FFT and Applications

Lecture ETC-2

Cheong-Eung Ahn

November 7, 2017

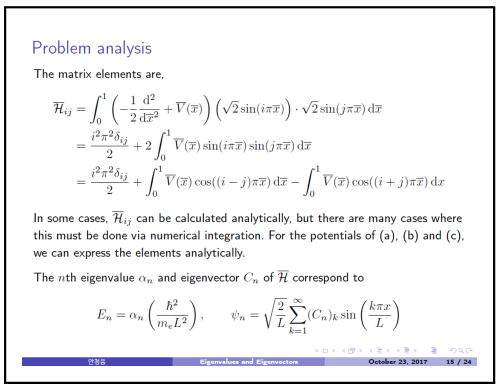
Definitions: Fourier Transform

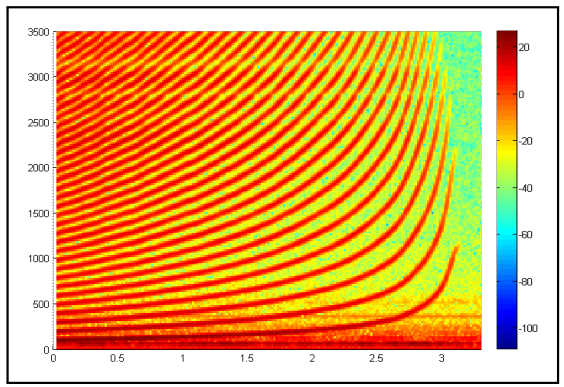
Fourier transform (FT) is the decomposition of a *function of time*, $f : \mathbb{R} \to \mathbb{C}$ to a *function of frequency* with the **integral transform**,

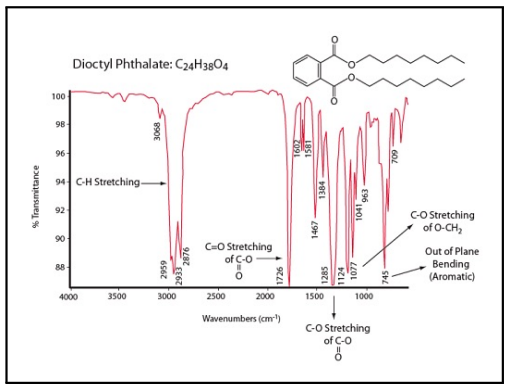
$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} \, \mathrm{d}x$$

Fourier transforms are extensively used in the solution of PDEs such as the heat equation (original development by Fourier), the wave equations of QM and QFT (Schördinger equation, Klein-Gordon-Fock equation, etc.), signal processing, spectroscopic methods such as FTIR, etc.

Examples:







Definitions: FFT

Fast Fourier transform (FFT) is often used synonymously with FT in the context of signal processing. However, it actually refers to a specific class of algorithms which computes the **discrete Fourier transform (DFT)**,

$$\mathcal{F} \ : \ X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N} = \sum_{n=0}^{N-1} x_n \left[\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)\right]$$

The **inverse discrete Fourier transform (iDFT)** can be done as

$$\mathcal{F}^{-1} \ : \ x_n = \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi nk/N}$$

The main use of FFT in context of competitve programming arises in calculating the **convolution** of two sequences. The convolution theorem shows a **duality** between the DFT and convolution.

Definitions: Convolution

The Polynomial Multiplication Problem

Given two polynomials $A(x)$ and $B(x)$, calculate $C(x) = A(x)B(x)$.

$$A(x) = \sum_{i=0}^{n-1} a_i x^i, \qquad B(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$C(x) = \sum_{k=0}^{2n-2} c_k x^k \quad \text{such that} \quad c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

Remark: We use the extension that $a_i = b_i = 0$ for $i < 0$ and $i \geq n$.

ex) Big number multiplication, generating functions, etc.

The sequence $\{c_k\}$ is the **discrete convolution** of $\{a_i\}$ and $\{b_i\}$. Another important convolution is the **cyclical convolution** where a_i for $i < 0$ and $i \geq n$ are defined as the remainder modulo n (hence the convention of $[n]$)[1].

[1] $[n] = \{0, 1, ..., n-1\}$

Convolution Theorem

The **convolution theorem** states that convolution in one domain equals pointwise multiplication in the other domain, i.e.

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}, \qquad \mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \{\{g\}$$

where $\cdot$ denotes pointwise multiplication (not polynomial multiplication). This works both for the discrete and cyclic convolutions. This theorem also holds for the inverse Fourier transform, Laplace transform, etc.

Therefore, our strategy for solving the polynomial multiplication problem is

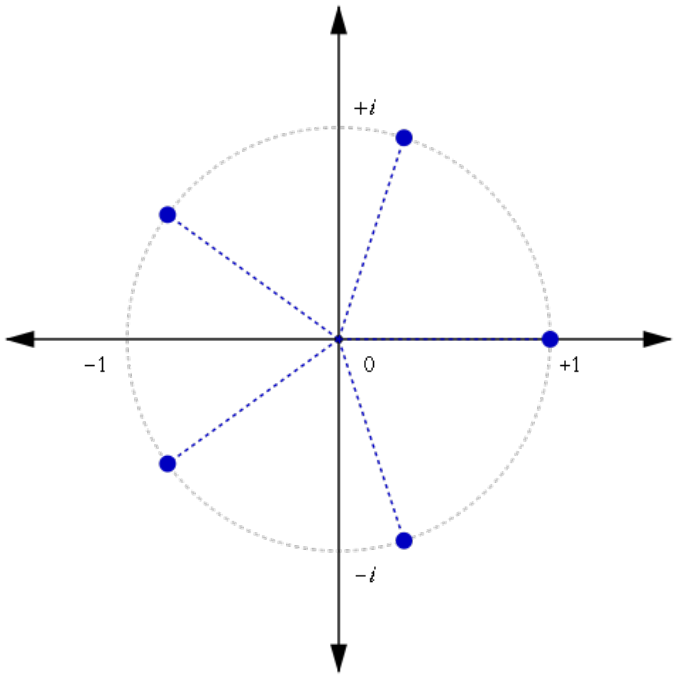
$$C(x) = A(x)B(x) \qquad \longleftrightarrow \qquad \mathbf{c} = \mathbf{a} * \mathbf{b} = \mathcal{F}^{-1}\left\{\mathcal{F}\{\mathbf{a}\} \cdot \mathcal{F}\{\mathbf{b}\}\right\}$$

As pointwise multiplication requires only $\mathcal{O}(n)$ time, the complexity is dominated by the time required for FFT which is $\mathcal{O}(n \lg n)$.

Primitive Roots of Unity in $\mathbb{C}$

Despite most input and output variables in CP problems being integers, or at the very least real, we consider $\mathbb{Z}_+ \to \mathbb{C}$ function to make use of the algebraic structure of $\mathbb{C}$.

For $\mathbb{C}$, we have the existence of **primitive roots of unity**, $\omega \in \mathbb{C}$ such that $\omega^N = 1$ and $\omega^k \neq 1$ for $k \in [N]$. For example, $\omega = e^{-i2\pi/N}$.



Divide and Conquer

Rewriting the DFT in context of roots of unity,

$$\mathcal{F} \ : \ A_i = \sum_{j=0}^{N-1} a_j \omega^{ij} = a(\omega^i), \qquad \mathcal{F}^{-1} \ : \ a_j = \sum_{i=0}^{N-1} A_i \omega^{-ij} = A(\omega^{-i})$$

where $a(x) = \sum a_k x^k$, $A(x) = \sum A_k x^k$. Trivially, we can calculate in $\mathcal{O}(n^2)$.

We consider the **radix-2 DIT Cooley–Tukey FFT algorithm** (most common implementation) which uses the *divide and conquer* paradigm to optimize.

Let $N_0 = 2N_1$. We can divide a polynomial into even and odd terms.

$$a_{0,0}(x) \quad = \quad \underbrace{\sum_{k=0}^{N_1-1} a_{2k} x^{2k}}_{=a_{1,0}(x^2)} \quad + \quad \underbrace{x \sum_{k=0}^{N_1-1} a_{2k+1} x^{2k}}_{=a_{1,1}(x^2)}$$

Assume we have the DFT of $\{a_{1,0}\}$ and $\{a_{1,1}\}$, $\{A_{1,0}\}$ and $\{A_{1,1}\}$ using the primitive root of unity $\omega_1 = e^{-i2\pi/N_1} = \omega^2$.

Divide and Conquer (cont.)

Then for $0 \leq i < N_1$,

$$A_{0,0,i} = a_{0,0}(\omega_0^i) = a_{1,0}(\omega_1^i) + \omega_0^i a_{1,1}(\omega_1^i) = A_{1,0,i} + \omega_0^i A_{1,1,i}$$

Similarly,

$$A_{0,0,N_1+i} = A_{1,0,i} - \omega_0^i A_{1,1,i}$$

The factor ω_0^i is sometimes refered to as the twiddle factor.

Therefore, if we know $\{\{A_{1,v}\}_{v=0}^{2^1-1=1}\}$ we can find $\{\{A_{0,v}\}_{v=0}^{2^0-1=0}\}$ in $\mathcal{O}(n)$.

Let us use $N = N_0 = 2^n$ and $N_p = 2^{n-p}$. We can recursively calculate,

$$\underbrace{\left\{\{A_{p,v,k}\}_{v=0}^{2^p-1}\right\}_{k=0}^{N_p-1}}_{2^p \times N^p = 2^n = N} \xleftarrow[\mathcal{O}(N)]{} \underbrace{\left\{\{A_{p+1,v,k}\}_{v=0}^{2^{p+1}-1}\right\}_{k=0}^{N_{p+1}-1}}_{2^{p+1} \times N^{p+1} = 2^n = N}$$

Therefore, overall complexity is $\mathcal{O}(nN) = \mathcal{O}(N \lg N)$.

FFT Implementation

content...

Primitive Roots of Unity in $\mathbb{Z}_p$ and NTT

One problem with DFT is that floating point arithmetic accumulates in error. One way to avoid this problem is to use the **number theoretic transform (NTT)** which considers functions $\mathbb{Z}_+ \to \mathbb{Z}_p$ not $\mathbb{Z}_+ \to \mathbb{C}$. However, this is not advisable for contests without access to notes due to suitable p being rare. For most cases, rounding will provide more than accurate results.

For $N = 2^n$, DFT uses $\omega = e^{\pm i2\pi/N}$. For NTT modulo $p = a \times 2^b + 1$, we use $\omega = (x^a)^{2^b/N}$ where x is a primitive root of p. ω is a primitive root of unity following from FLT. We must use p s.t. $b \geq n$.

Useful primes and root of unity:

Prime	a	b	Primitive root	Data type
3,221,225,473	3	30	5	64bit unsigned
2,281,701,377	17	27	3	64bit signed
2,013,265,981	15	27	31	64bit signed
469,762,049	7	26	3	64bit signed

NTT Implementation

content...

XOR Convolution

content...

Problems

Problem set by koosaga. Will select some from here and additional.

- https://www.acmicpc.net/workbook/view/824
- http://codeforces.com/problemset/tags/fft

Summary

- The Fourier transform is the decomposition of a function (sequence) in one domain to another using primitive roots of unity.
- By the convolution theorem, convolution (quadratic time) in one domain becomes pointwise multiplication (linear time) in another.
- FFT algorithms compute the DFT in $\mathcal{O}(n \lg n)$ time. Most common is the Cooley-Tukey algorithm which uses the divide and conquer paradigm.
- FFT can be generalized to various other convolutions and transforms, notably the number theoretic transform (NTT) and XOR convolution.
- For some problems, we must find the right convolution and transform to calculate the desired queries. This makes FFT problems hard to recognize.
- It is best to understand FFT/NTT conceptually and make use of team notes for the implementation.