,

Fast Fourier Transform

Revolutionizing algorithm

Md. Tanvirul Islam Turad (2005011)
Tanvir Hossain (2005014)
Md. Jakaria Hossain (2005026)

Department of CSE
Bangladesh University of Engineering and Technology

March 8, 2024

Introduction to FFT

- The Fast Fourier Transform (FFT) is an algorithm to compute the Discrete Fourier Transform (DFT) and its inverse.
- It drastically reduces the computational complexity of computing DFT, making it feasible for real-time processing.
- Developed by Cooley and Tukey in 1965, FFT has become a fundamental tool in various fields such as signal processing, image processing, and more.
- This presentation aims to provide an overview of FFT, its significance, and applications.

Table of Contents

1. Background
2. Motivation
3. The Fast Fourier Transform (FFT)
4. Polynomial Representation
5. Evaluation
6. Interpolation
7. Conclusion

Fourier Transform

- Fourier Transform decomposes a signal into its frequency components.
- It represents a signal in terms of sinusoidal basis functions.
- The continuous Fourier Transform is given by:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

where $f(t)$ is the signal, and $F(\omega)$ is its frequency domain representation.

Discrete Fourier Transform (DFT)

- DFT is the discrete counterpart of the continuous Fourier Transform.
- It is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi kn/N}$$

  where $x[n]$ is the discrete signal, and $X[k]$ is its frequency domain representation.
- Direct computation of DFT is of $O(N^2)$ complexity.

Table of Contents

1. Background
2. Motivation
3. The Fast Fourier Transform (FFT)
4. Polynomial Representation
5. Evaluation
6. Interpolation
7. Conclusion

Motivation

- Analyze signals in the frequency domain to understand their composition.
- Limitations of the Discrete Fourier Transform (DFT):
  - Quadratic time complexity
    $$O(N^2)$$
    computationally expensive for large signals.
  - Need for a faster and more efficient algorithm.
- Applications:
  - Signal processing (noise removal, filtering,...)
  - Image processing (compression, feature extraction,...)
  - Speech and audio processing (compression, synthesis,...)
  - Scientific computing (solving differential equations, analyzing time-series data)

Table of Contents

1. Background
2. Motivation
3. The Fast Fourier Transform (FFT)
4. Polynomial Representation
5. Evaluation
6. Interpolation
7. Conclusion

The Fast Fourier Transform (FFT)

- FFT is an efficient algorithm for computing DFT.
- It exploits the periodicity and symmetry properties of sinusoidal functions to reduce the number of computations.
- The Cooley-Tukey algorithm is the most popular FFT algorithm.
- It divides the DFT computation into smaller DFTs, recursively applies FFT, and combines the results to obtain the final DFT.
- FFT reduces the computational complexity to $O(N \log N)$, making it feasible for real-time applications.

Table of Contents

1. Background
2. Motivation
3. The Fast Fourier Transform (FFT)
4. Polynomial Representation
5. Evaluation
6. Interpolation
7. Conclusion

An Optimization Problem

An Optimization Problem

Let's multiply two quadratic polynomials:

An Optimization Problem

Let's multiply two quadratic polynomials:

$$
\begin{aligned}
P(x) &= (a_0 + a_1 x + a_2 x^2) \times (b_0 + b_1 x + b_2 x^2) \\
&= c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4
\end{aligned}
$$

An Optimization Problem

Let's multiply two quadratic polynomials:

$$\begin{aligned} P(x) &= (a_0 + a_1x + a_2x^2) \times (b_0 + b_1x + b_2x^2) \\ &= c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 \end{aligned}$$

$$\begin{aligned} c_0 &= a_0 b_0 \\ c_1 &= a_0 b_1 + a_1 b_0 \\ c_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\ c_3 &= a_1 b_2 + a_2 b_1 \\ c_4 &= a_2 b_2 \end{aligned}$$

$$O(n^2)$$

An Optimization Problem

Let's multiply two quadratic polynomials:

$$P(x) = (a_0 + a_1x + a_2x^2) \times (b_0 + b_1x + b_2x^2)$$
$$= c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$$

$$\begin{aligned}
c_0 &= a_0 b_0 \\
c_1 &= a_0 b_1 + a_1 b_0 \\
c_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\
c_3 &= a_1 b_2 + a_2 b_1 \\
c_4 &= a_2 b_2
\end{aligned}$$

$$O(n^2)$$

Can we do better?

Polynomial Representation

Two Unique Representation for Polynomials:

$$P(x) = p_0 + p_1 x + p_1 x^2 + ... + p_d x^d$$

- Coefficient Representation:

$$[p_0, p_1, p_2, ...p_d]$$

- Value Representation:

$$[(x_0, P(x_0)), (x_1, P(x_1)), ...(x_d, P(x_d))]$$

Why Value Representation

$$A(x) = x^2 + 2x + 1$$

[(-2, 1), (-1, 0), (0,1), (1,4), (2, 9)]

$$B(x) = x^2 - 2x + 1$$

[(-2, 9), (-1,4), (0,1), (1,0), (2, 1)]

$$C(x) = A(x).B(x)$$

(-2, 1), (-1, 0), (0,1), (1,4), (2, 9)
x (-2, 9), (-1,4), (0,1), (1,0), (2, 1)

C(x) =(-2, 9), (-1,0), (0,1), (1,0), (2, 9)

Multiplication only needs O(d) time

Work Flow

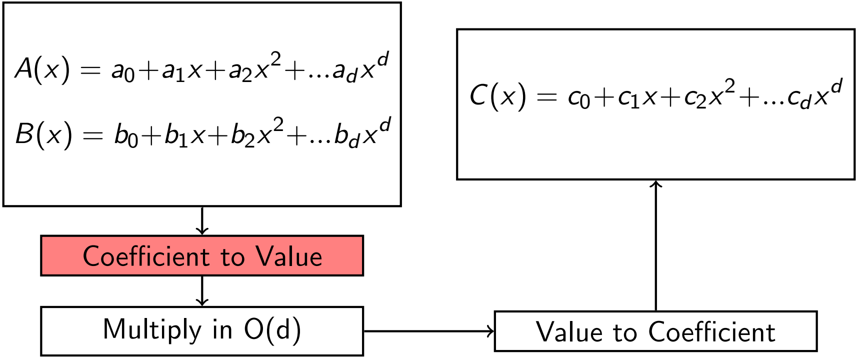

$A(x) = a_0 + a_1 x + a_2 x^2 + \dots a_d x^d$
$B(x) = b_0 + b_1 x + b_2 x^2 + \dots b_d x^d$
Coefficient to Value
Multiply in O(d)
Value to Coefficient
$C(x) = c_0 + c_1 x + c_2 x^2 + \dots c_d x^d$

Table of Contents

1. Background
2. Motivation
3. The Fast Fourier Transform (FFT)
4. Polynomial Representation
5. Evaluation
6. Interpolation
7. Conclusion

Evaluation

$$P(x) = x^2$$

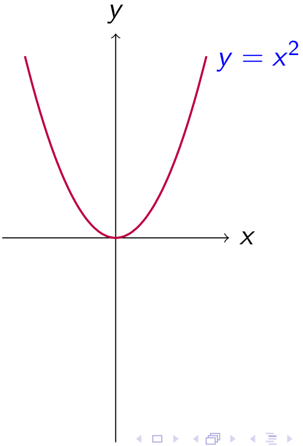
Evaluate at n = 8 points

Which Point Should We Pick?

(1,1)	(-1,1)
(2,4)	(-2,4)
(3,9)	(-3,9)
(4,16)	(-4,16)

$$P(x) = P(-x)$$

We need only 4 points!



Evaluation

$$P(x) = x^3$$

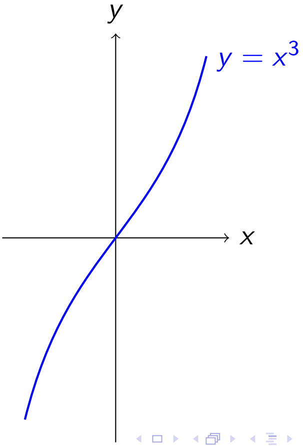
Evaluate at n = 8 points

Which Point Should We Pick?

(1,1)	(-1,-1)
(2,4)	(-2,-4)
(3,9)	(-3,-9)
(4,16)	(-4,-16)

$$P(x) = -P(-x)$$

We need only 4 points!



Evaluation

$$P(x) = p_0 + p_1 x + p_2 x^2 + ... + p_{n-1} x^{n-1}$$

Evaluate at n points $\pm x_1$,$\pm x_1$, $\pm x_2 ... \pm x_{n/2}$

$$P(x) = P_e(x^2) + xP_o(x^2)$$

$$P(x_i) = P_e(x_i^2) + xP_o(x_i^2)$$

$$P(-x_i) = P_e(x_i^2) - xP_o(x_i^2)$$

$P_e(x_i^2)$ and $P_o(x_i^2)$ have degree $n/2 - 1$

Evaluate $P_e(x_i^2)$ and $P_o(x_i^2)$ each at x_1^2, x_2^2, x_3^2, ..., $x_{n/2}^2$

Evaluation

$$P(x) = P_e(x^2) + xP_o(x^2)$$

$$P(x_i) = P_e(x_i^2) + xP_o(x_i^2)$$

$$P(-x_i) = P_e(x_i^2) - xP_o(x_i^2)$$

Points[$\pm\ x_1$, $\pm\ x_2$, $\pm\ x_3$, ... ,$\pm\ x_{n/2}$,] are $\pm$ paired.

Points[x_1^2, x_2^2, x_3^2, ... ,$x_{n/2}^2$,] are not $\pm$ paired.

Evaluation

$$P(x) = P_e(x^2) + xP_o(x^2)$$

$$P(x_i) = P_e(x_i^2) + xP_o(x_i^2)$$

$$P(-x_i) = P_e(x_i^2) - xP_o(x_i^2)$$

Points[$\pm\ x_1$, $\pm\ x_2$, $\pm\ x_3$, ... ,$\pm\ x_{n/2}$,] are $\pm$ paired.

Points[x_1^2, x_2^2, x_3^2, ... ,$x_{n/2}^2$,] are not $\pm$ paired.

Recursion breaks!!!

Is it possible to make [x_1^2, x_2^2, x_3^2, ... ,$x_{n/2}^2$,]$\pm$ paired?

Evaluation

$$P(x) = P_e(x^2) + xP_o(x^2)$$

$$P(x_i) = P_e(x_i^2) + xP_o(x_i^2)$$

$$P(-x_i) = P_e(x_i^2) - xP_o(x_i^2)$$

Points[$\pm\ x_1$, $\pm\ x_2$, $\pm\ x_3$, ... ,$\pm\ x_{n/2}$,] are $\pm$ paired.

Points[x_1^2, x_2^2, x_3^2, ... ,$x_{n/2}^2$,] are not $\pm$ paired.

Recursion breaks!!!

Is it possible to make [x_1^2, x_2^2, x_3^2, ... ,$x_{n/2}^2$,]$\pm$ paired?
Some of original [$\pm\ x_1$, $\pm\ x_2$, $\pm\ x_3$, ... ,$\pm\ x_{n/2}$,] need to be complex numbers!

Evaluation

$$P(x) = x^3 + x^2 - x - 1$$

We take 4 points : $\pm x_1, \pm x_2$

Evaluation

$$P(x) = x^3 + x^2 - x - 1$$

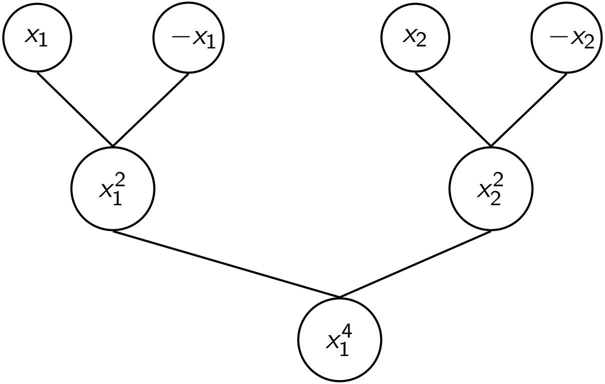
We take 4 points : $\pm x_1, \pm x_2$

x_1 $-x_1$ x_2 $-x_2$

Evaluation

$$P(x) = x^3 + x^2 - x - 1$$

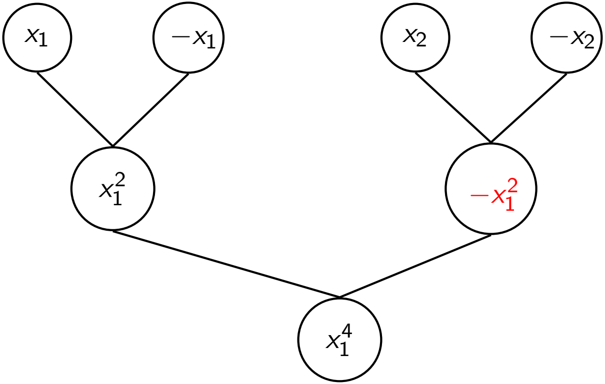
We take 4 points : $\pm x_1, \pm x_2$



Evaluation

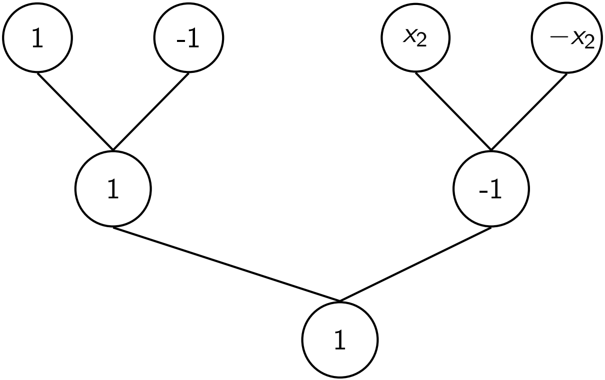
$$P(x) = x^3 + x^2 - x - 1$$

We take 4 points : $\pm x_1, \pm x_2$



Evaluation

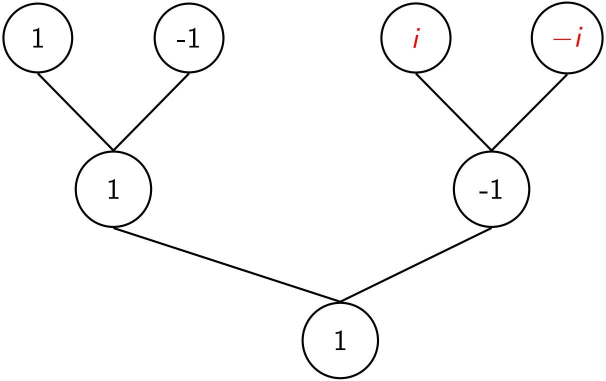
$$P(x) = x^3 + x^2 - x - 1$$



let $x_1 = 1 \implies x_2 = i$

Evaluation

$$P(x) = x^3 + x^2 - x - 1$$

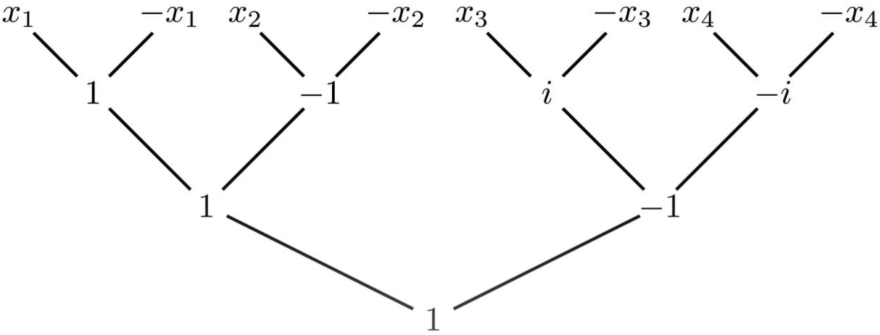


We take roots of $x^4 = 1$

Evaluation

$$P(x) = x^5 + 2x^4 - x^3 + x^2 + x + 1$$

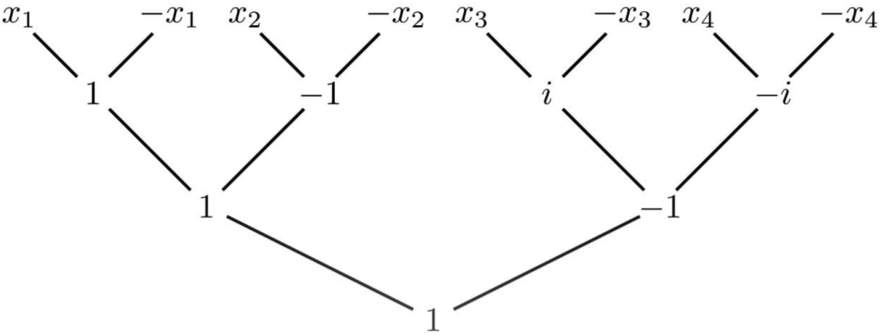
Need $n >= 6$ points, We take 8 points (powers of 2 are convenient)
Points are 8th roots of unity.



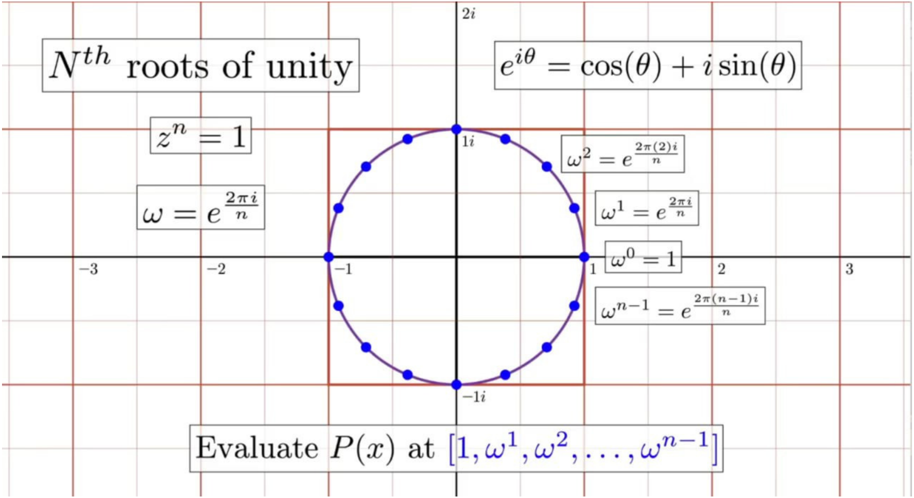
Evaluation

$$P(x) = p_0 + p_1 x + p_2 x^2 + ... + p_d x^d$$

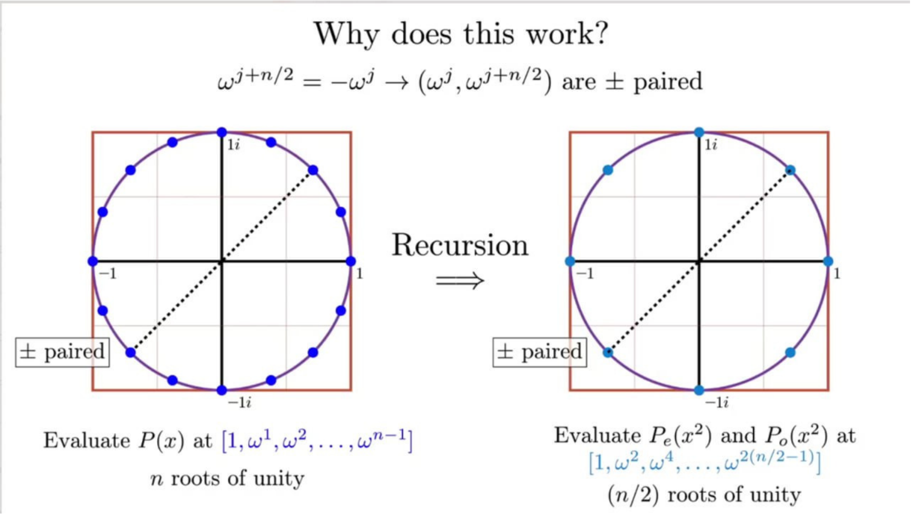
Need $n >= d + 1$ points to evaluate
where $n = 2^k, k \epsilon \mathbb{Z}$
The Points are nth roots of unity.



nth Roots of Unity



nth Roots of Unity



Implementation

$$\boxed{\text{FFT} \quad \begin{matrix} P(x) : [p_0, p_1, \ldots, p_{n-1}] \\ \omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \ldots, \omega^{n-1}] \end{matrix}}$$

$$\boxed{n = 1 \Rightarrow P(1)}$$

$$\boxed{\text{FFT} \quad \begin{matrix} P_e(x^2) : [p_0, p_2, \ldots, p_{n-2}] \\ [\omega^0, \omega^2, \ldots, \omega^{n-2}] \end{matrix}} \qquad \boxed{\text{FFT} \quad \begin{matrix} P_o(x^2) : [p_1, p_3, \ldots, p_{n-1}] \\ [\omega^0, \omega^2, \ldots, \omega^{n-2}] \end{matrix}}$$

$$y_e = [P_e(\omega^0), P_e(\omega^2), \ldots, P_e(\omega^{n-2})] \qquad y_o = [P_o(\omega^0), P_o(\omega^2), \ldots, P_o(\omega^{n-2})]$$

$$\boxed{\begin{matrix} P(x_j) = P_e(x_j^2) + x_j P_o(x_j^2) \\ P(-x_j) = P_e(x_j^2) - x_j P_o(x_j^2) \\ j \in \{0, 1, \ldots (n/2 - 1)\} \end{matrix}}$$

Implementation

$$\boxed{\text{FFT} \quad \begin{array}{c} P(x) : [p_0, p_1, \ldots, p_{n-1}] \\ \omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \ldots, \omega^{n-1}] \end{array}}$$

$$\boxed{n = 1 \Rightarrow P(1)}$$

$$\boxed{\text{FFT} \quad \begin{array}{c} P_e(x^2) : [p_0, p_2, \ldots, p_{n-2}] \\ [\omega^0, \omega^2, \ldots, \omega^{n-2}] \end{array}} \qquad \boxed{\text{FFT} \quad \begin{array}{c} P_o(x^2) : [p_1, p_3, \ldots, p_{n-1}] \\ [\omega^0, \omega^2, \ldots, \omega^{n-2}] \end{array}}$$

$$y_e = [P_e(\omega^0), P_e(\omega^2), \ldots, P_e(\omega^{n-2})] \qquad y_o = [P_o(\omega^0), P_o(\omega^2), \ldots, P_o(\omega^{n-2})]$$

$$x_j = \omega^j$$

$$\boxed{\begin{array}{c} P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j}) \\ P(-\omega^j) = P_e(\omega^{2j}) - \omega^j P_o(\omega^{2j}) \\ j \in \{0, 1, \ldots (n/2 - 1)\} \end{array}}$$

Implementation

FFT $\quad P(x) : [p_0, p_1, \ldots, p_{n-1}]$

$\omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \ldots, \omega^{n-1}]$

$$\boxed{n = 1 \Rightarrow P(1)}$$

FFT $\quad P_e(x^2) : [p_0, p_2, \ldots, p_{n-2}]$

$[\omega^0, \omega^2, \ldots, \omega^{n-2}]$

$$y_e = [P_e(\omega^0), P_e(\omega^2), \ldots, P_e(\omega^{n-2})]$$

FFT $\quad P_o(x^2) : [p_1, p_3, \ldots, p_{n-1}]$

$[\omega^0, \omega^2, \ldots, \omega^{n-2}]$

$$y_o = [P_o(\omega^0), P_o(\omega^2), \ldots, P_o(\omega^{n-2})]$$

$$x_j = \omega^j$$

$$-\omega^j = \omega^{j+n/2}$$

$$P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j})$$

$$P(\omega^{j+n/2}) = P_e(\omega^{2j}) - \omega^j P_o(\omega^{2j})$$

$$j \in \{0, 1, \ldots (n/2 - 1)\}$$

$$y_e[j] = P_e(\omega^{2j})$$

$$y_o[j] = P_o(\omega^{2j})$$

Implementation

$$\boxed{\text{FFT} \quad \begin{matrix} P(x) : [p_0, p_1, \ldots, p_{n-1}] \\ \omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \ldots, \omega^{n-1}] \end{matrix}}$$

$$\boxed{n = 1 \Rightarrow P(1)}$$

$$\boxed{\text{FFT} \quad \begin{matrix} P_e(x^2) : [p_0, p_2, \ldots, p_{n-2}] \\ [\omega^0, \omega^2, \ldots, \omega^{n-2}] \end{matrix}} \qquad \boxed{\text{FFT} \quad \begin{matrix} P_o(x^2) : [p_1, p_3, \ldots, p_{n-1}] \\ [\omega^0, \omega^2, \ldots, \omega^{n-2}] \end{matrix}}$$

$$y_e = [P_e(\omega^0), P_e(\omega^2), \ldots, P_e(\omega^{n-2})] \qquad y_o = [P_o(\omega^0), P_o(\omega^2), \ldots, P_o(\omega^{n-2})]$$

$$x_j = \omega^j$$

$$-\omega^j = \omega^{j+n/2}$$

$$\boxed{\begin{matrix} P(\omega^j) = y_e[j] + \omega^j y_o[j]) \\ P(\omega^{j+n/2}) = y_e[j] - \omega^j y_o[j]) \\ j \in \{0, 1, \ldots (n/2 - 1)\} \end{matrix}}$$

$$y_e[j] = P_e(\omega^{2j})$$

$$y_o[j] = P_o(\omega^{2j})$$

$$\boxed{y = [P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]}$$

Implementation

```
def FFT(P):
    # P - [p_0, p_1, ..., p_{n-1}] coeff representation
    n = len(P) # n is a power of 2
    if n == 1:
        return P
    ω = e^(2πi/n)
    P_e, P_o = [p_0, p_2, ..., p_{n-2}], [p_1, p_3, ..., p_{n-1}]
    y_e, y_o = FFT(P_e), FFT(P_o)
    y = [0] * n
    for j in range(n/2):
        y[j] = y_e[j] + ω^j y_o[j]
        y[j + n/2] = y_e[j] − ω^j y_o[j]
    return y
```

FFT $P(x) : [p_0, p_1, \ldots, p_{n-1}]$, $\omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \ldots, \omega^{n-1}]$

$n = 1 \Rightarrow P(1)$

FFT $P_e(x^2) : [p_0, p_2, \ldots, p_{n-2}]$, $[\omega^0, \omega^2, \ldots, \omega^{n-2}]$

$y_e = [P_e(\omega^0), P_e(\omega^2), \ldots, P_e(\omega^{n-2})]$

FFT $P_o(x^2) : [p_1, p_3, \ldots, p_{n-1}]$, $[\omega^0, \omega^2, \ldots, \omega^{n-2}]$

$y_o = [P_o(\omega^0), P_o(\omega^2), \ldots, P_o(\omega^{n-2})]$

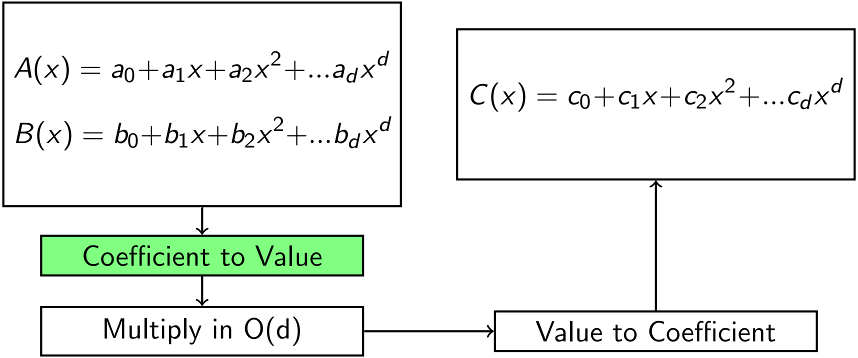
$P(\omega^j) = y_e[j] + \omega^j y_o[j])$

$P(\omega^{j+n/2}) = y_e[j] - \omega^j y_o[j])$

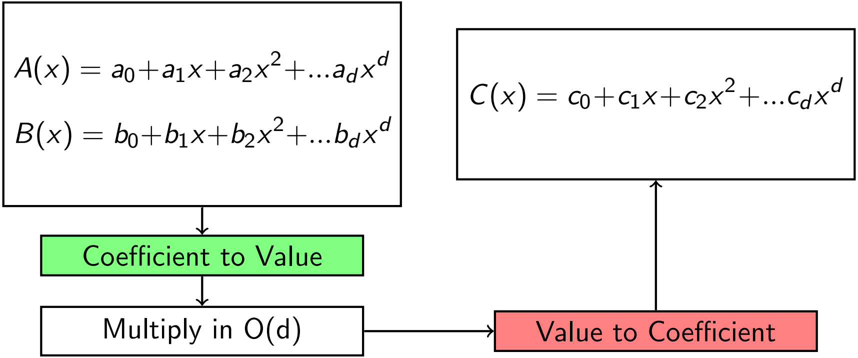
$j \in \{0, 1, \ldots (n/2 - 1)\}$

$y = [P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]$

Work Flow


$A(x) = a_0 + a_1x + a_2x^2 + \ldots a_dx^d$
$B(x) = b_0 + b_1x + b_2x^2 + \ldots b_dx^d$
Coefficient to Value
Multiply in O(d)
Value to Coefficient
$C(x) = c_0 + c_1x + c_2x^2 + \ldots c_dx^d$

Work Flow



$A(x) = a_0 + a_1x + a_2x^2 + \dots a_dx^d$
$B(x) = b_0 + b_1x + b_2x^2 + \dots b_dx^d$
Coefficient to Value
Multiply in O(d)
Value to Coefficient
$C(x) = c_0 + c_1x + c_2x^2 + \dots c_dx^d$

Work Flow

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_d x^d$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots b_d x^d$$

Coefficient to Value

Multiply in O(d)

Value to Coefficient

$$C(x) = c_0 + c_1 x + c_2 x^2 + \ldots c_d x^d$$

HOW!

Table of Contents

1. Background
2. Motivation
3. The Fast Fourier Transform (FFT)
4. Polynomial Representation
5. Evaluation
6. Interpolation
7. Conclusion

Interpolation

Alternative Perspective on Evaluation/FFT

$$P(x) = p_0 + p_1 x + p_2 x^2 + ... + p_{n-1} x^{n-1}$$

$$P(x_0) = p_0 + p_1 x_0 + p_2 x_0^2 + ... + p_{n-1} x_0^{n-1}$$

$$P(x_1) = p_0 + p_1 x_1 + p_2 x_1^2 + ... + p_{n-1} x_1^{n-1}$$

$$P(x_2) = p_0 + p_1 x_2 + p_2 x_2^2 + .. + p_{n-1} x_2^{n-1}$$

$$P(x_{n-1}) = p_0 + p_1 x_{n-1} + p_2 x_{n-1}^2 + ... + p_{n-1} x_{n-1}^{n-1}$$

Alternative Perspective on Evaluation

$$P(x) = p_0 + p_1 x + p_2 x^2 + \dots + p_{n-1} x^{n-1}$$

$$\begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

$$x_k = \omega^k, \text{ where } \omega = e^{\frac{2\pi i}{n}}$$

Interpolation

Alternative Perspective on Evaluation/FFT

$$P(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1}$$

$$\begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}}_{\text{Discrete Fourier Transform (DFT) matrix}} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

$$x_k = \omega^k, \text{ where } \omega = e^{\frac{2\pi i}{n}}$$

Interpolation

Alternative Perspective on Evaluation/FFT

$$P(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1}$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

$$x_k = \omega^k, \text{ where } \omega = e^{\frac{2\pi i}{n}}$$

Interpolation

$$x_k = \omega^k \text{ where } \omega = e^{\frac{2\pi i}{n}}$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

The inverse matrix and original matrix look quiet similar!

Every ωin original matrix is now $\frac{1}{n}\omega^{-}1$

Interpolation

$$x_k = \omega^k \text{ where } \omega = e^{\frac{2\pi i}{n}}$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

The inverse matrix and original matrix look quiet similar!

Every ωin original matrix is now $\frac{1}{n}\omega^-1$

$$
\begin{aligned}
&\text{def FFT}(P):\\
&\quad \#\ P\ \text{-}\ [p_0, p_1, \ldots, p_{n-1}]\ \text{coeff rep}\\
&\quad n = \text{len}(P)\ \#\ n \text{ is a power of } 2\\
&\quad \text{if } n == 1:\\
&\quad\quad \text{return } P\\
&\quad \omega = e^{\frac{2\pi i}{n}}\\
&\quad P_e, P_o = P[::2], P[1::2]\\
&\quad y_e, y_o = \text{FFT}(P_e), \text{FFT}(P_o)\\
&\quad y = [0] * n\\
&\quad \text{for } j \text{ in range}(n/2):\\
&\quad\quad y[j] = y_e[j] + \omega^j y_o[j]\\
&\quad\quad y[j + n/2] = y_e[j] - \omega^j y_o[j]\\
&\quad \text{return } y
\end{aligned}
$$

Interpolation

$$\text{IFFT}(\textless\text{values}\textgreater) \Leftrightarrow \text{FFT}(\textless\text{values}\textgreater) \text{ with } \omega = \frac{1}{n} e^{\frac{-2\pi i}{n}}$$

```
def FFT(P):
    # P - [p_0, p_1, ..., p_{n-1}] coeff rep
    n = len(P) # n is a power of 2
    if n == 1:
        return P
    ω = e^(2πi/n)
    P_e, P_o = P[::2], P[1::2]
    y_e, y_o = FFT(P_e), FFT(P_o)
    y = [0] * n
    for j in range(n/2):
        y[j] = y_e[j] + ω^j y_o[j]
        y[j + n/2] = y_e[j] - ω^j y_o[j]
    return y
```

```
def IFFT(P):
    # P - [P(ω^0), P(ω^1), ..., P(ω^(n-1))] value rep
    n = len(P) # n is a power of 2
    if n == 1:
        return P
    ω = (1/n) * e^(-2πi/n)
    P_e, P_o = P[::2], P[1::2]
    y_e, y_o = IFFT(P_e), IFFT(P_o)
    y = [0] * n
    for j in range(n/2):
        y[j] = y_e[j] + ω^j y_o[j]
        y[j + n/2] = y_e[j] - ω^j y_o[j]
    return y
```

Example Problems

All possible sums

We are given two arrays $a[]$ and $b[]$. We have to find all possible sums $a[i] + b[j]$, and for each sum count how often it appears.
For example for $a = [1,\ 2,\ 3]$ and $b = [2,\ 4]$ we get: then sum 3 can be obtained in 1 way, the sum 4 also in 1 way, 5 in 2, 6 in 1, 7 in 1.

Hint :

Construct for the arrays a and b two polynomials A and B. The numbers of the array will act as the exponents in the polynomial ($a[i] \Rightarrow x^{a[i]}$); and the coefficients of this term will be how often the number appears in the array.

Example Problems

String matching

We are given two strings, a text T and a pattern P, consisting of lowercase letters. We have to compute all the occurrences of the pattern in the text.

Hint :

Create a polynomial for each string ($T[i]$ and $P[I]$ are numbers between 0 and 25 corresponding to the 26 letters of the alphabet):

$$A(x) = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1}, \quad n = |T|$$

with $a_i = \cos(\alpha_i) + i\sin(\alpha_i), \quad \alpha_i = \frac{2\pi T[i]}{26}$.
And

$$B(x) = b_0x^0 + b_1x^1 + \cdots + b_{m-1}x^{m-1}, \quad m = |P|$$

with $b_i = \cos(\beta_i) - i\sin(\beta_i), \quad \beta_i = \frac{2\pi P[m-i-1]}{26}$.

More Problems to try:

POLYMUL - Polynomial Multiplication

MAXMATCH - Maximum Self-Matching

ADAMATCH - Ada and Nucleobase

Yet Another String Matching Problem

Lightsabers (hard)

Running Competition

Table of Contents

1. Background
2. Motivation
3. The Fast Fourier Transform (FFT)
4. Polynomial Representation
5. Evaluation
6. Interpolation
7. Conclusion

Conclusion

- FFT is a powerful algorithm for efficiently computing the Discrete Fourier Transform.
- It has revolutionized various fields by enabling fast and accurate frequency domain analysis.
- Understanding FFT and its applications is essential for anyone working in signal processing, communications, image processing, and related domains.

Thank You for your patience!