1

Algorithms
Professor John Reif

ALG 3.3

Newton Iteration and Polynomial Computation:

(a) Newton Iteration: application to division
(b) Polynomial Evaluation and Interpolation (Chinese Remaindering)

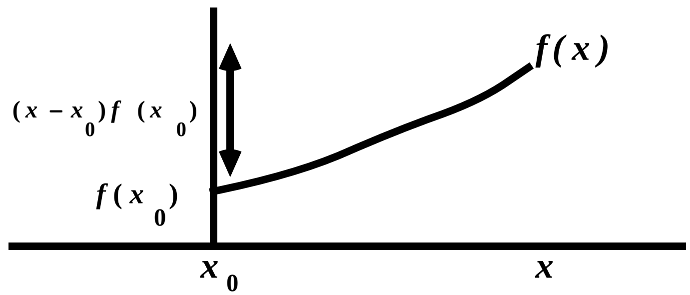
Reading Selection:

AHU-Data Chapter 8

2

Taylor Expansion

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2} f''(x_0) + \ldots$$



To find root of f(x),

use Newton iteration:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Example:

To find reciprocal of x

choose $f(y) = 1 - \frac{1}{x\,y}$ **find root** $y = \frac{1}{x}$

$$y_{i+1} = y_i - \frac{f(y_i)}{f'(y_i)} = y_i(2 - y_i x)$$

3

Application of Newton Iteration to *Reciprocal of an Integer*

input integer x, accuracy bound k

initialize $y_0 = 2^{-n}$ if x has n bits

for i=1 to k *do*

$$y_{i+1} \leftarrow y_i(2 - y_i x)$$

output y_k where $\left| 1 - y_k x \right| \le \frac{1}{2^{2^k}}$

proof let error $\varepsilon_k = 1 - y_k x$

then $\varepsilon_{k+1} = 1 - y_{k+1} x$

$$= 1 - x\, y_k (2 - y_k x)$$

$$= (\varepsilon_k)^2$$

$$= (\varepsilon_0)^{2^k}$$

$$= 2^{-2^k} \text{ since } \varepsilon_0 \le \frac{1}{2}$$

Theorem Integer Reciprocal can be computed to accuracy 2^{-n} in O(log n) integer mults and additions.

4

Steven Cook's Improvement (his Harvard Ph.D. thesis)

observe that since $\left| 1 - y_k x \right| \le \frac{1}{2^{2^k}}$ we need only compute y_k up to 2^{k+1} *bit accuracy.*

Total Time Cost

$$c \cdot \sum_{k=0}^{\log n} M(2^{k+1}) \le O(M(n))$$

where M(n) is time cost to multiply two n bit integers.

5

Other Applications of Newton Iteration on Integers

O(M(n)) time algorithms:

- **quotent + divisor of integer division**
- **square root**
- **sin, cosine, etc.**

used in practice!

6

Algorithm: **Reciprocal (P(x))**

input **polynomial** $P(x) = \sum_{i=0}^{n-1} a_i x^i$

degree n-1, n is power of 2

[1] ***if*** **n=1** ***then return*** $\frac{1}{a_0}$ **else**

[2] q(x) ← Reciprocal $(P_1(x))$

where $P_1(x) = \sum_{i=\frac{n}{2}}^{n-1} a_i x^{i-\frac{n}{2}}$

[3] $R(x) \leftarrow 2r_1(x)x^{(3/2)n-2} - r_1(x)^2 P(x)$

[4] ***return*** $\left\lfloor \frac{R(x)}{x^{n-2}} \right\rfloor$

7 ***Theorem:*** **The algorithm computes**

$$\text{reciprocal}\ (P(x)) = \left\lfloor \frac{x^{2n-2}}{P(x)} \right\rfloor$$

$= r(x)$ **where**

$$r(x) \cdot p(x) = x^{2n-2} + \varepsilon(x)$$

and $\varepsilon(x)$ **has degree** $< n-1$

proof **by induction**

basis $n=1 \Rightarrow P(x) = a_0$ *so* $r(x) = \dfrac{1}{a_0}$

inductive step **let** $P(x) = P_1(x)x^{\frac{n}{2}} + P_2(x)$

where $\deg(P_1) = \frac{n}{2} - 1$, $\deg(P_2) \leq \frac{n}{2} - 1$

By induction hypothesis, if

$r_1(x) = \text{reciprocal}\ (P_1(x))$ **then**

$$r_1(x) \cdot P_1(x) = x^{n-2} + \varepsilon_1(x)$$

where $\varepsilon_1(x)$ **has degree** $< \frac{n}{2} - 1$

8 **at line [3] we compute**

$$R(x) \leftarrow 2r_1(x)x^{(3/2)n-2} - r_1(x)^2 P(x)$$

since $P(x) = P_1(x)\, x^{\frac{n}{2}} + P_2(x)$

$$P(x)R(x) = 2r_1(x)P_1(x)x^{2n-2} + 2r_1(x)P_2(x)x^{\left(\frac{3}{2}\right)n-2}$$

$$-\left(r_1(x)p_1(x)x^{\frac{n}{2}} + r_1(x)p_2(x)\right)^2$$

Substituting $x^{n-2} + \varepsilon_1(x)$ **for** $r_1(x)p_1(x)$,

we get

$$R(x)p(x) = x^{3n-4} - \left(\varepsilon_1(x)x^{\frac{n}{2}} + r_1(x)p_2(x)\right)^2$$

$$= x^{3n-4} - 0\left(x^{2n-4}\right)$$

But $r(x) = \left\lfloor \dfrac{R(x)}{x^{n-2}} \right\rfloor$

so

$$r(x) \cdot p(x) = x^{2n-2} + o\left(x^{n-2}\right)$$

9 ***Modular Arithmetic***

assume **relatively prime** $p_0, p_1, \ldots, p_{k-1}$

$$\text{let } p = \prod_{i=0}^{k-1} p_i$$

given x , $0 \le x < p$,

$$x \underset{1-1}{\longleftrightarrow} (x_0, \ldots, x_{k-1}) \quad \text{where}$$

$$x_i = x \bmod p_i \quad \text{for } i=0,\ldots,k-1$$

Applications to Arithmetic:

$$u \text{ op } v \underset{1-1}{\longleftrightarrow} (\omega_0, \omega_1, \ldots, \omega_{k-1})$$

where $w_i = u_i$ **op** $v_i \bmod p_i$

$$u_i = u \bmod p_i$$

$$v_i = v \bmod p_i$$

$$\text{op} \in \{ +, -, \times \}$$

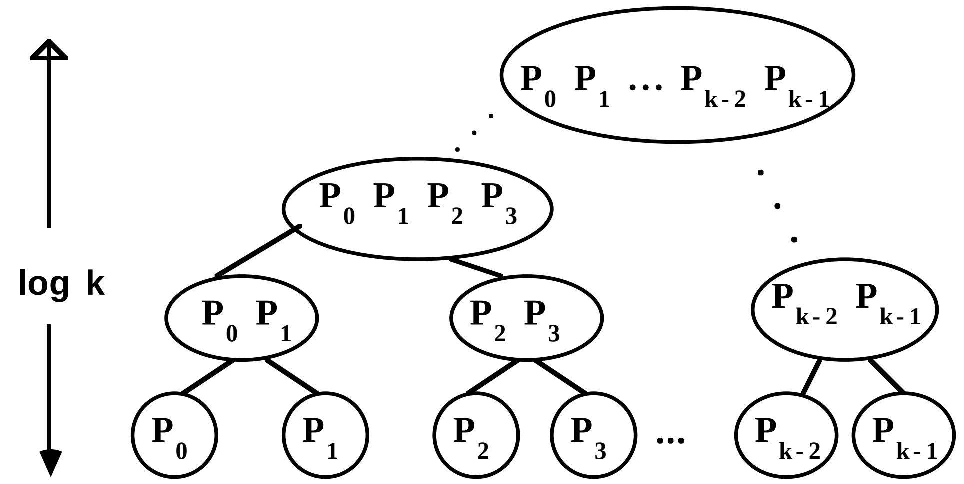
But doesn't extend to *division*

(overflow problems)

10 ***Super Moduli Computation***

input $p_0, p_1, \ldots, p_{k-1}$, **where** $p_i < 2^b$

output **Super modular Tree:**



Time Cost

$$\sum_{i=0}^{\log k} \left(\frac{k}{2^i}\right) M\left(2^i b\right)$$

$$= O\left(M(k\, b) \log k\right)$$

Algorithm Residue Computation

input x, $0 \le x < P = \prod_{i=0}^{k-1} P_i$

output $x_0, x_1, \ldots, x_{k-1}$ when

$x_i = x \bmod P_i$ i=0, and k-1

recursive algorithm

[1] compute quotent and remainders:

$$x = q_1 v_1 + r_1 \ , \quad v_1 = \prod_{i=0}^{(k-1)/2} P_i$$

$$x = q_2 v_2 + r_2, v_2 = \prod_{i=\frac{(k-1)}{2}+1}^{k-1} P_i$$

[2] recursively compute

(2.1) $r_1 \bmod P_i$ for $i = 0, \ldots, \frac{(k-1)}{2}$

(2.2) $r_2 \bmod P_i$ for $i = \frac{(k-1)}{2} + 1, \ldots, k-1$

[3] output for i = 0,1,..., k-1

$$x \bmod P_i = \begin{cases} r_1 \bmod P_i & \text{for } i \le \frac{(k-1)}{2} \\ r_2 \bmod P_i & \text{for } i > \frac{(k-1)}{2} \end{cases}$$

Time Cost for Residue Computation

let D(n) = time cost for integer division

= O(M(n))

Total Time $n = k \cdot b$

$$T(n) = 2T\left(\frac{n}{2}\right) + 2k\,D(b) + O(n)$$

$$\le 2T\left(\frac{n}{2}\right) + O(M(n))$$

$$\le O(M(n)\log n)$$

proof of algorithm:

uses fact if x = q v + r

and v mod P_i = 0, then

x mod P_i = r mod P_i

13

Residue Computation of Polynomials

input moduli $P_0(x), P_1(x), \ldots, P_{k-1}(x)$
relatively prime
each degree $\leq d$

$Q(x)$ degree $\leq kd$

output for $i=0, \ldots, k-1$

$$Q_i(x) = Q(x) \bmod P_i(x)$$

Theorem The Residue Computation can be done in time $O(M(n) \log n)$ where $n = k \cdot d$

proof use some algorithm as in integer case.

14

Multipoint Polynomial Evaluation

input polynomial $f(x) = \sum_{i=0}^{n-1} a_i x^i$

problem evaluate $f(x)$ at $x_0, x_1, \ldots, x_{n-1}$

Easy Cases:

FFT Case $x_i = \omega^i$

= principal root of unity

method $f(x) = f'(y) + x f''(y)$

where $y = x^2$

$f'(x), f''(x)$ both degree $\frac{(n-2)}{2}$

$\Rightarrow$ needed to only evaluate at

$\frac{(n-2)}{2}$ points $y_0, \ldots, y_{\frac{(n-1)}{2}-1}$

where $y_{i+\frac{n}{2}} = \omega^{2i+n} = \omega^{2i} = x_i^2 = y_i$

15

Other Easy Cases O(n log n) time

(1) $x_i = ba^i$ for i=0,..., n-1

(Chirp Transform) $\Rightarrow$ reduced to FFT_n

(2) $x_i = ba^{2i} + ca^i + d$ (Aho, Steiglitz, Ullman)

solve by divide & conquer similar to FFT

16

Single Point Evaluation of all Derivatives of Polynomial

input

$$f(x) = \sum_{i=0}^{n-1} a_i x^i \quad \text{and point } x_0$$

output $$f^{(k)}(x) = \left. \frac{d^k f(x)}{dx} \right|_{x = x_0} \quad \text{for } k=0,\dots n-1$$

Taylor Series Representation of

$$f(x) = \sum_{i=0}^{n-1} c_i (x - x_0)^i$$

Then

$$f^{(k)}(x_0) = k! c_k$$

$\Rightarrow$ reduces to case of evaluation at

$x_i = ab^i$ for $i = 0,\dots,n-1$

17

Multipoint Evaluation by Residue Computation

input polynomial f(x) degree n-1
and points $x_0, x_1, \ldots, x_{n-1}$

[1] for i=0,..., n-1
let $P_i(x) = (x-x_i)$

[2] By Residue Algorithm

Computer for i=0,..., n-1

$f(x_i) = f(x) \bmod P_i(x)$

[3] *output* $f(x_0), \ldots, f(x_{n-1})$

Time Cost O(M(n) log n),

$=O(n(\log n)^2)$

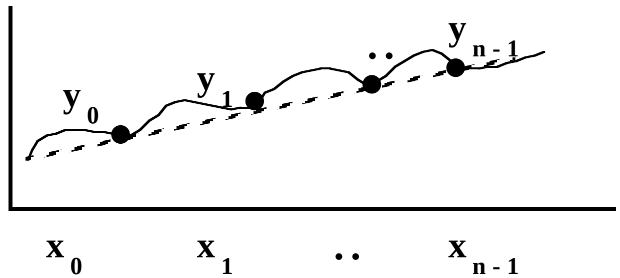
18

Polynomial (Lagrangian) Interpolation

input $x_0, \ldots, x_{n-1}$

$y_0, \ldots, y_{n-1}$

output P(x) where $y_k = P(x_k)$ for k=0,...,n-1



Interpolation formula:

$$P(x) = \sum_{k=0}^{n-1} y_k a_k \prod_{i \neq k} (x - x_i)$$

where

$$a_k = \frac{1}{\prod_{i \neq k} (x_k - x_i)}$$

proof use identities:

$$\left(a_k \prod_{i \neq k} (x - x_i)\right) \bmod (x - x_k) = 1$$

$$\Rightarrow \left(a_k \prod_{i \neq k} (x = x_i)\right) \bmod (x - x_j) = 0$$

19 ## *Integer Interpolation* (Chinese Remaindering)

input **relatively prime** $P_0, P_1, \ldots, P_{n-1}$

and $y_i \in \{0,\ldots,P_{i-1}\}$ **for** $i=0,\ldots,n-1$

problem **compute** $y < \prod P_i$ **s.t.**

$$y_i = y \bmod P_i \quad i=0,\ldots,n-1$$

Generalized Interpolation Formula:

$$y = \sum_{k=0}^{n-1} y_k a_k \prod_{i \neq k} P_i$$

where

$$a_k = \prod_{i \neq k} s_{i,k}$$ **and**

$$s_{i,k} = (P_i)^{-1} \bmod P_k$$

proof

$$\left(a_k \prod_{i \neq k} P_i \right) \bmod P_j = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases}$$

20 ## Preconditioned Interpolation

assume coefficients $\{a_k \mid k=0,\ldots,n-1\}$

precomputed

Use Divide & Conquer:

$$Y = \sum_{k=0}^{n-1} y_k a_k \prod_{i \neq k} P_i$$

$$= \left(\sum_{k=0}^{(n-1)/2} y_k a_k \prod_{\substack{i \neq k \\ i=0}}^{n-1} P_i \right) + \left(\sum_{k=\frac{(n-1)}{2}+1}^{n-1} y_k a_k \prod_{\substack{i \neq k \\ i=0}}^{n-1} P_i \right)$$

$$= \left(\prod_{i=0}^{(n-1)/2} P_i \right) \left[\sum_{k=\frac{(n-1)}{2}+1}^{n-1} y_k a_k \prod_{\substack{i \neq k \\ i=\frac{(n-1)}{2}+1}}^{(n-1)} P_i \right]$$

$$+ \left(\prod_{i=\frac{(n-1)}{2}+1}^{n-1} P_i \right) \left[\sum_{k=0}^{(n-1)/2} y_k a_k \prod_{\substack{i \neq k \\ i=0}}^{(n-1)/2} P_i \right]$$

21

Preconditioned Interpolation

$$T(n) = 2\ T\left(\frac{n}{2}\right) + 2\ M\left(\frac{n}{2}\right)$$

$$= O(M(n)\ \log\ n)$$

assuming $\{a_0, .., a_{n-1}\}$ **precomputed**

22 ***Precomputation of*** $\{a_0, \ldots, a_{n-1}\}$

(1) Compute $P = \prod_{i=0}^{n-1} P_i$

(2) Compute b_k **where**

$$b_k\ P_k = P \bmod (P_k)^2$$

by Residue Computation O(M(n) log n)

(3) Compute $a_k = \left(b_k\right)^{-1} \bmod P_k$

by Extended GCD algorithm

proof **since** $b_k\ P_k = P \bmod \left(P_k\right)^2$

then $P = d\left(P_k\right)^2 + b_k\ P_k$

so $\prod_{i \neq k} P_i = dP_k + b_k$

so $b_k = \prod_{i \neq k} P_i \bmod P_k$

23

Precomputation of $\{a_0,\ldots,a_{n-1}\}$ for Polynomial Interpolations

here $P_i = (x-x_i)$ for $i=0,\ldots,n-1$

$$b_k = \frac{Q(x)}{(x-x_k)} \quad \text{where} \quad Q(x) = \prod_{j=0}^{n-1} (x-x_j)$$

$$= \frac{Q(x)-Q(x_k)}{(x-x_k)} \quad \text{since} \quad Q(x_k) = 0$$

$$= \frac{d\ Q(x)}{dx}\Big]_{x = x_k}$$

$\Rightarrow$ reduces to multipoint evaluation of derivative of Q(x)
O(M(n) log n) time!

24

Conclusion:

Polynomial Computation and Arithmetic essentially

Equivalent Class of Problems:

Examples:

(1) multiplication

(2) division

(3) interpolation evaluation

Open Problem:

reduce from time O(M(n)log n) to O(M(n))