Algorithms
Professor John Reif

ALG 3.2

The Fast Fourier Transform and Applications to Multiplication

Reading Selections:
CLR: chapter 32

Auxiliary Reading:
AHU-Design Chapters 7 and 8
BB Section 7.1.3 and Chapter 9

Assume Commutative Ring

$(R,+,\cdot,0,1)$

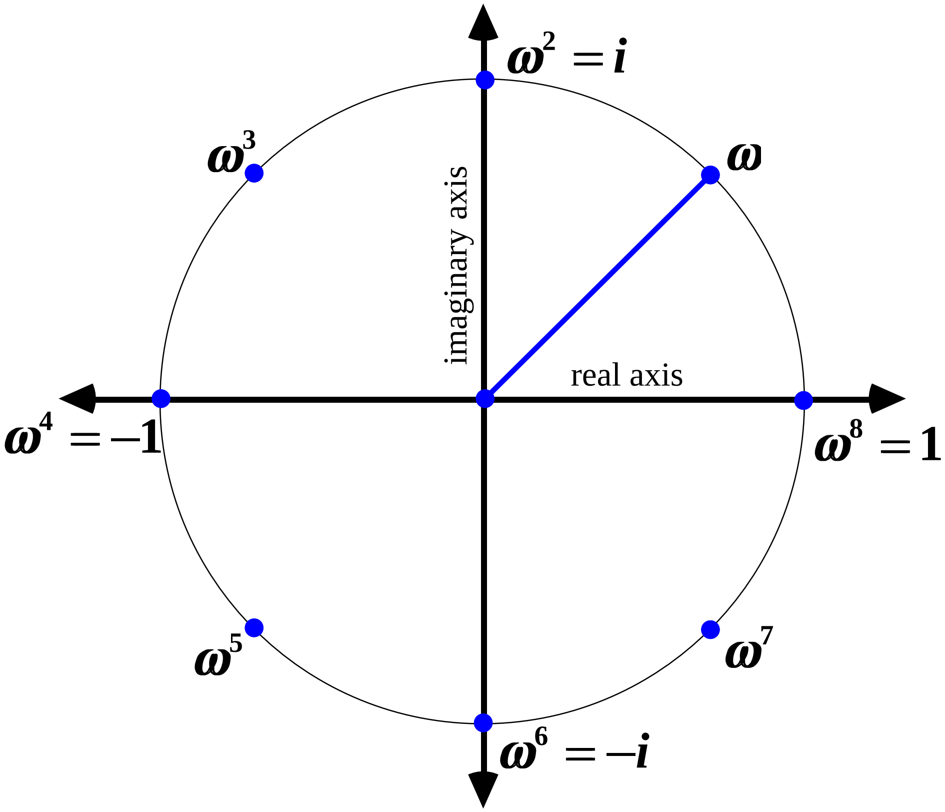
ω is *principal nth root of unity*
$\omega^i \neq 1$ for $i=1,\ldots,n-1$

if $\omega \neq 1$, $\omega^n = 1$, and

$$\sum_{j=0}^{n-1} \omega^{jp} = 0 \quad \text{for } 1 \le p < n$$

Examples:

$\omega = e^{2\Pi i/n}$ for complex numbers



Example

$\omega = e^{2\pi i/8}$ is the 8th root of unity

Fourier Matrix

$$M_n(\omega) = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ 1 & \omega & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \ldots & \omega^{2(n-1)} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ 1 & \omega^{n-1} & \ldots & \omega^{(n-1)(n-1)} \end{pmatrix}$$

so $M(\omega)_{ij} = \omega^{ij}$ for $o \le i,j < n$

given $a = \begin{pmatrix} a_o \\ \cdot \\ \cdot \\ \cdot \\ a_{n-1} \end{pmatrix}$

Discrete Fourier Transform is

$$DFT_n(a) = M(\omega) \times a$$

$$= \begin{pmatrix} f_o \\ \cdot \\ \cdot \\ \cdot \\ f_{n-1} \end{pmatrix} \quad \text{where}$$

$$f_i = \sum_{k=0}^{n-1} a_k \, \omega^{ik}$$

Inverse Fourier Transform

$$DFT_n^{-1}(a) = M(\omega)^{-1} \times a$$

Theorem $M(\omega)^{-1}_{ij} = \frac{1}{n} \omega^{-ij}$

proof We must show $M(\omega) \cdot M(\omega)^{-1} = I$

$$\frac{1}{n} \sum_{k=0}^{n-1} \omega^{ik} \, \omega^{-kj} = \frac{1}{n} \sum_{k=0}^{n-1} \omega^{k(i-j)}$$

$$= \begin{cases} 0 & \text{if } i-j \neq 0 \\ 1 & \text{if } i-j = 0 \end{cases}$$

using identity $\sum_{k=0}^{n-1} \omega^{kp} = 0$

for $1 \le p < n$

input column vector a = $(a_0,\ldots,a_{n-1})^T$

$$\mathrm{DFT}_n(a) = \begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix} \quad \text{where}$$

$f_i = f(\omega^i)$ and

$$f(x) = \sum_{j=0}^{n-1} a_j \cdot x^j$$

Viewed as *Evaluation Problem:*

naive algorithm **takes n^2 ops**

Divide and Conquer **gives FFT**

with O(n log n) ops

for n a power of 2

Key Idea:

If ω is nth root of unity

then ω^2 is $\frac{n}{2}$th root of unity

Algorithm FFT$_n$

Input $a = (a_0,\ldots,a_{n-1})^T$, n **a power of 2**

[1] If $n = 1$ then **output** a

[2]

$$\left(f_0^{\copyright},\ldots,f_{\frac{n}{2}-1}^{\copyright}\right)^T \leftarrow FFT_{\frac{n}{2}}\left((a_0,a_2,\ldots,a_{n-2})^T\right)$$

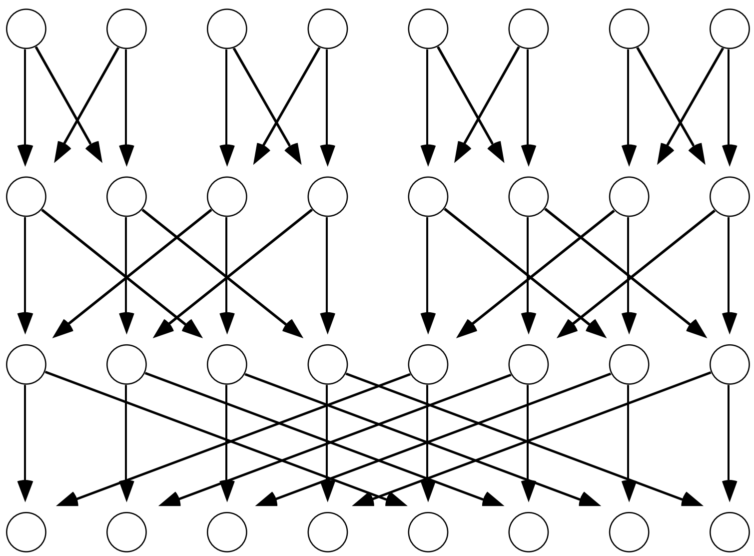
$$\left(f_0^{''},\ldots,f_{\frac{n}{2}-1}^{''}\right)^T \leftarrow FFT_{\frac{n}{2}}\left((a_1,a_3,\ldots,a_{n-1})^T\right)$$

[3] **For** $i = 0,\ldots,\frac{n}{2}-1$ **do** $f_i \leftarrow f_i^{\copyright} + \omega^i f_i^{''}$

$$f_{i+\frac{n}{2}} \leftarrow f_i^{\copyright} - \omega^i f_i^{''}$$

[4] **Output** $(f_0, f_1,\ldots,f_{n-1})$

FFT Circuit = Butterfly



Total Recursion depth = $\mathbf{log}\, \boldsymbol{n}$

Communication Distance $\mathbf{2}^{d}$ at depth $\boldsymbol{d}$

$$f_i = a_0 + a_1\omega^i + a_2\omega^{2i} + \dots + a_{n-1}\,\omega^{(n-1)i}$$

$$f_i = f'_i + \omega^i f''_i \quad \text{where}$$

$$f'_i = a_0 + a_2(\omega^2)^i + a_4(\omega^2)^{2i} + \dots + a_{n-2}(\omega^2)^{\frac{i(n-2)}{2}}$$

$$f''_i = a_1 + a_3(\omega^2)^i + \dots + a_{n-1}(\omega^2)^{\frac{i(n-2)}{2}}$$

$$\begin{bmatrix} f_0^{\copyright} \\ \vdots \\ f_{\frac{n}{2}-1}^{\copyright} \end{bmatrix} = M_{\frac{n}{2}}(\omega^2)\begin{bmatrix} a_0 \\ a_2 \\ \vdots \\ a_{n-2} \end{bmatrix} = DFT_{\frac{n}{2}}\left((a_0, a_2, \dots, a_{n-2})^T\right)$$

$$\begin{bmatrix} f_0'' \\ \vdots \\ f_{\frac{n}{2}-1}'' \end{bmatrix} = M_{\frac{n}{2}}(\omega^2)\begin{bmatrix} a_1 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = DFT_{\frac{n}{2}}\left((a_1, a_3, \dots, a_{n-1})^T\right)$$

Note: $f'_{\frac{n}{2}+i} = f'_i,\ f''_{\frac{n}{2}+i} = f''_i,\ i=0,\ldots \frac{n}{2}-1$

But $\omega^n = 1$, **so** $(\omega^2)^{\frac{n}{2}+i} = \omega^n \cdot (\omega^2)^i = \omega^{2i}$

$$\text{for } i=0,\ldots,\frac{n}{2}-1$$

Thus, $f_i = f'_i + \omega^i f''_i$ **for** $i=0,\ldots,\frac{n}{2}-1$

and $f_{i+\frac{n}{2}} = f'_i + \omega^{i+\frac{n}{2}} \cdot f''_i$

$= f'_i - \omega^i f''_i$ **for** $i=0,\ldots,\frac{n}{2}-1$

since $(\omega^{\frac{n}{2}})^2 = \omega^n = 1$, **so** $\omega^{\frac{n}{2}} = -1$

Operation Counts

Assume $n = 2^k$

additions

$$Add(n) = 2 \cdot Add\left(\frac{n}{2}\right) + n$$
$$= n \log n$$

#*multiplications*

$$Mult(n) = 2 \cdot Mult\left(\frac{n}{2}\right) + \frac{n}{2}$$
$$= \frac{1}{2} n \log n$$

Total Time **O(n log n)**

note **in complex FFT,**

 # real ops is 5 n log n.

Applications of FFT

(1) *Filtering* on infinite input streams

(2) *Convolution:*
Products and Powers of Polynomials

(3) Division and Inversion of Polynomials

(4) Multipoint Evaluation and Interpolation

Products and Powers of Polynomials

input vectors $a = (a_0, a_1, \ldots, a_{n-1})^T$

$b = (b_0, b_1, \ldots, b_{n-1})^T$

convolution $c = a \otimes b$

where $c_i = \sum_{j=0}^{n-1} a_j \, b_{i-j}$ for $i=0, \ldots, 2n-1$

(and define $a_k = b_k = 0$ if $k<0$ or $k \geq n$)

Convolution Theorem

$$a \otimes b = FFT_{2n}^{-1}\left(FFT_{2n}(a) \cdot FFT_{2n}(b)\right)$$

Application to Polynomial Products:

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

$$q(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$p(x) \cdot q(x) = \sum_{i=0}^{2n-2} c_i x^i \quad \text{where } c_i = \sum_{j=0}^{n-1} a_j b_{i-j}$$

Products of m polynomials

$$\text{for } k=1,\ldots,m \text{ let } P_k(x) = \sum_{i=0}^{n-1} a_{k,i} x^i$$

$$\prod_{k=1}^{m} P_k(x) = \sum_{i=0}^{m(n-1)} c_i\, x^i, \text{ when } c_i = \sum_{\Sigma j_k = 1} \prod_{k=1}^{m} a_{k,j_k}$$

Generalized Convolution Theorem

$$a_1 \otimes a_2 \otimes \ldots \otimes a_m =$$

$$FFT_{n\,m}^{-1}(FFT_{n\,m}(a_1) \bullet FFT_{n\,m}(a_2) \ldots FFT_{n\,m}(a_m))$$

$$a = (a_0,\ a_1,\ \ldots,\ a_{n-1})^T,\ b = (b_0,\ b_1,\ \ldots,\ b_{n-1})^T$$

positive wrapped convolution is $c = (c_0,\ c_1, \ldots, c_{n-1})^T$

$$c_i = \sum_{j=0}^{i} a_j\, b_{i-j} + \sum_{j=i+1}^{n-1} a_j\, b_{n+i-j}$$

negative wrapped convolution is $d = (d_0, d_1, \ldots, d_{n-1})^T$

$$d_i = \sum_{j=0}^{i} a_j\, b_{i-j} - \sum_{j=i+1}^{n-1} a_j\, b_{n+i-j}$$

"Wrapped Convolution"

Application to Modular Polynomial Products:

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

$$q(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$p(x) \cdot q(x) \mod(x^n+1)$$

$$= \sum_{i=0}^{n-1} d_i x^i \quad \text{since} \quad x^n = -1 \mod(x^n+1)$$

Theorem If ω = principal nth root of unity and $\Psi^2 = \omega$ and n has multiplicative inverse, then

$$c = FFT_n^{-1}\left(FFT_n(a) \cdot FFT_n(b)\right)$$ is the positive wrapped convolution of a, b. Also

$$\hat{d} = FFT_n^{-1}\left(FFT_n(\hat{a}) \cdot FFT_n(\hat{b})\right)$$ is the negatively wrapped convolution of a,b

where $\hat{a} = \left(a_0, \Psi a_1, \ldots, \Psi^{n-1} a_{n-1}\right)^T$ and

$$\hat{b} = \left(b_0, \Psi b_1, \ldots, \Psi^{n-1} b_{n-1}\right)^T$$

Integer Multiplication by FFT

input n bit integers a,b

define polynomials degree $k = n/L$

$$a(x) = \sum_{i=0}^{k-1} a_i x^i, \quad 0 \le a_i < 2^L$$

$$b(x) = \sum_{i=0}^{k-1} b_i x^i, \quad 0 \le b_i < 2^L$$

so $a = a(2^L)$, $b = b(2^L)$

idea

(1) compute $c(x) = a(x) \cdot b(x)$ by convolution

(2) evaluate $c(2^L) = a \cdot b$

Integer Multiplication Algorithms

	Time	
Pollard Alg.	$O(n(\log n)^2(\log\log n)^{\epsilon}$	use $L=\log n$
Karp Alg.	$O(n(\log n)^2)$	use $L=\sqrt{n}$
Strassen Alg.	$O(n(\log n)(\log\log n))$	use $L=\sqrt{n}$ and wrapped convolution

Pollard Algorithm

$n=kL, \quad L=1+\log k$

[1] Choose primes P_1, P_2, P_3 where

$$P_1 \cdot P_2 \cdot P_3 \geq 4 \cdot k^3$$

and $P_i = \alpha_i \cdot 2^L + 1, \; \alpha_i = O(1)$

[2] Compute $C(x)$ by convolution over finite field Z_{p_i} $i=1,2,3$

(requires k mults on 2L bit integers)

[3] Evaluate $C(2^L)$

Time Bounds

$$T(n) = 3k \overbrace{T(2L)}^{\text{recursive mults}} + O\overbrace{(k \log k)}^{FFT} \cdot O(L)$$

$$= 3k\,T(2(1+\log k)) + O(k(\log k)^2)$$

$$\leq O(n(\log n)^2(\log\log n)^{\epsilon}) \text{ for any } \epsilon > 0$$

Korp's Algorithm

$n = 2^s = kL$

$$k = \begin{cases} 2^{\frac{s}{2}} & \text{if } s \text{ even} \\ 2^{\frac{(s-1)}{2}} & \text{else} \end{cases}$$

(1) Compute C(x) modulo k by convolution

(2) Compute C(x) modulo $(2^{2L}+1)$ by convolution (requires 2k recursive mults)

(3) Compute C(x) coefficients from c(x) mod k, c(x) mod $(2^{2L}+1)$ by chinese remaindering

(4) evaluate $C(2^L)$

Time

$$T(n) = 2k\,\overbrace{T(2L)}^{\text{recursive mults}} + \overbrace{O(k \log k)}^{FFT} O(L)$$

$$= 2\sqrt{n}\,T(2\sqrt{n}) + O(n \log n)$$

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

Schönage –Strassen Algorithm

(2') Compute C(x) mod (x^k+1) modulo $(2^{2L}+1)$ by wrapped convolution

$\Rightarrow$ requires only k recursive mults on 2L bit numbers

Time

$$T(n) = k\,\overbrace{T(2L)}^{\text{recursive mults}} + \overbrace{O(k \log k)}^{FFT} O(L)$$

$$= \sqrt{n}\,T(2\sqrt{n}) + O(n \log n)$$

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

Open Problem: Can you mult n bit integers in O(n log n) time?