Polynomial Multiplication: DFT

Monday, 28 August 2023 12:11 PM

I/P: $n-1$ degree polynomials $A(x)$ & $B(x)$:

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

$$B(x) = b_0 + b_1 x + b_2 x^2 + \dots + b_{n-1} x^{n-1}$$

(i.e., we are given coeffts. $a_0 \dots a_{n-1}$, $b_0 \dots b_{n-1}$)

O/P: Product $C(x) = A(x)\, B(x)$

$$= c_0 + c_1 x + c_2 x^2 + \dots + c_{2n-2} x^{2n-2}$$

where: $c_0 = a_0 b_0$

$c_1 = a_0 b_1 + a_1 b_0$

...

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}$$

Naively, time taken to compute all coeffts. of c is $O(n^2)$

Polynomials can also be represented as pt-value pairs

$A(x) \rightarrow (x_0, A(x_0)), (x_1, A(x_1)), \dots$

$B(x) \rightarrow \dots$

how many pts. are required to uniquely identify a polynomial?

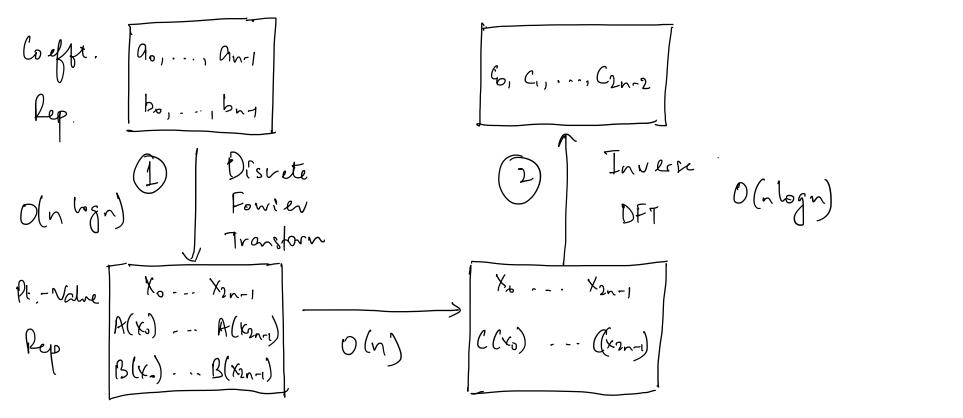
Theorem: Given n distinct pts. in $\mathbb{R}^2$ $\{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$ there is a unique $(n-1)$-degree polynomial $P(x)$, s.t. $P(x_i) = y_i$ for $i = 0 \dots n-1$.

PROBLEM 1: Prove the above theorem (you can look up Vandermonde matrices for help)

So, if we are given $A(x)$, $B(x)$ as $2n-1$ point-value pairs $\{(x_0, A(x_0)), \dots, (x_{2n-1}, A(x_{2n-1}))\}$
$\{(x_0, B(x_0)), \dots, (x_{2n-1}, B(x_{2n-1}))\}$
then can compute $C(x)$ in linear time:

$$C(x) = \{(x_0, A(x_0) B(x_0)), \dots, (x_{2n-1}, A(x_{2n-1}) B(x_{2n-1}))\}$$

But what if we want i/ps & o/ps as coeffts. of the polynomials?



Will show how to do DFT, inverse DFT in $O(n \log n)$ time

① Given coeffts of $(n-1)$-degree polynomial $A(x)$ evaluate $A(x)$ at $n-1$ distinct pts.

ASSUME: n is a power of 2.

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

Let $A^0(x) = a_0 + a_2 x + \dots + a_{n-2} x^{\frac{n}{2}-1}$

$A^1(x) = a_1 + a_3 + \dots + a_{n-1} x^{\frac{n}{2}-1}$

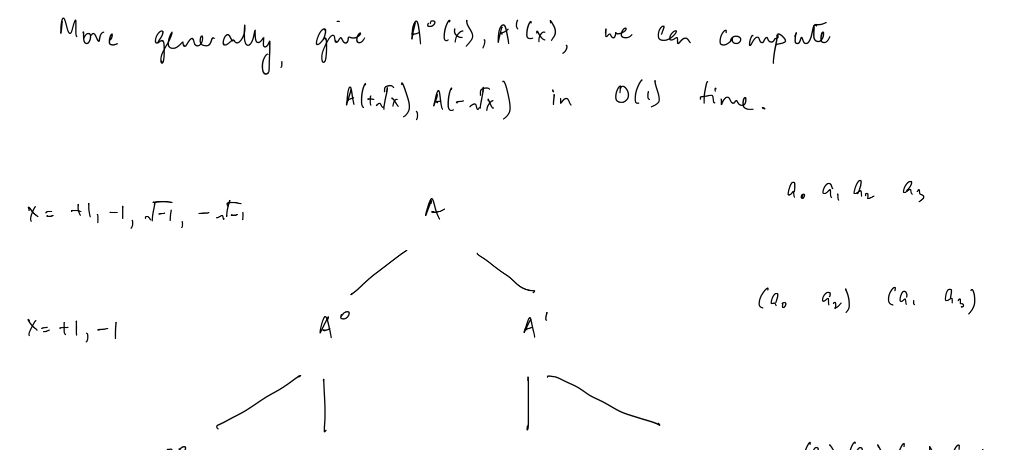
Then $A(x) = A^0(x^2) + x A^1(x^2)$

$A(1) = A^0(1) + 1 \cdot A^1(1)$

$A(-1) = A^0(1) + -1 \cdot A^1(1)$

Thus, by evaluating $A^0(x)$, $A^1(x)$ at $x = +1$, we get $A(x)$ at $x = +1$, $x = -1$!

More generally, give $A^0(x), A^1(x)$, we can compute $A(+\sqrt{x}), A(-\sqrt{x})$ in $O(1)$ time.



If A has degree $n-1$, can compute A at pts. $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$

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

$$= \cos\left(\frac{2\pi}{n}\right) + j \sin\left(\frac{2\pi}{n}\right) \qquad j = +\sqrt{-1}$$

So, A_0 & A_1 are evaluated at $\frac{n}{2}$th roots of unity

i.e., $x = \omega_{n/2}^0, \dots, \omega_{n/2}^{n/2-1}$

How do we get A from this?

$$A(\omega_n^k) = A_0(\omega_n^{2k}) + \omega_n^k A_1(\omega_n^{2k})$$

$$\& \quad A(-\omega_n^k) = A_0(\omega_n^{2k}) - \omega_n^k A_1(\omega_n^{2k})$$

$$\Rightarrow A(\omega_n^{k+n/2}) = A_0(\omega_n^{2k}) - \omega_n^k A_1(\omega_n^{2k})$$

$$k = 0 \dots \frac{n}{2} - 1$$

also $\omega_n^{2k} = e^{j \frac{2\pi}{n} \cdot 2k} = e^{j \frac{2\pi}{n/2} k} = \omega_{n/2}^k$

Thus our algo is:

```
Recursive - DFT (A)
// Takes coefft. of (n-1)-degree polynomial A, returns A
   evaluated at nth root of unity
   n = |A| - 1
   if (n == 0)  return A(0)
   A0 <- (A(0), A(2), ..., A(n-2))
   A1 <- (A(1), ..., A(n-1))
   Y0 <- Recursive-DFT (A0)
   Y1 <- Recursive-DFT (A1)
   for k = 0 ... n/2 - 1
        A(w_n^k)       = A0(w_{n/2}^k) + w_n^k A1(w_{n/2}^k)
        A(w_n^{k+n/2}) = A0(w_{n/2}^k) - w_n^k A1(w_{n/2}^k)
      // Y(k)     = Y0(k) + w_n^k Y1(k)
         Y(k+n/2) = Y0(k) - w_n^k Y1(k)
```

Running Time: $T(n) = O(n) + 2T(n/2)$

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

PROBLEM 2: Remove assumption that is a power of 2.

Inverse DFT:

Given: $(n-1)$-degree C as point-value pairs, i.e.,

For ω_n^j, $j = 0 \dots n-1$

given $C(\omega_n^j) = c_0 + c_1 \omega_n^j + c_2 \omega_n^{2j} + \dots + c_{n-1} \omega_n^{(n-1)j}$

$y_j =$

Want: $c_0, c_1, c_2 \dots c_{n-1}$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{1\cdot 1} & \omega_n^{1\cdot 2} & \dots & \omega_n^{1(n-1)} \\ 1 & \omega_n^{2\cdot 1} & \omega_n^{2\cdot 2} & \dots & \omega_n^{2(n-1)} \\ \vdots & & & & \\ 1 & \omega_n^{(n-1)1} & \omega_n^{(n-1)2} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

$\uparrow Y \qquad \uparrow V \qquad \uparrow C$

now $C = V^{-1} Y$

can see that $V_{jk} = \omega_n^{jk}$

Claim: $(V^{-1})_{jk} = \frac{1}{n} \frac{1}{\omega_n^{jk}}$

(verify yourself)

Now, to compute $V^{-1} Y$ in $O(n \log n)$ time

$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-jk} =$$

$$= \frac{1}{n} \left[y_0 + y_1 \omega_n^{-j} + y_2 \omega_n^{-2j} + \dots + y_{n-1} \omega_n^{-(n-1)j} \right]$$

Can think Y as a polynomial w/ coeffts. $y_0, y_1 \dots y_{n-1}$

Want to evaluate Y at $\omega_n^0, \omega_n^{-1}, \omega_n^{-2}, \dots, \omega_n^{-(n-1)}$

This can be done in $O(n \log n)$ time by slight modification of Recursive DFT algorithm.

PROBLEM 3: How can one change Recursive DFT algorithm to accommodate this modification?