

MODULE 2 : FOURIER TRANSFORMS

Today: Fast Integer multiplication,
Fast polynomial multiplication,
(FAST FOURIER TRANSFORM)

Today's Reference

- Brent Zimmerman's "Modern Computer Arithmetic"
- Knuth's "Art of Computer Programming II", 4.3.3

Sample question: How much time does it take to add two n -bit numbers?

Bit representation: $A = \sum_{i=0}^{n-1} a_i 2^i$, $B = \sum_{i=0}^{n-1} b_i 2^i$

(Typical $n \rightarrow 32, 64, \text{etc}$)

How do you count time here? Well, that depends on "word size".

Adding 1-bit = 1 time unit (word size = 1)
then adding A & B takes time $O(n)$.

Remark: If you took $w = \log n$ (say) - bit sized words (with 1 time unit = Adding 2 w -bit words) then your time is $O\left(\frac{n}{\log n}\right)$

→ this basically means the numbers are represented in base n instead of base 2. But to make this base transformation → you will spend $O(n)$ -time.

Now, Question of the day: How long does it take to multiply 2 n-bit integers?

Base-2 or Base-10 (which you study in school) doesn't matter $\rightarrow$ time you saw was $O(n^2)$

Base-10 $\leftarrow$ Examples $\rightarrow$ Base-2

$$\begin{array}{r} 12 \\ \times 13 \\ \hline 36 \\ 12 \\ \hline 156 \end{array}$$

$n=2$ #operations = $O(n^2)$

$$\begin{array}{r} 1100 \\ \times 1101 \\ \hline 1100 \\ 0000 \\ 1100 \\ 1100 \\ \hline 10011100 \end{array}$$

$n=4$
#operations = $O(n^2)$

What we will see today: use Discrete Fourier Transform (DFT) to do this in $O(n \log n)$ ops.

What about other important operations you need?

- $\rightarrow$ Integer division
- $\rightarrow$ Square root, kth root
- $\rightarrow$ Extended GCD
- $\rightarrow$ Base conversion
- $\rightarrow$ Primality testing?

}

Important operations often used in crypto amongst other areas.

(Refer to Brent-Zimmerman)
All can be expressed in terms of multiplication time.

Discrete Fourier Transform

Multiplying integers same as multiplying two polynomials $\rightarrow$ common operation is convolution

$$\begin{aligned} A &= \sum_{i=0}^{n-1} a_i 2^i & B &= \sum_{i=0}^{n-1} b_i 2^i & (\text{numbers}) \\ \left\{ \begin{array}{l} \rightarrow P(x) = \sum_{i=0}^{n-1} a_i x^i \\ \rightarrow Q(x) = \sum_{i=0}^{n-1} b_i x^i \end{array} \right. & & & & (\text{polynomials}) \end{aligned}$$

Multiplying either of them involves computing

Convolution

$$C_k = \sum_{(i,j): i+j=k} a_i b_j \quad \forall k=0, 1, \dots, 2n-2$$

$$P(x) \cdot Q(x) = \sum_{k=0}^{2n-2} C_k x^k$$

what about $A \times B$? $\rightarrow C_k$'s are the bits of this product (almost!) $\rightarrow$ account for carry bits

So, now we want a fast algorithm for convolution

POLYNOMIALS

degree $n-1$ polynomial specified by n coefficients

Gauss

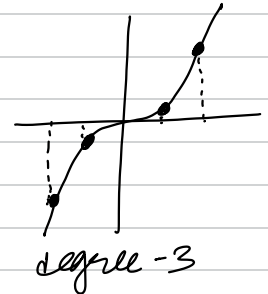
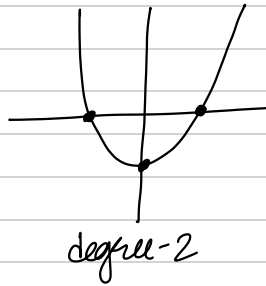
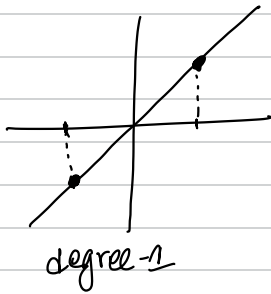
$\Leftrightarrow$

Evaluations of the polynomial at n distinct points

Two different representations of a polynomial.
(Fundamental Thm of Algebra)

COEFFICIENT REPRESENTATION

EVALUATION REPRESENTATION



time to multiply = $O(n^2)$

n coefficients
of polynomial
 $P(x)$

$O(n \log n) \rightarrow$ DFT
evaluation

interpolation
 $O(n \log n) \rightarrow$ DFT⁻¹

time to multiply = $O(n)$

n evaluations
(distinct) of
 $P(x)$

Naïve $\rightarrow O(n^2)$ operations for both
DFT $\rightarrow O(n \log n)$ for both.

Why care about switching representations?
polynomial multiplication in evaluation representation
takes $O(n)$ -time. Why?

$(P(1), \dots, P(2n-1)) \cdot (Q(1), \dots, Q(2n-1)) = (P(1)Q(1), \dots, P(2n-1)Q(2n-1))$
 $\Rightarrow \therefore O(n)$ operations
evaluation representation
of $P(x) \cdot Q(x)$.

Let points of evaluating be $w_0, w_1, \dots, w_{n-1}$.
Evaluation can be written as the following
matrix operation:

$$\begin{bmatrix} P(w_0) \\ P(w_1) \\ P(w_2) \\ \vdots \\ P(w_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & w_0 & w_0^2 & \dots & w_0^{n-1} \\ 1 & w_1 & w_1^2 & \dots & w_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_{n-1} & w_{n-1}^2 & \dots & w_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} \rightarrow \textcircled{I}$$

evaluations Vandermonde Matrix: V coefficients

* For interpolating, multiply evaluations by v^{-1} .

But for matrix operation ①, still need $O(n^2)$ ops.
What can we do? → Choose w_p 's cleverly.

We need $2n$ points to evaluate $p(x)$ & $q(x)$ on!

*** (take # evaluations = power of 2)

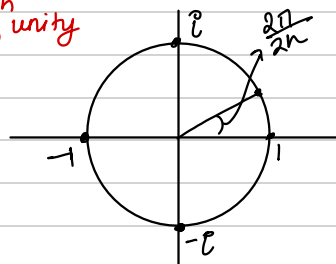
ROOTS OF UNITY

Take them to be $2n$ -th roots of unity.

* $j = 0, 1, \dots, 2n-1$

$w_p = w^j$ where $w = e^{\frac{2\pi i}{2n}}$ → primitive $2n$ -th root of unity

($2n$ -distinct complex roots of)
 $x^{2n} = 1$



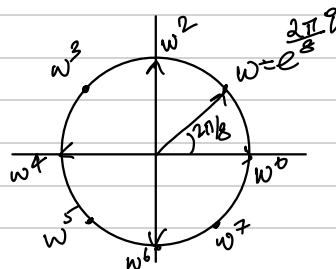
For these w_p 's the Vandermonde matrix V is DFT matrix.

$$(\text{DFT}_n [v_j]) = w^{jp \bmod n}$$

jth-th entry

Properties of the DFT matrix

$$\text{DFT}_8 = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^7 \\ 1 & w^2 & w^4 & \dots & \\ \vdots & & & & \\ 1 & w^7 & w^6 & \dots & \end{bmatrix}$$



① Inner product of two distinct columns is 0

$$\begin{aligned} \text{eg: } \left\langle \begin{pmatrix} 1 \\ w \\ w^2 \\ \vdots \\ w^7 \end{pmatrix}, \begin{pmatrix} 1 \\ w^3 \\ w^6 \\ \vdots \\ w^5 \end{pmatrix} \right\rangle &= \overline{1} \cdot 1 + \overline{w} \cdot w^3 + \overline{w^2} \cdot w^6 + \dots + \overline{w^7} \cdot w^5 \\ &= (w^0 + w^2 + w^4 + w^6) \cdot 2 = 0 \end{aligned}$$

Exercise: Prove generally.

② Inner product of a column & itself is n .

Pf: For column j $\langle \text{col}_j, \text{col}_j \rangle = \sum_{i=0}^{n-1} w^{(i-j)^2} = n$

$\Rightarrow \frac{1}{\sqrt{n}} \text{DFT}_n$ is a unitary matrix (Read-up unitary matrix)

Moreover, $\underbrace{\text{DFT}_n^+}_{\text{conjugate transpose}} \cdot \text{DFT}_n = \begin{pmatrix} n & & 0 \\ & n & \\ 0 & & n \end{pmatrix}$

$\Rightarrow \text{DFT}_n^{-1} = \frac{1}{n} \cdot \text{DFT}_n^+$

Hence, Interpolating $\equiv \text{DFT}_n^{-1} = \frac{1}{n} \cdot \text{DFT}_n^+$ | essentially the same.
& evaluating $\equiv \text{DFT}_n$

FAST FOURIER TRANSFORM (FFT)

Divide & conquer Algorithm

$p(x) = a_0 + a_1 x + \dots + a_7 x^7$ (eg: $n=8$)

$P_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$

$P_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$

$p(x) = P_{\text{even}}(x^2) + x \cdot P_{\text{odd}}(x^2)$

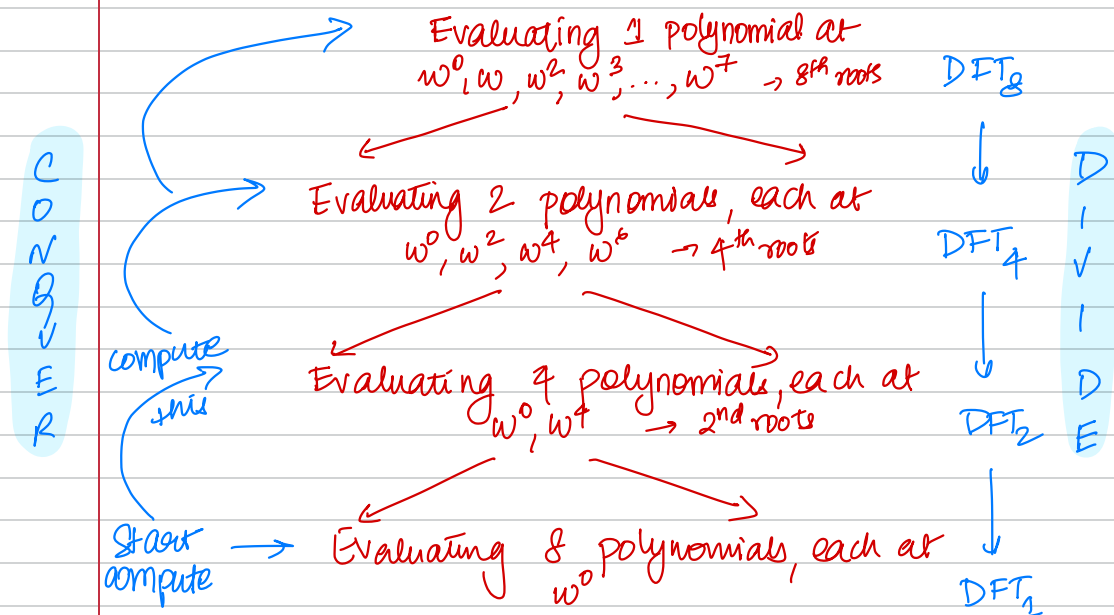
recurse

Why does this splitting help? Take the roots of unity.

Evaluating $P_{\text{even}}(x)$ & $P_{\text{odd}}(x)$ at ± 1 gives.

$$\left. \begin{aligned} P(1) &= P_{\text{even}}(1) + P_{\text{odd}}(1) \\ P(-1) &= P_{\text{even}}(1) + (-1) \cdot P_{\text{odd}}(1) \\ P(i) &= P_{\text{even}}(-1) + i \cdot P_{\text{odd}}(-1) \\ P(-i) &= P_{\text{even}}(-1) + (-i) \cdot P_{\text{odd}}(-1) \end{aligned} \right\} \begin{array}{l} \text{evaluation} \\ \text{of } P(x) \text{ at} \\ \pm 1, \pm i \end{array}$$

In general, evaluating $P_{\text{even}}(x)$ & $P_{\text{odd}}(x)$ at 4 points gives evaluations of $P(x)$ at 8 points!



$\therefore$ The recursion gives

$$\begin{array}{ccc} T(n) = & 2 \cdot T(n/2) & + O(n) \\ \uparrow & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} \\ \text{DFT}_n\text{'s} & 2 \times \text{DFT}_{n/2} & \text{additional } O(n) \\ \text{time} & & \text{to combine} \end{array}$$

$\Rightarrow T(n) = O(n \log n) \rightarrow \text{Prove it! (EXERCISE)}$

HOMEWORK BEFORE NEXT CLASS

↳ Brush-up basic linear algebra.

- Vector spaces
- Linear Independence
- Span
- Basis
- Orthogonal / Orthonormal basis
- Inner product
- Unitary matrix
- etc.

If you want
a crash-course
- contact Krishna

NEXT TWO LECTURES

↳ Main reference: Chapter 1
Analysis of Boolean Functions
Ryan O'Donnell