Polynomial + Fast Fourier Transform

Michael Tsai

2017/06/13

Polynomials

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

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

- Coefficients: $a_0, a_1, \ldots, a_{n-1}$
- Degree: highest order term with nonzero coefficient (k if highest nonzero term is a_k)
- Degree-bound: any integer strictly greater than the degree

Coefficient Representation

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad \Rightarrow \quad (a_0, a_1, \ldots, a_{n-1})$$

Vector

- Using it:
- Evaluation: Horner's rule: $\boxed{O(n)}$

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x0(a_{n-2} + x_0(a_{n-1}))\ldots))$$

- Addition: $A(x) + B(x)$ $\boxed{O(n)}$

$$(a_0, a_1, \ldots, a_{n-1})$$

+

$$(b_0, b_1, \ldots, b_{n-1})$$

$$(a_0 + b_0, a_1 + b_1, \ldots, a_{n-1} + b_{n-1})$$

Prob.: Polynomial Multiplication

- Example: $A(x) = 6x^3 + 7x^2 - 10x + 9$

  $B(x) = -2x^3 + 4x - 5$

  $C(x) = A(x)B(x)$

$$
\begin{array}{rrrrrrrl}
 & & & 6x^3 & +\ 7x^2 & -\ 10x & +\ 9 & \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 & \\
\hline
 & & & -\ 30x^3 & -\ 35x^2 & +\ 50x & -\ 45 & O(n) \\
 & & 24x^4 & +\ 28x^3 & -\ 40x^2 & +\ 36x & & O(n) \\
-\ 12x^6 & -\ 14x^5 & +\ 20x^4 & -\ 18x^3 & & & & \cdots \\
\hline
-\ 12x^6 & -\ 14x^5 & +\ 44x^4 & -\ 20x^3 & -\ 75x^2 & +\ 86x & -\ 45 & O(n)
\end{array}
$$

$$O(n^2)$$

Prob.: Polynomial Multiplication

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j \qquad c_j = \sum_{k=0}^{j} a_k b_{j-k}$$

- Degree(C)=degree(A)+degree(B)
- Degree bound: $n_a + n_b$

- $O(n^2)$!
- Problem: can we reduce this time complexity?

Point-Value Representation

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad \Rightarrow \quad \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

$$y_k = A(x_k) \qquad k = 0, 1, \ldots, n-1$$

Degree-bound: n

- Computing a point-value representation:
- Calculate each $A(x_k)$ using Horner's rule takes $O(n)$
- Thus calculate all n $A(x_k)$ values take $O(n^2)$
- If we choose the points x_k wisely, we can reduce this to $O(n \log n)$!

Interpolation

- **Interpolation** is the inverse operation of **evaluation**
- **Interpolation** is **well-defined** when the interpolating polynomial have a **degree-bound** == given **number of point-value pairs**

Theorem:
For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ of n point-value pairs such that all the x_k values are **distinct,** there is a unique polynomial A(x) of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \ldots, n-1$. (see p. 902 in Cormen for the proof)

Point-Value Representation

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

Evaluation

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

$$y_k = A(x_k) \qquad k = 0, 1, \ldots, n-1$$

Interpolation

Degree-bound: n

- Using it:
- Addition:
  since $C(x) = A(x) + B(x)$, $C(x_k) = A(x_k) + B(x_k)$

A:
$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

B:
$$\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1})\}$$

$$O(n)$$

C:
$$\{(x_0, y_0 + y'_0), (x_1, y_1 + y'_1), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1})\}$$

Point-Value Representation: Multiplication

- Multiplication:
  since $C(x) = A(x)B(x)$, $C(x_k) = A(x_k)B(x_k)$

A:
$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ n pairs

B:
$\{(x_0, y_0'), (x_1, y_1'), \ldots, (x_{n-1}, y_{n-1}')\}$ n pairs

C: n pairs
$\{(x_0, y_0 y_0'), (x_1, y_1 y_1'), \ldots, (x_{n-1}, y_{n-1} y_{n-1}')\}$

- Problem!
  We need 2n point-value pairs so that C(x) is well-defined!

Point-Value Representation: Multiplication

- Multiplication:
  since $C(x) = A(x)B(x)$, $C(x_k) = A(x_k)B(x_k)$

A:
$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\}$ 2n pairs

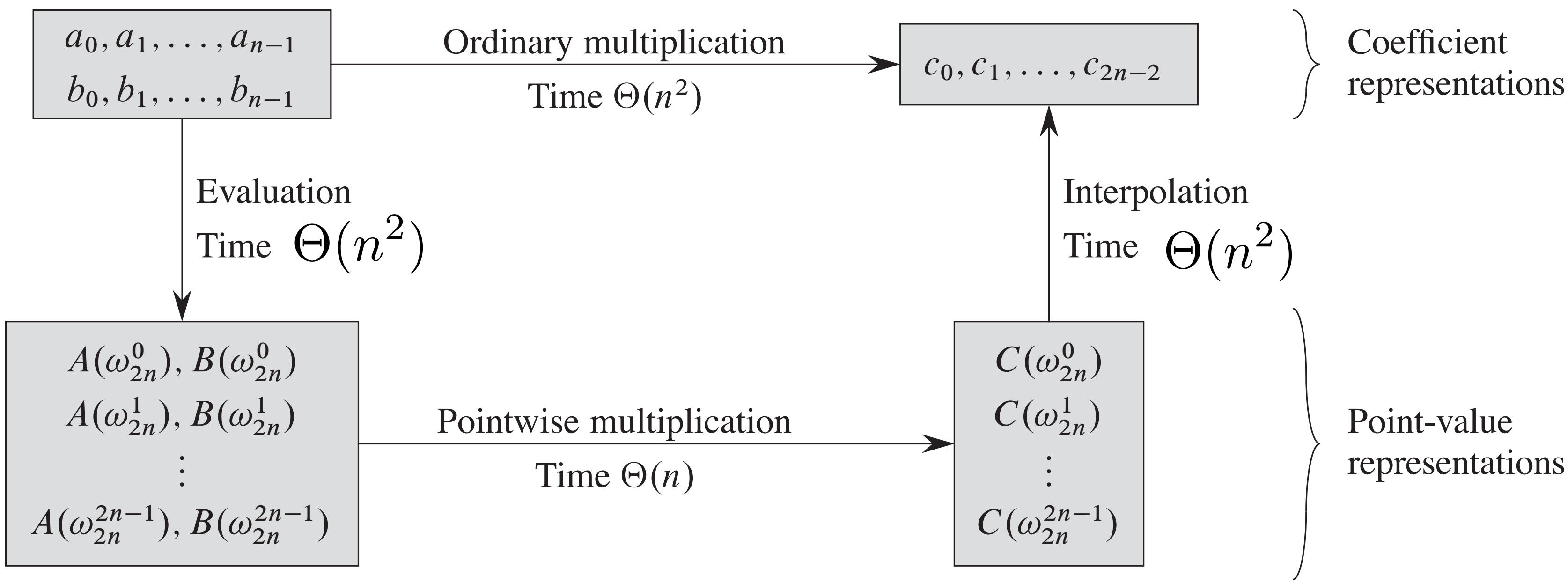
B:
$\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{2n-1}, y'_{2n-1})\}$ 2n pairs

$\boxed{O(n)}$

C: 2n pairs

$\{(x_0, y_0y'_0), (x_1, y_1y'_1), \ldots, (x_{2n-1}, y_{2n-1}y'_{2n-1})\}$

- Solution: Extend A and B to 2n point-value pairs (add n zero coefficient high-order terms)
- C(x) is now well-defined with 2n point-value pairs



Can we improve evaluation and interpolation time to

$$\Theta(n \log n)$$?

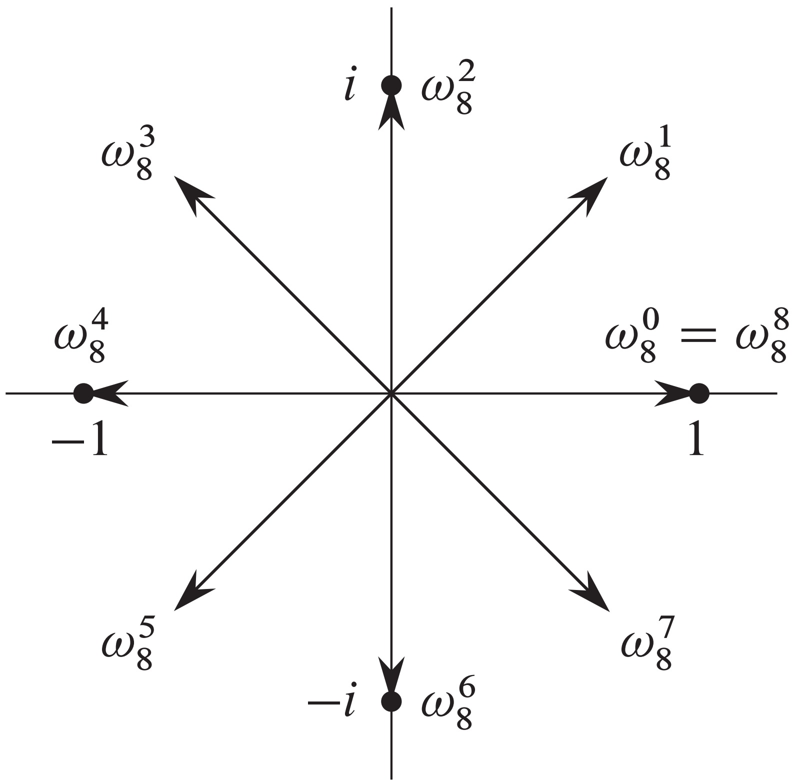
Complex Roots of Unity

- A complex n-th root of unity is a complex number ω such that
$$\omega^n = 1$$

- Exponential of a complex number:
$$e^{iu} = \cos(u) + i\sin(u)$$

- There are exactly n complex n-th roots of unity:
$$e^{2\pi ik/n} \qquad k = 0, 1, \ldots, n-1$$

Example



Principle n-th root of unity:

$$\omega_n = e^{2\pi i/n}$$

n complex n-th roots of unity:

$$\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$$

Discrete Fourier Transform

- Evaluate A(x) of degree-bound n at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$

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

$$\Rightarrow \quad y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \qquad k = 0, 1, \ldots, n-1$$

- The vector $y = (y_0, y_1, \ldots, y_{n-1})$ is the discrete Fourier Transform (DFT) of coefficient vector $a = (a_0, a_1, \ldots, a_{n-1})$

still $O(n^2)$

Physical Meaning of DFT

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$$

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}$$

Inverse DFT

Weight of that frequency

Signal at different frequencies

Fast Fourier Transform

- Taking advantage of the special properties of the complex roots of unity, we can compute DFT in $\Theta(n \log n)$!
- Assumption: n is a power of 2.
- Split A(x) into two parts:

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$$

$$A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$$

$$A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$$

$$A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2)$$

Fast Fourier Transform

$$A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$$
$$A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$$

$$A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2) \qquad (ㄅ)$$

- How do we evaluate A(x) at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$?

1. Evaluate $A^{[0]}(x)$ and $A^{[1]}(x)$ at $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_{n-1}^0)^2$
2. Combine the result using (ㄅ)

What is Divide-and-Conquer?

- When dealing with a problem:
  1. Divide the problem into smaller, but same type of, problems
  2. If the problem is small enough to solve (Conquer),
     - then solve it
     - Else recursively call itself to solve smaller sub-problems
  3. Combine the solutions of smaller sub-problems into the solution of the original, larger, problem

Base case

Recursive case

Fast Fourier Transform

$$A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$$

$$A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$$

$$A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2) \qquad (ㄅ)$$

- How do we evaluate A(x) at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$?

1. Evaluate $A^{[0]}(x)$ and $A^{[1]}(x)$ at $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_{n-1}^0)^2$

   Divide and Conquer 2 n/2-sized problems

2. Combine the result using (ㄅ)

   Combine the sub-problem solutions

$O(n \log n)$

Pseudo-code

RECURSIVE-FFT(a)

$n = a.length$ // n is a power of 2
if $n == 1$
  return a
$\omega_n = e^{2\pi i/n}$
$\omega = 1$
$a^{[0]} = (a_0, a_2, \ldots, a_{n-2})$
$a^{[1]} = (a_1, a_3, \ldots, a_{n-1})$

Divide: 2x n/2

$y^{[0]} =$ RECURSIVE-FFT$(a^{[0]})$
$y^{[1]} =$ RECURSIVE-FFT$(a^{[1]})$

Conquer

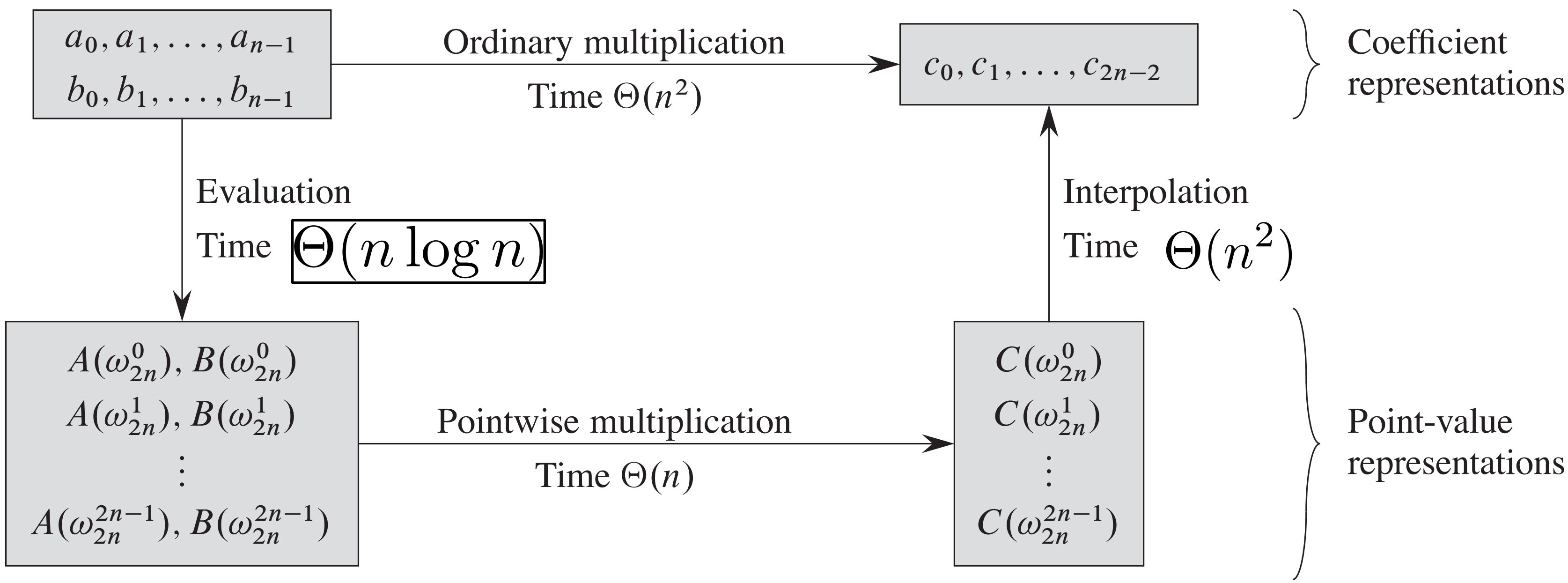
for $k = 0$ **to** $n/2 - 1$

Evaluate at ω_n^k

  $y_k = y_k^{[0]} + \omega\, y_k^{[1]}$
  $y_{k+(n/2)} = y_k^{[0]} - \omega\, y_k^{[1]}$
  $\omega = \omega\, \omega_n$

Combine $O(n)$

return y // y is assumed to be a column vector



Can we improve evaluation and interpolation time to

$$\Theta(n \log n) \text{ ?}$$

How about interpolation?? (p. 912 on Cormen)