Lecture 4

2/21/2012

Note Title 2/18/2012

Today

- Polynomial Operations
- Complexity of Multiplication
  - with nice roots of unity
  - without nice roots ....
- (time permitting): Division, GCD, ...

Basic Setup

- Some ring $R[x]$ (commutative)
- Mostly interested in $R = \mathbb{F}$ (field)
- But general setting will help anyway.
- Will consider only monic polynomials though

Main Operations

$n \triangleq \deg(f), \deg(g)$

- Addition : linear time $O(n)$
- Multiplication
  - $O(n \log n)$ time in "nice" case
  - $O(n \log n (\log\log n))$ time in general case.

  (slightly faster known?)
- Division : $f, g \Rightarrow q, r$ s.t.

  $f = q \cdot g + r$

  $\deg(r) < \deg(g)$

  - $O(\text{mult}(n))$ time
- Multipoint evaluation : $\alpha_1 \ldots \alpha_n$; f

  $\Rightarrow f(\alpha_1) \ldots f(\alpha_m)$

  - $O(n \operatorname{poly} \log n)$ time

- Interpolation : $\alpha_1 \ldots \alpha_n ; \beta_1 \ldots \beta_n$

  $\Rightarrow f$ s.t. $f(\alpha_i) = \beta_i \quad \forall i$

  - $O(n \operatorname{polylog} n)$ time

- GCD : $f, g \Rightarrow a, b, h$ s.t.

  $h \mid f, \quad h \mid g$

  $h = a \cdot f + b \cdot g$

  - $O(n \cdot \operatorname{polylog} n)$ time

- Modular composition : $f, g, h \Rightarrow (f \circ g) \bmod h$

  - $O(n \operatorname{polylog} n)$ time [Kedlaya, Umans]

TODAY : Mostly Multiplication

"Review of FFT-based Multiplication"

- Let ω be $(2n)^{th}$ primitive root of unity

$$\boxed{\begin{array}{l} \omega \text{ is } m^{th} \text{ root of unity if } \omega^m = 1 \\ \text{"primitive" if } \omega^i \neq 1 \text{ for } i \in \{1 \ldots m-1\} \end{array}}$$

- FFT: Compute f, g at $1, \omega, \omega^2, \ldots \omega^{2n-1}$
(aka "Multipoint Evaluation")
- "multiply": $\beta_i \stackrel{\circ}{=} f(\omega^i) \cdot g(\omega^i)$
- FFT^{-1}: Compute h s.t. $h(\omega^i) = \beta_i$
(aka "interpolation")

FFT

Key idea:

$$x \longmapsto x^2$$

is a 2-1 map on $\{1 \ldots \omega^{2n}\}$

also on $\{1, \omega^i, \omega^{2i}, \ldots (\omega^i)^{\frac{2n}{i}}\}$ if

i, n are powers of 2

Algorithm

Input: $f = c_0, c_1, \ldots c_{n-1}$

ω - primitive n^{th} root.

Step 1: Write $f(x) = f_0(x^2) + x f_1(x^2)$

$\deg(f_0), \deg(f_1) < \frac{n}{2}$

Step 2: Recursively compute

(f_0, f_1) on $\{1, \omega^2, \omega^4, \omega^6, \ldots \omega^{n-2}\}$

Step 3: Return $f(\omega^i) = f_0(\omega^{2i}) + \omega^i f_1(\omega^{2i})$

FFT^{-1}

$$\begin{array}{c} M \\ \overbrace{\hphantom{\quad\quad\omega^{ij}\quad\quad}} \end{array}$$

FT: $\begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} = \begin{bmatrix} \\ \omega^{ij} \\ \\ \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix}$

$\uparrow$ output $\qquad\qquad$ $\uparrow$ Input

FT^{-1}: input $\qquad\qquad$ Output

So $M^{-1} = ?$

Claim: $M^{-1} = \frac{1}{n} \begin{bmatrix} \omega^{-ij} \end{bmatrix}$

Proof: $(M^{-1} M)_{ii} = \frac{1}{n} \sum_j \omega^{-ij} \omega^{ij} = \frac{n}{n} = 1$

$$(M^{-1} M)_{ij} = \frac{1}{n} \sum_k \omega^{(j-i)k} = ?$$

Lemma next claims it is zero, if...

Lemma: R = commutative ring

ω = primitive n^{th} root of unity

$2 \neq$ zero divisor. $n = 2^m$

Then $\forall\, \ell \in \{1 \ldots n-1\}$

① $\omega^{\ell} - 1 \neq$ zero divisor

② $\displaystyle\sum_{i=0}^{n-1} \omega^{\ell i} = 0$

Proof ① $\Rightarrow$ ②

$$(\omega^{\ell} - 1) \sum_{i=0}^{n-1} \omega^{\ell i} = \omega^{\ell n} - 1 = 0$$

But $\omega^{\ell} - 1$ is not a zero divisor, so

$$\sum_{i=0}^{n-1} \omega^{\ell i} = 0.$$

Proof of ①:

Let u, k be s.t. u = odd, $\ell = u \cdot 2^k$

Proof by reverse induction on k.

- $k = m-1$: $\omega^{\ell} = -1$; $\omega^{\ell} - 1 = -2 \neq$ zero divisor

- $k+1 \to k$:

Suppose $(\omega^{\ell}-1)\cdot a = 0$ for some $a \neq 0$

Then $(\omega^{\ell}+1)(\omega^{\ell}-1)a = 0$ $[0 \cdot x = 0]$

$\Rightarrow (\omega^{2\ell}-1)\cdot a = 0$ violating induction

$\boxtimes$

Conclusion:

- FFT can be computed in $O(n \log n)$ time
- FFT^{-1} is just FFT, can be also " .
- Multiplication in $R[x]$ takes

  n general multiplications in R

  $+ O(n \log n)$ additions, multiplications by ω^i
- Needs R to have primitive 2^m-th root.

  & 2 as a unit (non zero-div)

Issues with FFT-based Multiplication

- Hard to find R with 2^mth root of unity.
- Fields of char 2 can't have 2 as unit.

Exercise:

Defn: $S \triangleq \mathbb{F}_2$-subspace of $\mathbb{F}_{2^t}$ if

$$\forall \alpha, \beta \in S, \quad \alpha + \beta \in S$$

Task: (1) Given $f \in \mathbb{F}_2[x]$, $\deg(f) < n$, $\alpha \in \mathbb{F}_{2^n}$

Compute f_0, f_1 $\deg(f_0), \deg(f_1) < \frac{n}{2}$

s.t. $f(x) = f_0(x^2 - \alpha x) + x f_1(x^2 - \alpha x)$

in $O(n \log n)$ time

(2) Use above to do multipoint evaluation, interpolation over $\mathbb{F}_2$ subspace S, $|S| = n$, and thus multiplication in $\mathbb{F}_2[x]$ in $O(n \log^2 n)$ time.

General Multiplication

(in R where $2 \neq$ zero-divisor)

[Schönhage-Strassen]

Key idea

- Extend ring to have some big root of unity.
- Problem: Usually makes ring bigger; to have ℓ^{th} root of unity new ring $\approx R^{\ell}$.
- Resolution: Reduce mult. of deg n polys in R, to mult. of deg $\frac{n}{\ell}$ polys in R'_{ℓ}, where $|R'_{\ell}| \approx R^{\ell}$, & R'_{ℓ} has ℓ^{th} root of unity.
- Complication: R'_{ℓ} multiplication is like multiplication in $R[x]$, fortunately we can recurse. (polys are of deg ℓ)

Details

$R'_\ell = R[y] / (y^\ell + 1)$ $\qquad \ell$ = power of 2.

- y = primitive $2\ell^{th}$ root of unity.
- So multiplication of deg k polys takes $O(k \log k)$ additions in R'_ℓ etc. $+ k$ multiplications in R'_ℓ
- R'_ℓ multiplication is multiplication of deg ℓ polys. in R

Reduction: Let $n = \dfrac{\ell \cdot k}{2}$

$$f, g \in R[x] \longrightarrow f', g' \in R[x,y]$$

s.t. $f(x) = f'(x, x^k)$

$$R[x,y] \supseteq R'[x]$$

$\Downarrow$ using FFT + recursion

$$h \in R[x] \xleftarrow{h' = h(x, x^k)} h' = f' \cdot g' \in R'[x] \subseteq R[x,y]$$

Analysis/Correctness: Omitted.

Appendix : Web of interconnections

- Algebraic algorithms mix interpolation, multipoint evaluation, division & multiplication intricately
- Already saw that mult $\leq$ special multeval & special interp.

- Next Division $\leq$ Mult.
- General Multi Point Eval $\leq$ Mult + Div
- General Interpolation $\leq$ Mult, Div, M.P.E

...

Division $\leq$ Multiplication

Problem

Input: f, g

Output: q, r s.t. $\deg(r) < \deg(g)$

$$f = q \cdot g + r$$

Step 1: Division $\leq$ Special Modular Inversion

Define: $\mathrm{Rev}(f) \triangleq x^{\deg(f)} \cdot f\left(\frac{1}{x}\right)$

(i.e. reverse coefficients)

Easy Identity

$$\mathrm{Rev}(f) = \mathrm{Rev}(q) \cdot \mathrm{Rev}(g) + x^{\ell} \cdot \mathrm{Rev}(r)$$

$$\ell \triangleq \deg(f) - \deg(g)$$

Utility?

$$\mathrm{Rev}(q) = \mathrm{Rev}(f) \cdot \mathrm{Rev}(g)^{-1} \pmod{x^{\ell}}$$

Can compute q from above, and thus r.

(end Step 1)

Step 2: Special Modular Inversion $\pmod{x^\ell}$

Input: $h(x) = \sum h_i x^i$; $h_0 = 1$; ℓ

Output: $h(x)^{-1} \pmod{x^\ell}$

Algorithm: "Newton's Iterations"
or "Hensel lifting"

Inductively lift solution $\pmod{x^t}$
to solution $\pmod{x^{2t}}$

Suppose $a_0 \in R[x]$ s.t.

$$a_0 h = 1 \pmod{x^t}$$

Write $h = h_0 + x^t h_1$ $\deg(h_0) < t$

w.l.o.g $\deg(a_0) < t$

Want a_1, $\deg(a_1) < t$ s.t.

$$(a_0 + x^t a_1)(h_0 + x^t h_1) = 1 \pmod{x^{2t}}$$

Hensel/Newton: Can solve for a_1 above

Let $a_0 h_0 = 1 + x^t \cdot b$

Then

$(a_0 + x^t \underline{\underline{a_1}})(h_0 + x^t h_1)$

$= a_0 h_0 + x^t(\underline{\underline{a_1}} h_0 + h_1 a_0) + x^{2t} \cdot \text{stuff}.$

$= 1 + x^t(a_0 h_1 + b + \underline{\underline{a_1}} h_0)$

Need to set a_1 s.t. $a_1 h_0 = -(a_0 h_1 + b) \pmod{x^t}$

Can we multiply by h_0^{-1} ?

Yes: we know that is $a_0 \pmod{x^t}$!

- Conclude: $a_1 = -a_0^2 h_1 - b a_0$.

  Thus $O(1)$ multiplications reduce problem to half the size.

- Big Conclusion: Division time $= O(\text{Mult. time})$.

Multipoint Evaluation

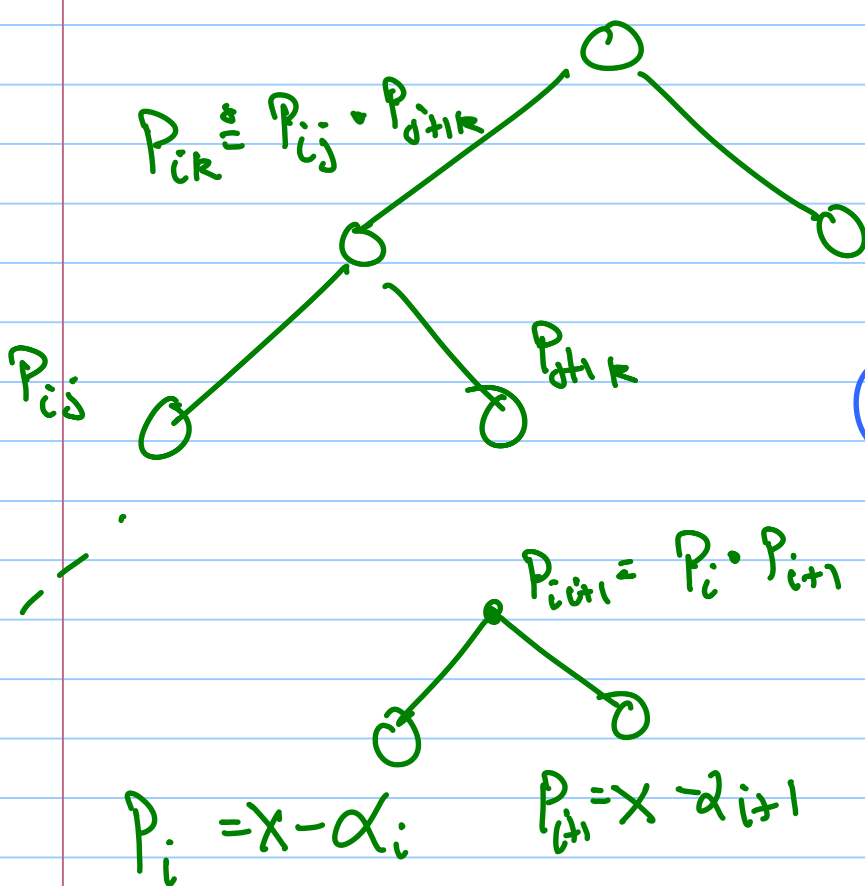
Input: $f = \sum_{i=0}^{n-1} c_i x^i$; $\alpha_1 \ldots \alpha_n$

Output: $\beta_1 \ldots \beta_n$; $\beta_i = f(\alpha_i)$

Key idea: Evaluation = Modular Reduction

$$f(\alpha_i) = f \pmod{(x-\alpha_i)}$$

Alg + Proof + Analysis by Picture



① Starting at leaves compute P_i's at nodes & go up to root.

② Starting at root go down computing $f_i \bmod P_i$

Conclude: Multi Point Eval = $O(\text{Mult.} \log n)$

General Interpolation

Input : $\alpha_1 \dots \alpha_n$; $\beta_1 \dots \beta_n$; α_i's distinct

Output $c_0 \dots c_{n-1}$ s.t. $f(\alpha_i) = \beta_i \; \forall i$, $f(x) = \sum c_i x^i$

Idea: Compute z_1, z_2, f_1, f_2 s.t.

$$① \; z_1(\beta_1) \dots z_1(\beta_{\frac{n}{2}}) = 0$$

$$② \; z_2(\beta_{\frac{n}{2}+1}) \dots z_2(\beta_n) = 0$$

$$③ \; f = f_1 z_2 + f_2 z_1$$

$$\deg(f_1) \; \deg(f_2) < \frac{n}{2}$$

$$\deg(z_1) \; \deg(z_2) = \frac{n}{2}$$

Key step:

Given z_2, f_1 is solution to

Interpolation of input

$$\alpha_1 \dots \alpha_{\frac{n}{2}}, \quad \frac{\beta_1}{z_2(\beta_1)} \; \dots \; \frac{\beta_{\frac{n}{2}}}{z_2(\beta_{\frac{n}{2}})}$$

none of these are zero →

Rest is details

REFERENCES

① Text by [Gerhard & von zur Gathen]

② Text by [Burgisser, Clausen, Shokrollahi]

③ Algorithms text by [Aho, Hopcroft, Ullman]