Fast Fourier Transform*

Yanheng Wang

March 27, 2024

A *polynomial* $\boldsymbol{p} = (p_0, \ldots, p_n)$ is a finite sequence of real numbers. The *function* $\boldsymbol{p}(x)$ induced by $\boldsymbol{p}$ is defined by

$$\boldsymbol{p}(x) := \sum_{i=0}^{n} p_j x^j.$$

For polynomials $\boldsymbol{a} = (a_0, \ldots, a_n)$ and $\boldsymbol{b} = (b_0, \ldots, b_n)$, we define their *sum* as $\boldsymbol{a} + \boldsymbol{b} := (a_0 + b_0, \ldots, a_n + b_n)$, and their *product* as $\boldsymbol{a} * \boldsymbol{b} := (c_0, \ldots, c_{2n})$ where $c_k := \sum_{j+j'=k} a_j b_{j'}$. It is easy to verify that

$$\begin{aligned} (\boldsymbol{a}+\boldsymbol{b})(x) &= \boldsymbol{a}(x) + \boldsymbol{b}(x) \\ (\boldsymbol{a}*\boldsymbol{b})(x) &= \boldsymbol{a}(x)\,\boldsymbol{b}(x) \end{aligned} \tag{1}$$

for all x. So polynomial addition and multiplication are in line with the usual notions of function addition and multiplication.

Multiplying two polynomials needs $\Theta(n^2)$ time if we follow the definition plainly. Can we do it faster? To this end we need an alternative representation.

Let us fix $m+1$ points $X = \{x_0, \ldots, x_m\}$. Given an arbitrary polynomial $\boldsymbol{p} = (p_0, \ldots, p_n)$ where $n \leqslant m$, we evaluate the function $\boldsymbol{p}(x)$ on X. This can be expressed as

$$\begin{pmatrix} \boldsymbol{p}(x_0) \\ \boldsymbol{p}(x_1) \\ \vdots \\ \boldsymbol{p}(x_m) \end{pmatrix} = \begin{pmatrix} 1 & x_0 & \cdots & x_0^m \\ 1 & x_1 & \cdots & x_1^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^m \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \\ \mathbf{0} \end{pmatrix} \tag{2}$$

where $\mathbf{0}$ pads the vector with $m-n$ zeros. The van der Monde matrix in the middle is invertible as $x_0, \ldots, x_m$ are distinct. Hence $(p_0, \ldots, p_n)$ and $(\boldsymbol{p}(x_0), \ldots, \boldsymbol{p}(x_m))$ uniquely determine each other, and we shall call them the *standard* and *functional* representations, respectively, of the same polynomial $\boldsymbol{p}$.

*. The note is inspired by a lecture by Erik Demaine.

In the functional representation, polynomial product becomes point-wise product; see (1). This suggests the following method for computing $\boldsymbol{a} * \boldsymbol{b} =: \boldsymbol{c}$.

- Fix $m := 2n$ and distinct points $x_0, \ldots, x_m$.
- Convert $\boldsymbol{a}$ and $\boldsymbol{b}$ to functional representations $(\boldsymbol{a}(x_0), \ldots, \boldsymbol{a}(x_m))$ and $(\boldsymbol{b}(x_0), \ldots, \boldsymbol{b}(x_m))$.
- Multiply point-wise to obtain $(\boldsymbol{c}(x_0), \ldots, \boldsymbol{c}(x_m))$, the functional representation of $\boldsymbol{c}$.
- Convert $\boldsymbol{c}$ back to its standard representation.

The "multiply" step costs merely $\Theta(n)$ time. Next we will show how to implement the conversions in $\Theta(n \log n)$ time.

We begin with the forward conversion, i.e. evaluating a polynomial $\boldsymbol{p} = (p_0, \ldots, p_n)$ at points X. A naïve implementation would incur $\Theta(n^2)$ cost. To speed it up, let us try divide-and-conquer by breaking the evaluation at $x \in X$ into odd and even parts:

$$\begin{aligned}\boldsymbol{p}(x) = \sum_{j=0}^{n} p_j x^j &= \sum_{j=0}^{n/2} p_{2j} x^{2j} + \sum_{j=0}^{n/2} p_{2j+1} x^{2j+1} \\ &= \sum_{j=0}^{n/2} p_{2j} (x^2)^j + x \cdot \sum_{j=0}^{n/2} p_{2j+1} (x^2)^j\end{aligned}$$

Hence, denoting $\boldsymbol{p}_{\text{even}} := (p_0, p_2, \ldots)$ and $\boldsymbol{p}_{\text{odd}} := (p_1, p_3, \ldots)$, we have the identity $\boldsymbol{p}(x) = \boldsymbol{p}_{\text{even}}(x^2) + x \cdot \boldsymbol{p}_{\text{odd}}(x^2)$. This immediately leads to the following algorithm:

```
fn evaluate(p, X)
  if p = (p_0) then
    return (p_0, ..., p_0)
  else
    X' := {x^2 : x ∈ X}
    p_even := (p_0, p_2, ...)
    p_odd := (p_1, p_3, ...)
    return evaluate(p_even, X') + X · evaluate(p_odd, X')
```

Let $T(n)$ denote the running time. Clearly we have the recursion

$$T(n) = 2T(n/2) + \Theta(m),$$

thus $T=\Theta(n\,m)=\Theta(n^2)$. Unfortunately it is no better than the naïve solution.

But we have a last resort. So far we did not assume any specific property of the evaluation points X. Can we craft X so that it halves in size after each recursive call?

Yes! If we set $X := \{(m+1)\text{-th roots of unity}\} \subseteq \mathbb{C}$, then after squaring, one half of the points fold into the other half, and we obtain $X' = \left\{\left(\frac{m+1}{2}\right)\text{-th roots of unity}\right\}$. With this choice, the running time recursion becomes

$$T(n+m) = 2\,T((n+m)/2).$$

Hence $T=\Theta(n \log n)$.

The divide-and-conquer algorithm invoked on such X is dubbed *Fast Fourier Transform*. Why is the name? Recall equation (2): the van der Monde matrix contains entries $x_j^k = \exp\left(\mathrm{i}\cdot\frac{2\pi jk}{m+1}\right) = \cos\left(\frac{2\pi jk}{m+1}\right) + \mathrm{i}\cdot\sin\left(\frac{2\pi jk}{m+1}\right)$, which form a Fourier basis. So evaluating $\boldsymbol{p}$ at X is equivalent to mixing the sine/cosine waves of different frequencies via coefficients $\boldsymbol{p}$. In Fourier analysis jargon, the standard representation lives in *frequency domain*, and the functional representation lives in *time domain*.

How do we convert from functional representation back to standard representation? Let us derive an inverse formula based on equation (2). We claim that, with our choice of X,

$$\begin{pmatrix} 1 & \overline{x_0} & \cdots & \overline{x_0^m} \\ 1 & \overline{x_1} & \cdots & \overline{x_1^m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{x_m} & \cdots & \overline{x_m^m} \end{pmatrix} \begin{pmatrix} 1 & x_0 & \cdots & x_0^m \\ 1 & x_1 & \cdots & x_1^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^m \end{pmatrix} = (m+1)\cdot I$$

where the bars denote complex conjugate. Indeed, we calculate the result at cell (j,k) by

$$\begin{aligned} \sum_{\ell=0}^{m} \overline{x_j^\ell}\cdot x_\ell^k &= \sum_{\ell=0}^{m} \exp\left(-\mathrm{i}\cdot\frac{2\pi j\,\ell}{m+1}\right)\cdot\exp\left(\mathrm{i}\cdot\frac{2\pi\,\ell k}{m+1}\right) \\ &= \sum_{\ell=0}^{m} \exp\left(\frac{2\,\pi\,\ell\,\mathrm{i}}{m+1}\cdot(k-j)\right). \end{aligned}$$

If $j = k$ then the result is $m+1$. Otherwise, the summands rotate around the unit circle in the complex plane and cancel each other, so we get a zero.

With the claim we derive

$$(m+1) \cdot \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} 1 & \overline{x_0} & \cdots & \overline{x_0^m} \\ 1 & \overline{x_1} & \cdots & \overline{x_1^m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{x_m} & \cdots & \overline{x_m^m} \end{pmatrix} \begin{pmatrix} \boldsymbol{p}(x_0) \\ \boldsymbol{p}(x_1) \\ \vdots \\ \boldsymbol{p}(x_m) \end{pmatrix}.$$

Now comes the punchline. If we pretend $(\boldsymbol{p}(x_0), \ldots, \boldsymbol{p}(x_m))$ as a polynomial, then $(m+1) \cdot (p_0, \ldots, p_n, \mathbf{0})$ is exactly its evaluation at points $\overline{x_0}, \ldots, \overline{x_m}$. Since $\{\overline{x_0}, \ldots, \overline{x_m}\} = X$, we can recover all information by just calling evaluate$((\boldsymbol{p}(x_0), \ldots, \boldsymbol{p}(x_m)), X)$!