1 Fast Fourier Transform (Section 6.12)

Fast Fourier Transform can be used to efficiently evaluate an interpolating trigonometric polynomial,

$$P(x_j) = f(x_j),$$

for $x_j = \frac{2\pi j}{N}$, $E_k(x) = e^{ikx}$, and $i = \sqrt{-1}$, for $0 \le j \le N-1$ (recall again that N is the number of nodes). Then, we can define P as

$$P(x) = \sum_{k=0}^{N-1} c_k E_k(x), \qquad c_k = \langle f, E_k \rangle_N = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) \overline{E_k(x_j)}. \tag{1}$$

Computing $P(x)$ (i.e., computing the c_k coefficients) without structure results in $\mathcal{O}(N^2)$ flops operations. How can we improve this runtime?

1.1 Algorithm for Power of 2 Polynomial

The algorithm we'll consider is when $N = 2^m$, with $m \in \mathbb{N}$. This algorithm for evaluating such a polynomial depends on the recursion on intermediate polynomials. In particular, we'll be working with the polynomial,

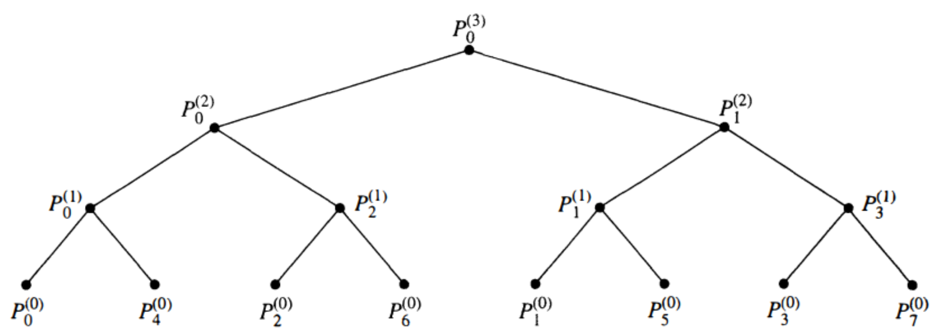
$$P_k^{(n)}(x), \qquad 0 \le n \le m, \quad 0 \le k \le 2^{m-n} - 1.$$

Keep in mind that N and m are *fixed*, but n and k are *not* fixed. In any case, the formula is given by

$$P_k^{(n+1)}(x) = \frac{1}{2}\left(1 + e^{i2^n x}\right) P_k^{(n)}(x) + \frac{1}{2}\left(1 - e^{i2^n x}\right) P_{k+2^{m-n-1}}^{(n)}\left(x - \frac{\pi}{2^n}\right).$$

Note that the superscript (m) is just an iteration counter.

(Example.) Suppose we wanted to compute $P_0^{(3)}$. This can be obtained by computing $P_0^{(2)}$ and $P_1^{(2)}$. Each of these can be obtained from the four polynomials of lower order. For example, $P_0^{(2)}$ can be obtained by $P_0^{(1)}$ and $P_{0+2^{3-1-1}}^{(1)} = P_2^{(1)}$.



1.1.1 The Idea

Each $P_k^{(n)}$ can be represented by a set of coefficients, $C_{kj}^{(n)}$, for $0 \le n \le m$ and $0 \le k \le 2^{m-n} - 1$ and $0 \le j \le 2^n - 1$. More specifically, the polynomial can be written as

$$P_k^{(n)}(x) = \sum_{j=0}^{2^n-1} C_{kj}^{(n)} E_j(x) = \sum_{j=0}^{2^n-1} C_{kj}^{(m)} e^{ijx}.$$

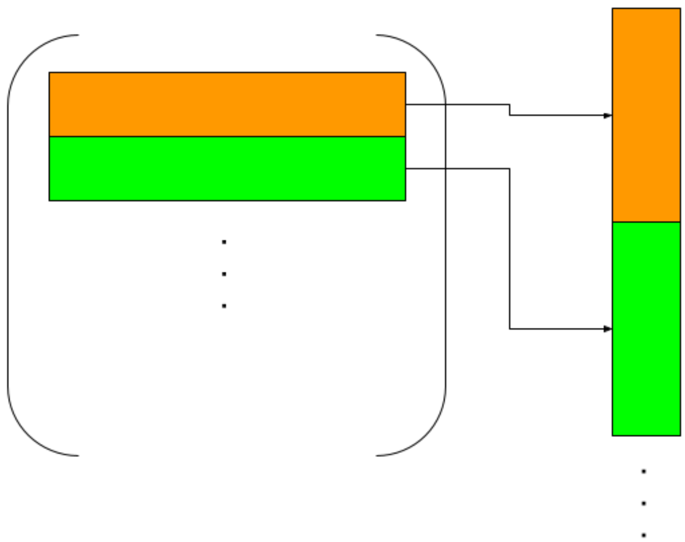
Note that

$$C_{kj}^{(n+1)} = \frac{1}{2}\left(C_{kj}^{(n)} + e^{\frac{-ij\pi}{2^n}} C_{k+2^{m-n-1},j}^{(m)}\right)$$

and

$$C^{(n+1)}_{k,j+2^n} = \frac{1}{2}\left(C^{(n)}_{kj} - e^{\frac{-ij\pi}{2^n}} C^{(m)}_{k+2^{m-n-1},j}\right)$$

So, for a fixed n where $0 \le k \le 2^{m-n} - 1$ and $0 \le j \le 2^n - 1$, we have a matrix with 2^{m-n} rows and 2^n columns. Together, there's $2^{m-n}2^n = 2^m = N$ entries. Instead, we'd like to map each matrix row to some elements in an array, something like



The long array has N elements in total. To get each element in this array, we can write

$$C(2^n k + j) = C^{(n)}_{kj} \qquad 0 \le k \le 2^{m-n} = 1, 0 \le j \le 2^n - 1$$

$$D(2^{n+1} k + j) = C^{(n+1)}_{kj} \qquad 0 \le k \le 2^{m-n-1} - 1, 0 \le j \le 2^{n+1} - 1.$$

Next, we want to precompute $Z(j)$; that is,

$$Z(j) = e^{-\frac{2\pi ij}{N}}$$

and write

$$Z(j2^{m-n-1}) = e^{-\frac{ij\pi}{2^n}}.$$

1.2 The Algorithm

With the ideas in mind, we can write the algorithm below. Note that this has inputs m and $f(x)$.

Algorithm 1 Fast Fourier Transform

function FFT(m, f)
    $N \leftarrow 2^m$
    $w \leftarrow e^{-\frac{2\pi i}{N}}$
    for $k \leftarrow 0$ to $N-1$ **do**
        $Z(k) \leftarrow w^k$
        $C(k) \leftarrow f\left(\frac{2\pi k}{N}\right)$
    end for
    for $n \leftarrow 0$ to $m-1$ **do**
        for $k \leftarrow 0$ to $2^{m-n-1} = 1$ **do**
            for $j \leftarrow 0$ to $2^n - 1$ **do**
                $u \leftarrow C(2^n k + j)$
                $v \leftarrow Z(j 2^{m-n-1}) C(2^n k + 2^{m-1} + j)$
                $D(2^{n+1} k + j) \leftarrow \frac{u+v}{2}$
                $D(2^{n+1} k + j + 2^n) \leftarrow \frac{u-v}{2}$
            end for
        end for
        for $j \leftarrow 0$ to $N-1$ **do**
            $C(j) \leftarrow D(j)$
        end for
    end for
    return C
end function

The elements in C can then be used in the interpolating formula (1).