The Fast Fourier Transform

An essential algorithm for signal processing in engineering, music, and other applications

A Senior Poster Presentation by Rebecca M. Riley, candidate for the Bachelor of Arts degree in Computer Science and Mathematics at Wellesley College, Spring 2020

INTRODUCTION

The Fast Fourier Transform was popularized by American Mathematicians J.W. Cooley and John Turkey in 1965 and is now extensively used in signal processing and numerous other computer science applications. The FFT allows us to apply the Discrete Fourier Transformation (change of bases) in $\theta(n\,log\,n)$ time instead of $\theta(n^2)$ time. In signal processing, this is often a transformation between time and frequency. This poster will use the simpler example of converting between two formats of polynomials.

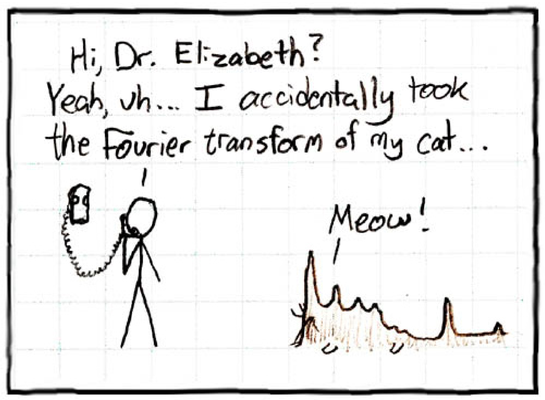
THE DISCRETE FOURIER TRANSFORM

We want to transform the polynomial $A(x)$ of degree-bound n from coefficient form to point-value form at the n complex nth roots of unity, $\omega_n^0, \omega_n^1, \dots \omega_n^{n-1}$. We can assume n is a power of 2, because the degree bound can always be raised by adding coefficients of 0 (e.g. $A(x) + 0 * x^{n+1} = A(x)$)

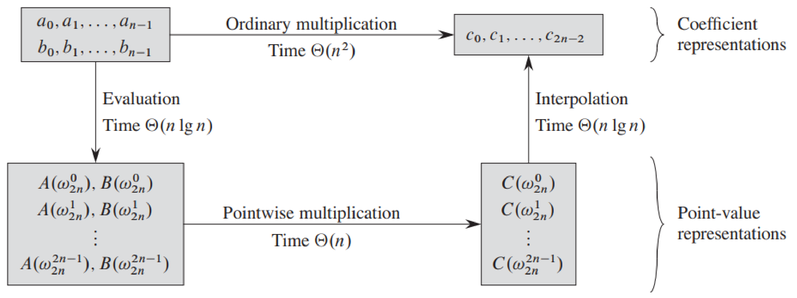
Let $y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$. As before, this operation takes n time, so calculating each $k = 0,1, \dots n-1$ will take n^2 time.

The vector $y = (y_0, y_1, \dots, y_{n-1})$ is the Discrete Fourier Transform (DFT) of the coefficient vector $a = (a_0, a_1, \dots, a_{n-1})$. We can also write $y = DFT_n(a)$.

Observe we now have a point-value form of a:
$\{(\omega_n^0, y_0), (\omega_n^1, y_1), \dots (\omega_n^{n-1}, y_{n-1})\}$



THE FAST FOURIER TRANSFORM - DIAGRAM



Corem, et. al. Introduction to Algorithms, 902

THE FAST FOURIER TRANSFORM

Apply, Divide, and Conquer!

Let $A^{[0]}(x)$ be the even indices of $A(x)$, and $A^{[1]}(x)$ be the odd indices of $A(x)$, so that

$$A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \dots + a_{(n-2)} x^{n/2-1}$$
$$A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 \dots + a_{(n-2)} x^{n/2-1}$$

Observe $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$, so we can calculate $DFT(a)$ by evaluating $A^{[0]}(x)$ and $A^{[0]}(x)$ at $(\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n-1})^2$ and combining the results.

Calculating a root of unity at each A would take $n/2$ time, as they are degree n/2, and then n combining time for a total of $\frac{n}{2} * 2 * n = n^2$ time.

However, by the Halving Lemma, the $(\omega_n^k)^2$ are not distinct, but exactly a double set of the $n/2$ complex $(n/2)$th roots of unity.

Therefore, each subproblem is half the size of the original problem, resulting in $\log(n)$ calculation time and n combining time, for a total of $nlog(n)$ time.

THE FAST FOURIER TRANSFORM - PSEUDOCODE

RECURSIVE-FFT(a) function pseudocode

```
1  n = a.length                 // a is an array representing the
                                // polynomial in coefficient form
2  if n==1
3     return a
4  ω_n = e^(2πi/d)
5  ω = 1                        // this is ω_n^0
6  a^[0]=(a_0 , a_2 ,..., a_(n-2))   // even indices of a
7  a^[1]=(a_1 , a_3 ,..., a_(n-1))   // odd indices of a
8  y^[0]= RECURSIVE-FFT(a^[0] )
9  y^[1]= RECURSIVE-FFT(a^[1] )
10 for k = 0 up to (n/2)        // combining step
11    y_k = y_k^[0] + ω y_k^[1]
12    y_k + (n/2) = y_k^[0] − ω y_k^[1]
13    ω = ωω_n
14 return y
```

Corem, et. al. Introduction to Algorithms, 911

TERMINOLOGY

- Polynomial: a polynomial of degree-bound n, $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$
  - Degree-bound n: n is greater than or equal to the degree of A
  - Coefficient form: $a = (a_0, a_1, \dots, a_{n-1})$
  - Point-Value form: $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$ such that each x_k is distinct and $y_k = A(x_k)$ for all $k = 0, 1, \dots, n-1$
    - It can be proven that for any set $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$ of n point-value pairs, there exists one unique polynomial $A(x)$ of degree less than or equal to n such that $y_k = A(x_k)$ for all $k = 0, 1, \dots, n-1$
    - Given $A(x)$ in coefficient form, for a specific x it takes n time to calculate $A(x)$. Thus, it takes n^2 to calculate the point-value form of $A(x)$
- Complex Roots of Unity
  - $i = \sqrt{-1}$
  - A complex number is a number of the form $a + bi$ or $e^{iu} = \cos(u) + i\,sin(u)$
  - A complex nth root of unity is a complex number ω such that $\omega^n = 1$
    - There are exactly n complex nth roots of unity: $e^{2\pi ik/n}$ for $k = 0,1, \dots, n-1$
    - The value $\omega_n = e^{2\pi i/n}$ $(k = 1)$ is called the *principal nth root of unity* and all other complex nth roots of unity are powers of ω_n
    - Thus, the n complex nth roots of unity are $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$
  - Note: for $a > n, a\ mod\ n = k$, $\omega_n^a = \omega_n^k$
    - Proof: Let $a > n$ and $n * j + k = a$ for some $j > 0$. Recall $e^{2\pi i} = 1$ and observe $\omega_n^a = \omega_n^{n*j+k} = \omega_n^{n*j} * \omega_n^k = e^{(2\pi i/n)*n*j} * \omega_n^k = e^{2\pi i*j} * \omega_n^k = 1^j * \omega_n^k = \omega_n^k$
  - Cancellation Lemma: for any integers $n \geq 0, k \geq 0$, and $d > 0$, $\omega_{d*n}^{d*k} = \omega_n^k$
    - Proof: $\omega_{d*n}^{k*d} = \left(e^{2\pi i/dn}\right)^{dk} = \left(e^{2\pi i/n}\right)^k = \omega_n^k$
  - Halving Lemma: If $n > 0$ is even, then the squares of the n complex nth roots of unity are the $n/2$ complex $(n/2)$th roots of unity
    - For example, there are two complex 2^{nd} roots of unity, ω_2^0 and ω_2^1. The squares of these are $(\omega_2^0)^2 = \omega_2^{0*2} = \omega_1^0$ and $(\omega_2^1)^2 = \omega_2^{1*2} = \omega_1^1 = \omega_1^0$, the single first root of unity, by the cancellation lemma.