Lecture 4: Discrete Fourier Transform

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

January 20, 2021

Overview

- Polynomial Evaluation & Interpolation
- Basics for DFT
- Fast Fourier Transform (FFT)
- Conclusion
- Acknowledgements

Representing Polynomials

- In previous classes saw two different ways of representing polynomials, given an upper bound d on degree

Representing Polynomials

- In previous classes saw two different ways of representing polynomials, given an upper bound d on degree
  1. As a list of coefficients $p_0 + p_1 x + \cdots + p_d x^d$

$$p(x) \leftrightarrow (p_0, p_1, \ldots, p_d)$$

  2. As evaluations at $d + 1$ distinct points

$$\det(V) = \prod_{i>j} (\alpha_i - \alpha_j) \neq 0$$

$$(\alpha_0, y_0), \ldots, (\alpha_d, y_d)$$

  where $y_i := p(\alpha_i)$

- Proof of equivalence:

$$V \quad \underbrace{\begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^d \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^d \\ 1 & \alpha_2 & & \ddots & \vdots \\ \vdots & \vdots & & & \\ 1 & \alpha_d & \cdots & & \alpha_d^d \end{pmatrix}}_{\text{invertible}} \overbrace{\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{pmatrix}}^{1} = \overbrace{\begin{pmatrix} p(\alpha_0) \\ p(\alpha_1) \\ \vdots \\ p(\alpha_d) \end{pmatrix}}^{2}$$

Multiplying Polynomials via Interpolation

- From previous lecture, saw how to multiply two polynomials via interpolation
- Given $p, q \in R[x]$ of degree $\leq d$

Multiplying Polynomials via Interpolation

- From previous lecture, saw how to multiply two polynomials via interpolation
- Given $p, q \in R[x]$ of degree $\leq d$
  1. Compute $2d + 1$ evaluations of p, q (why $2d + 1$?)

$p(x) \cdot q(x) \leftarrow$ degree $\leq 2d$

$\updownarrow$

$2d+1$ evaluations
to uniquely determine it

Multiplying Polynomials via Interpolation

- From previous lecture, saw how to multiply two polynomials via interpolation
- Given $p, q \in R[x]$ of degree $\leq d$
  1. Compute $2d+1$ evaluations of p, q (why $2d+1$?)
  2. Compute the products $\gamma_i := p(\alpha_i) \cdot q(\alpha_i)$

Multiplying Polynomials via Interpolation

- From previous lecture, saw how to multiply two polynomials via interpolation
- Given $p, q \in R[x]$ of degree $\leq d$
  1. Compute $2d + 1$ evaluations of p, q (why $2d + 1$?)
  2. Compute the products $\gamma_i := p(\alpha_i) \cdot q(\alpha_i)$ $\leftarrow$
  3. Use Lagrange's interpolation polynomials

$$L_i(x) = \prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

$L_i(\alpha_i) = 1$

$L_i(\alpha_j) = 0$ if $j \neq i$

Multiplying Polynomials via Interpolation

- From previous lecture, saw how to multiply two polynomials via interpolation
- Given $p, q \in R[x]$ of degree $\leq d$
  1. Compute $2d + 1$ evaluations of p, q (why $2d + 1$?)
  2. Compute the products $\gamma_i := p(\alpha_i) \cdot q(\alpha_i)$
  3. Use Lagrange's interpolation polynomials

$$L_i(x) = \prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

  4. From previous slide's equivalence, we know

$$p(x) \cdot q(x) = \sum_{i=0}^{2d} \gamma_i L_i(x)$$

$$p(\alpha_i) \cdot q(\alpha_i) = \gamma_i \cdot \underbrace{L_i(\alpha_i)}_{1}$$

they evaluate to same values at $2d+1$ pts.

Multiplying Polynomials via Interpolation

- From previous lecture, saw how to multiply two polynomials via interpolation
- Given $p, q \in R[x]$ of degree $\leq d$
  1. Compute $2d + 1$ evaluations of p, q (why $2d + 1$?)
  2. Compute the products $\gamma_i := p(\alpha_i) \cdot q(\alpha_i)$
  3. Use Lagrange's interpolation polynomials

$$L_i(x) = \prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

  4. From previous slide's equivalence, we know

$$p(x) \cdot q(x) = \sum_{i=0}^{2d} \gamma_i L_i(x)$$

- We saw the analysis of this in Ostrowski's non-scalar model

Multiplying Polynomials via Interpolation

- From previous lecture, saw how to multiply two polynomials via interpolation
- Given $p, q \in R[x]$ of degree $\leq d$
  1. Compute $2d + 1$ evaluations of p, q (why $2d + 1$?)
  2. Compute the products $\gamma_i := p(\alpha_i) \cdot q(\alpha_i)$
  3. Use Lagrange's interpolation polynomials

$$L_i(x) = \prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

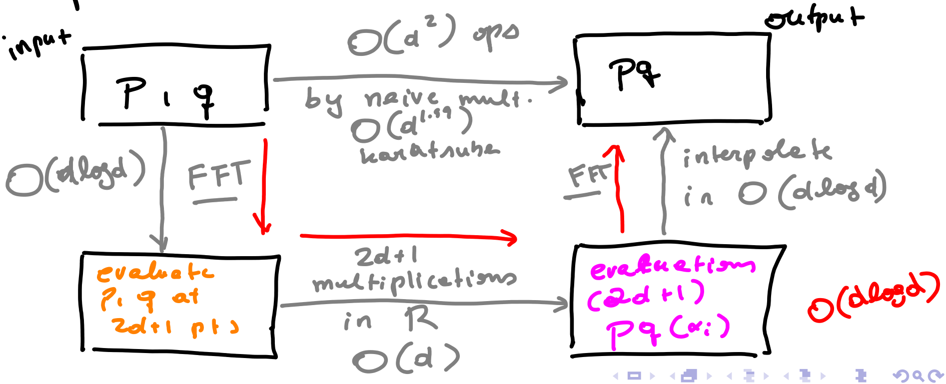
  4. From previous slide's equivalence, we know

$$p(x) \cdot q(x) = \sum_{i=0}^{2d} \gamma_i L_i(x)$$

- We saw the analysis of this in Ostrowski's non-scalar model
- To get a fast algorithm in the scalar model (where we count all ring operations) we will use *Fast Fourier Transform*

Fast Fourier Transform Idea

If we can evaluate poly of deg d and interpolate poly of deg d fast, then can multiply fast



Roots of Unity

- $\mathbb{F}$ a field, $n \in \mathbb{N}$

Roots of Unity

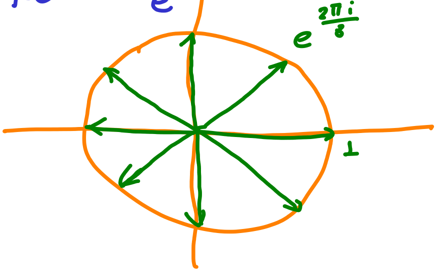
- $\mathbb{F}$ a field, $n \in \mathbb{N}$
- We say $\omega \in \mathbb{F}$ is a primitive n^{th} *root of unity* (n-PRU) if
  1. $\omega^n = 1$
  2. $\omega^k \neq 1$ for $1 \leq k < n$

$x^n - 1$ ω root

Roots of Unity

- $\mathbb{F}$ a field, $n \in \mathbb{N}$
- We say $\omega \in \mathbb{F}$ is a primitive n^{th} *root of unity* (n-PRU) if
  1. $\omega^n = 1$
  2. $\omega^k \neq 1$ for $1 \leq k < n$
- Examples:
  1. $\mathbb{F} = \mathbb{C}$ and $\omega = e^{2\pi i/n}$ $= \cos\left(\frac{2\pi}{n}\right) + i\sin\left(\frac{2\pi}{n}\right)$

4-PRU $\leftarrow e^{\frac{2\pi i}{4}}$



8-PRU

Roots of Unity

- $\mathbb{F}$ a field, $n \in \mathbb{N}$
- We say $\omega \in \mathbb{F}$ is a primitive n^{th} *root of unity* (n-PRU) if
  1. $\omega^n = 1$
  2. $\omega^k \neq 1$ for $1 \leq k < n$
- Examples:
  1. $\mathbb{F} = \mathbb{C}$ and $\omega = e^{2\pi i/n}$
  2. If p is a prime and $\mathbb{F} = \mathbb{Z}_p$, then $\mathbb{F}$ has a $(p-1)$-PRU. How many such roots does $\mathbb{Z}_p$ have?

in $\mathbb{Z}_p$ we know that $a^{p-1} = 1 \quad \forall a \in \mathbb{Z}_p^*$

$\underline{p(x) = x}^{p-1} - 1$ has exactly $p-1$ roots over $\mathbb{Z}_p$

$$p(x) = \underbrace{\left(x^{\frac{p-1}{2}} - 1\right)}_{\leq \frac{p-1}{2} \text{ roots } = \frac{p-1}{2} \text{ roots over } \mathbb{Z}_p} \underbrace{\left(x^{\frac{p-1}{2}} + 1\right)}_{\text{has } \frac{p-1}{2} \text{ roots over } \mathbb{Z}_p \;\; \alpha \text{ root}}$$

$\alpha^{\frac{p-1}{2}} \equiv -1$

$\alpha^{p-1} \equiv 1$

$\therefore \alpha$ is $(p-1)$-PRU

More on Roots of Unity

Vandermonde Matrix

- Given elements $u_0, \ldots, u_d$

$$V(u_0, \ldots, u_d) = \begin{pmatrix} u_0^0 & u_0^1 & \cdots & u_0^d \\ u_1^0 & u_1^1 & \cdots & u_1^d \\ \vdots & \vdots & \vdots & \vdots \\ u_d^0 & u_d^1 & \cdots & u_d^d \end{pmatrix}$$

$(d+1) \times (d+1)$ matrix

Vandermonde Matrix

- Given elements $u_0, \ldots, u_d$

$$V(u_0, \ldots, u_d) = \begin{pmatrix} u_0^0 & u_0^1 & \cdots & u_0^d \\ u_1^0 & u_1^1 & \cdots & u_1^d \\ \vdots & \vdots & \vdots & \vdots \\ u_d^0 & u_d^1 & \cdots & u_d^d \end{pmatrix}$$

- From previous lecture, maps coefficients to evaluations:

$$\begin{pmatrix} u_0^0 & u_0^1 & \cdots & u_0^d \\ u_1^0 & u_1^1 & \cdots & u_1^d \\ \vdots & \vdots & \ddots & \vdots \\ u_d^0 & u_d^1 & \cdots & u_d^d \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} p(u_0) \\ p(u_1) \\ \vdots \\ p(u_d) \end{pmatrix}$$

Vandermonde Matrix

- Given elements $u_0, \ldots, u_d$

$$V(u_0, \ldots, u_d) = \begin{pmatrix} u_0^0 & u_0^1 & \cdots & u_0^d \\ u_1^0 & u_1^1 & \cdots & u_1^d \\ \vdots & \vdots & \vdots & \vdots \\ u_d^0 & u_d^1 & \cdots & u_d^d \end{pmatrix}$$

- From previous lecture, maps coefficients to evaluations:

$$\begin{pmatrix} u_0^0 & u_0^1 & \cdots & u_0^d \\ u_1^0 & u_1^1 & \cdots & u_1^d \\ \vdots & \vdots & \ddots & \vdots \\ u_d^0 & u_d^1 & \cdots & u_d^d \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} p(u_0) \\ p(u_1) \\ \vdots \\ p(u_d) \end{pmatrix}$$

- For interpolation, we can choose our evaluation points. Choosing roots of unity give rise to the FFT!

- Polynomial Evaluation & Interpolation
- Basics for DFT
- Fast Fourier Transform (FFT)
- Conclusion
- Acknowledgements

Vandermonde with Roots of Unity

- Given d, Take ω to be a $(d+1)$-PRU, and let

$$V(\omega) = V(\omega^0, \omega, \ldots, \omega^d) = \begin{array}{cc} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \cdots & \omega^d \\ 1 & \omega^2 & \cdots & \omega^{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^d & \cdots & \omega^{d^2} \end{pmatrix} & \begin{matrix} \omega^0 \\ \omega \\ \omega^2 \\ \\ \omega^d \end{matrix} \end{array}$$

$$\underbrace{1, \omega, \ldots \omega^d}_{\substack{\text{all } d+1 \\ \text{roots of unity}}}$$

Vandermonde with Roots of Unity

$x^{d+1}-1 =$
$= (x-1)\underbrace{(x^d+x^{d-1}+\dots+x+1)}_{\omega^{a-b} \text{ root}}$

- Given d, Take ω to be a $(d+1)$-PRU, and let

ω^{-1} also $(d+1)$-PRU

$$V(\omega) = V(\omega^0, \omega, \dots, \omega^d) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \cdots & \omega^d \\ 1 & \omega^2 & \cdots & \omega^{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^d & \cdots & \omega^{d^2} \end{pmatrix}$$

$V(\omega^{-1})$

- **Lemma:** $\underbrace{V(\omega) \cdot V(\omega^{-1})}_{W} = (d+1) \cdot I$

$$W_{\ell\ell} = \sum_{i=0}^{d} \underbrace{\omega^{i\ell}}_{\substack{i\ell \text{ entry} \\ V(\omega)}} \cdot \underbrace{\omega^{-i\ell}}_{\substack{\ell i \text{ entry} \\ \text{of } V(\omega^{-1})}} = \sum_{i=0}^{d} 1 = d+1$$

$(\omega^{a-b})^{d+1} = 1$ $\quad a \neq b$ $\quad \omega^{a-b} \neq 1$

$$W_{ab} = \sum_{i=0}^{d} \omega^{ia} \cdot \omega^{-ib} = \sum_{i=0}^{d} \omega^{(a-b)\cdot i} = 0 .$$

Proof of lemma

upshot of Lemma

if use PRU then

$$\underbrace{V(\omega^{-1})}_{\text{evaluation}} = (d+1) \cdot \underbrace{V(\omega)^{-1}}_{\text{interpolation}}$$

$\Rightarrow$ interpolation (inverse of Vandermonde)

is equivalent to

evaluation (matrix-vector multiplication by Vandermonde matrix)

Discrete Fourier Transform (DFT)

- Given ω a $(d+1)$-PRU, the DFT is given by $DFT(\omega) : \underline{\mathbb{F}^{d+1}} \to \underline{\mathbb{F}^{d+1}}$

$$DFT(\omega) \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} p(1) \\ p(\omega) \\ \vdots \\ p(\omega^d) \end{pmatrix}$$

$$V(\omega) \begin{pmatrix} p_0 \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} p(1) \\ \vdots \\ p(\omega^d) \end{pmatrix}$$

Discrete Fourier Transform (DFT)

- Given ω a $(d+1)$-PRU, the DFT is given by $DFT(\omega) : \mathbb{F}^{d+1} \to \mathbb{F}^{d+1}$

$$DFT(\omega) \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} p(1) \\ p(\omega) \\ \vdots \\ p(\omega^d) \end{pmatrix}$$

- Fast Fourier Transform is a way of computing the DFT in $O(n \log n)$ operations!

Discrete Fourier Transform (DFT)

- Given ω a $(d+1)$-PRU, the DFT is given by $DFT(\omega) : \mathbb{F}^{d+1} \to \mathbb{F}^{d+1}$

$$\underline{DFT(\omega)} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} p(1) \\ p(\omega) \\ \vdots \\ p(\omega^d) \end{pmatrix}$$

- Fast Fourier Transform is a way of computing the DFT in $O(n \log n)$ operations!
- $DFT(\omega)$ computes polynomial evaluation
- $DFT(\omega^{-1})$ computes polynomial interpolation

Fast Fourier Transform

- Assume that $d = 2^k$
- Idea:
  - $p(x) = p_{even}(x^2) + \underline{x} \cdot \underline{p_{odd}(x^2)}$

$$P(x) = \underline{p_0} + \underline{p_1 x} + \underline{p_2 x^2} + \underline{p_3 x^3} + \underline{p_4 x^4}$$

$$\text{degree } \le \frac{d}{2} \begin{cases} P_{even}(y) = p_0 + p_2 y + p_4 y^2 \\ P_{odd}(y) = p_1 + p_3 y \end{cases}$$

Fast Fourier Transform

- Assume that $d = 2^k$
- Idea:
  - $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$
  - Reduced problem of

$$\text{evaluating } \underbrace{p(x)}_{\text{deg } d} \text{ at } \underbrace{1, \omega, \ldots, \omega^d}_{d+1 \text{ values}}$$

into

$$\text{evaluating } p_{even}(x), p_{odd}(x) \text{ (of degree } d/2) \text{ at } \underbrace{1, \omega^2, \omega^4, \omega^6, \ldots, \omega^{2d}}_{d+1 \text{ values}}$$

Fast Fourier Transform

$\omega^d = 1$ $\quad \omega$ d-PRU $\quad \omega^2$ $\frac{d}{2}$-PRU

- Assume that $d = 2^k$
- Idea:
  - $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$
  - Reduced problem of

$$\text{evaluating } p(x) \text{ at } 1, \omega, \ldots, \omega^{d-1}$$

ω^2

into

$$\text{evaluating } p_{even}(x), p_{odd}(x) \text{ (of degree } d/2\text{) at } \underbrace{\overbrace{1, \omega^2, \omega^4, \omega^6, \ldots, \omega^{2(d-1)}}^{\text{even powers}}}_{\text{not } d \text{ points!}}$$

  - are we evaluating it at d points?

they are actually $\frac{d}{2}$ points!

$$\left(\omega^{d/2}\right)^2 = \omega^d = 1$$

$$\left.\begin{array}{l} \to \left(\omega^{d/2+1}\right)^2 = \omega^{d+2} = \omega^2 \\ \vdots \\ \to \left(\omega^{d-1}\right)^2 = \omega^{2d-2} = \left(\omega^2\right)^{\frac{d}{2}-1} \end{array}\right\}$$

ω^ℓ $\ell > \frac{d}{2}$ then evaluated it already $(\omega^2)^k$

Fast Fourier Transform

- Assume that $d = 2^k$
- Idea:
  - $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$
  - Reduced problem of

$$\text{evaluating } p(x) \text{ at } 1, \omega, \ldots, \omega^d$$

into

$$\text{evaluating } p_{even}(x), p_{odd}(x) \text{ (of degree } d/2\text{) at } 1, \omega^2, \omega^4, \omega^6, \ldots, \omega^{2d}$$

  - are we evaluating it at d points? no $d/2$ pts
  - need to combine them back

d evaluations

$$\boxed{p(\omega^t)} = p_{even}(\omega^{2t}) + \underline{\omega^t} \cdot p_{odd}(\omega^{2t})$$

Two more operations to get $p(\omega^t)$ plus d operations to get $1, \omega, \ldots, \omega^d$

d evaluations of p

$\frac{d}{2}$ evaluations of p_{even}

$\frac{d}{2}$ evaluations of p_{odd}

Fast Fourier Transform

- Assume that $d = 2^k$
- Idea:
  - $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$
  - Reduced problem of

$$\text{evaluating } p(x) \text{ at } 1, \omega, \ldots, \omega^d$$

$$\text{into}$$

$$\text{evaluating } p_{even}(x), p_{odd}(x) \text{ (of degree } d/2) \text{ at } 1, \omega^2, \omega^4, \omega^6, \ldots, \omega^{2d}$$

  - are we evaluating it at d points?
  - need to combine them back

$$p(\omega^t) = p_{even}(\omega^{2t}) + \omega^t \cdot p_{odd}(\omega^{2t})$$

Two more operations to get $p(\omega^t)$ plus d operations to get $1, \omega, \ldots, \omega^d$

  - Recurrence:

$$\underbrace{T(d)}_{\substack{\text{\# operations} \\ \text{eval } p(\omega^i)}} = 2\underbrace{T(d/2)}_{\substack{P_{even} \\ P_{odd}}} + \underbrace{3d}_{\text{put everything together}}$$

Fast Fourier Transform

- Assume that $d = 2^k$
- Idea:
  - $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$
  - Reduced problem of

$$\text{evaluating } p(x) \text{ at } 1, \omega, \dots, \omega^d$$

$$\text{into}$$

$$\text{evaluating } p_{even}(x), p_{odd}(x) \text{ (of degree } d/2\text{) at } 1, \omega^2, \omega^4, \omega^6, \dots, \omega^{2d}$$

  - are we evaluating it at d points?
  - need to combine them back

$$p(\omega^t) = p_{even}(\omega^{2t}) + \omega^t \cdot p_{odd}(\omega^{2t})$$

Two more operations to get $p(\omega^t)$ plus d operations to get $1, \omega, \dots, \omega^d$

  - Recurrence:

$$T(d) = 2T(d/2) + 3d$$

  - Master's theorem gives us $O(n \log n)$

Fast Polynomial Multiplication

- Now that we know how to evaluate and interpolate polynomials fast, can use it to multiply faster!
- Given p, q of degree $< 2^{k-1}$, take ω to be a 2^k-PRU.

Fast Polynomial Multiplication

- Now that we know how to evaluate and interpolate polynomials fast, can use it to multiply faster!
- Given p, q of degree $< 2^{k-1}$, take ω to be a 2^k-PRU.
- $c(x) = p(x)q(x)$ has degree $\underline{< 2^k}$ so it is uniquely determined by the evaluations $\underbrace{c(1), \ldots, c(\omega^{2^k-1})}_{\text{all roots of unity evaluations}}$

Fast Polynomial Multiplication

$d = 2^k$

- Now that we know how to evaluate and interpolate polynomials fast, can use it to multiply faster!
- Given p, q of degree $< 2^{k-1}$, take ω to be a 2^k-PRU.
- $c(x) = p(x)q(x)$ has degree $< 2^k$ so it is uniquely determined by the evaluations $c(1), \ldots, c(\omega^{2^k-1})$
- use FFT to evaluate $p(1), p(\omega), \ldots, p(\omega^{2^k-1})$ and $q(1), q(\omega), \ldots, q(\omega^{2^k-1})$

$O(d \log d)$ operations

Fast Polynomial Multiplication

- Now that we know how to evaluate and interpolate polynomials fast, can use it to multiply faster!
- Given p, q of degree $< 2^{k-1}$, take ω to be a 2^k-PRU.
- $c(x) = p(x)q(x)$ has degree $< 2^k$ so it is uniquely determined by the evaluations $c(1), \ldots, c(\omega^{2^k-1})$
- use FFT to evaluate $p(1), p(\omega), \ldots, p(\omega^{2^k-1})$ and $q(1), q(\omega), \ldots, q(\omega^{2^k-1})$
- Multiply $c(\omega^t) = p(\omega^t) \cdot q(\omega^t)$

d operations

Fast Polynomial Multiplication

- Now that we know how to evaluate and interpolate polynomials fast, can use it to multiply faster!
- Given p, q of degree $< 2^{k-1}$, take ω to be a 2^k-PRU.
- $c(x) = p(x)q(x)$ has degree $< 2^k$ so it is uniquely determined by the evaluations $c(1), \ldots, c(\omega^{2^k-1})$
- use FFT to evaluate $p(1), p(\omega), \ldots, p(\omega^{2^k-1})$ and $q(1), q(\omega), \ldots, q(\omega^{2^k-1})$
- Multiply $c(\omega^t) = p(\omega^t) \cdot q(\omega^t)$
- use FFT to interpolate $c(1), \ldots, c(\omega^{2^k-1})$

$O(d \log d)$ operations

Fast Polynomial Multiplication

- Now that we know how to evaluate and interpolate polynomials fast, can use it to multiply faster!
- Given p, q of degree $< 2^{k-1}$, take ω to be a 2^k-PRU.
- $c(x) = p(x)q(x)$ has degree $< 2^k$ so it is uniquely determined by the evaluations $c(1), \ldots, c(\omega^{2^k-1})$
- use FFT to evaluate $p(1), p(\omega), \ldots, p(\omega^{2^k-1})$ and $q(1), q(\omega), \ldots, q(\omega^{2^k-1})$ $O(d \log d)$
- Multiply $c(\omega^t) = p(\omega^t) \cdot q(\omega^t)$ $O(d)$
- use FFT to interpolate $c(1), \ldots, c(\omega^{2^k-1})$ $O(d \log d)$
- running time: $\cancel{O(n \log n)}$ $O(d \log d)$

Fast Polynomial Multiplication

- Now that we know how to evaluate and interpolate polynomials fast, can use it to multiply faster!
- Given p, q of degree $< 2^{k-1}$, take ω to be a 2^k-PRU.
- $c(x) = p(x)q(x)$ has degree $< 2^k$ so it is uniquely determined by the evaluations $c(1), \ldots, c(\omega^{2^k-1})$
- use FFT to evaluate $p(1), p(\omega), \ldots, p(\omega^{2^k-1})$ and $q(1), q(\omega), \ldots, q(\omega^{2^k-1})$
- Multiply $c(\omega^t) = p(\omega^t) \cdot q(\omega^t)$
- use FFT to interpolate $c(1), \ldots, c(\omega^{2^k-1})$
- running time: $O(n \log n)$
- **Remark:** FFT can be (and is) used in fast integer multiplication - starting from the works of Strassen and Schönhage in 1971 up to the recent work of Harvey and van der Hoeven in 2019.

Read more on Anne's notes

Conclusion

In today's lecture, we learned

- Discrete Fourier Transform for evaluation and interpolation
- Properties of roots of unity
- Fast Fourier Transform
- Multiplying polynomials in $O(n \log n)$ operations

Acknowledgement

- Based largely on Arne's notes

  `https://cs.uwaterloo.ca/~r5olivei/courses/2021-winter-cs487/lec4-ref.pdf`

$$T(n) = \alpha \, T\left(\frac{n}{\beta}\right) + f(n)$$

Master's theorem

α, β f gives you
upper bound on running time
of $T(n)$

$f(n) = 3n \quad \alpha = 2 \quad \beta = 2$

$$T(n) = 2T\left(\frac{n}{2}\right) + 3n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + 3\frac{n}{2}\right) + 3n$$

$$= 4 \cdot T\left(\frac{n}{4}\right) + 2 \cdot (3n)$$

$$\cdots = 2^k \cdot T\left(\frac{n}{2^k}\right) + K \cdot (3n) \quad \text{after } k \text{ steps}$$

$k = \log n$

$$= n \cdot \overbrace{T(1)}^{\text{const.}} + 3n \cdot \log n$$

$$= O(n \log n)$$