CSC 721 Algorithms Fall 2017

Recursive Radix-2 FFT

An n^{th} principal root of unity ω satisfies:

1. $\omega^n = 1$
2. $\omega \neq 1$
3. Two equivalent properties that characterize principality of the root:
   (a) $\omega^0, \omega^1, \omega^2, ..., \omega^{n-1}$ are all distinct
   (b) For every $1 \leq p < n$, $\sum_{j=0}^{n-1} \omega^{jp} = 0$

The most often used n^{th} principal root of unity is in the complex plane:

$$\omega = e^{-2\pi\imath/n}$$

We define the $n \times n$ discrete Fourier matrix as follows:

$$F_{i,j} = \omega^{i \cdot j}$$

Let $a \in \mathcal{C}^n$ be an $n \times 1$ column vector over the complex numbers. The $n \times n$ discrete Fourier transform of a is the matrix-vector multiply:

$$b = Fa$$

There is a natural connection between the discrete Fourier transform and polynomials. Observe:

$$b_i \;=\; \sum_{j=0}^{n-1} \omega^{i \cdot j} a_j \;=\; \sum_{j=0}^{n-1} \left(\omega^i\right)^j a_j$$

We can define a polynomial whose coefficients are the entries in the vector a. Let,

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

The entries in the transformed vector b can be then written:

$$b_i \;=\; p(\,\omega^i\,)$$

We derive a radix-2 recursive divide and conquer algorithm by defining two new polynomials:

$$p_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + ... + a_{n-2} x^{n/2-1}$$
$$p_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + ... + a_{n-1} x^{n/2-1}$$

Observe that for any x:

$$p_(x) \;=\; p_{\text{even}}(x^2) \;+\; x\, p_{\text{odd}}(x^2) \tag{1}$$

Equation (1) suggests a divide and conquer approach since $p_{\text{even}}(x)$ and $p_{\text{odd}}(x)$ each have only half as many coefficients as $p(x)$. However, there are special properties of powers of an n^{th} principal root of unity that make our task even more computationally efficient.

If we list all the values of x for which we want to evaluate the polynomal $p(x)$ and compare them to x^2, we have:

x	ω^0	ω^1	ω^2	...	$\omega^{n/2-1}$	$\omega^{n/2}$	$\omega^{n/2+1}$	...	ω^{n-1}
$\mathbf{x^2}$	ω^0	ω^2	ω^4	...	ω^{n-2}	ω^n	ω^{n+2}	...	ω^{2n-2}
$\mathbf{x^2}$	ω^0	ω^2	ω^4	...	ω^{n-2}	$= \omega^0$	$= \omega^2$	...	$= \omega^{n-2}$

We see that there are only $n/2$ distinct values of x^2 because $\omega^n = 1 = \omega^0$. Therefore, we can reduce the problem of evaluating a polynomial with n coefficients at n distinct point to the sub-problems of evaluating two polynomials, each with $n/2$ coefficients, at $n/2$ distinct points. This yields the following recursive algorithm.

```
// Assume n is a power of 2.
// Assume omega is a global array of constants containing:
// [ ω^0, ω^1, ..., ω^(n-1) ]
// Input:
// a[] – array of input values of length n
// m – supplemental variable for indexing ϒ
// Output:
// b[] – output array, b = Fa
procedure fft( array a[], integer n, integer m ; array b[] )
{
    array evens ; array odds ; // Local arrays.

    if ( n == 1 ) {
        b[0] = a[0] ;
    }
    else {
        f( [ a_0, a_2, ..., a_(n-2) ], n/2, 2*m ; evens ) ;
        f( [ a_1, a_3, ..., a_(n-1) ], n/2, 2*m ; odds ) ;
        for ( j = 0 ; j < n/2 ; j++ ) {
            temp = omega[ m * j ] * odds[j] ;
            b[j] = evens[j] + temp ;
            b[j+n/2] = evens[j] - temp ;
        }
    }
```