

Algorithm Design and Analysis

CSE
565

LECTURES 17 and 18
Divide and Conquer
• Fast Fourier Transform

Sofya Raskhodnikova

10/5/2007

S. Raskhodnikova; based on slides by K. Wayne.

Fast Fourier Transform: Applications

Applications.

- Optics, acoustics, quantum physics, telecommunications, control systems, signal processing, speech recognition, data compression, image processing.
- DVD, JPEG, MP3, MRI, CAT scan.
- Numerical solutions to Poisson's equation.

The FFT is one of the truly great computational developments of this [20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. -Charles van Loan

Fast Fourier Transform: Brief History

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.

Runge-König (1924). Laid theoretical groundwork.

Danielson-Lanczos (1942). Efficient algorithm.

Cooley-Tukey (1965). Monitoring nuclear tests in Soviet Union and tracking submarines. Rediscovered and popularized FFT.

Importance not fully realized until advent of digital computers.

Polynomials: Coefficient Representation

Polynomial. [coefficient representation]

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

Add: $O(n)$ arithmetic operations.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \dots + (a_{n-1} + b_{n-1})x^{n-1}$$

Evaluate: $O(n)$ using Horner's method.

$$A(x) = a_0 + (x(a_1 + x(a_2 + \dots + x(a_{n-2} + x(a_{n-1}))))))$$

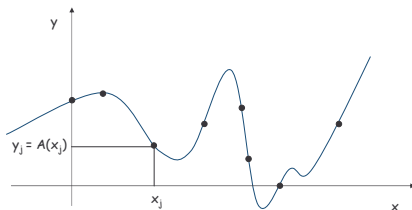
Multiply (convolve): $O(n^2)$ using brute force.

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^i a_j b_{i-j}$$

Polynomials: Point-Value Representation

Fundamental theorem of algebra. [Gauss, PhD thesis] A degree n polynomial with complex coefficients has n complex roots.

Corollary. A degree $n-1$ polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x .



Polynomials: Point-Value Representation

Polynomial. [point-value representation]

$$A(x) = (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$$

$$B(x) = (x_0, z_0), \dots, (x_{n-1}, z_{n-1})$$

Add: $O(n)$ arithmetic operations.

$$A(x) + B(x) = (x_0, y_0 + z_0), \dots, (x_{n-1}, y_{n-1} + z_{n-1})$$

Multiply: $O(n)$, but need $2n-1$ points.

$$A(x) \times B(x) = (x_0, y_0 \times z_0), \dots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

Evaluate: $O(n^2)$ using Lagrange's formula.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

Converting Between Two Polynomial Representations

Tradeoff. Fast evaluation or fast multiplication. We want both!

Representation	Multiply	Evaluate
Coefficient	$O(n^2)$	$O(n)$
Point-value	$O(n)$	$O(n^2)$

Goal. Make all ops fast by efficiently converting between two representations.

a_0, a_1, \dots, a_{n-1}
coefficient representation

↔

$(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$
point-value representation

Converting Between Two Polynomial Representations: Brute Force

Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} .

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$O(n^2)$ for matrix-vector multiply

$O(n^3)$ for Gaussian elimination

Vandermonde matrix is invertible iff x_i distinct

Point-value to coefficient. Given n distinct points x_0, \dots, x_{n-1} and values y_0, \dots, y_{n-1} , find unique polynomial $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$ that has given values at given points.

Coefficient to Point-Value Representation: Intuition

Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} .

Divide. Break polynomial up into even and odd powers.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{\text{even}}(x) = a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6$.
- $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^3 + a_7 x^7$.
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$.
- $A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2)$.

Intuition. Choose two points to be ± 1 .

- $A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1)$.
- $A(-1) = A_{\text{even}}(1) - 1 A_{\text{odd}}(1)$.

Can evaluate polynomial of degree $\leq n$ at 2 points by evaluating two polynomials of degree $\leq \frac{n}{2}$ at 1 point.

Coefficient to Point-Value Representation: Intuition

Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} .

Divide. Break polynomial up into even and odd powers.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{\text{even}}(x) = a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6$.
- $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^3 + a_7 x^7$.
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$.
- $A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2)$.

Intuition. Choose four points to be $\pm 1, \pm i$.

- $A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1)$.
- $A(-1) = A_{\text{even}}(1) - 1 A_{\text{odd}}(1)$.
- $A(i) = A_{\text{even}}(-1) + i A_{\text{odd}}(-1)$.
- $A(-i) = A_{\text{even}}(-1) - i A_{\text{odd}}(-1)$.

Can evaluate polynomial of degree $\leq n$ at 4 points by evaluating two polynomials of degree $\leq \frac{n}{2}$ at 2 points.

Discrete Fourier Transform

Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} .

Key idea: choose $x_k = \omega^k$ where ω is principal n th root of unity.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

\uparrow
 Discrete Fourier transform

\uparrow
 Fourier matrix F_n

Roots of Unity

Def. An n th root of unity is a complex number x such that $x^n = 1$.

Fact. The n th roots of unity are: $\omega^0, \omega^1, \dots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.
 Pf. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{2\pi i})^{2k} = (-1)^{2k} = 1$.

Fact. The $\frac{n}{2}$ th roots of unity are: $v^0, v^1, \dots, v^{n/2-1}$ where $v = e^{4\pi i / n}$.
 Fact. $\omega^2 = v$ and $(\omega^2)^k = v^k$.

Fast Fourier Transform

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \dots, \omega^{n-1}$.

Divide. Break polynomial up into even and odd powers.

- $A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n/2-2}x^{(n-1)/2}$.
- $A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n/2-1}x^{(n-1)/2}$.
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$.

Conquer. Evaluate degree $A_{\text{even}}(x)$ and $A_{\text{odd}}(x)$ at the $\frac{1}{2}n^{\text{th}}$ roots of unity: $v^0, v^1, \dots, v^{n/2-1}$.

Combine.

- $A(\omega^k) = A_{\text{even}}(v^k) + \omega^k A_{\text{odd}}(v^k), \quad 0 \leq k < n/2$
 - $A(\omega^{k+n/2}) = A_{\text{even}}(v^k) - \omega^k A_{\text{odd}}(v^k), \quad 0 \leq k < n/2$
- $v^k = (\omega^k)^2 = (\omega^{k+n/2})^2$
 $\omega^{k+n/2} = -\omega^k$

FFT Algorithm

```
fft(n, a_0, a_1, ..., a_{n-1}) {
  if (n == 1) return a_0

  (e_0, e_1, ..., e_{n/2-1}) ← FFT(n/2, a_0, a_2, a_4, ..., a_{n-2})
  (d_0, d_1, ..., d_{n/2-1}) ← FFT(n/2, a_1, a_3, a_5, ..., a_{n-1})

  for k = 0 to n/2 - 1 {
    ω^k ← e^{2πik/n}
    y_k ← e_k + ω^k d_k
    y_{k+n/2} ← e_k - ω^k d_k
  }

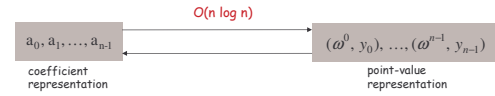
  return (y_0, y_1, ..., y_{n-1})
}
```

FFT Summary

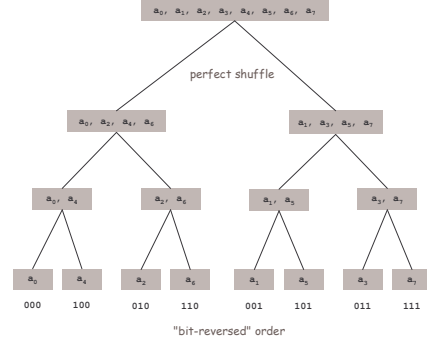
Theorem. FFT algorithm evaluates a degree $n-1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ steps.

assumes n is a power of 2

Running time. $T(2n) = 2T(n) + O(n) \Rightarrow T(n) = O(n \log n)$.



Recursion Tree



Point-Value to Coefficient Representation: Inverse DFT

Goal. Given the values y_0, \dots, y_{n-1} of a degree $n-1$ polynomial at the n points $\omega^0, \omega^1, \dots, \omega^{n-1}$, find unique polynomial $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ that has given values at given points.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Inverse DFT Fourier matrix inverse $(F_n)^{-1}$

Inverse FFT

Claim. Inverse of Fourier matrix is given by following formula.

$$G_n = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \dots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \dots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \dots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Consequence. To compute inverse FFT, apply same algorithm but use $\omega^{-1} = e^{-2\pi i/n}$ as principal n^{th} root of unity (and divide by n).

Inverse FFT: Proof of Correctness

Claim. F_n and G_n are inverses.
 Pf.

$$(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-j k'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{j(k-k')} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma

Summation lemma. Let ω be a principal n^{th} root of unity. Then

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \pmod n \\ 0 & \text{otherwise} \end{cases}$$

Pf.

- If k is a multiple of n then $\omega^k = 1 \Rightarrow$ sums to n .
- Each n^{th} root of unity ω^k is a root of $x^n - 1 = (x - 1)(1 + x + x^2 + \dots + x^{n-1})$.
- if $\omega^k \neq 1$ we have: $1 + \omega^k + \omega^{k(2)} + \dots + \omega^{k(n-1)} = 0 \Rightarrow$ sums to 0 .

Inverse FFT: Algorithm

```

ifft(n, a_0, a_1, ..., a_{n-1}) {
  if (n == 1) return a_0

  (e_0, e_1, ..., e_{n/2-1}) ← FFT(n/2, a_0, a_2, a_4, ..., a_{n-2})
  (d_0, d_1, ..., d_{n/2-1}) ← FFT(n/2, a_1, a_3, a_5, ..., a_{n-1})

  for k = 0 to n/2 - 1 {
    ω^k ← e^{-2πi k/n}
    y_k ← (e_k + ω^k d_k) / n
    y_{k+n/2} ← (e_k - ω^k d_k) / n
  }

  return (y_0, y_1, ..., y_{n-1})
}
  
```

Inverse FFT Summary

Theorem. Inverse FFT algorithm interpolates a degree $n-1$ polynomial given values at each of the n^{th} roots of unity in $O(n \log n)$ steps.

\n
assumes n is a power of 2

Polynomial Multiplication

Theorem. Can multiply two degree $n-1$ polynomials in $O(n \log n)$ steps.

Integer Multiplication

Integer multiplication. Given two n bit integers $a = a_{n-1} \dots a_0$ and $b = b_{n-1} \dots b_0$, compute their product $c = a \times b$.

Convolution algorithm.

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

- Form two polynomials.
- Note: $a = A(2)$, $b = B(2)$.
- Compute $C(x) = A(x) \times B(x)$.
- Evaluate $C(2) = a \times b$.
- Running time: $O(n \log n)$ complex arithmetic steps.

Theory. [Schönhage-Strassen 1971] $O(n \log n \log \log n)$ bit operations.
 [Martin Fürer (Penn State) 2007] $O(n \log n 2^{\log^3 n})$ bit operations.

Practice. [GNU Multiple Precision Arithmetic Library] GMP proclaims to be "the fastest bignum library on the planet." It uses brute force, Karatsuba, and FFT, depending on the size of n .