sidhant's blog

Tutorial on FFT/NTT — The tough made simple. (Part 2)

By sidhant, history, 8 years ago,

Welcome to Part 2

Firstly, I am assuming that you have been through the Part 1 of this blog.

Okay so in hindsight I now see the drawbacks there were in my explanation of the roots of unity and how the divide and conquer works in FFT.

So in this blog I would be aiming to give you a visual intuition of what really FFT exploits over the trivial classical DFT. Also I would be covering up NTT and sharing a nice trick that I got to know about while learning NTT.

Visual Intuition of FFT $\rightarrow$

The part of converting the polynomial from coefficient form to point value form in $N \cdot \log_2 N$ instead of the trivial N^2 would be elaborated upon in this section as I find the dry mathematical explanation to be rigorous but not intuitive. In the notation below I have referred to n complex n^{th} roots of unity as powers of ${w_n}^1 = e^{2\pi \cdot i / n}$

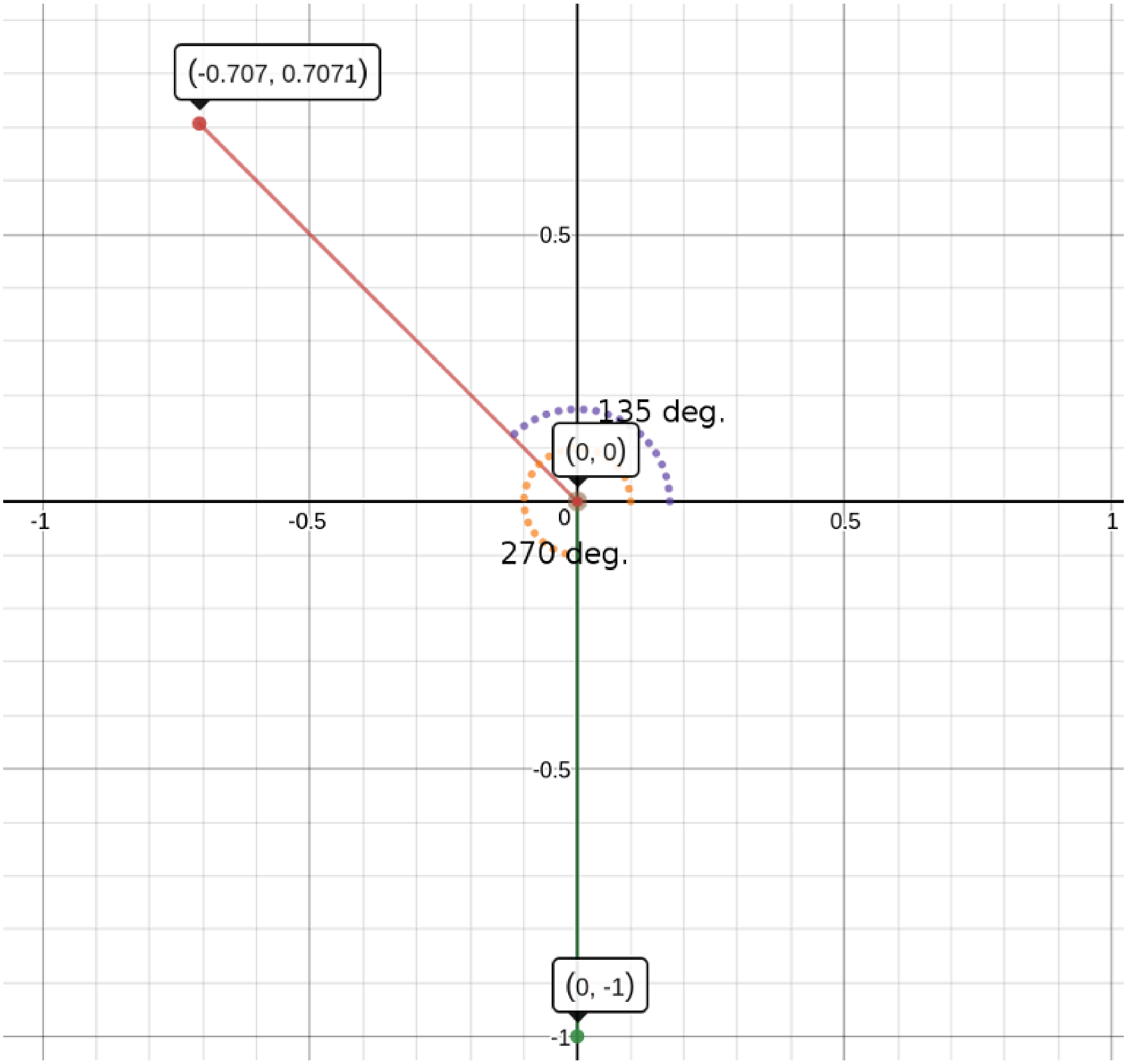
Assume that you have a 8 terms (7 degree) polynomial you need to convert from coefficient form to point value form. So firstly you choose the powers of the 8^{th} complex root of unity as the x_i's that you will be plugging in the polynomial. Initially this might seem random but hopefully by the end of this topic it would be clear to you.

Okay so now we can reduce our polynomial $A(x)$ as $A_{even}(x^2) + x{\cdot}A_{odd}(x^2)$ where $A_{even}(y)$ is a 4 terms (3 degree) polynomial in y and $A_{odd}(y)$ is also a 4 terms (3 degree) polynomial in y.
Now solving the original problem of converting $A(x)$ from coefficient form to point value form for each of the power of the 8^{th} root of unity is exactly equivalent to solving this new problem of converting $A_{even}(x^2)$ and $A_{odd}(x^2)$ for powers the 8^{th} root of unity.
So really this doesn't help us much. *But the trick* is that when we square (because x^2) these powers of the 8^{th} root of unity, they come out to be same pairwise, i.e. the 0^{th} and 4^{th} powers of the 8^{th} root of unity when squared give the same result, similarly, the 1^{st} and 5^{th} powers of 8^{th} root of unity when squared give the same result and so on.
Also note that these results actually turn out to be the powers of the 4^{th} root of unity.

Okay, so I gave the dry mathematical proof for this in the previous part, but that time I did not have a visual understanding of it. Well now I do so refer to the animations and diagrams below to get the idea but first read through the below subtopic of Rotation in Imaginary Plane.

Rotation in Imaginary Plane $\rightarrow$

Okay so as a rule of thumb remember that if you have a complex number lets say Z and you multiply it with $e^{i\theta}$ then it basically rotates it by θ angle in anti clockwise direction with respect to the origin.
So lets say you have 3^{rd} power of 8^{th} root of unity, i.e. ${w_n}^3 = e^{i\cdot 2\pi \cdot (3/8)}$ and you wish to exponentiate it. Then currently the θ is $2\pi \cdot (3/8) = 360 \cdot 3/8 = 135^\circ$.
So if you square it you are actually multiplying the θ by 2 so you are basically rotating it by θ degrees in anti clockwise direction with respect to the origin, i.e. it will now be make an angle of 270° with the positive x axis in the anti clockwise direction.
Raising it by power x is $({w_n}^3)^x = e^{i\cdot 2\pi \cdot (3/8) \cdot x} = e^{i\cdot 2\pi \cdot (3/8)} \cdot e^{i\cdot 2\pi \cdot (3/8) \cdot (x-1)}$ which is equivalent to rotating $e^{i\cdot 2\pi \cdot (3/8)}$ by $(x-1) \cdot \theta^\circ$ in anti clockwise direction with respect to positive x axis.



So given this understanding we can now easily exponentiate roots of unity. So here is the key, when you mark the 8th roots of unity on a graph it is regular octagon whose vertices are making angle $0^\circ, 45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ, 315^\circ$ degree with respect to positive x axis.
Now when you square each of these, the angles double as shown above, so they become 0 * 2, 45 * 2, 90 * 2, 135 * 2, 180 * 2, 225 * 2, 270 * 2, 315 * 2 = 0, 90, 180, 270, 360, 450, 540, 630 = $0^\circ, 90^\circ, 180^\circ, 270^\circ, 0^\circ, 90^\circ, 180^\circ, 270^\circ$ which are basically the points of the 4^{th} root of unity. Below is a beautiful animation showing the same (You would need to open the link below and play the animation from the top left hand side)

Animation here

Now you can clearly grasp the complexity of $T(n) = T(n\,/\,2)$ (For A_{even}) + $T(n\,/\,2)$ (For A_{odd}) + $O(2 \cdot n)$(For combining the results as $A_{even}(x^2) + x \cdot A_{odd}(x^2)$). So $T(n) = O(N \cdot \log_2 N)$

NTT (Number Theoretic Transform) →

The objective of NTT is to multiply 2 polynomials such that the coefficient of the resultant polynomials are calculated under a particular modulo. The benefit of NTT is that there are no precision errors as all the calculations are done in integers. A major drawback of NTT is that generally we are only able to do NTT with a prime modulo of the form $2^k \cdot c + 1$, where k and c are arbitrary constants. So for doing it for a random mod we would need to use CRT (Chinese Remainder Theorem).

Firstly, n^{th} roots of unity under a primitive field, i.e. $\mathrm{mod}\, P$ are defined as $z^n = 1 \bmod P$, where P is only considered as prime for simplicity.
Here we are assuming that $P = 2^k \cdot c + 1$, where c, k are positive integers and P is prime.

Note ⟶ All the calculation done below are done under modulo P including the operations in the exponents.
Example ⟶ $r^{P+5} = r^5$. The modulo has been intentionally skipped sometimes as it makes the notation way too ugly.

So we first find a r such that $r^x \bmod P$ goes through all the numbers from 1 to $P - 1$ when x goes from 1 to $P - 1$. Note that $r^{P-1} = 1 \bmod P$ is already a fact using Fermat's Little Theorem which states that $a^{P-1} = 1 \bmod P$ if $gcd(a, P) = 1$
After finding one such r, the $(2^k)^{th}$ root of unity under the modulo field of P will be r^c.
The powers will be as $(r^c)^0 = 1 \bmod P, (r^c)^1 \bmod P, (r^c)^2 \bmod P, ..., (r^c)^{2^k-1} \bmod P$
Notice that as $w_n{}^n = 1$ for complex roots similarly here $(r^c)^{2^k} = r^{2^k \cdot c} = r^{P-1} = 1 \bmod P$

Lemma 1 ⟶ Here $(r^c)^x \neq 1 \bmod P$ when $1 \le x < 2^k$. this is because r is itself defined as the $(P - 1)^{th}$ root of unity, i.e. any positive power of r less than $P - 1$ will not be equal to $1 \bmod P$, and as $P - 1 = 2^k \cdot c$, therefore any power of r where c is multiplied by anything less than 2^k would not give $1 \bmod P$.

Lemma 2 ⟶ Another key observation is that $(r^c)^\alpha = (r^c)^\beta$ where $\alpha \neq \beta$ and both lie between $[0, 2^k)$ can never be true, this observation can be proven by contradiction, assuming that $r^{c \cdot \alpha} = r^{c \cdot \beta}$ under $\bmod P$ then $r^{c \cdot \alpha} \cdot r^c = r^{c \cdot \beta} \cdot r^c \rightarrow (r^c)^{\alpha+1} = (r^c)^{\beta+1}$ which can be generalised to $(r^c)^{\alpha + \gamma} = (r^c)^{\beta + \gamma}$ where γ is an integer. But we already know for a fact that $(r^c)^{2^k} = 1 \bmod P$, so lets put $\gamma = 2^k - \alpha$, so now $(r^c)^{\alpha+\gamma} = (r^c)^{2^k} = (r^c)^{\beta+\gamma} = 1 \bmod P$ where $\beta + \gamma \neq 2^k$ because $\alpha \neq \beta$. This is contradictory to the result mentioned in Lemma 1, hence proved.

So, r^c is now the $(2^k)^{th}$ root of unity under the modulo field of P. Now the only difference between NTT and FFT will be that the n^{th} root of unity changes from w to r^c, where r is found by hit and trial method.

Trick to handle more prime modulos →

Instead of breaking the sub-problems as powers of 2 (into two halves), we can generalise it to powers of any number.
Then we can basically solve NTT for a prime of the form $B^k \cdot c + 1$, obviously if $B = 2$, then it gives the best time complexity and it would be slower when we opt for different and bigger B's. But it would still work which is the key.
Okay so how ?
Let's take an example for 3
So we have a polynomial of lets say $3^3 = 27$ terms, i.e. it is a 26 degree polynomial.
Now $A(x) = a_0 + a_1 \cdot x^1 + a_2 \cdot x^2 + ... + a_{26} \cdot x^{26}$
$A(x) = (a_0 + a_3 \cdot x^3 + ... + a_{24} \cdot x^{24}) + (a_1 \cdot x + a_4 \cdot x^4 + ... + a_{25} \cdot x^{25}) + (a_2 \cdot x^2 + a_5 \cdot x^5 + ... + a_{26} \cdot x^{26}) = A_{0mod3}(x^3) + x \cdot A_{1mod3}(x^3) + x^2 \cdot A$

Now the neat trick is that the complex roots not only coincide in one another when squared but also when exponentiated to any other power, in this case when cubed.

Below is an animation showing a regular 9-gon denoting the 9^{th} roots of unity and when cubed, forming an equilateral triangle denoting the 3^{rd} roots of unity.

Animation here

So basically $T(n) = 3 \cdot T(n/3) + O(2 \cdot 3 \cdot N) = 2 \cdot 3 \cdot N \cdot \log_3 N = O(3 \cdot N \cdot \log_3 N)$
Here it is $O(2 \cdot 3 \cdot N)$ as on each step of conquer for the N terms we need to do 3 multiplications i.e.
$A_{0mod3}(x^3) \cdot 1, A_{1mod3}(x^3) \cdot x, A_{2mod3}(x^3) \cdot x^2$ and 3 additions, i.e. adding all the terms together.

For a general B the time complexity will be $T(n) = B \cdot T(n/B) + O(2 \cdot B \cdot N) = 2 \cdot B \cdot N \cdot \log_B N = O(B \cdot N \cdot \log_B N)$

Note ⟶ The underlying idea behind using roots of unity in any kind of transform is that the roots of unity under any field (complex numbers or under modulo P) form in themselves a cyclic group over multiplication, i.e if any 2 elements of these roots of unity are operated using multiplication then it is guaranteed that their product will be element of this set as well. Also this set has an identity element. So the property of these roots of unity of coinciding into one another when exponentiated is satisfied because they are forming a cyclic group over multiplication and also because the size of this group is of the form B^k

Problems for Practice →

To be added.

> Thanks to **Baba** and **polingy** for assisting me in understanding the concepts mentioned in this blog and to **NibNalin** for proof reading the blog.
>
> References and resources used ⟶ Introduction to Algorithms (aka CLRS), Better Explained, Apfloat blog on NTT, Desmos Graphing Calculator
>
> Any feedback would be appreciated. Please notify me about any typos or errors in the comments or in PM.

fft, ntt, tutorial, polynomials