Fast Fourier Transform

Katerina Mamali

Algorithms and Complexity, book Dasgupta *et al.* 2008

ALMA, NKUA

October 20, 2022

Summary

Setting and Motivation

Three problems

- Number Multiplication
  $13 = 1101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
  $7 = 0111 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

- Polynomial Multiplication
  $A(x) = 5x^3 + 4x^2 + 3x + 2$
  $B(x) = x^3 + 2x^2 + 3x + 4$

- Convolution

$$\left.\begin{matrix}(a_0, a_1, \ldots, a_d)\\(b_0, b_1, \ldots, b_d)\end{matrix}\right\} \rightarrow \quad (c_0, c_1, \ldots, c_{2d}) : c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

$O(d^2)$ multiplications

Three problems

- Number Multiplication
  $13 = 1101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
  $7 = 0111 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

- Polynomial Multiplication
  $A(x) = 5x^3 + 4x^2 + 3x + 2$
  $B(x) = x^3 + 2x^2 + 3x + 4$

- Convolution

$$\left.\begin{matrix}(a_0, a_1, \ldots, a_d) \\ (b_0, b_1, \ldots, b_d)\end{matrix}\right\} \rightarrow \quad (c_0, c_1, \ldots, c_{2d}) : c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

$O(d^2)$ multiplications

Three problems

- Number Multiplication
  $13 = 1101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
  $7 = 0111 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

- Polynomial Multiplication
  $A(x) = 5x^3 + 4x^2 + 3x + 2$
  $B(x) = x^3 + 2x^2 + 3x + 4$

- Convolution

$$\left.\begin{matrix}(a_0, a_1, \ldots, a_d) \\ (b_0, b_1, \ldots, b_d)\end{matrix}\right\} \rightarrow \quad (c_0, c_1, \ldots, c_{2d}) : c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

$O(d^2)$ multiplications

Can we do better?

Why would we? $\rightarrow c_k$ not independent!

$$c_0 = a_0 b_0$$
$$c_1 = a_0 b_1 + a_1 b_0$$
$$c_2 = a_0 b_2 + a_1 b_1 + a_2 b_0$$
$$\ldots$$

But $a_0 b_1 + a_1 b_0 = (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0$.
c_1 needs by a single multiplication!

Can we do better?

Why would we? $\rightarrow c_k$ not independent!

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

But $a_0 b_1 + a_1 b_0 = (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0$.
c_1 needs by a single multiplication!

Our problem

Convolution!
But

$$(a_1, a_2, \ldots, a_d) \iff A(x) = \sum_{i=0}^{d} a_i x^i$$

Setting

Input:

- $A(x) = a_0 + a_1 x + \ldots + a_d x^d$
- $B(x) = b_0 + b_1 x + \ldots + b_d x^d$

Output:

- $C(x) = A(x)B(x) = c_0 + c_1 x + \ldots + c_{2d} x^{2d}$

Fast Fourier Transform solves this problem in $O(d \log d)$.

Fast Fourier Transform

Two simple facts

Fact 1: $A(x) = \sum_{i=0}^{d} a_i x^i \iff A(x_0), \ldots, A(x_d)$

Why?

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^d \\ 1 & x_1 & x_1^2 & \cdots & x_1^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^d \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{pmatrix} = \begin{pmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_d) \end{pmatrix}$$

We will determine $x_0, \ldots, x_d$ so as the above matrix (let's call it M) has full rank!

$C(x_i) = A(x_i)B(x_i) \rightarrow O(d)$ multiplications!

Fact 2: $A(x) = A_e(x^2) + xA_o(x^2)$

where $A_e(x)$ contains all even-terms and $A_o(x)$ contains all the odd-terms.

Two simple facts

Fact 1: $A(x) = \sum_{i=0}^{d} a_i x^i \iff A(x_0), \dots, A(x_d)$

Why?

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^d \\ 1 & x_1 & x_1^2 & \cdots & x_1^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^d \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{pmatrix} = \begin{pmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_d) \end{pmatrix}$$

We will determine $x_0, \dots, x_d$ so as the above matrix (let's call it M) has full rank!

$C(x_i) = A(x_i)B(x_i) \rightarrow O(d)$ multiplications!

Fact 2: $A(x) = A_e(x^2) + xA_o(x^2)$

where $A_e(x)$ contains all even-terms and $A_o(x)$ contains all the odd-terms.

Two simple facts

Fact 1: $A\left(x\right)=\sum_{i=0}^{d} a_i x^i \iff A\left(x_0\right),\ldots,A\left(x_d\right)$

Why?

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^d \\ 1 & x_1 & x_1^2 & \cdots & x_1^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^d \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{pmatrix} = \begin{pmatrix} A\left(x_0\right) \\ A\left(x_1\right) \\ \vdots \\ A\left(x_d\right) \end{pmatrix}$$

We will determine $x_0,\ldots,x_d$ so as the above matrix (let's call it M) has full rank!

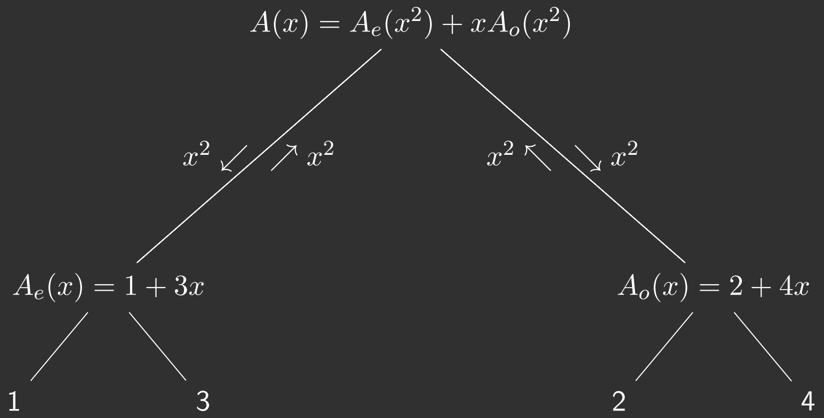
$$C(x_i)=A(x_i)B(x_i)\rightarrow O(d) \text{ multiplications!}$$

Fact 2: $A\left(x\right)=A_e\left(x^2\right)+xA_o\left(x^2\right)$

where $A_e(x)$ contains all even-terms and $A_o(x)$ contains all the odd-terms.

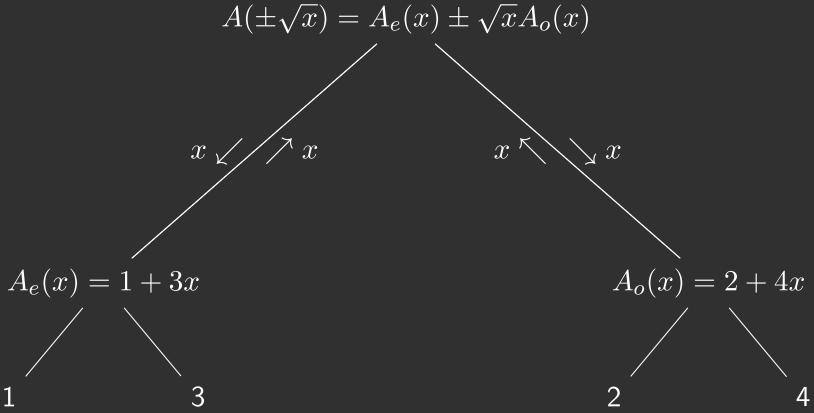
A Naive Approach

$A(x) = 1 + 2x + 3x^2 + 4x^3$, $\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}$.



Key Observation

$A(x) = 1 + 2x + 3x^2 + 4x^3$, $\sqrt{\mathbf{x_0}}, -\sqrt{\mathbf{x_0}}, \sqrt{\mathbf{x_1}}, -\sqrt{\mathbf{x_1}}$.



Fast Fourier Transform

$A(x) = 1 + 2x + 3x^2 + 4x^3 \,,\; \sqrt[4]{x_0}, -\sqrt[4]{x_0}, i\sqrt[4]{x_0}, -i\sqrt[4]{x_0}.$

$$A(\pm\sqrt[4]{x}) = A_e(\sqrt{x}) \pm \sqrt[4]{x}A_o(\sqrt{x})$$
$$A(\pm i\sqrt[4]{x}) = A_e(-\sqrt{x}) \pm i\sqrt[4]{x}A_o(-\sqrt{x})$$

$x \swarrow \;\; \nearrow \pm\sqrt{x}$ $\qquad$ $\pm\sqrt{x} \nwarrow \;\; \searrow x$

$$A_e(\pm\sqrt{x}) = 1 \pm 3\sqrt{x} \qquad\qquad A_o(\pm\sqrt{x}) = 2 \pm 4\sqrt{x}$$

1 $\quad$ 3 $\qquad\qquad$ 2 $\quad$ 4

Fast Fourier Transform

- We can calculate all $d+1$ values by traversing the tree just once $\Rightarrow O(d \log d)$.
- We can simply set $x = 1$. Then the $d+1$ points will be the $(d+1)$-th roots of unity *i.e.* $e^{2\pi ij/(d+1)}$.
- We have specified the matrix M.

$$M = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{2\pi i/(d+1)} & e^{4\pi i/(d+1)} & \ldots & e^{2\pi id/(d+1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{2\pi id/(d+1)} & e^{4\pi id/(d+1)} & \ldots & e^{2\pi id^2/(d+1)} \end{pmatrix}$$

- To multiply M with any vector $(\alpha_0, \alpha_1, \ldots, \alpha_d)^T$ the *FFT* algorithm can be used!

Fast Fourier Transform

- We can calculate all $d+1$ values by traversing the tree just once $\Rightarrow O(d \log d)$.
- We can simply set $x = 1$. Then the $d+1$ points will be the $(d+1)$-th roots of unity *i.e.* $e^{2\pi ij/(d+1)}$.
- We have specified the matrix M.

$$M = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{2\pi i/(d+1)} & e^{4\pi i/(d+1)} & \cdots & e^{2\pi id/(d+1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{2\pi id/(d+1)} & e^{4\pi id/(d+1)} & \cdots & e^{2\pi id^2/(d+1)} \end{pmatrix}$$

- To multiply M with any vector $(\alpha_0, \alpha_1, \ldots, \alpha_d)^T$ the *FFT* algorithm can be used!

Fast Fourier Transform

- We can calculate all $d+1$ values by traversing the tree just once $\Rightarrow O(d \log d)$.
- We can simply set $x = 1$. Then the $d+1$ points will be the $(d+1)$-th roots of unity *i.e.* $e^{2\pi i j/(d+1)}$.
- We have specified the matrix M.

$$M = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{2\pi i/(d+1)} & e^{4\pi i/(d+1)} & \cdots & e^{2\pi i d/(d+1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{2\pi i d/(d+1)} & e^{4\pi i d/(d+1)} & \cdots & e^{2\pi i d^2/(d+1)} \end{pmatrix}$$

- To multiply M with any vector $(\alpha_0, \alpha_1, \ldots, \alpha_d)^T$ the *FFT* algorithm can be used!

Fast Fourier Transform

- We can calculate all $d+1$ values by traversing the tree just once $\Rightarrow O(d\log d)$.
- We can simply set $x=1$. Then the $d+1$ points will be the $(d+1)$-th roots of unity *i.e.* $e^{2\pi ij/(d+1)}$.
- We have specified the matrix M.

$$M = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{2\pi i/(d+1)} & e^{4\pi i/(d+1)} & \cdots & e^{2\pi id/(d+1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{2\pi id/(d+1)} & e^{4\pi id/(d+1)} & \cdots & e^{2\pi id^2/(d+1)} \end{pmatrix}$$

- To multiply M with any vector $(\alpha_0, \alpha_1, \ldots, \alpha_d)^T$ the *FFT* algorithm can be used!

Fast Fourier Transform

Theorem

$$MM^* = (d+1)I$$

proof

Consider the k-th row of M and j-th column of M^*. Then

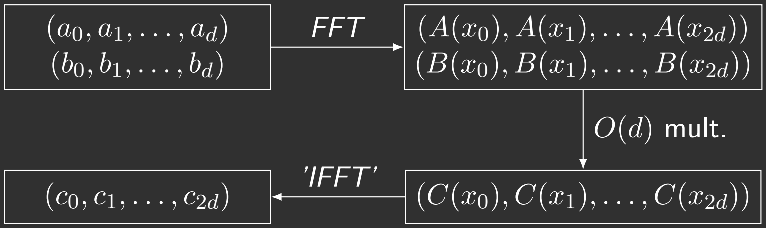
$$\begin{pmatrix} 1 & e^{2\pi ik/(d+1)} & \dots & e^{2\pi idk/(d+1)} \end{pmatrix} \begin{pmatrix} 1 \\ e^{-2\pi ij/(d+1)} \\ \vdots \\ e^{-2\pi idj/(d+1)} \end{pmatrix} = \sum_{i=0}^{d} e^{2\pi i(k-j)/(d+1)}.$$

If $k = j$ then the inner product equals $d+1$.

If $k \neq j$ then we have the sum of a geometric series with ratio $e^{2\pi(k-j)/(d+1)}$ and equals zero.

Problem Solved!

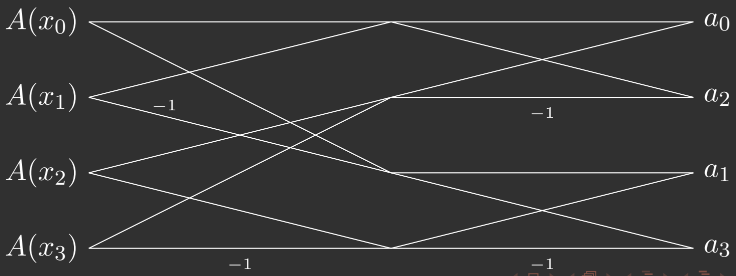
- $M^{-1} = \frac{1}{d+1} M^*$
  So to retrieve the coefficients of a polynomial A we just need to run *FFT* using $e^{-2\pi i/(d+1)}$ as the $(d+1)$-th roots of unity.



Can we do better?

Not clear yet!

- Morgenstern 1975, Winograd 1978
  Need $O(d \log d)$ additions and $O(d)$ multiplications when d not a power of 2.
- Papadimitriou 1979
  FFT optimal when $d = 2^N$ under certain assumptions on the FFT circuit.



References

1. Dasgupta, S., Papadimitriou, C. H. & Vazirani, U. V. *Algorithms.* ISBN: 978-0-07-352340-8 (McGraw-Hill, 2008).
2. Morgenstern, J. The linear complexity of computation. *Journal of the ACM (JACM)* **22,** 184–194 (1975).
3. Papadimitriou, C. H. Optimality of the fast Fourier transform. *Journal of the ACM (JACM)* **26,** 95–102 (1979).
4. Winograd, S. On computing the discrete Fourier transform. *Mathematics of computation* **32,** 175–199 (1978).

The End