Chapter 5
Divide and Conquer

5.6 Convolution and FFT

Fast Fourier Transform: 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.

Applications.

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.

---

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*
Polynomials: Coefficient Representation

Polynomials: Point-Value Representation

Polynomials: Point-Value Representation

Evaluate: O(n) arithmetic operations.

Evaluate: O(n) using Horner’s method.

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

\[ A(x) = a_0 + (a_1 + (a_2 + \cdots + a_{n-1} + a_n)) x = a_0 + \sum_{i=0}^{n-1} a_i x^i \]

Polynomials: Coefficient Representation

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

Converting Between Two Polynomial Representations

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

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

**Conversion between two polynomial representations**

<table>
<thead>
<tr>
<th>Representation</th>
<th>Multiply</th>
<th>Evaluate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficient</td>
<td>( O(n^2) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Point-value</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>
Converting Between Two Polynomial Representations: Brute Force

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

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

\( O(n^3) \) for matrix-vector multiply

Vandermonde matrix is invertible iff \( x_i \) distinct

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

**Coefficient to point-value Representation: Intuition**

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

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

\[ A_{\text{even}}(x) = a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6. \]

\[ A_{\text{odd}}(x) = a_1 + a_3 x^3 + a_5 x^5 + a_7 x^7. \]

\[ A(x) = A_{\text{even}}(x^2) + A_{\text{odd}}(x^2). \]

**Divide.** Break polynomial up into even and odd powers.

**Intuition.** Choose four points to be \( \pm 1, \pm i \).

\[ A(1) = A_{\text{even}}(1) + 1 \cdot A_{\text{odd}}(1). \]

\[ A(-1) = A_{\text{even}}(-1) - 1 \cdot A_{\text{odd}}(-1). \]

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

\[ A(-i) = A_{\text{even}}(-1) - i \cdot A_{\text{odd}}(-1). \]

Can evaluate polynomial of degree \( \leq n \) at 4 points by evaluating two polynomials of degree \( \leq \frac{1}{2} n \) at 2 points.

**Coefficient to Point-Value Representation: Intuition**

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

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

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

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

\[ A_{\text{even}}(x) = a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6. \]

\[ A_{\text{odd}}(x) = a_1 + a_3 x^3 + a_5 x^5 + a_7 x^7. \]

\[ A(x) = A_{\text{even}}(x^2) + A_{\text{odd}}(x^2). \]

**Divide.** Break polynomial up into even and odd powers.

**Intuition.** Choose two points to be \( \pm 1 \).

\[ A(1) = A_{\text{even}}(1) + 1 \cdot A_{\text{odd}}(1). \]

\[ A(-1) = A_{\text{even}}(1) - 1 \cdot A_{\text{odd}}(1). \]

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

**Discrete Fourier Transform**

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

**Key idea:** choose \( x_k = \omega^k \) where \( \omega \) is principal \( n^{th} \) root of unity.

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
\]

\( \Downarrow \) Discrete Fourier transform

\( \Downarrow \) Fourier matrix \( F_n \)
**FFT Algorithm**

```plaintext
fft(n, a0,a1,…,an-1) {
    if (n == 1) return a0
    (e0,e1,…,en/2-1) & FFT(n/2, a0,a2,a4,…,an-2)
    (d0,d1,…,dn/2-1) & ... n/2 - 1 {
        “k & e2$ik/n
        yk+n/2 & ek + “k dk
        yk+n/2 & ek - “k dk
    }
    return (y0,y1,…,yn-1)
}
```

**Fast Fourier Transform**

**Goal.** Evaluate a degree n-1 polynomial \( A(x) = a_0 + ... + a_{n-1} x^{n-1} \) at its \( n \)th roots of unity: \( a_0, a_1, ..., a_{n-1} \).

**Divide.** Break polynomial up into even and odd powers.
- \( A_{even}(x) = a_0 + a_2 x + a_4 x^2 + ... + a_{n/2-1} x^{(n/2)-1} \).
- \( A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + ... + a_{n/2} x^{(n/2)-1} \).
- \( A(x) = A_{even}(x^2) + x A_{odd}(x^2) \).

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

**Combine.**
- \( A(\omega^k) = A_{even}(\omega^k) + \omega^k A_{odd}(\omega^k), \ 0 \leq k < n/2 \)
- \( A(\omega^{kn}) = A_{even}(\omega^k) - \omega^k A_{odd}(\omega^k), \ 0 \leq k < n/2 \)

\[ \omega^k \equiv (\omega^k)^2 \equiv (\omega^{kn})^2 \]

**Theorem.** FFT algorithm evaluates a degree n-1 polynomial at each of the \( n \)th roots of unity in \( O(n \log n) \) steps. 1 assumes \( n \) is a power of 2

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

**FFT Summary**

- \( a_0, a_1, ..., a_{n-1} \) \( \rightarrow \) \( (\omega^0, y_0), ..., (\omega^{n-1}, y_{n-1}) \)
- coefficient representation
- point-value representation
- \( O(n \log n) \)
**Inverse FFT**

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

\[
G_n = \frac{1}{n} \begin{bmatrix}
1 & 1 & 1 & 1 & \ldots & 1 \\
1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \ldots & \omega^{-(n-1)} \\
1 & \omega^{-2} & \omega^{-3} & \omega^{-4} & \ldots & \omega^{-(2n-1)} \\
1 & \omega^{-3} & \omega^{-4} & \omega^{-5} & \ldots & \omega^{-(3n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-(n-1)} & \omega^{-(n-2)} & \omega^{-(n-3)} & \ldots & \omega^{-(n(n-1)-1)}
\end{bmatrix}
\]

**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 \( n \)).

**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 unique polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \) that has given values at given points.

\[
\begin{bmatrix}
\omega^0 \\
\omega^1 \\
\omega^2 \\
\vdots \\
\omega^{n-1}
\end{bmatrix} \begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix} = \begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{bmatrix}
\]

**Inverse FFT: Proof of Correctness**

**Claim.** \( F_n \) and \( G_n \) are inverses.

**Pf.**

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

**Summation lemma.** Let \( \omega \) be a principal \( n \)th root of unity. Then

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

**Pf.**

- If \( k \) is a multiple of \( n \) then \( \omega^k = 1 \Rightarrow \text{sums to } n. \)
- Each \( n \)th root of unity \( \omega^k \) is a root of \( x^n - 1 = (x-1)(1 + x + x^2 + \ldots + x^{n-1}) \).
- If \( \omega^k = 1 \) we have: \( 1 + \omega^k + \omega^{2k} + \ldots + \omega^{(n-1)k} = 0 \Rightarrow \text{sums to } 0. \)
- If \( k \) is not a multiple of \( n \) then \( \sum_{j=0}^{n-1} \omega^{kj} = 0 \).
Inverse FFT: Algorithm

```
ifft(n, a0, a1, ..., an-1) {
    if (n == 1) return a0
    (e0, e1, ..., en/2-1) ← FFT(n/2, a0, a1, ..., an-2)
    (d0, d1, ..., dn/2-1) ← FFT(n/2, a1, a2, ..., an-1)
    for k = 0 to n/2 - 1 {
        w k ← e-2πik/n
        yk ← (ek + w k dk) / n
        yk+n/2 ← (ek - w k dk) / n
    }
    return (y0, y1, ..., yn-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.

\[a_0, a_1, ..., a_n \rightarrow (\omega^0, y_0), ..., (\omega^{n-1}, y_{n-1})\]

Polynomial Multiplication

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

FFT in Practice

- **Fastest Fourier transform in the West.** [Frigo and Johnson]
- 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.
- Core algorithm is nonrecursive version of Cooley-Tukey radix 2 FFT.
- \(O(n \log n)\), even for prime sizes.

Reference: [http://www.fftw.org](http://www.fftw.org)
**Integer Multiplication**

**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 \( c = a \times b \).

**Convolution algorithm.**

- 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.

**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 \( n \).