Fast Fourier Transform & Number Theoretic Transform

SIGma UIUC



Outline

The Core Idea: Coefficient Space vs. Value Space

Fast Value Space Evaluation: Split and Merge

Returning to Coefficient Space (Vandermonde & IFFT)

Number Theoretic Transform (NTT)



Section 1

The Core Idea: Coefficient Space vs. Value Space



The Problem: Multiplying Two Polynomials

- Suppose we are given two polynomials of degree $< n$:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

- Our goal is to compute their product polynomial $C(x) = A(x) \cdot B(x)$:

$$C(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{2n-2} x^{2n-2}$$

- How fast can we find all the coefficients $c_0, c_1, \ldots, c_{2n-2}$?



The Schoolbook Algorithm: $O(n^2)$ Time

- In the standard schoolbook algorithm, we expand $A(x) \cdot B(x)$ by multiplying every term of $A(x)$ with every term of $B(x)$:

$$c_k = \sum_{i+j=k} a_i b_j = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0$$

- Because there are n terms in $A(x)$ and n terms in $B(x)$, cross-multiplying every pair (a_i, b_j) requires $n \times n = n^2$ multiplications.
- In this standard representation—called Coefficient Space—addition takes $O(n)$ time, but schoolbook multiplication is stuck at $O(n^2)$ time.



Speeding Up: From Coefficient Space to Value Space

- We can speed up polynomial multiplication by transforming the polynomials from Coefficient Space into Value Space:
  - Coefficient Space: A polynomial is represented by its coefficients $(a_0, a_1, \ldots, a_{n-1})$. Multiplying requires $O(n^2)$ cross-multiplications.
  - Value Space: A polynomial is represented by its evaluations at a set of points $(A(x_0), A(x_1), \ldots)$.
- In Value Space, multiplying $C(x) = A(x) \cdot B(x)$ is easy: we just multiply the point values straight across in $O(n)$ time:

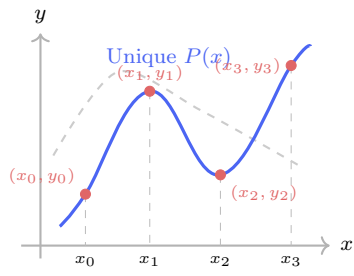
$$C(x_i) = A(x_i) \cdot B(x_i)$$

- If we pick the right set of evaluation points, we can move from Coefficient Space to Value Space, multiply in $O(n)$ time, and convert back much faster than $O(n^2)$.



Why Value Space Works: Polynomial Interpolation

- Why can we recover the coefficients of $C(x)$ from its point values?
- So given enough evaluation points ($N \geq 2n - 1$), the point values $C(x_i) = A(x_i) \cdot B(x_i)$ uniquely determine the product polynomial $C(x)$.





Defining the n-th Roots of Unity

- An *n-th root of unity* is any complex number z satisfying:

$$z^n = 1$$

- A *primitive n-th root of unity* is

$$\omega_n = e^{2\pi i/n} = \cos\left(\frac{2\pi}{n}\right) + i\sin\left(\frac{2\pi}{n}\right)$$

- Its powers $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ are equally spaced around the complex unit circle. Each successive power rotates by $\frac{360^\circ}{n}$.



From Roots of Unity to the DFT

- Recall that we can represent a polynomial by its coefficients:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j.$$

- Instead of working only with its coefficients, we can evaluate the polynomial at the n-th roots of unity:

$$1,\ \omega_n,\ \omega_n^2,\ \ldots,\ \omega_n^{n-1}.$$

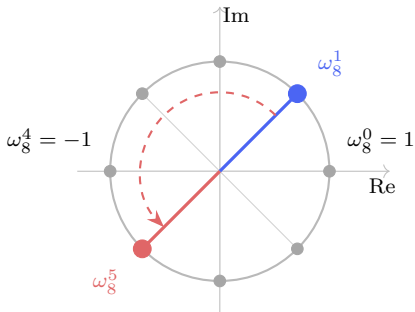
- This gives us a vector of n evaluation values:

$$\begin{bmatrix} A(1) \\ A(\omega_n) \\ A(\omega_n^2) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix}.$$

- This operation is called the Discrete Fourier Transform (DFT).



Visualizing the Roots of Unity



- The roots are evenly spaced around the circle.
- Every root has an opposite point across the circle.
- Opposite roots have the same square because $z^2 = (-z)^2$.
- These geometric patterns lead to two identities that make the FFT efficient.



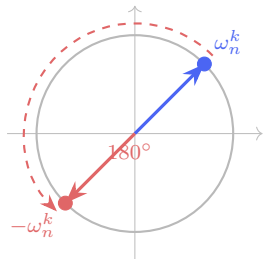
Identity 1: Halfway Around the Circle

- There are n equally spaced roots around a full 360° circle.
- Moving halfway around means taking $n/2$ steps, or rotating by 180°.
- Therefore, for even n:

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

- Shifting any exponent by $n/2$ reaches the opposite point:

$$\begin{aligned} \omega_n^{k+n/2} &= \omega_n^k \omega_n^{n/2} \\ &= -\omega_n^k. \end{aligned}$$





Identity 2: Squaring the Roots Halves the Set

- Opposite roots differ by a factor of -1.
- Squaring removes that sign:

$$(\omega_n^{k+n/2})^2 = (-\omega_n^k)^2 = (\omega_n^k)^2.$$

- Squaring also doubles the angle, so

$$(\omega_n^k)^2 = (\omega_n^2)^k = (\omega_{n/2})^k.$$

- The n roots therefore map onto only $n/2$ distinct roots when squared.
- This lets us evaluate the even-indexed and odd-indexed coefficients using a smaller transform of size $n/2$, then combine their results.



Section 2

Fast Value Space Evaluation: Split and Merge



The Strategy: Split into Two Halves, Then Merge

- Instead of evaluating all n terms of $A(x)$ at once, we break the work into three simple steps:
  1. Divide the polynomial into two smaller halves of size $n/2$ (even terms and odd terms).
  2. Compute the values for each half separately.
  3. Merge the results from the two halves together to get all n values of $A(x)$.



Step 1: Splitting into Even and Odd Halves

- Given a polynomial $A(x)$ of degree $< n$, $\mathrm{DFT}_n(A)$ returns the vector of evaluations of $A(x)$ at all n-th roots of unity from ω_n^0 to ω_n^{n-1}:

$$\mathrm{DFT}_n(A) = \left[A(\omega_n^0),\ A(\omega_n^1),\ A(\omega_n^2),\ \ldots,\ A(\omega_n^{n-1})\right]$$

- Assume n is a power of 2. We divide $A(x)$ into two halves: its even-indexed terms and odd-indexed terms:

$$A(x) = \underbrace{(a_0 + a_2x^2 + a_4x^4 + \ldots)}_{A_{\mathrm{even}}(x^2)} + x\,\underbrace{(a_1 + a_3x^2 + a_5x^4 + \ldots)}_{A_{\mathrm{odd}}(x^2)}$$

- Each half ($A_{\mathrm{even}}(u)$ and $A_{\mathrm{odd}}(u)$) is a smaller polynomial with only $n/2$ coefficients, evaluated at $u = x^2$.



Step 2: Computing the Two Halves

- When we plug $x = \omega_n^k$ into $A(x) = A_{\text{even}}(x^2) + x \cdot A_{\text{odd}}(x^2)$, the input passed into A_{even} and A_{odd} is $x^2 = (\omega_n^k)^2$.
- By the Halving Identity, squaring an n-th root of unity gives an $(n/2)$-th root of unity:

$$(\omega_n^k)^2 = (\omega_n^2)^k = \omega_{n/2}^k$$

- For $k = 0, 1, \ldots, \frac{n}{2} - 1$, the inputs are $\omega_{n/2}^0, \omega_{n/2}^1, \ldots, \omega_{n/2}^{n/2-1}$ (all $\frac{n}{2}$-th roots of unity).
- So computing the values for the even half and odd half is the exact same DFT problem, just on half the size $(n/2)$:

$$E = \text{DFT}_{n/2}(A_{\text{even}}), \quad O = \text{DFT}_{n/2}(A_{\text{odd}})$$



Step 3: Merging the Two Halves Together

- Once we have the answers from the even half (E) and odd half (O), how do we merge them into all n values of $A(x)$?
- First half of points ($0 \le k < \frac{n}{2}$) — Use the Halving Identity $(\omega_n^k)^2 = \omega_{n/2}^k$:

$$y_k = A(\omega_n^k) = A_{\text{even}}(\omega_{n/2}^k) + \omega_n^k \cdot A_{\text{odd}}(\omega_{n/2}^k) = E_k + \omega_n^k \cdot O_k$$

- Second half of points ($k + \frac{n}{2}$) — Symmetry ($\omega_n^{k+n/2} = -\omega_n^k$) flips the $+$ to a $-$:

$$y_{k+n/2} = A(\omega_n^{k+n/2}) = A_{\text{even}}(\omega_{n/2}^k) - \omega_n^k \cdot A_{\text{odd}}(\omega_{n/2}^k) = E_k - \omega_n^k \cdot O_k$$

- Merging the two halves takes just $O(n)$ additions/multiplications across $\log_2 n$ levels $\implies$ $O(n \log n)$ total time.



Worked Example ($n = 4$): Split & Evaluate Halves

- Evaluate $A(x) = 1 + 2x + 3x^2 + 4x^3$ at $\omega_4^0 = 1$, $\omega_4^1 = i$, $\omega_4^2 = -1$, $\omega_4^3 = -i$.
- Step 1 (Split): $A(x) = (1 + 3x^2) + x(2 + 4x^2)$
  - $A_{\text{even}}(u) = 1 + 3u$ and $A_{\text{odd}}(u) = 2 + 4u$
- Step 2 (Evaluate Halves at $\omega_2^0 = 1, \omega_2^1 = -1$):
  - Even Half $E = \text{DFT}_2(A_{\text{even}})$: $E_0 = 1 + 3(1) = 4$, $E_1 = 1 + 3(-1) = -2$
  - Odd Half $O = \text{DFT}_2(A_{\text{odd}})$: $O_0 = 2 + 4(1) = 6$, $O_1 = 2 + 4(-1) = -2$



Worked Example ($n = 4$): Merge the Halves

- We have $E = [4, -2]$, $O = [6, -2]$, and roots $\omega_4^0 = 1, \omega_4^1 = i$.
- Step 3 (Merge via $y_k = E_k \pm \omega_4^k O_k$):
  - For $k = 0$ ($\omega_4^0 = 1$): $\quad y_0 = 4 + 1(6) = 10, \quad y_2 = 4 - 1(6) = -2$
  - For $k = 1$ ($\omega_4^1 = i$): $\quad y_1 = -2 + i(-2) = -2 - 2i, \quad y_3 = -2 - i(-2) = -2 + 2i$
- Final Output: $\mathrm{DFT}_4(A) = [10,\ -2 - 2i,\ -2,\ -2 + 2i]^T$.



Section 3

Returning to Coefficient Space (Vandermonde & IFFT)



How Do We Return to Coefficient Space?

- Our split-and-merge FFT algorithm moves us from Coefficient Space to Value Space in $O(n \log n)$ time.
- After multiplying $C(\omega_n^k) = A(\omega_n^k) \cdot B(\omega_n^k)$, how do we invert the process (IFFT)?
- Evaluating $A(x) = \sum_{k=0}^{n-1} a_k x^k$ at $\omega^0, \omega^1, \ldots, \omega^{n-1}$ is equivalent to multiplying the coefficient vector by a Vandermonde Matrix V_n.
- Therefore, returning to Coefficient Space is simply multiplying the values vector by the inverse matrix V_n^{-1}.



Writing Out the Forward Vandermonde Matrix (V_n)

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$



Deriving the IFFT (Step 1): A Geometric Series of Roots of Unity

- To construct V_n^{-1}, we first need a key property about summing powers of roots of unity.
- Consider the geometric series $S(d) = \sum_{k=0}^{n-1} (\omega_n^d)^k$ for an integer power d:
  - Case 1 ($d = 0$): Every term is $(\omega_n^0)^k = 1^k = 1$, so $S(0) = \sum_{k=0}^{n-1} 1 = n$.
  - Case 2 ($0 < |d| < n$): Since $r = \omega_n^d \neq 1$ and $r^n = (\omega_n^n)^d = 1^d = 1$:

$$S(d) = \sum_{k=0}^{n-1} r^k = \frac{1 - r^n}{1 - r} = \frac{1 - 1}{1 - r} = 0$$

- A geometric series of the roots of unity sums to 0 whenever $0 < |d| < n$, and sums to n when $d = 0$.



Deriving the IFFT (Step 2): Guessing the Inverse Structure

- How can we use this geometric series fact to invert V_n?
- We want a matrix W such that $W \cdot V_n$ has n on the diagonal ($j = m$) and 0 everywhere else ($j \neq m$).
- In V_n, column m has entries ω^{km}. If we define row j of W using negative exponents ω^{-jk}, multiplying row j of W by column m of V_n forms the exact geometric series from Step 1:

$$W = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \dots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$



Deriving the IFFT (Step 3): Multiplying $W \cdot V_n$

- Check entry (j, m) of the product $W \cdot V_n$ (row j of W times column m of V_n):

$$(W \cdot V_n)_{j,m} = \sum_{k=0}^{n-1} W_{j,k}(V_n)_{k,m} = \sum_{k=0}^{n-1} \omega^{-jk} \cdot \omega^{km} = \sum_{k=0}^{n-1} (\omega^{m-j})^k$$

- Since $0 \leq j, m < n$, the difference $d = m - j$ satisfies $|d| < n$. By our Geometric Series rule:
  - Diagonal ($j = m \implies d = 0$): Sums to $\sum_{k=0}^{n-1} (\omega^0)^k = n$.
  - Off-diagonal ($j \neq m \implies 0 < |d| < n$): Geometric series sums to 0.
- All off-diagonal terms vanish, giving $W \cdot V_n = \text{diag}(n, n, \ldots, n) = n \cdot I_n$.



Deriving the IFFT (Step 4): The Inverse Vandermonde Matrix (V_n^{-1})

- Since $W \cdot V_n = n \cdot I_n$, dividing by n gives the inverse matrix $V_n^{-1} = \frac{1}{n}W$:

$$V_n^{-1} = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

- V_n^{-1} is the exact same Vandermonde matrix as V_n, except ω_n is replaced by $\omega_n^{-1} = e^{-2\pi i/n}$ and the output is divided by n.
- We run our exact same $O(n \log n)$ split-and-merge FFT function with ω_n^{-1} and divide the outputs by n.



Pseudocode: Recursive FFT

RECURSIVE-FFT(a)

1 $n \leftarrow length[a]$ ▷ n is a power of 2.
2 **if** $n = 1$
3 **then return** a
4 $\omega_n \leftarrow e^{2\pi i/n}$
5 $\omega \leftarrow 1$
6 $a^{[0]} \leftarrow (a_0, a_2, \ldots, a_{n-2})$
7 $a^{[1]} \leftarrow (a_1, a_3, \ldots, a_{n-1})$
8 $y^{[0]} \leftarrow$ RECURSIVE-FFT($a^{[0]}$)
9 $y^{[1]} \leftarrow$ RECURSIVE-FFT($a^{[1]}$)
10 **for** $k \leftarrow 0$ **to** $n/2 - 1$
11 **do** $y_k \leftarrow y_k^{[0]} + \omega\, y_k^{[1]}$
12 $y_{k+(n/2)} \leftarrow y_k^{[0]} - \omega\, y_k^{[1]}$
13 $\omega \leftarrow \omega\, \omega_n$
14 **return** y ▷ y is assumed to be column vector.



Section 4

Number Theoretic Transform (NTT)



Number Theoretic Transform (NTT)

- Standard FFT uses complex roots of unity ($\omega_n = e^{2\pi i/n}$), which can suffer from floating-point precision errors.
- For the modular version (NTT), we work over a finite field $\mathbb{Z}_p$ for a prime $p = c \cdot 2^k + 1$ (e.g., $998244353 = 119 \cdot 2^{23} + 1$).
- If g is a primitive root of $\mathbb{Z}_p^\times$ (like $g = 3$), then $\omega_n \equiv g^{(p-1)/n} \pmod{p}$ satisfies the exact same Halving, Symmetry, and Orthogonality identities.
- The Vandermonde matrices and split-and-merge code remain completely unchanged.

