COMP3121: Algorithms & Programming Techniques
Summary notes – Week 3

Gerald Huang

Updated: June 19, 2020

Contents

1 Lecture 5A – Fast Fourier Transform (Part I)

Date: June 18, 2020

1.1 Multiplying polynomials quickly

Given two polynomials of degree at most n,

$$P_A(x) = A_n x^n + \cdots + A_0, \quad P_B(x) = B_n x^n + \cdots + B_0,$$

1. Convert them into value representation at $2n+1$ distinct points $x_0, x_1, \ldots, x_{2n}$

$$P_A(x) \leftrightarrow \{(x_0, P_A(x_0)), (x_1, P_A(x_1)), \ldots, (x_{2n}, P_A(x_{2n}))\}.$$
$$P_B(x) \leftrightarrow \{(x_0, P_B(x_0)), (x_1, P_B(x_1)), \ldots, (x_{2n}, P_B(x_{2n}))\}.$$

   - The reason why we have $2n+1$ distinct points is because we eventually want to multiply $P_A(x)$ with $P_B(x)$ which will be of degree $n+n=2n$. Hence we require $2n+1$ distinct points to uniquely determine $P_A(x)P_B(x)$.

2. Multiply them **point by point** using $2n+1$ multiplications to form the value representation of $P_C(x)$

$$P_C(x) \leftrightarrow \{(x_0, \underbrace{P_A(x_0)P_B(x_0)}_{P_C(x_0)}), (x_1, \underbrace{P_A(x_1)P_B(x_1)}_{P_C(x_1)}), \dots, (x_{2n}, \underbrace{P_A(x_{2n})P_B(x_{2n})}_{P_C(x_{2n})})\}.$$

3. Convert such value representation of $P_C(x)$ to its coefficient form

$$P_C(x) = C_{2n}x^{2n} + C_{2n-1}x^{2n-1} + \cdots + C_1x + C_0.$$

- Key question: What values should we take for $x_0, \dots, x_{2n}$ to avoid "explosion" of size when evaluating x_i^n? If n is arbitrarily large, computing n^n will explode in size.
  Introduce **roots of unity**!

1.2 Complex numbers revisited

- Complex numbers $z = a + ib$ can be represented using their modulus $|z| = \sqrt{a^2 + b^2}$ and argument which is angle taking values in $(-\pi, \pi]$ (includes π and excludes $-\pi$)

$$z = |z|e^{i\theta} = |z|(\cos\theta + i\sin\theta).$$

- Recall that

$$z^n = \left(|z|e^{i\theta}\right)^n = |z|^n e^{i(n\theta)} = |z|^n[\cos(n\theta) + i\sin(n\theta)].$$

1.2.1 Complex roots of unity

- Roots of unity of order n are complex numbers which satisfy the equation $z^n = 1$.
- If $z^n = |z|^n e^{i(n\theta)} = 1$, then

$$|z|^n = 1 \implies |z| = 1$$
$$n\theta = 0 + 2\pi k \implies \theta = \frac{2\pi k}{n}, \quad k \in \mathbb{Z}.$$

- Denote $\omega_n = e^{i(2\pi/n)}$. Such ω_n is called the primitive root of unity of order n.
- A root of unity ω is primitive if all other roots of unity of the same order can be obtained as its powers ω^k.

- For $\omega_n = e^{i(2\pi/n)}$ and for all k such that $0 \le k \le n-1$,

$$((\omega_n)^k)^n = (\omega_n)^{nk} = ((\omega_n)^n)k = 1^k = 1.$$

  - Thus $\omega_n^k = (\omega_n)^k$ is also a **root of unity**.
  - Since ω_n^k are roots of unity for $k = 0, 1, \dots, n-1$ and there are **at most** n roots of unity of order n, we conclude that every root of unity of order n must be of the form ω_n^k.

- For the product of any two roots of unity ω_n^k and ω_n^m of the same order, we have $\omega_n^k \omega_n^m = \omega_n^{k+m}$.
- If $k + m \geq n$, then write $k + m = n + l$ for some $l \in [0, n)$. Then

$$\omega_n^k \omega_n^m = \omega_n^{k+m} = \omega_n^{n+l} = \omega_n^n \omega_n^l = 1 \cdot \omega_n^l = \omega_n^l.$$

- Thus, the product of any two roots of unity of the same order is just **another root of unity** of the same order!

- **Cancellation Lemma**:
  - For all integers k, m, n,

$$\omega_{kn}^{km} = \omega_n^m.$$

Proof.

$$\omega_{kn}^{km} = (\omega_{kn})^{km} = \left(e^{i\frac{2\pi}{kn}}\right)^{km} = e^{i\frac{2\pi km}{kn}} = e^{i\frac{2\pi m}{n}} = \left(e^{i\frac{2\pi}{n}}\right)^m = \omega_n^m.$$

□

1.3 Discrete Fourier Transform

- Let $A = \langle A_0, A_1, \ldots, A_{n-1} \rangle$ be a sequence of n real or complex numbers.
- Form the corresponding polynomial $P_A(x) = \sum_{j=0}^{n-1} A_j x^j$.
- We can evaluate it at all complex roots of unity of order.
  - Compute $P_A(\omega_n^k)$ for all $0 \leq k \leq n - 1$.
- The sequence of values $\langle P_A(1), P_A(\omega_n), P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1}) \rangle$ is called the Discrete Fourier Transform (DFT) of the sequence $A = \langle A_0, A_1, \ldots, A_{n-1} \rangle$.

- $P_A(\omega_n^k)$ is usually denoted by $\hat{A}_k$ and the sequence of values $\langle P_A(1), P_A(\omega_n), P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1}) \rangle$ is denoted by $\hat{A} = \langle \hat{A}_0, \hat{A}_1, \ldots, \hat{A}_{n-1} \rangle$.
- The transform takes $O(n^2)$ time to compute. But the Fast Fourier Transform is an algorithm that efficiently computes the DFT in $O(n \log n)$ time!

- Note: Use complex roots of unity for fast multiplication instead of the Karatsuba trick to avoid explosion in size!

- Define the two polynomials $P_A(x) = A_0 + \cdots + A_{n-1} x^{n-1}$ and $P_B(x) = B_0 + \cdots + B_{m-1} x^{m-1}$.
- Note that we defined the **DFT** of a sequence of length n as the values of the corresponding polynomial of degree $n - 1$ at the n roots of unity of order n.
- So the DFT of a sequence A is another sequence $\hat{A}$ of exactly the same length!
- In order to be able to call a function for evaluating the DFT and since we need $n + m - 1$ values of $P_C(x) = P_A(x)P_B(x)$, we pad A with $m - 1$ zeros at the end, $(A_0, A_1, \ldots, A_{n-1}, \underbrace{0, \ldots, 0}_{m-1})$ to make it of length $n + m - 1$, and

similarly, we pad B with $n-1$ zeros at the end, $(B_0, B_1, \ldots, B_{m-1}, \underbrace{0, \ldots, 0}_{n-1})$ to also obtain a sequence of length $n+m-1$.

- Compute the DFTs of the two (0 padded) sequences

$$DFT(\langle A_0, A_1, \ldots, A_{n-1}, \underbrace{0, \ldots, 0}_{m-1}\rangle) = \langle \hat{A}_0, \hat{A}_1, \ldots, \hat{A}_{n+m-2}\rangle,$$
$$DFT(\langle B_0, B_1, \ldots, B_{m-1}, \underbrace{0, \ldots, 0}_{n-1}\rangle) = \langle \hat{B}_0, \hat{B}_1, \ldots, \hat{B}_{n+m-2}\rangle.$$

- Define $\hat{C}_k = \hat{A}_k \hat{B}_k = P_A(\omega^k_{n+m-1}) P_B(\omega^k_{n+m-1}) = P_C(\omega^k_{n+m-1})$.
- Apply the inverse transformation for DFT, called the IDFT, to recover coefficients $\langle C_0, C_1, \ldots, C_{n+m-1}\rangle$ of the product polynomial $P_C(x)$ from the sequence $\langle \hat{C}_0, \hat{C}_1, \ldots, \hat{C}_{n+m-1}\rangle$ of its values $C_k = P_C(\omega^k_{n+m-1}$ at the roots of unity of order $n+m-1$.

Extra reading:

- *Wikipedia article*
- *Fourier Transforms: Discrete Fourier Transform Part 1, Part 2*
- *Algorithm Design Chpt 5.6: Convolutions and the Fast Fourier Transform*
- *Introduction to Algorithms Chpt 30.2: The DFT and FFT*

2 Lecture 5B – Fast Fourier Transform (Part II)

- **Problem**: Given a sequence $A = \langle A_0, A_1, \ldots, A_{n-1} \rangle$, compute its DFT. Also assume that n is a power of 2 so that $(n-1)$ is odd and $(n-2)$ is even.
- *Main idea*: divide and conquer by splitting the polynomial $P_A(x)$ into the even and odd powers:

$$P_A(x) = \underbrace{\left(A_0 + A_2x^2 + A_4x^4 + \cdots + A_{n-2}x^{n-2}\right)}_{\text{even powers}} + \underbrace{\left(A_1x + A_3x^3 + A_5x^5 + \cdots + A_{n-1}x^{n-1}\right)}_{\text{odd powers}}$$
$$= \left(A_0 + A_2x^2 + A_4\left(x^2\right)^2 + \cdots + A_{n-2}\left(x^2\right)^{n/2-1}\right) + x\left(A_1 + A_3x^2 + A_5\left(x^2\right)^2 + \cdots + A_{n-1}\left(x^2\right)^{n/2-1}\right).$$

- Define

$$A^{[0]} = \underbrace{\langle A_0, A_2, A_4, \ldots, A_{n-2} \rangle}_{\text{even coefficients of } P_A(x)}, \quad A^{[1]} = \underbrace{\langle A_1, A_3, A_5, \ldots, A_{n-1} \rangle}_{\text{odd coefficients of } P_A(x)}.$$

  Then construct the polynomials

$$P_{A^{[0]}}(y) = A_0 + A_2y + A_4y^2 + \cdots + A_{n-2}y^{n/2-1}$$
$$P_{A^{[1]}}(y) = A_1 + A_3y + A_5y^2 + \cdots + A_{n-1}y^{n/2-1}.$$

  So

$$\boxed{P_A(x) = P_{A^{[0]}}(x^2) + xP_{A^{[1]}}(x^2).}$$

- Note that the number of coefficients of the polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ is $n/2$ each, while the number of coefficients of the polynomial $P_A(x)$ is n.
- **Problem of size** n: Evaluate a polynomial with n coefficients at n many roots of unity.
- **Problem of size** $n/2$: Evaluate a polynomial with $n/2$ coefficients at $n/2$ many roots of unity.

We have reduced evaluation of our polynomial $P_A(x)$ with n coefficients at inputs $x = \omega_n^0, x = \omega_n^1, \ldots, x = \omega_n^{n-1}$ to evaluation of two polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ each with $n/2$ coefficients at points $y = x^2$ for the same values of inputs x.

- However, as x ranges through values $\{\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}\}$, the value $y = x^2$ ranges through $\{\omega_{n/2}^0, \omega_{n/2}^1, \ldots, \omega_n^{n-1}\}$, and there are only $n/2$ distinct such values.

Once we get these $n/2$ values of $A^{[0]}(x^2)$ and $A^{[1]}(x^2)$, we need n additional multiplications with numbers ω_n^k to obtain the values of

$$P_A(\omega_n^k) = P_{A^{[0]}}((\omega_n^k)^2) + \omega_n^k \cdot P_{A^{[1]}}((\omega_n^k)^2)$$
$$= P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k \cdot P_{A^{[1]}}(\omega_{n/2}^k)$$

2.1 Simplifying the Fast Fourier Transform (FFT)

Apply the **Cancellation Lemma** to obtain the result

$$\omega_n^{n/2} = \omega_{2n/2}^{n/2} = \omega_2 = -1.$$

Thus, we have the result

$$\boxed{\omega_n^{k+n/2} = \omega_n^{n/2} \cdot \omega_n^k = \omega_2 \cdot \omega_n^k = -\omega_n^k.}$$

We now simplify evaluation of

$$P_A(\omega_n^k) = P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k \cdot P_{A^{[1]}}(\omega_{n/2}^k)$$

for $n/2 \le k < n$ as follows.

- Let $k = n/2 + m$ for some $m \in [0, n/2)$. Then

$$\begin{aligned} P_A\left(\omega_n^{n/2+m}\right) &= P_{A^{[0]}}\left(\omega_{n/2}^{n/2+m}\right) + \omega_n^{n/2+m} \cdot P_{A^{[1]}}\left(\omega_{n/2}^{n/2+m}\right) \\ &= P_{A^{[0]}}(\underbrace{\omega_{n/2}^{n/2}}_{=1} \cdot \omega_{n/2}^m) + \underbrace{\omega_n^{n/2}}_{-1} \cdot \omega_n^m \cdot P_{A^{[1]}}(\underbrace{\omega_{n/2}^{n/2}}_{=1} \cdot \omega_{n/2}^m) \qquad \text{(using } a^{b+c} = a^b \cdot a^c\text{)} \\ &= P_{A^{[0]}}\left(1 \cdot \omega_{n/2}^m\right) - \omega_n^m \cdot P_{A^{[1]}}\left(1 \cdot \omega_{n/2}^m\right) \\ &= P_{A^{[0]}}(\omega_{n/2}^m) - \omega_n^m \cdot P_{A^{[1]}}(\omega_{n/2}^m). \end{aligned}$$

- Compare this with

$$P_A(\omega_n^m) = P_{A_{[0]}}(\omega_{n/2}^m) + \omega_n^m \cdot P_{A^{[1]}}(\omega_{n/2}^m) \quad \text{for } 0 \le m < n/2.$$

- Hence, replace evaluations of

$$P_A(\omega_n^k) = P_{A_{[0]}}(\omega_{n/2}^k) + \omega_n^k \cdot P_{A^{[1]}}(\omega_{n/2}^k)$$

  for $k = 0$ to $k = n - 1$ with such evaluations only for $k = 0$ to $k = n/2 - 1$

$$\begin{aligned} P_A(\omega_n^k) &= P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k \cdot P_{A^{[1]}}(\omega_{n/2}^k) \\ P_A(\omega_n^{n/2+k}) &= P_{A^{[0]}}(\omega_{n/2}^k) - \omega_n^k \cdot P_{A^{[1]}}(\omega_{n/2}^k). \end{aligned}$$

2.2 Analysis of the FFT

- We have recursively reduced evaluation of a polynomial $P_A(x)$ with n coefficients at n roots of unity of order n to evaluations of two polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$, each with $n/2$ coefficients.
- Once we get these $n/2$ values, we need $n/2$ additional multiplications to obtain the values of

$$P_A(\omega_n^k) = P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k \cdot P_{A^{[1]}}(\omega_{n/2}^k) \quad \text{and} \quad P_A(\omega_n^{n/2+k}) = A^{[0]}(\omega_{n/2}^k) - \omega_n^k A^{[1]}(\omega_{n/2}^k)$$

  for all $0 \le k < n/2$.

- Thus, we have reduced a problem of size n to two subproblems of size $n/2$ plus a linear overhead. So our algorithm's run time satisfies the recurrence

$$T(n) = 2T(n/2) + cn.$$

  By the Master Theorem, $T(n) = \Theta(n \log n)$.

2.3 Matrix representation of polynomial evaluation

- Evaluation of a polynomial $P_A(x) = A_0 + A_1 x + \cdots + A_{n-1} x^{n-1}$ at roots of unity of order n can be represented in the matrix form as follows

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}}_{\text{Vandermonde matrix}} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{pmatrix} = \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix} = \begin{pmatrix} \hat{A}_0 \\ \hat{A}_1 \\ \hat{A}_2 \\ \vdots \\ \hat{A}_{n-1} \end{pmatrix}$$

- The FFT is just replacing matrix-vector multiplication with an $n \log n$ procedure.
- From $P_A(1), P_A(\omega_n), P_A(\omega_n^2), \dots, P_A(\omega_n^{n-1})$, we get the coefficients from

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} \hat{A}_0 \\ \hat{A}_1 \\ \hat{A}_2 \\ \vdots \\ \hat{A}_{n-1} \end{pmatrix} \tag{1}$$

To obtain the inverse matrix from (1), all we have to do is just change the signs of the exponents and divide everything by n:

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

Proof. Compute the matrix product:

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}.$$

The (i, j) entry in the product matrix is equal to a product of ith row and jth column:

$$\begin{pmatrix} 1 & \omega_n^i & \omega_n^{2i} & \dots & \omega_n^{i(n-1)} \end{pmatrix} \begin{pmatrix} 1 \\ \omega_n^{-j} \\ \omega_n^{-2j} \\ \vdots \\ \omega_n^{-(n-1)j} \end{pmatrix} = \sum_{k=0}^{n-1} \omega_n^{ik} \omega_n^{-jk} = \sum_{k=0}^{n-1} \omega_n^{(i-j)k}.$$

We now have two possibilities.

1. If $i = j$, then

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

2. If $i \neq j$, then $\sum_{k=0}^{n-1} \omega_n^{(i-j)k}$ becomes a geometric series with ratio $r = \omega_n^{i-j}$ and thus

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

So,

$$\sum_{k=0}^{n-1} \omega_n^{(i-j)k} = \begin{cases} n & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

So we get

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & \dots & 0 \\ 0 & n & 0 & \dots & 0 \\ 0 & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & n \end{pmatrix} = n \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}.$$

In other words, we obtain

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2(n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}.$$

□

This means we can use the same FFT algorithm with the only change that

1. the root of unity ω_n is replaced by $\omega_n^{-1} = e^{-i\frac{2\pi}{n}}$.
2. the resulting output values are divided by n.

2.4 Computing Convolutions

- To compute convolutions, proceed in exactly the same manner as FFT's except this time, we start with sequences $A = \langle A_0, \dots A_{n-1} \rangle$ and $B = \langle B_0, \dots, B_{n-1} \rangle$.

2.5 Applications of convolutions

- The degree of similarity of two strings A and B of the same length n can be taken to be the number of places i where $A[i] = B[i]$. Assume you are given two binary strings A and B; A has n^2 bits, B has n bits. Your task is to find all substrings of consecutive bits in A which are of length n that are the most similar to string B. Your algorithm should run in time $O(n^2 \log n)$.

- Replace 0's everywhere in both A and B with -1 to obtain sequences A' and B'. Look at the convolution $C = A' * \tilde{B'}$ where $\tilde{B'}$ is the mirror image of B' (ie B' written in reverse). Find indices j such that $C(j) = \max\{C(m) : 0 \leq m \leq 2n-2\}$.

  - Assume you are given a map of a straight sea shore of length n metres as a sequence on n numbers such that A_i is the number of fish between ith metre of the shore and the $(i+1)$th metre, $0 \leq i \leq n-1$. You also have a net of length m metres but unfortunately it has holes in it. Such a net is described as a sequence of m ones and zeros, where 0's denote where the holes are. If you throw such a net starting at metre k and ending at metre $k+m$, then you will catch only the fish in one metre stretches of the shore where the corresponding bit of the net is 1. Find the spot where you should place the left end of your net in order to catch the largest possible number of fish using an algorithm which runs in time $O(n \log n)$.

- Revert the recording of the net (write the bits from right to left). Find the convolution of the shore sequence with the sequence presenting the net and look for the peak value of the convolution.