CS6161: Design and Analysis of Algorithms (Fall 2020)

Fast Fourier Transform (FFT)

Instructor: Haifeng Xu

Outline

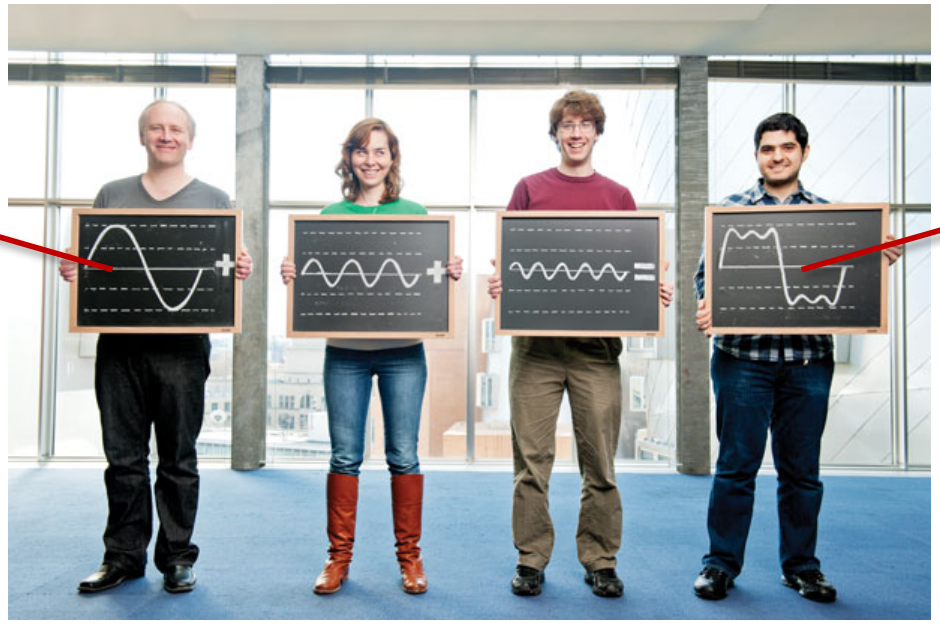
- Discrete Fourier Transform (DFT)
- Fast Fourier Transform
  - A fast way to compute DFT
- An Application of FFT

Fourier Transform

- Decomposes a function into linear combination of base functions

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \, e^{2\pi i \cdot \frac{kx}{n}}$$

Base "wave" function $e^{2\pi i \cdot \frac{kx}{n}}$



$f(x)$ is usually a "wave" function

What is $e^{2\pi i\frac{k}{n}}$?

It comes from complex analysis → $i = \sqrt{-1}$ is called *imaginary unit*

- It is okay, if you do not know

"**Definition**": $e^{xi} = \cos(x) + \sin(x) \cdot i$

Fact 1: $e^{2\pi i} = \cos(2\pi) + \sin(2\pi) \cdot i = 1$

Fact 2: $e^{2\pi i \cdot \frac{k}{n}}, \forall k = 0,1,\cdots,n-1$ are the roots of $x^n = 1$

- Proof: $\left(e^{2\pi i \cdot \frac{k}{n}}\right)^n = e^{2\pi i \cdot k}$

What is $e^{2\pi i\frac{k}{n}}$?

It comes from complex analysis → $i = \sqrt{-1}$ is called *imaginary unit*

- It is okay, if you do not know

"**Definition**": $e^{xi} = \cos(x) + \sin(x) \cdot i$

Fact 1: $e^{2\pi i} = \cos(2\pi) + \sin(2\pi) \cdot i = 1$

Fact 2: $e^{2\pi i \cdot \frac{k}{n}}, \forall k = 0,1,\cdots,n-1$ are the roots of $x^n = 1$

- Proof: $\left(e^{2\pi i \cdot \frac{k}{n}}\right)^n = e^{2\pi i \cdot k} = \cos(2\pi k) + \sin(2\pi k) \cdot i$

By **Definition**

What is $e^{2\pi i\frac{k}{n}}$?

It comes from complex analysis → $i = \sqrt{-1}$ is called *imaginary unit*

- It is okay, if you do not know

"**Definition**": $e^{xi} = \cos(x) + \sin(x) \cdot i$

Fact 1: $e^{2\pi i} = \cos(2\pi) + \sin(2\pi) \cdot i = 1$

Fact 2: $e^{2\pi i \cdot \frac{k}{n}}, \forall k = 0,1,\cdots,n-1$ are the roots of $x^n = 1$

- Proof: $\left(e^{2\pi i \cdot \frac{k}{n}}\right)^n = e^{2\pi i \cdot k} = \cos(2\pi k) + \sin(2\pi k) \cdot i = 1$

What is $e^{2\pi i\frac{k}{n}}$?

It comes from complex analysis → $i = \sqrt{-1}$ is called *imaginary unit*

- It is okay, if you do not know

"**Definition**": $e^{xi} = \cos(x) + \sin(x) \cdot i$

Fact 1: $e^{2\pi i} = \cos(2\pi) + \sin(2\pi) \cdot i = 1$

Fact 2: $e^{2\pi i \cdot \frac{k}{n}}, \forall k = 0,1,\cdots,n-1$ are the roots of $x^n = 1$

- When $k = 0$, $e^{2\pi i \cdot \frac{k}{n}} = \mathrm{e}^0 = 1$
- When $k = 1,\cdots,n-1$, $e^{2\pi i \cdot \frac{k}{n}}$ is a *complex number*

FYI: Complex Numbers

- Extension of real numbers
- Have format $a + bi$ where a, b are real numbers
- Many reasons to define complex numbers (won't cover here)
- Follows standard calculation

$$(a + bi) \cdot (c + di) = ac + bci + adi + bdi^2$$
$$= ac + bci + adi - bd$$

Use definition $i^2 = -1$

FYI: Complex Numbers

- Extension of real numbers
- Have format $a + bi$ where a, b are real numbers
- Many reasons to define complex numbers (won't cover here)
- Follows standard calculation

$$(a + bi) \cdot (c + di) = ac + bci + adi + bdi^2$$
$$= ac + bci + adi - bd$$
$$= ac - bd + (bc + ad)i$$

Rearrange terms

- If you are not familiar, knowing the following is sufficient

"**Definition**": $e^{xi} = \cos(x) + \sin(x) \cdot i$

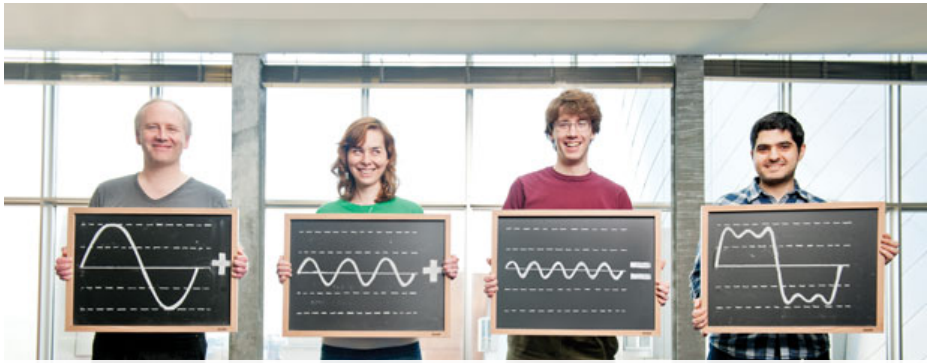
Fact 2: $e^{2\pi i \cdot \frac{k}{n}}, \forall k = 0,1,\cdots,n-1$ are the roots of $x^n = 1$

Remark: Calculations follows the same rule as real numbers, and $i^2 = -1$

Back to Fourier Transform…

- Decomposes a function into linear combination of base functions

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \, e^{2\pi i \cdot \frac{kx}{n}}$$



What if we only observe $f(x)$ at discrete time $0,1,\cdots,n-1$?

- Indeed, in applications computer can never record the entire $f(x)$

Discrete Fourier Transform (DFT)

- Decomposes data observations into linear combination of fnc $e^{2\pi i \frac{kx}{n}}$

Definition (DFT). Given data sequence $f = (f_0, \cdots, f_{n-1})^T$, its discrete Fourier transform (DFT) is the coefficients $\hat{f} = \left(\hat{f}_0, \cdots, \hat{f}_{n-1}\right)^T$ satisfying

$$f_j = \frac{1}{n} \sum_{k=0}^{n-1} \hat{f}_k \cdot e^{2\pi i \frac{kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

- Too many applications to mention
  - Audio/image compression, de-noising, computing derivatives, solving differential equations…
- All in all: we want a fast algorithm to computer $\hat{f}$

A Useful Fact

Theorem. DFT $\hat{f} = (\hat{f}_0, \cdots, \hat{f}_{n-1})^T$ for any sequence $f = (f_0, \cdots, f_{n-1})^T$ satisfies the following equation:

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

Compare with DFT definition:

$$f_j = \frac{1}{n} \sum_{k=0}^{n-1} \hat{f}_k \cdot e^{2\pi i \cdot \frac{kj}{n}}$$

A Useful Fact

Theorem. DFT $\hat{f} = (\hat{f}_0, \cdots, \hat{f}_{n-1})^T$ for any sequence $f = (f_0, \cdots, f_{n-1})^T$ satisfies the following equation:

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

Compare with DFT definition:

$$f_j = \frac{1}{n} \sum_{k=0}^{n-1} \hat{f}_k \cdot e^{2\pi i \cdot \frac{kj}{n}}$$

Therefore, f and its DFT $\hat{f}$ can be transformed to each other

Proof of $\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$

- Recall definition $f_k = \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}}$
- Just algebraic calculations

$$\sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}$$

Proof of $\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \quad \forall j = 0, \cdots, n-1$

- Recall definition $f_k = \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}}$

- Just algebraic calculations

$$\sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}} = \sum_{k=0}^{n-1} \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}} \cdot e^{2\pi i \cdot \frac{-kj}{n}}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k(k'-j)}{n}}$$

Combine the last two terms

Proof of $\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$

- Recall definition $f_k = \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}}$

- Just algebraic calculations

$$\sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}} = \sum_{k=0}^{n-1} \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}} \cdot e^{2\pi i \cdot \frac{-kj}{n}}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k(k'-j)}{n}}$$

$$= \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \left[\sum_{k=0}^{n-1} e^{2\pi i \cdot \frac{k(k'-j)}{n}} \right]$$

Switch the order of the sum

Proof of $\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \quad \forall j = 0, \cdots, n-1$

- Recall definition $f_k = \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}}$
- Just algebraic calculations

$$\sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}} = \sum_{k=0}^{n-1} \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}} \cdot e^{2\pi i \cdot \frac{-kj}{n}}$$

$$= \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \left[\sum_{k=0}^{n-1} e^{2\pi i \cdot \frac{k(k'-j)}{n}} \right]$$

Claim. For any $k' \neq j$, we have $\sum_{k=0}^{n-1} e^{2\pi i \cdot \frac{k(k'-j)}{n}} = 0$

- $\bar{x} = e^{2\pi i \cdot \frac{(k'-j)}{n}}$ is a solution to $0 = 1 - x^n = (1-x)(1 + x + \cdots x^{n-1})$
- So $\bar{x}$ satisfies $1 + \bar{x} + \cdots \bar{x}^{n-1} = 0$, as claimed

Proof of $\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \quad \forall j = 0, \cdots, n-1$

- Recall definition $f_j = \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'j}{n}}$

- Just algebraic calculations

$$\sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}} = \sum_{k=0}^{n-1} \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}} \cdot e^{2\pi i \cdot \frac{-kj}{n}}$$

$$= \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \underbrace{\left[\sum_{k=0}^{n-1} e^{2\pi i \cdot \frac{k(k'-j)}{n}} \right]}$$

Non-zero only when $k' = j$

Claim. For any $k' \neq j$, we have $\sum_{k=0}^{n-1} e^{2\pi i \cdot \frac{k(k'-j)}{n}} = 0$

Proof of $\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$

- Recall definition $f_j = \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'j}{n}}$

- Just algebraic calculations

$$\sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}} = \sum_{k=0}^{n-1} \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \cdot e^{2\pi i \cdot \frac{k'k}{n}} \cdot e^{2\pi i \cdot \frac{-kj}{n}}$$

$$= \frac{1}{n} \sum_{k'=0}^{n-1} \hat{f}_{k'} \left[\sum_{k=0}^{n-1} e^{2\pi i \cdot \frac{k(k'-j)}{n}} \right]$$

$$= \frac{1}{n} \hat{f}_j \left[\sum_{k=0}^{n-1} 1 \right]$$

$$= \hat{f}_j$$

Compute Discrete Fourier Transform (DFT)

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

Compute Discrete Fourier Transform (DFT)

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

- DFT in matrix form

$$\begin{pmatrix} \hat{f}_0 \\ \\ \end{pmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

$$\hat{f}_0 = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-k \cdot 0}{n}} = \sum_{k=0}^{n-1} f_k \cdot 1$$

Compute Discrete Fourier Transform (DFT)

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

- DFT in matrix form

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \\ \\ \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

$$\hat{f}_1 = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-k \cdot 1}{n}}$$

Compute Discrete Fourier Transform (DFT)

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

- DFT in matrix form

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \\ \\ \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

$\hat{f}_1 = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-k \cdot 1}{n}}$

Let $\omega_n = e^{\frac{-2\pi i}{n}}$

Compute Discrete Fourier Transform (DFT)

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

- DFT in matrix form

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \\ \\ \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

$$\hat{f}_1 = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-k \cdot 1}{n}} = \sum_{k=0}^{n-1} f_k \cdot \omega_n^k \qquad \text{Let } \omega_n = e^{\frac{-2\pi i}{n}}$$

Compute Discrete Fourier Transform (DFT)

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

- DFT in matrix form

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \\ \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & & & \\ & & & & \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

$\hat{f}_2 = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-k \cdot 2}{n}}$

Let $\omega_n = e^{\frac{-2\pi i}{n}}$

Compute Discrete Fourier Transform (DFT)

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1$$

- DFT in matrix form

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_{n-1} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & & \omega_n^{2(n-1)} \\ & \vdots & & \ddots & \vdots \\ 1 & \omega_n^{n-1} & & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

Straightforward multiplication takes $O(n^2)$ time

Can we do better?

Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform
  - A fast way to compute DFT
- An Application of FFT

Another Interpretation of DFT

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}}, \qquad \forall j = 0, \cdots, n-1 \qquad \text{Let } \omega_n = e^{\frac{-2\pi i}{n}}$$

Another Interpretation of DFT

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot \omega_n^{kj}, \qquad \forall j = 0, \cdots, n-1$$

Let $\omega_n = e^{\frac{-2\pi i}{n}}$

Another Interpretation of DFT

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot \left(\omega_n^j\right)^k, \qquad \forall j = 0, \cdots, n-1 \qquad \text{Let } \omega_n = e^{\frac{-2\pi i}{n}}$$

Observation. $\hat{f}_j$ is the value of the polynomial $\hat{f}(x) = \sum_{k=0}^{n-1} f_k\, x^k$ evaluated at $x = (\omega_n)^j$.

Instead of calculating following matrix, we can just compute $\hat{f}(x)$ at $x = 1, \omega_n, (\omega_n)^2, \ldots, (\omega_n)^{n-1}$

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_{n-1} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & & \omega_n^{2(n-1)} \\ & \vdots & & \ddots & \vdots \\ 1 & \omega_n^{n-1} & & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

Another Interpretation of DFT

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot \left(\omega_n^j\right)^k, \qquad \forall j = 0, \cdots, n-1 \qquad \text{Let } \omega_n = e^{\frac{-2\pi i}{n}}$$

Observation. $\hat{f}_j$ is the value of the polynomial $\hat{f}(x) = \sum_{k=0}^{n-1} f_k\, x^k$ evaluated at $x = (\omega_n)^j$.

Instead of calculating following matrix, we can just compute $\hat{f}(x)$ at $x = 1, \omega_n, (\omega_n)^2, \ldots, (\omega_n)^{n-1}$

This is the core idea of Fast Fourier Transform (FFT)

The Algorithmic Problem

Input: $f = (f_0, \cdots, f_{n-1})^T$

Output: $\hat{f} = \left(\hat{f}_0, \cdots, \hat{f}_{n-1}\right)^T$ where $\hat{f}_j = \hat{f}\left(\omega_n^j\right)$ and $\hat{f}(x) = \sum_{k=0}^{n-1} f_k \, x^k$

Natural first attempt: calculate each $\hat{f}_j$ separately

- This does not buy you anything, still $O(n^2)$ time

We need something cleverer

Divide and Conquer!

The Algorithm

- Assume $n = 2m$ is even, for convenience
- A closer look of $\hat{f}(x) = \sum_{k=0}^{n-1} f_k x^k$

$$\hat{f}(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots f_{n-2} x^{n-2} + f_{n-1} x^{n-1}$$

$$= \underbrace{f_0 + f_2 x^2 + \cdots f_{n-2} x^{n-2}}_{\text{Even terms}} + \underbrace{f_1 x + f_3 x^3 + \cdots f_{n-1} x^{2m-1}}_{\text{Odd terms}}$$

The Algorithm

- Assume $n = 2m$ is even, for convenience
- A closer look of $\hat{f}(x) = \sum_{k=0}^{n-1} f_k x^k$

$$\hat{f}(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots f_{n-2} x^{n-2} + f_{n-1} x^{n-1}$$

$$= f_0 + f_2 x^2 + \cdots f_{n-2} x^{n-2} + f_1 x + f_3 x^3 + \cdots f_{n-1} x^{2m-1}$$

$$= \widehat{f}_{even}(x^2) + x\widehat{f}_{odd}(x^2)$$

where $\widehat{f}_{even}(x) = f_0 + f_2 x + \cdots f_{n-2} x^{\frac{n-2}{2}}$

$\widehat{f}_{odd}(x) = f_1 + f_3 x + \cdots f_{n-1} x^{\frac{n-2}{2}}$

Note $\frac{n-2}{2} = m - 1$

The Algorithm

$$\hat{f}_{even}(x) = f_0 + f_2 x + \cdots f_{n-2} x^{m-1}$$

$$\hat{f}_{odd}(x) = f_1 + f_3 x + \cdots f_{n-1} x^{m-1}$$

- Assume $n = 2m$ is even, for convenience

Observation. $\hat{f}(x) = \hat{f}_{even}(x^2) + x\hat{f}_{odd}(x^2)$ effectively reduces the algorithmic problem to two sub-problems of half size

- We want to compute $\hat{f}(x)$ for $x = 1,\ \omega_n, \omega_n^2, \cdots, \omega_n^{n-1}$
- Importantly, for any $x = \omega_n^j$

$$x^2 = \left(\omega_n^j\right)^2 = \left(e^{-2\pi i \frac{j}{n}}\right)^2$$

The Algorithm

$$\hat{f}_{even}(x) = f_0 + f_2 x + \cdots f_{n-2} x^{m-1}$$

$$\hat{f}_{odd}(x) = f_1 + f_3 x + \cdots f_{n-1} x^{m-1}$$

- Assume $n = 2m$ is even, for convenience

Observation. $\hat{f}(x) = \hat{f}_{even}(x^2) + x\hat{f}_{odd}(x^2)$ effectively reduces the algorithmic problem to two sub-problems of half size

- We want to compute $\hat{f}(x)$ for $x = 1,\ \omega_n, \omega_n^2, \cdots, \omega_n^{n-1}$
- Importantly, for any $x = \omega_n^j$

$$x^2 = \left(\omega_n^j\right)^2 = \left(e^{-2\pi i \frac{j}{n}}\right)^2 = e^{-2\pi i \frac{2j}{n}}$$

The Algorithm

$$\hat{f}_{even}(x) = f_0 + f_2 x + \cdots f_{n-2} x^{m-1}$$

$$\hat{f}_{odd}(x) = f_1 + f_3 x + \cdots f_{n-1} x^{m-1}$$

- Assume $n = 2m$ is even, for convenience

Observation. $\hat{f}(x) = \hat{f}_{even}(x^2) + x\hat{f}_{odd}(x^2)$ effectively reduces the algorithmic problem to two sub-problems of half size

- We want to compute $\hat{f}(x)$ for $x = 1,\ \omega_n, \omega_n^2, \cdots, \omega_n^{n-1}$
- Importantly, for any $x = \omega_n^j$

$$x^2 = \left(\omega_n^j\right)^2 = \left(e^{-2\pi i \frac{j}{n}}\right)^2 = e^{-2\pi i \frac{2j}{n}} = e^{-2\pi i \frac{j}{m}}$$

The Algorithm

$$\hat{f}_{even}(x) = f_0 + f_2 x + \cdots f_{n-2} x^{m-1}$$

$$\hat{f}_{odd}(x) = f_1 + f_3 x + \cdots f_{n-1} x^{m-1}$$

- Assume $n = 2m$ is even, for convenience

Observation. $\hat{f}(x) = \hat{f}_{even}(x^2) + x\hat{f}_{odd}(x^2)$ effectively reduces the algorithmic problem to two sub-problems of half size

- We want to compute $\hat{f}(x)$ for $x = 1,\ \omega_n, \omega_n^2, \cdots, \omega_n^{n-1}$
- Importantly, for any $x = \omega_n^j$

$$x^2 = \left(\omega_n^j\right)^2 = \left(e^{-2\pi i \frac{j}{n}}\right)^2 = e^{-2\pi i \frac{2j}{n}} = e^{-2\pi i \frac{j}{m}} = \omega_m^j$$

- So, if we computed $\hat{f}_{even}(x)$ and $\hat{f}_{odd}(x)$ for $x = 1, \omega_m, \omega_m^2, \cdots, \omega_n^{n-1}$, then $\hat{f}(x)$ for $x = 1,\ \omega_n, \omega_n^2, \cdots, \omega_n^{n-1}$ can be computed in $O(n)$ time

FFT: Pseudo-Code

FFT: Pseudo-Code

FFT(f) // Compute DFT for f

1 Let f_{even}, f_{odd} be the even and off subsequences of f, respectively

2

3

4

5

FFT: Pseudo-Code

FFT(f) // Compute DFT for f

1 Let f_{even}, f_{odd} be the even and off subsequences of f, respectively

2 $\hat{f}_{even} = \text{FFT}(f_{even})$ → These are $\hat{f}_{even}\left(\omega_{n/2}^{j}\right), \forall j = 0, \cdots, \frac{n}{2}$

3 $\hat{f}_{odd} = \text{FFT}(f_{odd})$

4

5

FFT: Pseudo-Code

FFT(f) // Compute DFT for f

1 Let f_{even}, f_{odd} be the even and off subsequences of f, respectively

2 $\hat{f}_{even} = \text{FFT}(f_{even})$

3 $\hat{f}_{odd} = \text{FFT}(f_{odd})$

4 **For** $j = 0, \cdots, n-1$

5 $\hat{f}(\omega_n^j) = \hat{f}_{even}\left(\omega_{n/2}^j\right) + \omega_n^j \cdot \hat{f}_{odd}(\omega_{n/2}^j)$

- Correctness follows from our analysis
- Running time satisfies recursion $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$
  - Master theorem implies $T(n) = \Theta(n \log n)$

Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform
  - A fast way to compute DFT
- An Application of FFT

Multiplying Two Polynomials

- Two polynomials:

$$A(x) = a_0 + a_1 x + \cdots a_n x^n$$
$$B(x) = b_0 + b_1 x + \cdots b_n x^n$$

- Want to compute $A(x) \cdot B(x) = C(x) = c_0 + c_1 x + \cdots c_{2n} x^{2n}$ where

$$c_k = \sum_{i+j=k} a_i b_j$$

The Algorithmic Problem

Input: two polynomial represented by coefficient vectors a, b

Output: The coefficient vector c of their product polynomial

Directly calculate each c_k one by one → $O(n^2)$ time

Is There a Faster Algorithm?

- Recall discrete Fourier transform

$$\hat{f}_j = \sum_{k=0}^{n-1} f_k \cdot e^{2\pi i \cdot \frac{-kj}{n}} = \sum_{k=0}^{n-1} f_k \cdot \left(\omega_n^j\right)^k, \ \forall j = 0, \cdots, n-1$$

$$f_j = \frac{1}{n}\sum_{k=0}^{n-1} \hat{f}_k \cdot e^{2\pi i \cdot \frac{kj}{n}} = \frac{1}{n}\sum_{k=0}^{n-1} \hat{f}_k \cdot \left(\omega_n^{-j}\right)^k, \ \forall j = 0, \cdots, n-1$$

Learned Lessons

Coefficients of an n-degree polynomial and its values at ω_n^js can be transformed to each other *effectively* via FFT

$O(n \log n)$ for Polynomial Products

Input: two degree-$(n-1)$ polynomial with coefficients a, b

Output: coefficient vector c of their product

POLY-PRODUCT(a, b) // Compute DFT for f

1 Compute $A(x), B(x)$ for $x = x_j = e^{-2\pi i \frac{j}{2n-1}}, j = 0, \cdots, 2n-2$

2 Compute $C(x) = A(x)B(x)$ for $x = x_j, j = 0, \cdots, 2n-2$

3 c = FFT($\{C(x_j)\}_{j=0,\cdots,2n-2}$)

A Final Note

- For $a = (a_0, a_1, \cdots a_n)$, $b = (b_0, b_1, \cdots b_m)$, the vector $c = a \circ b$ defined as follows is called the convolution of a, b

$$c_k = \sum_{i+j=k} a_i b_j \ , \ \forall k = 0, \cdots, m+n$$

- Besides polynomial products, many other applications
  - See Kleinberg/Tardos book, Chapter 5.6

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu