FFT in PS

2019-09-04 | Hard-Algorithm

목차

convolution

이 글에서는 FFT(Fast Fourier Transform, 고속 푸리에 변환)이 PS에서 주로 어떻게 사용되는지 알아보도록 하겠습니다.
FFT는 합성곱(convolution), 그중 이산 합성곱(discrete convolution)을 계산하기 위해 고안되었습니다. 이름에 Fast가 붙었으니까 빠르게 구해줍니다.

길이가 N인 두 벡터 A, B가 있다고 합시다.

- $A = (a_0, a_1, a_2, \dots, a_{N-1})$
- $B = (b_0, b_1, b_2, \dots, b_{N-1})$

두 벡터 A, B를 convolution한 C = A * B는 길이가 2N-1이며 아래와 같은 값을 갖게 됩니다.

- $C_0 = A_0B_0$
- $C_1 = A_0B_1 + A_1B_0$
- $C_2 = A_0B_2 + A_1B_1 + A_2B_0$

  …
- $C_{2N-2} = A_{N-1}B_{N-1}$

벡터 A를 다항식 $f(x) = A_{N-1}x^{N-1} + A_{N-2}x^{N-2} + \dots + A_2x^2 + A_1x^1 + A_0x^0$
벡터 B를 다항식 $g(x) = B_{N-1}x^{N-1} + B_{N-2}x^{N-2} + \dots + B_2x^2 + B_1x^1 + B_0x^0$라고 나타내면

두 백터를 convolution한 벡터 C의 성분은 두 다항식 f(x)와 g(x)를 곱한 결과의 계수들이 됩니다.

두 다항식의 곱은 Naive하게 짜도 $O(N^2)$만에 구할 수 있으니 FFT는 $O(N^2)$보다 빠르게 구해줄 것입니다. 그러니까 Fast가 붙었겠지요.

지금부터 어떻게 convolution을 빠르게 해주는지, 그리고 얼마나 빠르게 구해주는지 알아봅시다.

다항식의 표현

다항식을 나타내는 방법은 크게 두 가지로 나눌 수 있습니다.

먼저, sigma($A_i x^i$) 형태, 즉 계수와 지수들로 나타내는 coefficient representation이 있습니다.

N차 다항식은 문자가 N+1개이므로 N+1개의 서로 다른 x값에 대해 f(x)의 값을 알고 있다면 연립해서 N차 다항식을 구할 수 있습니다.
그러므로 { (x_0, f(x_0)), (x_1, f(x_1)), … , (x_N, f(x_N)) } 형태의 point-value representation으로 나타낼 수도 있습니다.

이 파트에서는 다항식을 두 가지로 나타낼 수 있다는 것만 알고가면 됩니다.

DFT

DFT라는 개념을 들고 와봅시다. DFT는 N개의 서로 다른 복소수가 주어졌을 때, 어떤 규칙에 따라 이들을 각각 N개의 다른 복소수로 변환해줍니다. 역방향으로 복원하는 것은 Inverse DFT, 즉 IDFT라고 부릅니다.

N개의 복소수가 주어질 때 어떤 규칙에 따라 변환을 해준다. 어떤 규칙이 f(x)라면 서로 다른 N개의 X_k값들이 주어졌을 때 각각의 X_k값에 대하여 f(X_k)를 계산해주겠네요!

다항식 f(x)와 g(x)를 곱한 다항식을 h(x)라고 한다면, $f(X_k) * g(X_k) = h(X_k)$입니다.

- N개의 복소수가 주어지면 어떤 규칙에 따라 변환하는 과정
- 변환된 결과를 역방향으로 복원하는 과정

위 두 가지를 빠르게 처리할 수 있다면 다항식 곱셈을 빠르게 할 수 있게 됩니다.

n-th root of unity

지금부터 어떤 0이상인 정수 p에 대해 $n = 2^p$라고 하겠습니다. 만약 다항식의 길이가 2의 멱수가 아니라면, 더 큰 2^p를 잡아서 남는 공간을 0으로 채워주면 됩니다.

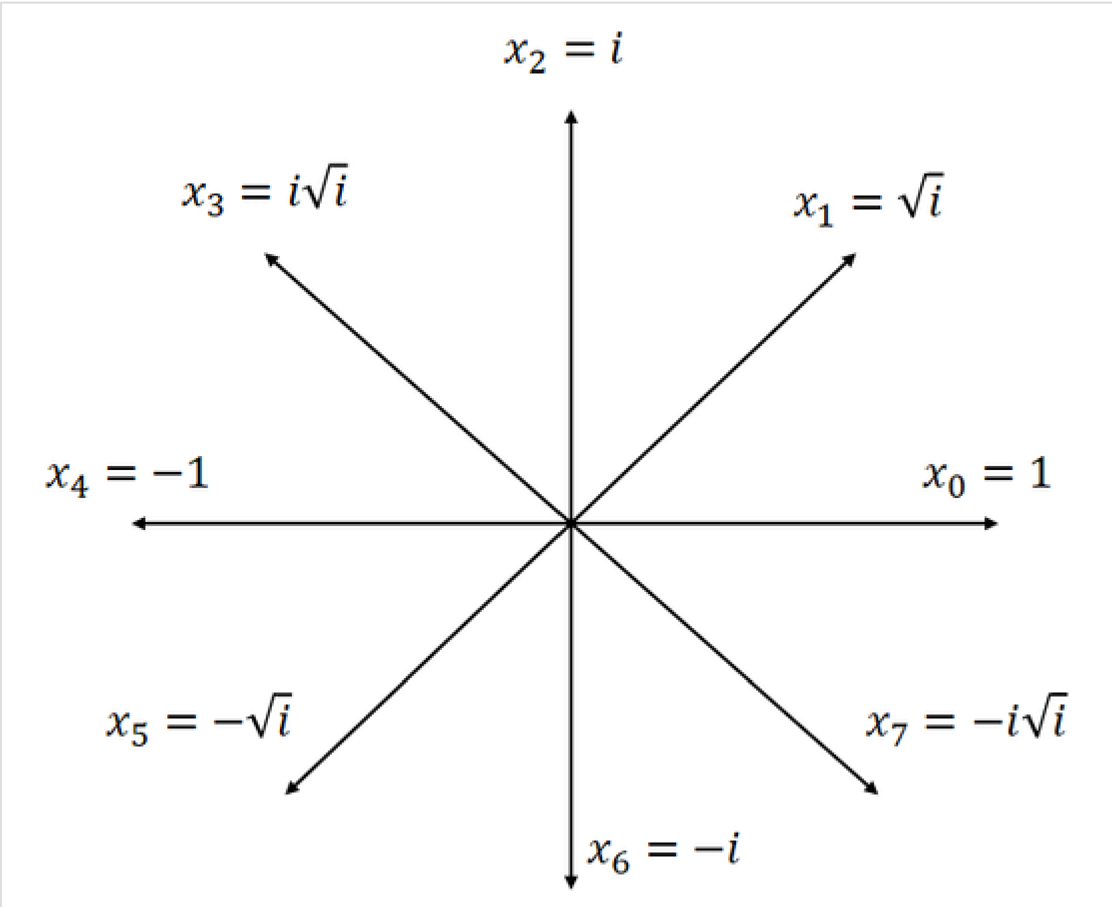
z^n = 1이 되는 서로 다른 n개의 복소수를 구해서 X_k로 쓸 것입니다.

n-th root of unity는 n승을 했을 때 1이 되는 복소수를 말합니다.

그중 특히 n보다 작은 자연수 k에 대해, k승을 하는 것으로는 1이 될 수 없는 복소수를 principal n-th root of unity 라고 합니다.

우리는 principal n-th root of unity를 X_1로 잡고, $X_k = X_1^K$라고 정의해서 계산을 해주면 N개의 n-th root of unity가 나옵니다.

이때 X_k는 오일러 공식에 의해 $e^{2k\pi i/n} = \cos(2k\pi/n) + \sin(2k\pi/n)i$가 됩니다.



n = 8인 경우의 n-th root of unity를 복소평면 위에 나타내면 위 그림과 같이 표현이 됩니다. 위 그림에서 $X_k = -X_{k+n/2}$라는 것을 알 수 있습니다.

이 파트에서는 X_k가 $e^{2k\pi i/n} = \cos(2k\pi/n) + \sin(2k\pi/n)i$인 것과, $X_k = -X_{k+n/2}$라는 것을 꼭 알고 가야합니다.

DFT와 n-th root of unity

n-th root of unity에서 나오는 $(X_0, X_1, X_2, \dots, X_{n-1})$을 $(f(X_0), f(X_1), f(X_2), \dots, f(X_{n-1}))$로 바꿔주는 작업을 DFT를 이용해 할 것입니다.

Naive하게 구현하면 함숫값을 한 번 구하는데 O(n)이 걸리니까 총 $O(n^2)$지만, FFT는 앞에 Fast가 붙으니까 더 빠르게 구해줄겁니다.

FFT의 세계로 들어가봅시다!

FFT

FFT는 분할 정복을 사용합니다.
길이가 n인 다항식을 n/2인 다항식 두 개로 쪼개는 방식으로 분할 정복을 할 것입니다.

$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_2x^2 + a_1x + a_0$
f(x)를 짝수차항과 홀수차항으로 분리해 $f_{even}(x^2) + x * f_{odd}(x^2)$라고 나타내봅시다.

- $f_{even} = a_{n-2}x^{n/2-1} + a_{n-4}x^{n/2-2} + ... + a_4x^2 + a_2x + a_0$
- $f_{odd} = a_{n-1}x^{n/2-1} + a_{n-2}x^{n/2-2} + ... + a_5x^2 + a_3x + a_1$

이런 식으로 n이 짝수라면 길이 n짜리 다항식을 길이 n/2짜리 다항식 두 개로 쪼갤 수 있습니다.

어떤 복소수 w와 -w를 뽑아서 함숫값을 계산해봅시다.

- $f(w) = f_{even}(w^2) + w * f_{odd}f(w^2)$
- $f(-w) = f_{even}(w^2) - w * f_{odd}f(w^2)$

$f_{even}(w^2)$와 $f_{odd}f(w^2)$의 값을 할고 있다면 f(w)와 f(-w)를 바로 구해낼 수 있고, $w = X_k$라고 하면 $-w = X_{k+n/2}$입니다.
n개의 X_k 중 n/2개만 구한다고 하면, 반대편에 있는 나머지 n/2개의 $X_{k+n/2}$는 동시에 구해줄 수 있습니다.

길이 n짜리 다항식을 두 개의 길이 n/2짜리 다항식으로 분해하는 과정과 n/2쌍의 복소수의 값을 구하는 과정 모두 O(n)에 가능합니다.
T(n) = 2T(n/2) + O(n)이 되고, 결국 FFT는 O(n log n)에 동작하게 됩니다.

```
typedef complex<double> cpx;
typedef vector<cpx> vec;

const double pi = acos(-1);

/*
input : f => Coefficient, w => principal n-th root of unity
output : f => f(x_0), f(x_1), f(x_2), ... , f(x_n-1)
T(N) = 2T(N/2) + O(N)
*/
void FFT(vec &f, cpx w){
	int n = f.size();
	if(n == 1) return; //base case
	vec even(n >> 1), odd(n >> 1);
	for(int i=0; i<n; i++){
		if(i & 1) odd[i >> 1] = f[i];
		else even[i >> 1] = f[i];
```

```
        }
        FFT(even, w*w); FFT(odd, w*w);
        cpx wp(1, 0);
        for(int i=0; i<n/2; i++){
                f[i] = even[i] + wp * odd[i];
                f[i+n/2] = even[i] - wp * odd[i];
                wp = wp * w;
        }
}
```

IDFT

위쪽 FFT 파트에서 구한 값들은 DFT한 결과이므로 $\{ h(X_0), h(X_1), \ldots, h(X_{n-1}) \}$입니다. 우리의 목표는 다항식 곱셈이므로 n개의 함숫값이 주어졌을 때 n-1차 다항식을 복원해야 합니다.

위키피디아의 "Inverse Transform"문단을 보면 X_1의 켤레 복소수를 넣고 돌려서 나온 결과를 n으로 나누면 된다고 하므로 그대로 하면 됩니다.

```
/*
input : a => A's Coefficient, b => B's Coefficient
output : A * B
*/
vec mul(vec a, vec b){
        int n = 1;

  //a * b 결과의 길이보다 길거나 같은 2의 멱수 찾기
        while(n <= a.size() || n <= b.size()) n <<= 1;
        n <<= 1;
        a.resize(n); b.resize(n); vec c(n);

  //principal n-th root of unity
        cpx w(cos(2*pi/n), sin(2*pi/n));

  //a와 b의 dft구하기
        FFT(a, w); FFT(b, w);

  //f(x) * g(x) = h(x)
        for(int i=0; i<n; i++) c[i] = a[i] * b[i];

  //켤레 복소수로 dft -> idft
        FFT(c, cpx(w.real(), -w.imag()));
        for(int i=0; i<n; i++){
                c[i] /= cpx(n, 0);
                //c[i] = cpx(round(c[i].real()), round(c[i].imag())); 만약 정수 결과를 원
        }
        return c;
}
```

예제) 큰 수 곱셈

10진수는 x가 10인 다항식 형태로 바꿀 수 있습니다.
그러므로 다항식 곱셈같은 느낌으로 O(N log N)에 해줄 수 있습니다. (N은 숫자의 길이)

코드

FFT # Math

〈 백준13161 분단의 슬픔

백준13358 Exponial 〉