Following article is mainly from CLRS. introduction to algorithm
The purpose of this article is just learning

1 다항식과 FFT

차수가 n인 두 다항식을 더하는데에 필요한 복잡도는 $\Theta(n)$이다. 그러나 두 다항식을 곱하는데에는 $\Theta(n^2)$이 소요된다. **Fast Fourier Transform, FFT** 알고리즘을 이용해 두 다항식의 곱을 $\Theta(n \log n)$ 복잡도에 수행하는 것을 알아 볼 것이다.

Fourier trasnform은 신호 처리에서 가장 많이 사용된다. 신호란 정의역은 시간으로, 시간에서 진폭으로 맵핑하는 함수다. **해석 불가. FFT에 대해 더 이해하면 이해 가능, 그때 추가 하겠음.**

두 다항식 $A(x)$와 $B(x)$가 존재한다고 하자. 두 항의 곱을 $C(x)$라 할 때 $C(x)$의 최대 차수는 다항식 A와 B의 차수 합인 $deg(A) + deg(B)$이다. 아래는 $A(x) = 6x^3 + 7x^2 - 10x + 9$, $B(x) = -2x^3 + 4x - 5$일 때 연산법을 나열한 것이다.

$$
\begin{aligned}
A(x) &= 6x^3 + 7x^2 - 10x + 9 \\
B(x) &= -2x^3 + 4x - 5 \\
C(x) &= A(x)B(x) \\
&= (-12x^6 - 14x^5 + 20x^4 - 18x^3) \\
&+ (24x^4 + 28x^3 - 40x^2 + 36x) \\
&+ (-30x^3 - 35x^2 + 50x - 45) \\
&= ... \qquad (1)
\end{aligned}
$$

$C(x)$는 정형적으로 다음과 같이 표현할 수 있다. 이때 $A(x)$와 $B(x)$는 각 각 항이 n개라 하자.

$$C(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ and } c_i = \sum_{j=0}^{i} a_j b_{i-j} \qquad (2)$$

앞으로 **coefficient representation**과 **point-value representation**에 대해 이야기 해볼 것이다. 전자의 연산량은 앞서 (2)와 같이 $\Theta(n^2)$이다. 그리고 후자는 $\Theta(n)$ 복잡도를 가진다. 이 두 표현법을 바꿔가며 연산하면 앞서 $\Theta(n^2)$으로 표현할 수 있던 표현식을 $\Theta(n \log n)$ 복잡도에 풀어낼 수 있다. 이를 수행 하기 위해선, 제일 먼저 1의 거듭제곱근**complex root of unity**에 대해 학습해야한다. 그리고 FFT를 사용한 변환과 역변환도 이야기 해볼 것이며, FFT를 어떻게 빠르게 구현하는지에 대해서도 다룰 것이다. 이후로는 복소수를 빈번히 사용하기에, $sqrt{-1}$를 줄여 기호 i라고 표현하겠다.

1.1 다항식을 표현하는 법

다항식을 표현하는데엔 두 가지 방법이 있다. coefficient 와 point-value. 이 섹션에서 어떻게 두 표현식을 이용하여 $\Theta(n \log n)$ 복잡도에 두 다항식을 곱할 수 있는지 보여줄 것이다.

Coefficient Representation

coefficient representation은 다항식을 다음과 같이 표현하는 것이다.

$$A(x) = \sum_{i=0}^{n-1} a_i x^i, \text{ where } deg(A) = n$$

각 항의 계수를 다음과 같이 표현할 수 있다. $(a_0, a_1, a_2, \ldots, a_{n-1})$ 이러한 계수들의 집합을 벡터라고도 표현할 수 있다.
두 다항식 $A(x)$와 $B(x)$가 있을 때, 두 다항식의 합은 $\Theta(n)$에 수행될 수 있다. 또한, 다항식 $A(x)$에서 $x = x_0$일 때의 값도 $\Theta(n)$에 도출할 수 있다. 하지만 두 다항식의 곱은 그것보단 더 복잡하다. 두 다항식의 곱을 통해 생성되는 다항식 $C(x)$의 계수 벡터는 $(c_0, c_1, ..., c_{n-1})$이다. 우리는 앞서 $c_i = \sum_{j=0}^{i} a_j b_{i-j}$임을 보았다. 이러한 c 계수 벡터는 벡터 a와 b의 **convolution**라고도 불린다. $c = a \otimes b$. 다항식들을 곱하는 것이나 convolution을 연산하는 것은 상당히 실용적인 중요도가 있는 기초적인 컴퓨터 문제들이다.

Point-value representation

다항식 $A(x)$의 point-value 표현법을 이용하면 다음과 같이 표현할 수 있다.

$$\begin{aligned} &(x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1}) \\ &\text{where } x_i \neq x_j \text{ and } y_i = A(x_i)(0 \leq i, j \leq n-1, x_i \neq x_j) \end{aligned} \tag{3}$$

point-value란 $A(x)$ 다항식이 x가 특정 지점일 때의 값 즉, $A(x_i)$ 값을 n개를 추출하여 그것을 이용하는 것이다.
point-value set을 구하는 데엔 n개의 $A(x_i)$ 결과값을 도출해야하는데, 문제는 하나의 결과값을 구하는데에만 $\Theta(n)$ 복잡도가 소요되는 것이다. 총 n개의 결과값은 $\Theta(n^2)$ 복잡도를 형성한다. 미리 힌트를 주자면, 이 때 구하고자 하는 지점 x_i를 현명하게 선택할 수 있으면 이러한 연산이 $\Theta(n \log n)$이 소요된다.

Theorem 30.1 Uniqueness of an interpolating polynomial.
만약 임의의 set $(x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1})$, where $x_i \neq x_j$ 이 존재할 때, n 항의 다항식 $A(x)$가 고유하게 결정될 수 있다.
proof
증명은 특정한 행렬의 역행렬이 존재한다는 것에 기반한다. 수식 (3)을 행렬로 표현하면 다음과 같다.

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} \tag{4}$$

맨 왼쪽의 행렬은 간략하게 $V(x_0, x_1, ..., x_{n-1})$로 표현할 수 있는데, *Vandermonde matrix* 라고 불린다. 이 행렬의 determinant는 아래와 같다.

$$\prod_{0 \le j < k \le n-1} (x_k - x_j)$$

x_k가 모두 고유하다면, 해당 행렬은 역행렬을 구할 수 있다. [1] 따라서 우리는 계수 벡터 a에 대해 무엇인지 알 수 있다.

$$a = V(x_0, x_1, ..., x_{n-1})^{-}1y \tag{5}$$

이 a를 도출하는데엔 연산량 $\Theta(n^3)$이 필요하다. 좀더 빠른 알고리즘은 ***Lagrange's formula***를 이용하는 방법이다. Lagrange's formula는 (3)과 같은 집합이 주어졌을 때 해당 조건을 만족하는 다항식을 보간법으로 구해내는 것이다. 즉, (5)에서 a를 구하는 것과 같은 것이다. 아래는 Lagrange's formula이다.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \ne k}(x - x_j)}{\prod_{j \ne k}(x_k - x_j)}$$

위의 복잡도는 $\Theta(n^2)$이다. Lagrange's formula를 이용하여 계수를 구하는 것은 문제로 남겨두었다.

이제까지 이야기한 것을 정리하자면, n 지점에서 다항식의 값을 구하는 것과 다항식을 보간하는 법은 $\Theta(n^2)$이 걸린다.

point-value 방식의 관점은 다항식을 연산할 때 이점이 있다. 다음 예를 보자. $C(x) = A(x) + B(x)$라 하자. 그러면 $C(x_a) = A(x_a) + B(x_a)$이다. 이를 point-value 방식으로 표현하면 다음과 같다.

$$A = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_{n-1}, y_{n-1})\}$$
$$B = \{(x_0, y'_0), (x_1, y'_1), (x_2, y'_2), ..., (x_{n-1}, y'_{n-1})\}$$
$$C = \{(x_0, y_0 + y'_0), (x_1, y_1 + y'_1), (x_2, y_2 + y'_2), ..., (x_{n-1}, y_{n-1} + y'_{n-1})\}$$

최대 항이 n 개 인 두 다항식을 더하는 연산이 $\Theta(n)$ 복잡도임을 더 명확히 알 수 있다.

[1]determinant를 구하는 방법이나, 해당 조건을 만족할 때 역행렬이 존재한다는 정확한 증명은 CLRS의 부록의 D-1 problem 과 themorem D.5를 참조하시오.

두 다항식의 곱을 살펴보자. $C(x) = A(x)B(x)$라 하자. 그렇다면 $C(x_k) = A(x_k)B(x_k)$이다. 이 때, 각 항이 n개가 아닌 $2n$개라고 고려해보자.

$$A = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_{2n-1}, y_{2n-1})\}$$
$$B = \{(x_0, y'_0), (x_1, y'_1), (x_2, y'_2), ..., (x_{2n-1}, y_{2n-1})\}$$
$$C = \{(x_0, y'_0 + y'_0), (x_1, y'_1 + y'_1), (x_2, y'_2 + y'_2), ..., (x_{n-1}, y'_{n-1} + y'_{n'-1})\}$$