FFT

Xudong Hang

September 10, 2022

1 FFT

The discrete Fourier transform (DFT) of the n-vector $(a_0, a_1, \cdots, a_{n-1})$ is defined as [1]

$$y_k = \sum_{j=0}^{n-1} a_j e^{i\frac{2\pi}{n}jk}. \tag{1}$$

Let n be a power of 2 throughout the discussion. Note that $y_k = y_{k+n}$. So, the value of k can be confined to one periodicity—$k = 0, 1, \cdots, n-1$ being a good choice.

Let $\omega_n = e^{i\frac{2\pi}{n}}$. It is a n-th complex root of unity. ω_n^0, ω_n^1, ω_n^2, $\cdots$, ω_n^{n-1}, are the n-th complex roots of unity. The DFT can be rewritten as

$$y_k = \sum_{j=0}^{n-1} a_j \omega_n^{jk}, \tag{2}$$

for $k = 0, 1, \cdots, n-1$.

The task of computing the DFT of the n-length vector is the following matrix multiplication

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

A total of n^2 multiplication operations are needed for this computation. However, the $n \times n$ matrix in Eq. (3) has a unique structure which the FFT algorithm takes advantage of to reduce the computation to $\Theta(n \log n)$ time.

[1] Another popular definition puts a $-$ sign on the exponent, which will be our inverse DFT.

The DFT defined in Eq. (2) can be written as the sum of two series

$$y_k = y_k^{\text{even}} + \omega_n^k y_k^{\text{odd}}, \quad k = 0, 1, \cdots, n-1, \tag{4}$$

where

$$y_k^{\text{even}} = a_0 + a_2\omega_n^{2k} + \cdots + a_{n-2}\omega_n^{(n-2)k}, \tag{5}$$

and

$$y_k^{\text{odd}} = a_1 + a_3\omega_n^{2k} + \cdots + a_{n-1}\omega_n^{(n-2)k}. \tag{6}$$

Note that these two series have the same structure. Define $b_0 = a_0$, $b_1 = a_2$, $\cdots$, $b_{n/2-1} = a_{n-2}$ and use the property of the complex roots of unity $\omega_{2n}^{2k} = \omega_n^k$, Eq. (5) can be rewritten as

$$y_k^{\text{even}} = b_0 + b_1\omega_{n/2}^k + \cdots + b_{n/2-1}\omega_{n/2}^{(n/2-1)k} = \sum_{j=0}^{n/2-1} b_j\omega_{n/2}^{jk}, \tag{7}$$

which is the exact definition of DFT in Eq. (2), but for a vector of length $\frac{n}{2}$. y^{odd} is calculated in the similar way. So, computing the DFT of a length-n vector can be transformed into computing the DFT of two length-$\frac{n}{2}$ vectors. It takes $\Theta(n)$ time to combine the results of the two smaller problems to get the solution to the original problem, as described in Eq. (4). Thus, the running time of the divide and conquer approach taken by FFT is

$$T(n) = 2T(\frac{n}{2}) + \Theta(n), \tag{8}$$

which solves to $\Theta(n \log n)$.

1.1 Inverse DFT

Given the definition of DFT in Eq. (2), for $l = 0, \cdots, n-1$, we calculate the following series

$$\begin{aligned}
\sum_{k=0}^{n-1} y_k\omega_n^{-kl} &= \sum_{k=0}^{n-1}(\sum_{j=0}^{n-1} a_j\omega_n^{jk})\omega_n^{-kl} \\
&= \sum_{j=0}^{n-1} a_j \sum_{k=0}^{n-1} \omega_n^{(j-l)k} \\
&= \sum_{j=0}^{n-1} a_j(n\delta_{jl}) \\
&= na_l.
\end{aligned}$$

So,

$$a_l = \frac{1}{n}\sum_{k=0}^{n-1} y_k\omega_n^{-kl}, \quad l = 0, 1, \cdots, n-1 \tag{9}$$

is the inverse DFT. Note that the inverse DFT differs from the DFT defined in Eq. (2) only in the constant factor n and the change from ω_n to ω_n^{-1}. So, the FFT approach can be applied to computing the inverse DFT as well.

2 Application to polynomial multiplication

A polynomial of degree $n-1$, given by

$$A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}, \tag{10}$$

can be represented by the vector of its coefficients, $(a_0, a_1, \cdots, a_{n-1})$. A polynomial of degree $n-1$ is also uniquely defined by any n distinct points $(x_0, A(x_0))$, $\cdots$, $(x_{n-1}, A(x_{n-1}))$ on it [1]. The former is the coefficient representation and the latter is the point-value representation.

Multiplying two degree-$(n-1)$ polynomials $(a_0, a_1, \cdots, a_{n-1})$ and $(b_0, b_1, \cdots, b_{n-1})$, the result is a degree-$(2n-2)$ polynomial $(c_0, c_1, \cdots, c_{2n-2})$, in which

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

for $j = 0, 1, \cdots, 2n-2$.

The naïve approach of polynomial multiplication—calculating $c_0, \cdots, c_{2n-2}$—takes $\Theta(n^2)$ time for the above problem. However, if the point-value representation is given—$2n-1$ point values needed for both polynomials—the polynomial multiplication is just $2n-1$ number multiplication operations. The result is of course in point-value representation.

The idea is, to multiply two polynomials in coefficient representation, we first transform them into point-value representation, then compute the result in point-value representation, and finally transform the result back to coefficient representation. For an easy description, we append 0s to the given vectors to make their length $2n$, that is, $(a_0, a_1, \cdots, a_{2n-1})$ and $(b_0, b_1, \cdots, b_{2n-1})$, where a_n (b_n) through $a_{2n-1}(b_{2n-1})$ are zero.

Since *any* $\geq 2n-1$ points can be chosen for the point values, we choose the $2n$ $2n$-th complex roots of unity. The point values for the polynomial in Eq. (10) are

$$A(\omega_{2n}^k) = \sum_{j=0}^{2n-1} a_j \omega_{2n}^{jk}, \tag{11}$$

for $k = 0, 1, \cdots, 2n-1$. Note that Eq. (11) is exactly the DFT defined in Eq. (2) for the vector $(a_0, a_1, \cdots, a_{2n-1})$ of length $2n$. Hence transforming from the coefficient to the point-value representation can be done with FFT in $\Theta(n \log n)$ time.

Going from the point-value to the coefficient representation is thus an inverse DFT and can be achieved by using FFT again. As a result, the polynomial multiplication is decomposed into three steps, taking $\Theta(n \log n)$, $\Theta(n)$, and $\Theta(n \log n)$ time respectively. A graphical illustration of this idea, taken from [1], is shown in Fig. 1.

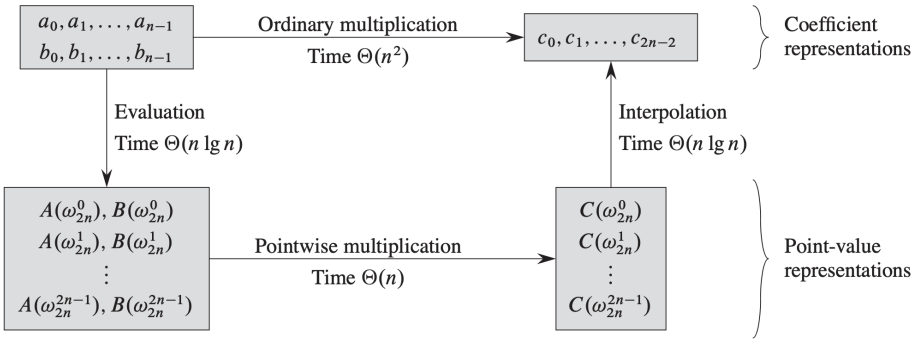

Figure 1: Efficient polynomial multiplication using FFT. Taken from CLRS.

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.