The Fast Fourier Transform

March 3, 2017

Polynomial representations

Representation 1: coefficients.

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

- Addition: $O(n)$
- Evaluation: $O(n)$
  Horner's method: $a_0 + x(a_1 + x(a_2 + \cdots + x(a_n) \dots))$
- Multiplication (convolution): $O(n^2)$
  $a_0 b_0 + (a_0 b_1 + a_1 b_0)x + (a_0 b_2 + a_1 b_1 + a_0 b_2)x^2 + \dots$

Polynomial representations

Theorem (Fundamental Theorem of Algebra)

Any nonzero degree-n polynomial with complex coefficients has exactly n complex roots.

Polynomial representations

Theorem (Fundamental Theorem of Algebra)

Any nonzero degree-n polynomial with complex coefficients has exactly n complex roots.

Corollary

A degree-n polynomial is uniquely specified by its value at $n + 1$ distinct x-coordinates.

Polynomial representations

Theorem (Fundamental Theorem of Algebra)

Any nonzero degree-n polynomial with complex coefficients has exactly n complex roots.

Corollary

A degree-n polynomial is uniquely specified by its value at $n+1$ distinct x-coordinates.

Proof.

If f and g are two degree-n polynomials which have the same value at each of $n+1$ distinct x-coordinates, consider the polynomial $f-g$: it has $n+1$ roots but degree $\leq n$; the only way for this to happen is if $f-g=0$, that is, $f=g$. $\square$

Polynomial representations

Representation 2: point-value

$$(x_0, f(x_0)), (x_1, f(x_1)), \ldots (x_n, f(x_n))$$

Polynomial representations

Representation 2: point-value

$$(x_0, f(x_0)), (x_1, f(x_1)), \ldots (x_n, f(x_n))$$

Addition and multiplication are easy:

$$(f+g)(x_i) = f(x_i) + g(x_i)$$
$$(fg)(x_i) = f(x_i)g(x_i)$$

Polynomial representations

Representation 2: point-value

$$(x_0, f(x_0)), (x_1, f(x_1)), \ldots (x_n, f(x_n))$$

Addition and multiplication are easy:

$$(f+g)(x_i) = f(x_i) + g(x_i)$$
$$(fg)(x_i) = f(x_i)g(x_i)$$

- Addition: $O(n)$.
- Evaluation: $O(n^2)$ (Lagrange's method)
- Multiplication: $O(n)$

Converting Between Polynomial Representations

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

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

$$a_0, a_1, \ldots, a_{n-1} \underset{?}{\overset{?}{\rightleftarrows}} (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

Converting Between Polynomial Representations

Brute Force

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

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

(**Vandermonde** matrix—invertible iff x_i distinct)

$O(n^2)$ for matrix-vector multiplication.

Coefficient to Point-Value: Intuition

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

Divide. Break polynomial up into even and odd powers.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.

Coefficient to Point-Value: Intuition

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

Divide. Break polynomial up into even and odd powers.

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

Coefficient to Point-Value: Intuition

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

Divide. Break polynomial up into even and odd powers.

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

Coefficient to Point-Value: Intuition

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

Divide. Break polynomial up into even and odd powers.

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

Coefficient to Point-Value: Intuition

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

Divide. Break polynomial up into even and odd powers.

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

Coefficient to Point-Value: Intuition

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

Divide. Break polynomial up into even and odd powers.

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

Intuition. For two points, choose ± 1.

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

Coefficient to Point-Value: Intuition

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

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

Coefficient to Point-Value: Intuition

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$
$$A(-x) = A_{\text{even}}(x^2) - xA_{\text{odd}}(x^2).$$

Intuition. For four points, choose

Coefficient to Point-Value: Intuition

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$
$$A(-x) = A_{\text{even}}(x^2) - xA_{\text{odd}}(x^2).$$

Intuition. For four points, choose $\pm 1, \pm i$.

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

Coefficient to Point-Value: Intuition

$$
\begin{aligned}
A(1) &= A_{\text{even}}(1) + 1A_{\text{odd}}(1) \\
A(-1) &= A_{\text{even}}(1) - 1A_{\text{odd}}(1) \\
A(i) &= A_{\text{even}}(-1) + iA_{\text{odd}}(-1) \\
A(-i) &= A_{\text{even}}(-1) - iA_{\text{odd}}(-1).
\end{aligned}
$$

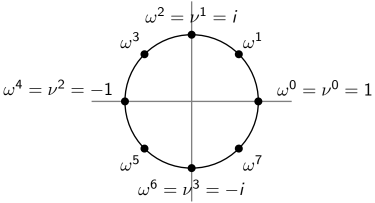
Can evaluate polynomial of degree $\leq n$ at 4 points by evaluating two polynomials of degree $\leq n/2$ at 2 points.

Roots of Unity

Definition

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

- The nth roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i/n}$.
- The $n/2$th roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = e^{4\pi i/n}$.
- $\omega^2 = \nu$.



Discrete Fourier Transform

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

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

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

(**Fourier matrix** F_n)

Fast Fourier Transform

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \cdots + a_{n-1}x^{n-1}$ at the nth roots of unity $\omega^0, \ldots, \omega^{n-1}$. (Assume n is power of 2.)

Divide. Break up polynomial into even and odd parts.

$$
\begin{aligned}
A_{\text{even}}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{n/2-2} x^{(n-1)/2} \\
A_{\text{odd}}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{n/2-1} x^{(n-1)/2} \\
A(x) &= A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).
\end{aligned}
$$

Conquer. Evaluate A_{even} and A_{odd} at $\nu^0, \ldots \nu^{n/2-1}$.

Combine. $(\omega^{k+n/2} = -\omega^k;\ \nu^k = (\omega^k)^2 = (\omega^{k+n/2})^2)$

$$
\begin{aligned}
A(\omega^k) &= A_{\text{even}}(\nu^k) + \omega^k A_{\text{odd}}(\nu^k) \qquad 0 \le k < n/2 \\
A(\omega^{k+n/2}) &= A_{\text{even}}(\nu^k) - \omega^k A_{\text{odd}}(\nu^k) \qquad 0 \le k < n/2
\end{aligned}
$$

FFT: Algorithm

Algorithm 1 FFT

Require: Size n, coefficients $a_0, a_1, \ldots, a_{n-1}$

if $n = 1$ **then**
    return a_0
end if
$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow FFT(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$
$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow FFT(n/2, a_1, a_3, a_5, \ldots, a_{n-1})$
for $k = 0$ to $n/2 - 1$ **do**
    $\omega^k \leftarrow e^{2\pi ik/n}$
    $y_k \leftarrow e_k + \omega^k d_k$
    $y_{k+n/2} \leftarrow e_k - \omega^k d_k$
end for
return $(y_0, y_1, \ldots, y_{n-1})$

FFT Summary

Theorem

The FFT algorithm evaluates a degree $n-1$ polynomial at each of the nth roots of unity in $O(n \log n)$ time (assuming n is a power of two).

Proof.

$T(n) = 2T(n/2) + O(n) \implies T(n) = O(n \log n)$. □

$$a_0, a_1, \ldots, a_{n-1} \underset{?}{\overset{O(n \log n)}{\rightleftarrows}} (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

Point-Value to Coefficient Representation: Inverse DFT

Goal. Given the values $y_0, \ldots, y_{n-1}$ of a degree $n-1$ polynomial at the n points $\omega^0, \omega^1, \ldots, \omega^{n-1}$, find the unique polynomial $a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ that fits the given points.

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

Inverse FFT!

Claim

The inverse of the Fourier matrix F_n is given by

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

Corollary

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

Inverse FFT: Proof of Correctness

Theorem

F_n and G_n are inverse.

Lemma

Let ω be a principal nth root of unity. Then

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

Proof.

- If k is a multiple of n then $\omega^{kj} = 1$.
- Every nth root of unity ω^k is a root of $x^n - 1 = (x-1)(1 + x + x^2 + \cdots + x^{n-1})$. Hence if $\omega^k \neq 1$, we have $1 + \omega^k + \omega^{k(2)} + \cdots + \omega^{k(n-1)} = 0$.

□

Inverse FFT: Proof of Correctness

Theorem

F_n and G_n are inverse.

Note $(F_n)_{ij} = \omega^{ij}$, and $(G_n)_{ij} = \frac{1}{n}\omega^{-ij}$.

Proof.

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

□

Inverse FFT: Algorithm

Algorithm 2 IFFT

Require: Size n, coefficients $a_0, a_1, \ldots, a_{n-1}$

if $n = 1$ **then**
    return a_0
end if
$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow IFFT(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$
$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow IFFT(n/2, a_1, a_3, a_5, \ldots, a_{n-1})$
for $k = 0$ to $n/2 - 1$ **do**
    $\omega^k \leftarrow e^{-2\pi ik/n}$
    $y_k \leftarrow (e_k + \omega^k d_k)/n$
    $y_{k+n/2} \leftarrow (e_k - \omega^k d_k)/n$
end for
return $(y_0, y_1, \ldots, y_{n-1})$

Inverse FFT Summary

Theorem

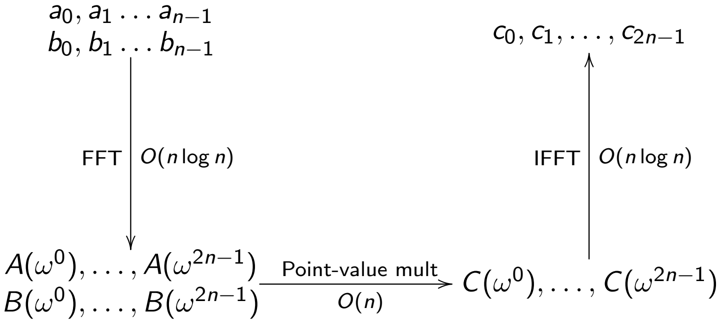
The inverse FFT algorithm interpolates a degree $n-1$ polynomial, given values at each of the nth roots of unity, in $O(n \log n)$ time.

$$a_0, a_1, \ldots, a_{n-1} \underset{O(n \log n)}{\overset{O(n \log n)}{\rightleftarrows}} (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

Polynomial Multiplication

Theorem

We can multiply two degree $n-1$ polynomials in $O(n \log n)$ time.



FFT in Practice

Fastest Fourier transform in the West.
[Frigo and Johnson, `http://www.fftw.org`]

- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

Implementation details.

- Instead of executing predetermined algorithm, it evaluates your hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- $O(n \log n)$, even for prime sizes.

Integer Multiplication

Given two n bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $c = a \times b$.
FFT algorithm.

- Form polynomials, **e.g.** $A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$.
- Note: $a = A(2)$, $b = B(2)$.
- Compute $C(x) = A(x)B(x)$ via FFT.
- Evaluate $C(2) = ab$.
- Running time: $O(n \log n)$ complex **arithmetic operations**.

Theory. [Schönhage-Strassen 1971] $O(n \log n \log \log n)$ **bit operations**.
Practice. GNU Multiple Precision Arithmetic Library (GMP) claims to be "the fastest bignum library on the planet." For multiplication it uses brute force ($O(n^2)$), Karatsuba ($O(n^{1.59})$), Toom-Cook (generalization of Karatsuba), and FFT, depending on n.