FFT,NTT

まつした

2021 年 8 月 13 日

1　定義とか数式とか

高速フーリエ変換（FFT) は離散フーリエ変換（DFT）を高速化したものです。離散フーリエ変換、逆変換、それらの性質等をなぞっていきます

1.1　離散フーリエ変換

n 次多項式 $f(x)$ に対して

$$\hat{f}(t) = \sum_{i=0}^{n-1} f(\zeta_n^i) t^i$$

と定義されます。ζ_nは 1 のn乗根です

1.2　1 の n 乗根

性質 1:

$$\zeta_n^i = \zeta_n^{n+i} \tag{1}$$

証明:

$$\zeta_n^i = \zeta_n^i \cdot 1 = \zeta_n^i \cdot \zeta_n^n = \zeta_n^{n+i}$$

性質 2:

$$\sum_{i=0}^{n-1} \zeta_n^{i(j-k)} = \begin{cases} n & (j \equiv k \mod n) \\ 0 & (otherwise) \end{cases} \tag{2}$$

証明:
$j \equiv k$ のとき、$j - k = mn (m : 整数)$ と書けて、

$$\sum_{i=0}^{n-1} \zeta_n^{i(j-k)} = \sum_{i=0}^{n-1} \zeta_n^{imn} = \sum_{i=0}^{n-1} (\zeta_n^n)^{mi} = n$$

そうでないとき、

$$\sum_{i=0}^{n-1} \zeta_n^{i(j-k)} = \frac{1 - (\zeta_n^{(j-k)})^n}{1 - \zeta_n^{(j-k)}} = \frac{1 - (\zeta_n^n)^{(j-k)}}{1 - \zeta_n^{(j-k)}} = 0$$

1.3 離散フーリエ逆変換

多項式 $f(t)$ を、

$$f(t) = \sum_{i=0}^{n-1} a_i t^i$$

とし、これにフーリエ変換を適用すると

$$\hat{f}(t) = \sum_{i=0}^{n-1} f(\zeta_n^i) t^i = \sum_{i=0}^{n-1} (\sum_{j=0}^{n-1} a_j \zeta_n^{ij}) t^i = \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} (\zeta_n^j t)^i$$

$t = \zeta_n^{-k}$とすると、

$$\hat{f}(\zeta_n^{-k}) = \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} (\zeta_n^{i(j-k)}) = n a_k (\because \text{式 (2)})$$

このことから、離散フーリエ逆変換は

$$f(t) = \frac{1}{n} \sum_{i=0}^{n-1} \hat{f}(\zeta_n^{-i}) t^i$$

と定義されます。

1.4 離散畳み込み

離散畳み込みは、数列 $a_0, a_1, , , , a_{n-1}$ と $b_0, b_1, b_2, , , , b_{n-1}$ から、数列 $c_0, c_1, , , , c_{n-1}$

$$c_k = \sum_{i+j=k} a_i b_j$$

を新しく生成することです。

1.5 原始根 (最後のほうで触れます)

1.5.1 定義

$1 \leqq g \leqq p-1$ の整数 g に対し、$g, g^2, , , , g^{p-2}$ のいずれも p で割った余りが 1 でないとき、g を p に対する原始根と呼ぶ。

1.5.2 性質

$1, g, g^2, g^3, , , g^{p-2}$ を $mod p$ すると、（順番は違えど）$1, 2, ...p-1$ がちょうどひとつずつ現れる。つまり、$x \equiv g^y (mod(P-1))$ となるような x と y は 1 対 1 に対応する

2　競プロにおけるフーリエ変換

競プロでは、DFT は主に離散畳み込みを行いたいときに登場します。ここで、多項式

$$f(t)=\sum_{i=0}^{n-1}a_it^i \quad g(t)=\sum_{i=0}^{m-1}b_it^i \quad (f\cdot g)(t)=\sum_{i=0}^{n+m-2}c_it^i$$

とすると、係数列 c を求めることは、係数列 a,b の畳み込みを行うことに他なりません。（以降、畳み込みを行うことを多項式 f と g の積を計算することと言い換えて説明することがあります）
ここで、多項式 f と g の積 $f\cdot g$ に対し離散フーリエ変換を行うと、

$$\begin{aligned}\widehat{(f\cdot g)}(t)&=\sum_{i=0}^{n+m-2}(f\cdot g)(\zeta_n^i)t^i\\&=\sum_{i=0}^{n+m-2}f(\zeta_n^i)g(\zeta_n^i)t^i\end{aligned}$$

と書けて、$\widehat{f\cdot g}$の係数は、$\hat{f}$と$\hat{g}$の同次の係数をただ掛けるだけで求められます（この証明は追いきれませんでした）
以上を踏まえ、フーリエ変換を用いた離散畳み込みの方法は以下の通りです

1.f と g をフーリエ変換する ($\hat{f}$ と $\hat{g}$ を求める)　→ O(nlogn)

2. 同次の係数を掛け、$\widehat{f\cdot g}$を計算する　→ O(n+m)

3.$\widehat{f\cdot g}$ をフーリエ逆変換し、$f\cdot g$ を求める　→ O(nlogn)

2 は $O(n+m)$ で計算できるので、フーリエ変換をいかに早く計算できるかが重要になることがわかります。ここで、愚直に計算すると $O(n^2)$ かかってしまうところを、$O(nlogn)$ で計算できる FFT の登場というわけです。

3 FFT

離散フーリエ変換

$$\hat{f}(t) = \sum_{i=0}^{n-1} f(\zeta_n^i) t^i$$

を高速に行うために、変換後の係数 $f(1), f(\zeta_n), f(\zeta_n^2), , ,$ を高速に求めたいです。f の係数列 $(a_i)_{i=0}^{n-1}$ を用いると、$f(\zeta_n^j)$ は

$$f(\zeta_n^j) = \sum_{i=0}^{n-1} a_i \zeta_n^{ij}$$

添え字の偶奇で分解すると、

$$f(\zeta_n^j) = \sum_{i=0}^{\frac{n}{2}-1} a_{2i} \zeta_n^{2ij} + \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} \zeta_n^{(2i+1)j}$$

$\frac{n}{2} = n'$ とすると、$\zeta_n^{2i} = \zeta_{n'}^i$ より

$$f(\zeta_n^j) = \sum_{i=0}^{n'-1} a_{2i} \zeta_{n'}^{ij} + \zeta_n^i \sum_{i=0}^{n'-1} a_{2i+1} \zeta_{n'}^{ij}$$

のように、サイズが半分になった二つのフーリエ変換に書き下せます。
$f(\zeta_n^j)$ を求めるのに $O(logn)$ ですから、FFT 全体の計算量は $O(nlogn)$ となります

4 NTT

4.1 FFT の代わり

FFT は 1 の n 乗根という複素数を使用しているので、実部、虚部ともに整数で表せないことがあるわけですが、それをプログラムで表現するとなると精度が不安な気持ちになります。
突然ですが、FFT での変換後の係数に使用されている 1 の n 乗根は、以下の性質があります。

・$\zeta_n^0, \zeta_n^1, \zeta_n^2, , , , \zeta_n^{n-1}$ の値はすべて異なる
・n 乗すると単位元になる
・式 (2) を満たす

実は、これを満たすようなものであれば、変換に使用できます。

4.2 素数 mod での FFT

$n = 2^m$ と書けるときを考えます。先ほどの要請から、$g^n \equiv 1$ となる必要があります。
フェルマーの小定理

$$g^{p-1} \equiv 1(modp)$$

とくに、$p = 2^m \times a + 1$ と書ける場合を考えると、

$$g^{2^m \times a} = (g^a)^{2^m} \equiv 1(modp)$$

よって、1 の 2^m 乗根は g^a になります。同様に、2^{m-1} 乗根、2^{m-2} 乗根、、、1 乗根も計算できます。四則演算も $modp$ では定義されているため、FFT を $modp$ で考えることができました。これが、数論変換 (NTT) になります。

4.3 NTT で使われる mod

NTT では、998244353 という素数が mod に使われることが多いです。これは

$$998244353 = 2^{23} \times 119 + 1$$

と書けるため、2^{23} 乗根まで計算することが可能です。つまり、n が $2^{23} = 8300608$ 以下となるようなものの畳み込みを計算できるというわけです。
一方、競プロで mod 演算をする際によく用いられる 1000000007 という素数は、

$$1000000007 = 2^1 \times 50000003 + 1$$

からわかるように、2 乗根までしか存在しないため、NTT には向いていません。

5 問題

5.1 AtCoder Typical Contest 001 C

AtCoder 食堂では、定食のメニューを検討している。$1 \leqq i \leqq N$ について、

・主菜は、価格が i 円のものが A_i 種類ある。

・副菜は、価格が i 円のものが B_i 種類ある。

主菜、副菜を一種類ずつ選んで、その価格の和が価格となるような定食を構成する。各 k について、価格が k 円になる定食の種類数を計算せよ。

制約　$1 \leqq N \leqq 10^5$

$A_i \times B_{k-i}$

$C_k = \sum_{i=0}^{k} A_i B_{k-i}$　→ 畳み込み

$k = 0 \sim 2N$

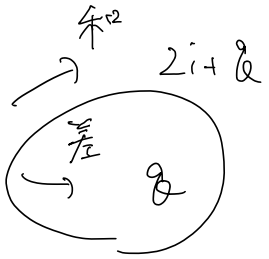
5.2 yukicoder 0206

どの二つの要素も相異なるサイズ L の数列 A と、どの二つの要素も相異なるサイズ M の数列 B が与えられる。$q = 0, 1, 2, , , , Q-1$ に対して、集合 $\{A[0], A[1], , , A[L-1]\}$ と $\{B[0]+q, B[1]+q, , , , B[M-1]+q\}$ の共通集合のサイズを求めよ。

制約 $1 \leqq Q \leqq N \leqq 10^5$

$0 \leq A[i] \leq 10^5$

$$ans_q = \sum_{i=0}^{L} A[i]B[i+q]$$



↓

Bを反転すると

$$B[N-i-q] = B[i+q]$$

$$\sum_{Sum=N-q} A[i]B[Sum-i]$$

5.3 yukicoder 931

素数 P と数列 $A[0], A[1], , , A[P-1], B[0], B[1], , , B[P-1]$ が与えられる。

$$C_k = \sum_{i \times j \equiv k \ modP} a_i b_j$$

となる数列 $C[0], C[1], , , C[P-1]$ を 998244353 で割った余りを求めよ。

5.4 AtCoder Grand Contest 047 C

$P = 200003$ とする。非負整数列 $A[0], A[1], , , A[N-1]$ に対し

$$\sum_{i<j}(A_i A_j \ \ mod\, P)$$

を求めよ。

制約 $1 \leqq N \leqq 10^5 \ \ A_i \leqq P-1$

6 参考文献

https://caddi.tech/archives/836

https://hcpc-hokudai.github.io/archive/math_fft_002.pdf

https://www.creativ.xyz/fast-fourier-transform/

https://maspypy.com/%E6%95%B0%E5%AD%A6%E3%83%BBnumpy-%E9%AB%98%E9%80%9F%E3%83%95%E3%83%BC%E3%83%AA%E3%82%A8%E5%A4%89%E6%8F%9Bfft%E3%81%AB%E3%82%88%E3%82%8B%E7%95%B3%E3%81%BF%E8%BE%BC%E3%81%BF#toc4