
- Prime power modular algorithms

48

Change of representation I

General idea:



- Problem is “hard” in the input domain and “easy” in the image domain
- Conversions are “cheap”

49

Change of representation II

Examples:

1) Polynomial multiplication

- input domain = $R[x]$
- conversion = FFT
- image domain = R^n (pointwise multiplication)

2) Polynomial gcd

- input domain = $\mathbb{Z}[x]$
- forward conversion = mod p
- image domain = $\mathbb{F}_p[x]$

50

The Euclidean algorithm I

$f, g \in R[x] \setminus \{0\}$, $m = \deg g \leq \deg f = n$

Compute monic $h = \gcd(f, g)$

$r_0 \leftarrow f$, $r_1 \leftarrow g$

for $i = 1, 2, \dots, \ell$ **do**

 let $r_{i-1} = q_i r_i + r_{i+1}$ such that either

$i < \ell$, $r_{i+1} \neq 0$, and $\deg r_{i+1} < \deg r_i$, or

$i = \ell$ and $r_{i+1} = 0$

return $\frac{r_\ell}{\text{lcoeff}(r_\ell)}$

Remark:

- $\text{lcoeff}(r_i)$ must be invertible, or at least not a zero divisor
- In the latter case, the coefficients of q_i, r_i may be in $\text{Quot}(R)$

51

The Euclidean algorithm II

$n_i = \deg r_i$ ($0 \leq i \leq \ell$)

$n = n_0 \geq m = n_1 > n_2 > \dots > n_\ell \geq 0$

$\deg q_i = n_{i-1} - n_i \geq 1$ ($1 < i \leq \ell$)

$$\sum_{1 < i \leq \ell} (n_{i-1} - n_i + 1)(2n_i + 1)$$

$$\leq \sum_{1 < i \leq \ell} 2(n_{i-1} - n_i)(2n_i + 1) = 2 \sum_{1 < i \leq \ell} \sum_{n_i \leq j < n_{i-1}} (2n_i + 1)$$

$$\leq 2 \sum_{1 < i \leq \ell} \sum_{n_i \leq j < n_{i-1}} (2j + 1) \leq 2 \sum_{0 \leq j < n_1} (2j + 1) = 2m^2$$

$$\text{Cost} \leq 2m^2 + (n - m + 1)(2m + 1) + (n_\ell + 1)$$

$$\leq 2nm + n + 2m + 2 = 2nm + O(n)$$

52

The Euclidean algorithm III

$f, g \in \mathbb{Z}[x]$, $\deg f, \deg g \leq n$, $\|f\|_1, \|g\|_1 < 2^b$

Compute *primitive* $h = \gcd(f, g) \in \mathbb{Z}[x]$

Issue:

coefficient growth in Euclidean algorithm: exponential for the naive algorithm, $O(nb)$ for a fraction-free version known as the subresultant algorithm (Collins)

But: coefficients of h are $O(n + b)$ (Mignotte) and in \mathbb{Z} !

53

A big prime modular algorithm

$f, g \in \mathbb{Z}[x]$, $\deg f, \deg g \leq n$, $\|f\|_2, \|g\|_2 < 2^b$

Compute $h = \gcd(f, g) \in \mathbb{Z}[x]$

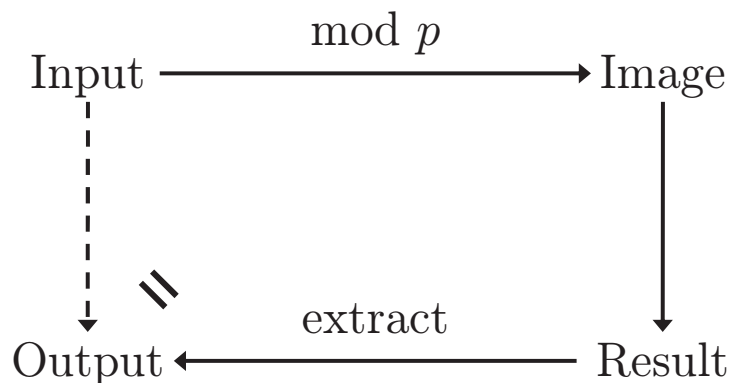
- Choose prime $2^{n+b+1} < p < 2^{n+b+2}$
- Compute monic $h_p = \gcd(f_p, g_p)$ in $\mathbb{F}_p[x]$
- $a \leftarrow \gcd(\text{lcoeff}(f), \text{lcoeff}(g))$
- Compute $a \cdot h_p$ in $\mathbb{F}_p[x]$

Cost: (assuming classical arithmetic and $O(n^2 + b^2)$ word operations for one operation in \mathbb{F}_p)

$O(n^4 + n^2b^2)$ \longleftrightarrow $O(n^4b^2)$ fraction-free

54

Big prime modular algorithm



Conversions are essentially free

Alternative approach for gcd: $p = x - 2^{n+b+2}$

(Kronecker substitution / heuristic gcd)

reduces polynomial gcd to integer gcd

55

Evaluation and interpolation

Instead of one “big” primes, use many “small” primes

One arithmetic operation takes constant time

Example (cont’d): polynomial multiplication

$\deg f + \deg g < n$, ω primitive n th root of unity

$$p_i = x - \omega^i \quad (0 \leq i < n)$$

Forward conversion = *multipoint evaluation*:

$$f \bmod p_i = f(\omega^i) \quad (\text{Exercise: why?}) \quad \forall i$$

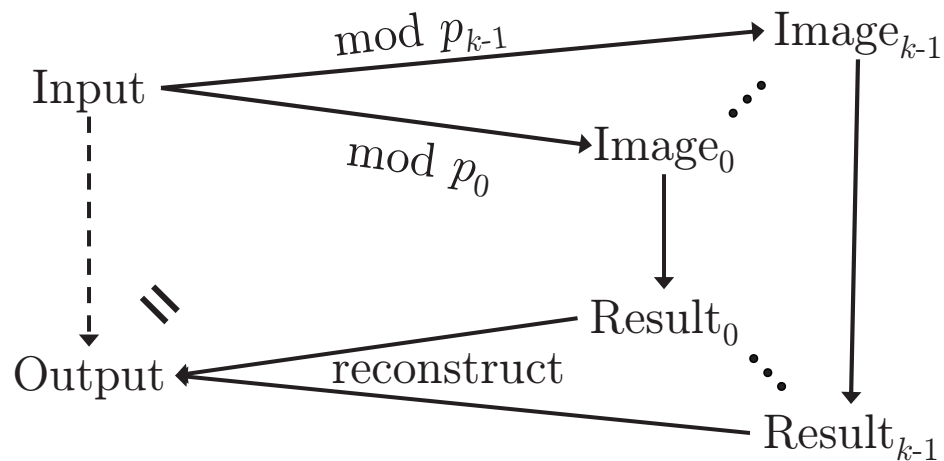
$$\text{Pointwise multiplication: } (fg)(\omega^i) = f(\omega^i) \cdot g(\omega^i) \quad \forall i$$

Backward conversion = *interpolation*:

$$\text{find } h \text{ (deg } h < n\text{): } h \bmod p_i = h(\omega^i) = (fg)(\omega_i) \quad \forall i$$

56

Small primes modular algorithms



$p_i = x - u_i$: reconstruct = interpolate

What if $\deg p_i > 1$, or if $p_i \in \mathbb{Z}$?

57

Extended Euclidean algorithm I

$f, g \in R[x] \setminus \{0\}$, $m = \deg g \leq \deg f = n$

Compute monic $h = \gcd(f, g) = sf + tg$

$r_0 \leftarrow f, s_0 \leftarrow 1, t_0 \leftarrow 0$

$r_1 \leftarrow g, s_1 \leftarrow 0, t_1 \leftarrow 1$

for $i = 1, 2, \dots, \ell$ **do**

$$r_{i-1} = q_i r_i + r_{i+1}$$

let $s_{i-1} = q_i s_i + s_{i+1}$ such that either

$$t_{i-1} = q_i t_i + t_{i+1}$$

$i < \ell$, $r_{i+1} \neq 0$, and $\deg r_{i+1} < \deg r_i$

or $i = \ell$ and $r_{i+1} = 0$

return $\frac{r_\ell}{\text{lcoeff}(r_\ell)}, \frac{s_\ell}{\text{lcoeff}(r_\ell)}, \frac{t_\ell}{\text{lcoeff}(r_\ell)}$

58

Extended Euclidean algorithm I

$f, g \in R[x] \setminus \{0\}$, $m = \deg g \leq \deg f = n$ ($0 < g \leq f \in \mathbb{N}$)
 Compute monic $h = \gcd(f, g) = sf + tg$ (positive if $f, g \in \mathbb{N}$)

$r_0 \leftarrow f$, $s_0 \leftarrow 1$, $t_0 \leftarrow 0$

$r_1 \leftarrow g$, $s_1 \leftarrow 0$, $t_1 \leftarrow 1$

for $i = 1, 2, \dots, \ell$ **do**

$$r_{i-1} = q_i r_i + r_{i+1}$$

let $s_{i-1} = q_i s_i + s_{i+1}$ such that either

$$t_{i-1} = q_i t_i + t_{i+1}$$

$i < \ell$, $r_{i+1} \neq 0$, and $\deg r_{i+1} < \deg r_i$ ($0 < r_{i+1} < r_i$ if $f, g \in \mathbb{N}$)

or $i = \ell$ and $r_{i+1} = 0$

return $\frac{r_\ell}{\text{lcoeff}(r_\ell)}$, $\frac{s_\ell}{\text{lcoeff}(r_\ell)}$, $\frac{t_\ell}{\text{lcoeff}(r_\ell)}$ ($\text{lcoeff}(r_\ell) = 1$ if $f, g \in \mathbb{N}$)

59

Extended Euclidean algorithm II

$f, g \in R[x] \setminus \{0\}$, $m = \deg g \leq \deg f = n$, $\ell \leq m + 1$

$\deg r_i = n_i$, $\deg q_i = n_{i-1} - n_i$,

$\deg s_i = n_1 - n_{i-1}$, $\deg t_i = n_0 - n_{i-1}$

Cost to compute all s_i :

$$\begin{aligned} & \sum_{1 < i \leq \ell} 2(n_{i-1} - n_i)(n_1 - n_{i-1}) + 2(n_1 - n_i + 1) \\ & \leq 4 \sum_{1 < i \leq \ell} n_1 - n_i + 1 \leq 4 \sum_{1 < i \leq \ell} i < 2\ell(\ell - 1) \leq 2m^2 + 2m \end{aligned}$$

Cost to compute all t_i : $4nm - 2m^2 + 4n - 2m$

60

Extended Euclidean algorithm III

Cost for EEA:

$$6nm + 5n + 2m + 2 = 6nm + O(n)$$

Cost for EEA without computing the t_i :

$$2nm + 2m^2 + n + 4m + 2 = 2nm + 2m^2 + O(n)$$

Cost for EEA in \mathbb{Z} :

$$O(nm)$$

61

Extended Euclidean algorithm IV

Corollary 1 $m = \deg g \leq \deg f = n$, $\gcd(f, g) = 1$

Then $\exists s, t$ with $\mathbf{1 = sf + tg}$, $\deg s < m$, $\deg t < n$

Cost: $2nm + O(n)$ for s and $6mn + O(n)$ for s, t

Corollary 2 f, g as before, $\deg h < n$

Then $\exists s^*, t^*$ with $\mathbf{h = s^*f + t^*g}$, $\deg s^* < m$, $\deg t^* < n$

Proof: $ht = qf + t^*$, $s^* = hs - qg$

Cost: $10mn + 4m^2 + O(n)$ for s^*, t^*

62

The Chinese remainder algorithm I

n pairwise coprime moduli $p_0, \dots, p_{k-1} \in \mathbb{N} \setminus \{0\}$

Given $h_0, \dots, h_{k-1} \in \mathbb{N}$, $0 \leq h_i < p_i$,

find $h \in \mathbb{N}$, $0 \leq h < m = p_0 \cdots p_{k-1}$

$$h \equiv h_i \pmod{p_i} \quad (0 \leq i < k)$$

for $i = 0, \dots, k - 1$ **do**

$$m_i \leftarrow \frac{m}{p_i}$$

compute $s_i \in \mathbb{Z}$ with $1 = s_i m_i + t_i p_i$

$$c_i \leftarrow h_i s_i \pmod{p_i}$$

return $\left(\sum_{0 \leq i < k} c_i m_i \right) \pmod{m}$

63

The Chinese remainder algorithm I

n pairwise coprime moduli $p_0, \dots, p_{k-1} \in \mathbb{N} \setminus \{0\}$ $(R[x] \setminus \{0\})$

Given $h_0, \dots, h_{k-1} \in \mathbb{N}$, $0 \leq h_i < p_i$, $(\deg h_i < \deg p_i)$

find $h \in \mathbb{N}$, $0 \leq h < m = p_0 \cdots p_{k-1}$ $(\deg h < \deg m)$

$$h \equiv h_i \pmod{p_i} \quad (0 \leq i < k)$$

for $i = 0, \dots, k - 1$ **do**

$$m_i \leftarrow \frac{m}{p_i}$$

compute $s_i \in \mathbb{Z}$ with $1 = s_i m_i + t_i p_i$ $(s_i \in R[x])$

$$c_i \leftarrow h_i s_i \pmod{p_i}$$

return $\left(\sum_{0 \leq i < k} c_i m_i \right) \pmod{m}$

64

The Chinese remainder algorithm II

Cost in the polynomial case (classical arithmetic):

Assuming $\deg p_i = b$, p_i monic, $\deg m = kb = n$

Computing $p_0p_1, p_0p_1p_2, \dots, p_0 \cdots p_{k-1} = m$:

$$\sum_{0 \leq i < k} 2ib^2 = (k-1)(k-2)b^2$$

Computing all m/p_i : $(k-1) \cdot 2b \cdot ((k-1)b+1)$

Computing all s_i : $k \cdot (2kb^2 + (k-1)b + 4b + 2)$

Computing all c_i : $k \cdot (b^2 + (b-1)^2 + (b-1) \cdot 2b)$

Computing all $c_i m_i$: $k \cdot ((k-1)b^2 + (k-1)b(b-1))$

Adding the results: $(k-1)kb$

65

The Chinese remainder algorithm III

Cost in the polynomial case (classical arithmetic, assuming $b \geq 1$ and $k \geq 2$):

$$(7b^2 + b)k^2 + (-5b^2 + b + 3)k + 4b^2 - 2b \leq 8n^2$$

Cost in the integer case (classical arithmetic, primes of the same word length b):

$$O(n^2)$$

66

The Chinese remainder algorithm IV

Linear polynomials $p_i = x - u_i$, $m = p_0 \cdots p_{n-1}$

Given $h_0, \dots, h_{n-1} \in R$,

find $h \in R$, $\deg h < n = \deg m$

$$h(u_i) = h_i \quad (0 \leq i < k)$$

for $i = 0, \dots, k - 1$ **do**

$$m_i \leftarrow \frac{m}{x - u_i} = \prod_{j \neq i} (x - u_j)$$

$$\mathbf{return} \sum_{0 \leq i < k} h_i \cdot \frac{1}{m_i(u_i)} \cdot m_i$$

(LAGRANGE)

67

Modular gcd algorithm I

$f, g \in \mathbb{Z}[x]$, $\deg f, \deg g \leq n$, $\|f\|_2, \|g\|_2 < 2^b$

Compute *primitive* $h = \gcd(f, g) \in \mathbb{Z}[x]$

choose single precision primes $p_0, \dots, p_{k-1} \in \mathbb{N}$

$a \leftarrow \gcd(\text{lcoeff}(f), \text{lcoeff}(g))$

for $i = 0, \dots, k - 1$ **do**

$f_i \leftarrow f \bmod p$, $g_i \leftarrow g \bmod p$

compute monic $\gcd(f_i, g_i)$ in $\mathbb{F}_p[x]$

$h_i \leftarrow a \cdot \gcd(f_i, g_i)$

compute $h \in \mathbb{Z}[x]$ with $h \equiv h_i \bmod p_i$ ($0 \leq i < k$)

compute $c = \gcd(\text{coefficients of } h)$ and **return** $\frac{h}{c}$

68

Modular gcd algorithm II

Cost:

Computing c	$O(b^2 + \log^2 n)$
Reducing f, g mod all p_i	$k \cdot O(n(b + \log n))$
Computing the gcd mod all p_i	$k \cdot O(n^2)$
Computing h_i mod all p_i	$k \cdot O(n)$
Chinese remainder algorithm	$n \cdot O(k^2)$
Making h primitive	$O(nk^2)$

In total $O(kn(n + b) + nk^2 + b^2)$

69

Modular gcd algorithm - the fine print

- We ignore the cost for finding the primes, which can be shown to be negligible. In practice, often a precomputed list of primes is used.
- How many primes do we need?
Mignotte's bound: $\|h\|_\infty \leq 2^{n+b}$, so $k = O(n + b)$
- It is customary to normalize the primitive gcd to have positive leading coefficient.

Cost: $O(n^3 + nb^2) \longleftrightarrow O(n^4 + n^2b^2)$ big prime

(COLLINS, BROWN)

70

The good, the bad, and the ugly

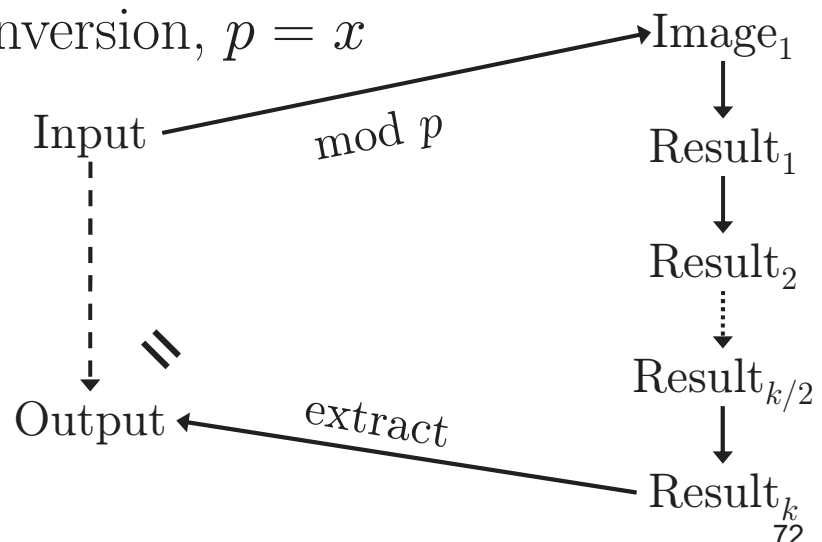
- Not every prime is good! Example:
 $\gcd(x^2 - 4, x + 1) = 1$ in $\mathbb{Z}[x]$, but $x + 1$ in $\mathbb{F}_3[x]$
However, there are only $O(nb)$ many bad primes
(subresultant theorem + Hadamard inequality).
- There are only finitely many single precision primes.
In practice, sufficiently many 64 bit primes exist.
In general, take primes of word length $O(\log(nb))$
(prime number theorem).
- The algorithm is probabilistic (Monte Carlo). It can be made Las Vegas by also computing f_i/h_i and g_i/h_i and performing a divisibility check at the end. This does not change the $O()$ estimate.

71

Prime power modular algorithms

Idea: Instead of one big prime or many small primes, use just *one* small prime p and then “lift” the result modulo p^2, p^4, \dots, p^k (Hensel lifting)

Example: Newton inversion, $p = x$



72

Advanced polynomial arithmetic

73

Contents

- Divide and conquer
- Subproduct trees
- Fast evaluation and interpolation
- Fast Chinese remainder algorithm
- The Half-gcd algorithm
- Fast modular gcd algorithm
- Squarefree factorization
- Full factorization

74

Divide and conquer

Well-known computer science paradigm

(example: Quicksort). Idea:

- Split original problem into r subproblems of the same type and of size n/s
- Solve the subproblems recursively
- Recombine the solutions

Typically, $s=2$ and the cost for splitting and recombining is $O(n)$.

Total cost: $O(n \log n)$ if $r = k$ and $O(n^{\log_k r})$ if $r > k$.

Computer algebra examples: Karatsuba ($r=3, s=2$),
Toom-Cook ($r=2s-1$), FFT ($r=s=2$)

75

Polynomial multiplication revisited

Given k moduli p_0, p_1, \dots, p_{k-1} , compute $m = p_0 \cdots p_{k-1}$

Divide and conquer:

Recursively compute $m_0 \leftarrow p_0 \cdots p_{k/2-1}$

and $m_1 \leftarrow p_{k/2} \cdots p_{k-1}$

$m \leftarrow m_0 \cdot m_1$

$M(n)$ multiplication time for polynomials of degree $\leq n$

Assuming $\deg p_i = b \forall i$ and superlinearity $M(\frac{1}{2}t) \leq \frac{1}{2}M(t)$:

$$S(n) = 2S(\frac{1}{2}n) + M(\frac{1}{2}n) \leq \frac{1}{2}M(n) \log_2 k$$

Naive approach $((p_0 p_1) p_2 \cdots) p_{k-1}$: - (n^2) ,

even with fast multiplication!

76

Exercise 6

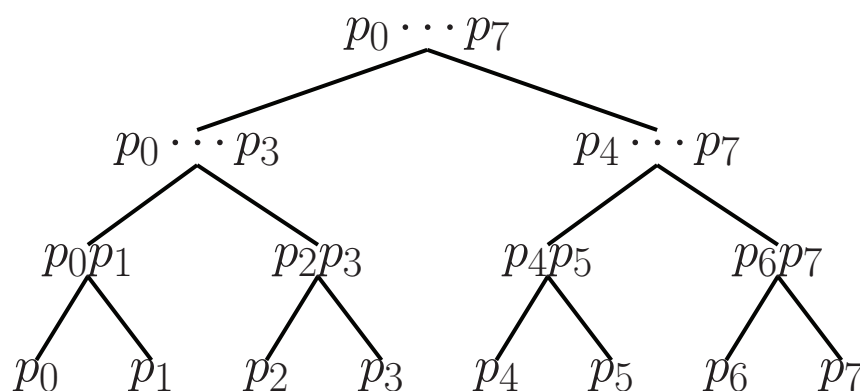
Given n monic linear moduli $p_i = x - u_i$ ($0 \leq i < n$), compute $m = (\cdots (p_0 p_1) p_2 \cdots) p_{n-1}$.

Show that this takes $n^2 - n$ additions and multiplications using classical arithmetic.

77

Subproduct tree

Certain subproducts of $p_0 \cdots p_{k-1}$ are computed along the way, yielding the following tree structure:



$m_{ij} := p_{j \cdot 2^i} \cdots p_{(j+1) \cdot 2^i - 1}$, the j th subproduct at level i
 $m_{0i} = p_i$, $m_{\ell 0} = m$, $\ell = \log_2 k$

79

Going down the subproduct tree

$u_0, \dots, u_{n-1} \in R, \deg f < n = 2^\nu$

compute $f(u_0), \dots, f(u_{n-1})$

Precomputation

0) Build the subproduct tree for $p_i = x - u_i$ ($0 \leq i < n$)

Divide & conquer

1) **if** $\ell = 0$ **then return** f

2) Compute $f_0 = f \bmod m_{\ell-1,0}$ and $f_1 = f \bmod m_{\ell-1,1}$

3) Recursively evaluate f_0 at $u_0, \dots, u_{n/2-1}$ and

f_1 at $u_{n/2}, \dots, u_{n-1}$

Proof: $f(u_i) = q(u_i)m_{\ell-1,0}(u_i) + f_0(u_i) = f_0(u_i)$ ($i < \frac{1}{2}n$)

80

Fast multipoint evaluation

Cost: $\deg f < n = 2^\nu$, assuming superlinearity of M

$$\begin{aligned} E(n) &= 2E\left(\frac{1}{2}n\right) + 10M\left(\frac{1}{2}n\right) + 2n \\ &\leq 5M(n) \log_2 n + 2n \log_2 n \end{aligned}$$

plus $\frac{1}{2}M(n) \log_2 n$ for the precomputation, in total

$$\frac{11}{2}M(n) \log_2 n + O(n \log n)$$

Compare with classical arithmetic (Horner): $2n^2 - 2n$

81

Fast multimodular reduction

$p_0, \dots, p_{k-1} \in R$, $\deg p_i = b$, $\deg f < n = bk = b2^\nu$
compute $f \bmod p_0, \dots, f \bmod p_{n-1}$

Precomputation

0) Build the subproduct tree p_i ($0 \leq i < k$)

Divide & conquer

1) **if** $\ell = 0$ **then return** f

2) Compute $f_0 = f \bmod m_{\ell-1,0}$ and $f_1 = f \bmod m_{\ell-1,1}$

3) Recursively compute $f_0 \bmod p_0, \dots, p_{n/2-1}$ and

$f_1 \bmod p_{n/2}, \dots, p_{n-1}$

Cost: $\frac{11}{2}M(n) \log_2 k + O(n \log k)$ (classical: $- (n^2)$)

82

Exercise 7

Devise a recursion for the FFT using the subproduct tree. Note:

$$m_{\ell-1,0} = \prod_{0 \leq i < n/2} (x - \omega^{2i}) = x^{n/2} - 1$$

$$m_{\ell-1,1} = \prod_{0 \leq i < n/2} (x - \omega^{2i+1}) = x^{n/2} + 1$$

83

Fast interpolation

$n = 2^\nu$, $u_0, \dots, u_{n-1} \in R$, $h_0, \dots, h_{n-1} \in R$

compute $h \in R[x]$, $\deg h < n$, with $h(u_i) = h_i$ ($0 \leq i < n$)

Cost: $\frac{13}{2}M(n) \log_2 n + O(n \log n)$ (classical: - (n^2))

“going up the subproduct tree”

(Section 10.2 in MCA)

85

Fast Chinese remaindering

$n = kb = 2^\nu b$, $p_0, \dots, p_{k-1} \in R[x]$, $h_0, \dots, h_{k-1} \in R[x]$

$\deg h_i < \deg p_i = b$

compute $h \in R[x]$, $\deg h < n$, with $h \bmod p_i = h_i$ ($0 \leq i < n$)

Cost: $O(M(n) \log k)$ (classical: - (n^2))

(Section 10.3 in MCA)

Remark Same estimate is valid for the unbalanced case (where not all p_i have the same degree) of fast multimodular reduction and fast Chinese remaindering

(LIPSON, FIDUCCIA, HOROWITZ, BORODIN & MOENCK)

86

Euclidean algorithm revisited I

$$r_{i-1} = q_i r_i + r_{i+1} \quad (1 \leq i \leq \ell)$$

In matrix form:

$$\begin{aligned} \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix} &= \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} \\ \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} &= \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \cdot \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} \cdot \begin{pmatrix} r_0 \\ r_1 \end{pmatrix} \\ &= \begin{pmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{pmatrix} \cdot \begin{pmatrix} r_0 \\ r_1 \end{pmatrix} \end{aligned}$$

87

Euclidean algorithm revisited II

Observation: The coefficients of $q_1, \dots, q_i, s_i, t_i$ and the highest coefficients of r_i, r_{i+1} depend only on the highest coefficients of r_0, r_1 .

Simplifying **assumption**: $\deg q_i = 1$

$$(1 \leq i \leq \ell = \deg r_1 + 1)$$

For the general case, see Chapter 11 in MCA.

Notation: $f = f_n x^n + \cdots + f_0, k \in \mathbb{N}$

$$f \upharpoonright k = f_n x^k + \cdots + f_{n-k}$$

88

Half-gcd algorithm I

$\deg r_0 = n, \deg r_i = n - i (1 \leq i \leq n), r_{n+1} = 0$

Given $1 \leq k = 2^\kappa \leq n$, compute $\text{hgcd}(r_0, r_1, k) = s_k, t_k, s_{k+1}, t_{k+1}$

1) **if** $k = 1$ **then** compute q_1 and **return** $0, 1, 1, -q_1$

2) $d \leftarrow \frac{1}{2}k$

3) $s_d, t_d, s_{d+1}, t_{d+1} \leftarrow \text{hgcd}(r_0 \upharpoonright 2d, r_1 \upharpoonright 2d - 1, d)$

4) $\begin{pmatrix} R_0 \\ R_1 \end{pmatrix} \leftarrow \begin{pmatrix} s_d & t_d \\ s_{d+1} & t_{d+1} \end{pmatrix} \cdot \begin{pmatrix} r_0 \upharpoonright \frac{3}{2}k \\ r_1 \upharpoonright \frac{3}{2}k - 1 \end{pmatrix} \bmod x^{k+1}$

8) $S_d, T_d, S_{d+1}, T_{d+1} \leftarrow \text{hgcd}(R_0, R_1, d)$

9) $\begin{pmatrix} s_k & t_k \\ s_{k+1} & t_{k+1} \end{pmatrix} \leftarrow \begin{pmatrix} S_d & T_d \\ S_{d+1} & T_{d+1} \end{pmatrix} \cdot \begin{pmatrix} s_d & t_d \\ s_{d+1} & t_{d+1} \end{pmatrix}$

(MOENCK, AHO, HOPCROFT & ULLMAN, SCHWARTZ, BRENT, GUSTAVSON & YUN, STRASSEN)

89

Half-gcd algorithm II

$\deg r_0 = n, \deg r_i = n - i (1 \leq i \leq n), r_{n+1} = 0$

Given $1 \leq k = 2^\kappa \leq n$, compute $\text{hgcd}(r_0, r_1, k) = s_k, t_k, s_{k+1}, t_{k+1}$

1) **if** $k = 1$ **then** compute q_1 and **return** $0, 1, 1, -q_1$ $O(1)$

2) $d \leftarrow \frac{1}{2}k$

3) $s_d, t_d, s_{d+1}, t_{d+1} \leftarrow \text{hgcd}(r_0 \upharpoonright 2d, r_1 \upharpoonright 2d - 1, d)$ $H(\frac{1}{2}k)$

4) $\begin{pmatrix} R_0 \\ R_1 \end{pmatrix} \leftarrow \begin{pmatrix} s_d & t_d \\ s_{d+1} & t_{d+1} \end{pmatrix} \cdot \begin{pmatrix} r_0 \upharpoonright \frac{3}{2}k \\ r_1 \upharpoonright \frac{3}{2}k - 1 \end{pmatrix} \bmod x^{k+1}$

$4M(k) + O(k)$

8) $S_d, T_d, S_{d+1}, T_{d+1} \leftarrow \text{hgcd}(R_0, R_1, d)$

$H(\frac{1}{2}k)$

9) $\begin{pmatrix} s_k & t_k \\ s_{k+1} & t_{k+1} \end{pmatrix} \leftarrow \begin{pmatrix} S_d & T_d \\ S_{d+1} & T_{d+1} \end{pmatrix} \cdot \begin{pmatrix} s_d & t_d \\ s_{d+1} & t_{d+1} \end{pmatrix}$

$8M(\frac{1}{2}k) + O(k)$

Cost (assuming superlinearity of M): $H(k) \leq 8M(k) \log_2 k + O(k \log k)$

90

Fast integer arithmetic

- multiplication, division: $O(M(n))$
- multimodular reduction, Chinese remaindering, (extended) Euclidean algorithm, modular inversion: $O(M(n) \log n)$
- All of the above take - (n^2) with classical arithmetic

91

Fast modular gcd algorithm

$f, g \in \mathbb{Z}[x]$, $\deg f, \deg g \leq n$, $\|f\|_2, \|g\|_2 < 2^b$

Compute *primitive* $h = \gcd(f, g) \in \mathbb{Z}[x]$

using $k = O(n + b)$ small primes $p_i \in \mathbb{N}$

Dominant costs:

- $\gcd(f \bmod p_i, g \bmod p_i) \forall i: k \cdot O(M(n) \log n)$
using fast polynomial arithmetic
- Chinese remaindering: $n \cdot O(M(k) \log k)$
using fast integer arithmetic

Total: $O(n \cdot M(n + b) \log(n + b)) = O^\sim(n^2 + nb)$

$\longleftrightarrow O(n^3 + nb^2)$ with classical arithmetic

92

Comparison of gcd algorithms

$f, g \in \mathbb{Z}[x]$, $\deg f, \deg g \leq n$, $\|f\|_2, \|g\|_2 < 2^b$

classical	$O(4^n nb^2)$
subresultant	$O(n^4 b^2)$
big prime	$O(n^4 + n^2 b^2)$
small primes	$O(n^3 + nb^2)$
fast	$O^\sim(n^2 + nb)$
output size	$O(n^2 + nb)$

93

Squarefree factorization

$f \in R[x]$, $\deg f = n < \text{char} R$

Compute $f = f_1 f_2^2 \cdots f_n^n$ with all f_i squarefree, i.e., $\gcd(f_i, f_i') = 1$, and pairwise coprime.

Let $v = f_1 f_2 \cdots f_n = \frac{f}{\gcd(f, f')}$, and observe:

$$\begin{aligned} \frac{v'}{v} &= \sum_{1 \leq i \leq n} \frac{f_i'}{f_i} & \frac{f'}{f} &= \sum_{1 \leq i \leq n} i \frac{f_i'}{f_i} \\ v' &= \sum_{1 \leq i \leq n} f_i' \frac{v}{f_i} & \frac{f'v}{f} &= \sum_{1 \leq i \leq n} i f_i' \frac{v}{f_i} \end{aligned}$$

So $f_i = \gcd\left(v, \frac{f'v}{f} - iv'\right)$

94

Yun's algorithm I

$$f = f_1 f_2^2 \cdots f_n^n, \deg f = n < \text{char} R$$

$$v_1 \leftarrow \frac{f}{\gcd(f, f')}, \quad w_1 \leftarrow \frac{f'}{\gcd(f, f')}$$

for $i = 1, \dots, n$ **do**

$$f_i \leftarrow \gcd(v_i, w_i - v_i'), \quad v_{i+1} \leftarrow \frac{v_i}{f_i}, \quad w_{i+1} \leftarrow \frac{w_i - v_i'}{f_i}$$

$$\text{Invariants: } v_i = \prod_{i \leq j \leq n} f_j, \quad w_i = \sum_{i \leq j \leq n} (j - i) f_j' \frac{v_i}{f_j}$$

95

Yun's algorithm II

Cost (assuming superlinearity $H(t + u) \geq H(t) + H(u)$)

$$d_i = \deg f_i, \quad e_i = \deg g_i = d_i + \cdots + d_n, \quad n = e_1 + \cdots + e_n$$

$$\begin{aligned} Y(n) &= H(n) + \sum_{1 \leq i \leq n} H(e_i) \leq H(n) + H(e_1 + \cdots + e_n) \\ &\leq 2H(n) = O(M(n) \log n) \end{aligned}$$

Remark: No polynomial-time algorithm for squarefree factorization of integers is known.

96

Modular squarefree factorization I

$f \in \mathbb{Z}[x]$ primitive, $\deg f \leq n$, $\|f\|_2 < 2^b$

Compute *primitive* squarefree decomposition $f = f_1 f_2^2 \cdots f_n^n$

choose single precision primes $p_0, \dots, p_{k-1} \in \mathbb{N}$, $p_i > n$

$a \leftarrow \text{lcoeff}(f)$

for $i = 0, \dots, k - 1$ **do**

$g_i \leftarrow f \bmod p$

 compute monic squarefree decomposition

$g_i = a g_{i1} g_{i2}^2 \cdots g_{in}^n$ in $\mathbb{F}_p[x]$

$h_{ij} \leftarrow a \cdot g_{ij}$ ($1 \leq j \leq n$)

for $j = 1, \dots, n$ **do**

 compute $h_j \in \mathbb{Z}[x]$ with $h_j \equiv h_{ij} \bmod p_i$ ($0 \leq i < k$)

 compute $c_j = \gcd(\text{coefficients of } h_j)$ and $f_j \leftarrow \frac{h_j}{c_j}$

97

Modular squarefree factorization II

$f \in \mathbb{Z}[x]$ primitive, $\deg f \leq n$, $\|f\|_2 < 2^b$

- Dominant cost: $k \cdot O(M(n) \log n)$ for the modular computation, $n \cdot O(M(k) \log k)$ for the Chinese remaindering
- Mignotte's bound: $\|f_i\|_\infty \leq 2^{n+b}$, so $k = O(n + b)$
Total cost $O(n \cdot M(n + b) \log(n + b))$ or $O^\sim(n^2 + nb)$
- There are $O(n^2 + nb)$ bad primes

As for the modular gcd algorithm:

- We neglect the cost for prime finding and assume there are sufficiently many.
- It is customary to assume that f and all f_i have positive leading coefficients.
- The algorithm is probabilistic, of Monte Carlo type, and can be made Las Vegas, within the same $O()$ estimate.

(G)

98

Full factorization

- $\mathbb{F}_p[x]$: distinct-degree factorization + equal-degree factorization (probabilistic); $O(n \cdot M(n) \log(pn))$ or better
See Chapter 14 in MCA
- $\mathbb{Q}[x]$: prime power modular (Hensel lifting) reduction to $\mathbb{F}_p[x]$ + “short vectors”
See Chapters 15-16 in MCA
- \mathbb{Z} : no polynomial time algorithm known

Optimizations

Contents

- What if n is not a power of 2?
- Short products
- Balancing
- Saving changes of representation
- Multi-algorithms
- Schönhage's golden rules

101

To pad or not to pad?

Example: $f \cdot g$ where $\deg f, \deg g < n < 2^\nu$

- Option 1: Pad inputs with 0s (trivial but inefficient)
$$F = 0 \cdot x^{2^\nu-1} + \dots + 0 \cdot x^n + f_{n-1}x^{n-1} + \dots + f_0$$
- Option 2: Extend algorithm design to work for odd n (often tricky)

102

Not to pad!

Example: $f \cdot g$ where $\deg f, \deg g < n < 2^\nu$

Divide & conquer (e.g., Karatsuba): split inputs into parts of size $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$, respectively

FFT:

- DFT of order $n = km$ “=” k DFTs of order m followed by m DFTs of order k
- Truncated FFT and inverse FFT (van der Hoeven, Harvey & Roche)

103

Short products

Sometimes only the upper or the lower coefficients are needed.

Example: $f \cdot g \bmod x^n$

- Option 1: Discard coefficients. Trivial but inefficient:
 $M(n)$

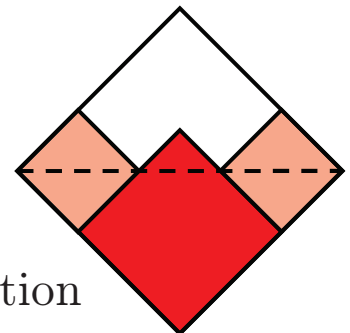
- Option 2: $S(n) = M(k) + 2S(n - k)$

$$M(n) = n^2: k = \frac{1}{2} \text{ and } S(n) = \frac{1}{2}M(n)$$

$$M(n) = n^{\log_2 3}: k = 2^{\lfloor \log_2 n \rfloor}$$

$$n = \frac{1}{3}(4^\nu - 1): S(n) \approx \frac{3}{5}M(n)$$

No gain for $n = 2^\nu$ or for FFT multiplication



(Krandick & Johnson, Mulders, Hanrot & Zimmermann)

104

Balancing

Asymptotically fast algorithms work best for *balanced* inputs, of roughly the same size.

Example: multiplication, $\deg f < kn$, $\deg g < n$

Solution: Break f into k blocks of degree $< n$

Cost: $k \cdot M(n) + k(n - 1)$

This leaves the range $n \leq \deg f < 2n$.

Example: $\deg f = n$, $\deg g = n - 1$

Cost: $M(n) + 2n + 1$

105

Saving changes of representation I

Example: Newton inversion revisited

$u^{-1} \bmod x^n$, $n = 2^k$

$v_0 \leftarrow u_0^{-1}$

for $i = 0, \dots, k - 1$ **do**

$v_{i+1} \leftarrow 2v_i - uv_i^2 \bmod x^{2^{i+1}}$

Cost $\approx 3M(n)$, actually $\approx 8F(2n)$

Let's do as much as possible by values.

106

Saving changes of representation II

ω primitive $2n$ -th root of unity

precompute ω^j and $u(\omega^j)$ for $0 \leq j < 2n$

$v_0 \leftarrow u_0^{-1}$

for $i = 0, \dots, k - 1$ **do**

$\eta \leftarrow \omega^{2^{k-i-1}}$

 compute $v_i(1), v_i(\eta), \dots, v_i(\eta^{2^{i+2}})$

for $j = 0, \dots, 2^{i+2} - 1$ **do** $w_{i+1}(\eta^j) \leftarrow -u(\eta^j)v_i(\eta^j)^2$

 interpolate w_{i+1} from all $w(\eta^j)$

$v_{i+1} \leftarrow 2v_i + w_{i+1} \bmod x^{2^{i+1}}$

Cost $\approx F(2n) + \sum_{0 \leq i < k} 2F(2^{i+2}) \leq 5F(2n) < 2M(n)$

107

Multi-algorithms

Use classical arithmetic for "small" inputs and fast arithmetic for "large" inputs

Example: multiplication, $\deg f, \deg g < n$

if $n < C_1$ **then return** classical(f, g)

elif $C_1 \leq n < C_2$ **then return** Karatsuba(f, g)

elif $n \geq C_2$ **then return** FFTmult(f, g)

Divide & conquer: Instead of

if $n = 1$ **then** ... **else** recurse

use

if $n < C_3$ **then** ... **else** recurse

"Magic constants" $C_1, C_2, C_3 \dots$ generally platform dependent

108

Golden rules

From Schönhage, Grotefeld & Vetter, Fast algorithms
– a multitape Turing machine implementation:

1. Do care about the size of $O(1)$!
2. Do not waste a factor of two!
3. Do trust the truth!
4. Do not raise the overall cost for speeding up rare cases - avoid lotteries!
5. Correctness implies termination in due time!
6. Don't forget the algorithms in object design!
7. Clean results by approximate methods is sometimes much faster!
8. The development of fast algorithms is slow!

109

Applications

110

Contents

- Hermite integration
- Modular Hermite integration
- Greatest factorial factorization
- Modular gff computation

111

Rational function integration

$r \in \mathbb{Q}(x)$, find $s, t \in \mathbb{Q}(x)$ with $\int r = s + \int t$ and h “minimal”
Equivalently: $r = s' + t$ and $\text{denom}(t)$ squarefree

Hermite's algorithm: $r = \frac{f}{g}$, $\deg f < \deg g = n$

0) Squarefree factorization $g = \prod_{1 \leq i \leq n} g_i^i$

1) Partial fraction decomposition $\frac{f}{g} = \sum_{1 \leq j \leq i \leq n} \frac{f_{ij}}{g_i^j}$

2) Extended Euclidean algorithm $f_{ij} = u g_i + v g_i'$

3) Integration by parts

$$\frac{f_{ij}}{g_i^j} = \frac{u}{g_i^{j-1}} + \frac{v g_i'}{g_i^j} = \frac{u + v'/(j-1)}{g_i^{j-1}} - \left(\frac{v}{(j-1)g_i^{j-1}} \right)'$$

112

Hermite integration

$$r = \frac{f}{g} = \frac{f}{\prod g_i^i}, \deg f < \deg g = n$$

$$\text{Hermite integration } \frac{f}{g} = \left(\sum_{1 \leq j < i \leq n} \frac{s_{ij}}{g_i^j} \right)' + \sum_{1 \leq i \leq n} \frac{t_i}{g_i}$$

$$\deg s_{ij}, \deg t_i < \deg g_i \quad \forall i, j$$

$$\text{Cost } O(M(n) \log n) \quad (O(n^2) \text{ classical})$$

113

Modular Hermite integration

$$f, g \in \mathbb{Z}[x], \deg f < \deg g = n, \|f\|_2, \|g\|_2 < 2^b$$

$$\text{Modular squarefree factorization } g = \prod_{1 \leq i \leq n} g_i^i$$

Choose single precision primes $p_0, \dots, p_{k-1} \in \mathbb{N}, p_q > n$

for $q = 0, \dots, k - 1$ **do**

$$\text{Hermite integration } \frac{f}{g} \equiv \left(\sum_{1 \leq j < i \leq n} \frac{s_{qij}}{g_i^j} \right)' + \sum_{1 \leq i \leq n} \frac{t_{qi}}{g_i} \pmod{p_q}$$

Chinese remaindering $s_{ij} \equiv s_{qij} \pmod{p_q}$ and $t_i \equiv t_{qi} \pmod{p_q}$

Cost: $O(k \cdot M(n) \log n + n \cdot M(k) \log k)$

Need $k = O(n^2 + nb) \longrightarrow O(n \cdot M(n^2 + nb) \log(n + b))$

or $O^\sim(n^3 + n^2b)$

(G)

114

Greatest factorial factorization I

$\deg g = n < \text{char} R, \quad i \in \mathbb{N}$

i th falling factorial $g^i(x) = g(x)g(x-1) \cdots g(x+i-1)$

greatest factorial factorization (gff): analog of squarefree factorization

$$g = \prod_{1 \leq i \leq n} g_i^i \text{ and}$$

$$\gcd(g_i^i(x), g_j(x+1)) = 1 = \gcd(g_i^i(x), g_j(x-j)) \quad (1 \leq i \leq j \leq n)$$

(Paule)

115

Greatest factorial factorization II

Fast gff computation a la Yun $g = \prod_{1 \leq i \leq n} g_i^i$

$$v_1(x) \leftarrow \frac{g(x)}{\gcd(g(x-1), g(x))}, \quad w_1(x) \leftarrow \frac{g(x)}{\gcd(g(x), g(x+1))}$$

for $i = 1, \dots, n$ **do**

$$g_i(x) \leftarrow \gcd(v_i(x), w_i(x)), \quad v_{i+1}(x) \leftarrow \frac{v_i(x)}{g_i(x)}, \quad w_{i+1}(x) \leftarrow \frac{w_i(x+1)}{g_i(x+1)}$$

$$\text{Invariants: } v_i(x) = \prod_{i \leq j \leq n} g_j(x), \quad w_i(x) = \sum_{i \leq j \leq n} g_j(x+i-j)$$

Cost: $O(M(n) \log n)$

(G)

116

Modular gff computation

$g \in \mathbb{Z}[x]$, $\deg g = n$, $\|g\|_2 < 2^b$

Choose single precision primes $p_0, \dots, p_{k-1} \in \mathbb{N}$, $p_q > n$

for $q = 0, \dots, k - 1$ **do**

 Compute gff $g \equiv \prod_{1 \leq i \leq n} g_{qi}^i \pmod{p_q}$

Chinese remaindering $g_i(x + j) \equiv g_{qi}(x + j) \pmod{p_q}$

Cost: $O(k \cdot M(n) \log n + n \cdot M(k) \log k)$

Need $k = O(n + b) \longrightarrow O(n \cdot M(n + b) \log(n + b))$

or $O^\sim(n^2 + nb)$

Need to choose among $O(n^2 + nb)$ primes

(G)