Chapter 1
Divide and Conquer

Polynomial Multiplication II

Algorithm Theory
WS 2015/16

Fabian Kuhn

Operations: Multiplication

- Given: Polynomials $p, q \in \mathbb{R}[x]$ of degree n

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
$$q(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_1 x + b_0$$

- Product $p(x) \cdot q(x)$:

$$p(x) \cdot q(x) = (a_n x^n + \cdots + a_0) \cdot (b_n x^n + \cdots + b_0)$$

$$= c_{2n} x^{2n} + c_{2n-1} x^{2n-1} + \cdots + c_1 x + c_0$$

- Obtaining c_i: what products of monomials have degree i?

$$\text{For } 0 \leq i \leq 2n\text{: } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

where $a_i = b_i = 0$ for $i > n$.

Polynomial Multiplication using DFT

Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):

p, q of degree $n-1$, n coefficients

↓ **Evaluation** at points $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$

$2 \times 2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right)\right)$ and $\left(\omega_{2n}^k, q\left(\omega_{2n}^k\right)\right)$

↓ **Point-wise multiplication**

$2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right)q\left(\omega_{2n}^k\right)\right)$

↓ **Interpolation**

$p(x)q(x)$ of degree $2n-2$, $2n-1$ coefficients

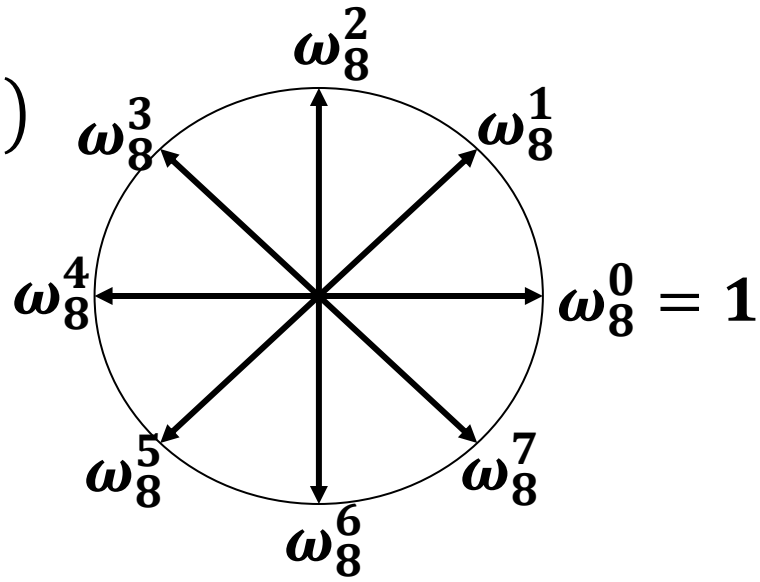
Point-Value Representation of p, q

- Select points $x_0, x_1, \dots, x_{N-1}$ to evaluate p and q in a clever way

Consider the N powers of the principle Nth root of unity:

Principle root of unity: $\omega_N = e^{2\pi i/N}$

$$\left(i = \sqrt{-1}, \qquad e^{2\pi i} = 1\right)$$



Powers of ω_N (roots of unity):

$$1 = \omega_N^0, \omega_N^1, \dots, \omega_N^{N-1}$$

Properties: $\omega_{dn}^{dk} = \omega_n^k, \quad \omega_n^{k \pm n} = \omega_n^k$

Discrete Fourier Transform

- The values $p\left(\omega_N^i\right)$ for $i = 0, \dots, N-1$ uniquely define a polynomial p of degree $< N$.

Discrete Fourier Transform (DFT):

- Assume $a = (a_0, \dots, a_{N-1})$ is the coefficient vector of poly. p

$$(p(x) = a_{N-1}x^{N-1} + \cdots + a_1 x + a_0)$$

$$\mathrm{DFT}_N(a) := \left(p\left(\omega_N^0\right), p(\omega_N^1), \dots, p(\omega_N^{N-1})\right)$$

Divide-and-Conquer Approach

- Divide $p(x)$ of degree $N-1$ (N is even) into 2 polynomials of degree ${}^{N}/_{2}-1$ differently than in Karatsuba's algorithm

- $p_0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{N-2} x^{{}^{N}/_{2}-1}$ (even coeff.)
  $p_1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{N-1} x^{{}^{N}/_{2}-1}$ (odd coeff.)

Recursively Computing the DFT

Polynomial p of degree $N-1$:

$$p(\omega_N^k) = \begin{cases} p_0(\omega_{N/2}^k) + \omega_N^k \cdot p_1(\omega_{N/2}^k) & \text{if } k < N/2 \\ p_0\left(\omega_{N/2}^{k-N/2}\right) + \omega_N^k \cdot p_1\left(\omega_{N/2}^{k-N/2}\right) & \text{if } k \geq N/2 \end{cases}$$

$$= \begin{cases} p_0(\omega_{N/2}^k) + \omega_N^k \cdot p_1(\omega_{N/2}^k) & \text{if } k < N/2 \\ p_0\left(\omega_{N/2}^{k-N/2}\right) - \omega_N^{k-N/2} \cdot p_1\left(\omega_{N/2}^{k-N/2}\right) & \text{if } k \geq N/2 \end{cases}$$

Need to compute $p_0(\omega_{N/2}^k)$ and $\omega_N^k \cdot p_1(\omega_{N/2}^k)$ for $0 \leq k < N/2$.

Example $n = 8$

$$p(\omega_8^0) = p_0(\omega_4^0) + \omega_8^0 \cdot p_1(\omega_4^0)$$
$$p(\omega_8^1) = p_0(\omega_4^1) + \omega_8^1 \cdot p_1(\omega_4^1)$$
$$p(\omega_8^2) = p_0(\omega_4^2) + \omega_8^2 \cdot p_1(\omega_4^2)$$
$$p(\omega_8^3) = p_0(\omega_4^3) + \omega_8^3 \cdot p_1(\omega_4^3)$$
$$p(\omega_8^3) = p_0(\omega_4^0) - \omega_8^0 \cdot p_1(\omega_4^0)$$
$$p(\omega_8^3) = p_0(\omega_4^1) - \omega_8^1 \cdot p_1(\omega_4^1)$$
$$p(\omega_8^3) = p_0(\omega_4^2) - \omega_8^2 \cdot p_1(\omega_4^2)$$
$$p(\omega_8^3) = p_0(\omega_4^3) - \omega_8^3 \cdot p_1(\omega_4^3)$$

Example $n = 4$

$$p(\omega_4^0) = p_0(\omega_2^0) + \omega_4^0 \cdot p_1(\omega_2^0)$$
$$p(\omega_4^1) = p_0(\omega_2^1) + \omega_4^1 \cdot p_1(\omega_2^1)$$
$$p(\omega_4^2) = p_0(\omega_2^0) - \omega_4^0 \cdot p_1(\omega_2^0)$$
$$p(\omega_4^3) = p_0(\omega_2^1) - \omega_4^1 \cdot p_1(\omega_2^1)$$

Example

- $p(x) = 3x^3 - 15x^2 + 18x + 0$, $a = [0{,}18, -15{,}3]$

Faster Polynomial Multiplication?

Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):

p, q of degree $n-1$, n coefficients

↓ **Evaluation** at $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$ using **FFT**

$2 \times 2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right)\right)$ and $\left(\omega_{2n}^k, q\left(\omega_{2n}^k\right)\right)$

↓ **Point-wise multiplication**

$2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right) q\left(\omega_{2n}^k\right)\right)$

↓ **Interpolation**

$p(x)q(x)$ of degree $2n-2$, $2n-1$ coefficients

Interpolation

Convert point-value representation into coefficient representation

Input: $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ with $x_i \neq x_j$ for $i \neq j$

Output:

Degree-$(n-1)$ polynomial with coefficients $a_0, \dots, a_{n-1}$ such that

$$\begin{aligned}
p(x_0) &= a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_{n-1} x_0^{n-1} = y_0 \\
p(x_1) &= a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_{n-1} x_1^{n-1} = y_1 \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
p(x_{n-1}) &= a_0 + a_1 x_{n-1} + a_2 x_{n-1}^2 + \cdots + a_{n-1} x_{n-1}^{n-1} = y_{n-1}
\end{aligned}$$

→ linear system of equations for $a_0, \dots, a_{n-1}$

Interpolation

Matrix Notation:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

- System of equations solvable iff $x_i \neq x_j$ for all $i \neq j$

Special Case $x_i = \omega_n^i$:

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Interpolation

- Linear system:

$$W \cdot \boldsymbol{a} = \boldsymbol{y} \quad \Longrightarrow \quad \boldsymbol{a} = W^{-1} \cdot \boldsymbol{y}$$

$$W_{i,j} = \omega_n^{ij}, \qquad \boldsymbol{a} = \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}, \qquad \boldsymbol{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Claim:

$$\boldsymbol{W_{ij}^{-1} = \frac{\omega_n^{-ij}}{n}}$$

Proof: Need to show that $W^{-1}W = I_n$

DFT Matrix Inverse

$$W^{-1}W = \begin{pmatrix} \frac{1}{n} & \frac{\omega_n^{-i}}{n} & \cdots & \frac{\omega_n^{-(n-1)i}}{n} \\ & & \cdots & \\ & & \vdots & \\ & & \cdots & \end{pmatrix} \cdot \begin{pmatrix} \cdots & 1 & \cdots \\ \cdots & \omega_n^{j} & \cdots \\ \cdots & \omega_n^{2j} & \cdots \\ & \vdots & \\ \cdots & \omega_n^{(n-1)j} & \cdots \end{pmatrix}$$

DFT Matrix Inverse

$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n}$$

Need to show $(W^{-1}W)_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

Case $i = j$:

DFT Matrix Inverse

$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n}$$

Case $i \neq j$:

Inverse DFT

- $W^{-1} = \begin{pmatrix} \frac{1}{n} & \frac{\omega_n^{-k}}{n} & \cdots & \frac{\omega_n^{-(n-1)k}}{n} \\ & & \vdots & \\ & & \cdots & \end{pmatrix}$
- We get $\boldsymbol{a} = W^{-1} \cdot \boldsymbol{y}$ and therefore

$$a_k = \begin{pmatrix} \frac{1}{n} & \frac{\omega_n^{-k}}{n} & \cdots & \frac{\omega_n^{-(n-1)k}}{n} \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$= \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-kj} \cdot y_j$$

DFT and Inverse DFT

Inverse DFT:

$$a_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-kj} \cdot y_j$$

- Define polynomial $q(x) = y_0 + y_1 x + \cdots + y_{n-1} x^{n-1}$:

$$a_k = \frac{1}{n} \cdot q(\omega_n^{-k})$$

DFT:

- Polynomial $p(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$:

$$y_k = p(\omega_n^k)$$

DFT and Inverse DFT

$$q(x) = y_0 + y_1 x + \cdots + y_{n-1} x^{n-1}, \qquad a_k = \frac{1}{n} \cdot q\left(\omega_n^{-k}\right):$$

- Therefore:

$$(a_0, a_1, \dots, a_{n-1})$$

$$= \frac{1}{n} \cdot \left(q(\omega_n^{-0}), q(\omega_n^{-1}), q(\omega_n^{-2}), \dots, q\left(\omega_n^{-(n-1)}\right)\right)$$

$$= \frac{1}{n} \cdot \left(q(\omega_n^{0}), q(\omega_n^{n-1}), q(\omega_n^{n-2}), \dots, q(\omega_n^{1})\right)$$

- Recall:

$$\mathrm{DFT}_n(\boldsymbol{y}) = \left(q(\omega_n^{0}), q(\omega_n^{1}), q(\omega_n^{2}), \dots, q(\omega_n^{n-1})\right)$$

$$= n \cdot (a_0, a_{n-1}, a_{n-2}, \dots, a_2, a_1)$$

DFT and Inverse DFT

- We have $\mathrm{DFT}_n(\boldsymbol{y}) = n \cdot (a_0, a_{n-1}, a_{n-2}, \dots, a_2, a_1)$:

$$a_i = \begin{cases} \frac{1}{n} \cdot (\mathrm{DFT}_n(\boldsymbol{y}))_0 & \text{if } i = 0 \\ \frac{1}{n} \cdot (\mathrm{DFT}_n(\boldsymbol{y}))_{n-i} & \text{if } i \neq 0 \end{cases}$$

- DFT and inverse DFT can both be computed using FFT algorithm in $O(n \log n)$ time.

- 2 polynomials of degr. $< n$ can be multiplied in time $O(n \log n)$.

Faster Polynomial Multiplication?

Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):

p, q of degree $n-1$, n coefficients

↓ **Evaluation** at $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$ using **FFT**

$2 \times 2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right)\right)$ and $\left(\omega_{2n}^k, q\left(\omega_{2n}^k\right)\right)$

↓ **Point-wise multiplication**

$2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right) q\left(\omega_{2n}^k\right)\right)$

↓ **Interpolation** using **FFT**

$p(x)q(x)$ of degree $2n-2$, $2n-1$ coefficients

Convolution

- More generally, the polynomial multiplication algorithm computes the convolution of two vectors:

$$\boldsymbol{a} = (a_0, a_1, \dots, a_{m-1})$$
$$\boldsymbol{b} = (b_0, b_1, \dots, b_{n-1})$$

$$\boldsymbol{a} * \boldsymbol{b} = (c_0, c_1, \dots, c_{m+n-2}),$$
$$\text{where } c_k = \sum_{\substack{(i,j): i+j=k \\ i<m, j<n}} a_i b_j$$

- c_k is exactly the coefficient of x^k in the product polynomial of the polynomials defined by the coefficient vectors $\boldsymbol{a}$ and $\boldsymbol{b}$

More Applications of Convolutions

Signal Processing Example:

- Assume $\boldsymbol{a} = (a_0, \dots, a_{n-1})$ represents a sequence of measurements over time
- Measurements might be noisy and have to be smoothed out
- Replace a_i by weighted average of nearby last m and next m measurements (e.g., Gaussian smoothing):

$$a_i' = \frac{1}{Z} \cdot \sum_{j=i-m}^{i+m} a_j e^{-(i-j)^2}$$

- New vector $\boldsymbol{a}'$ is the convolution of $\boldsymbol{a}$ and the weight vector $\frac{1}{Z} \cdot \left(e^{-m^2}, e^{-(m-1)^2}, \dots, e^{-1}, 1, e^{-1}, \dots, e^{-(m-1)^2}, e^{-m^2}\right)$
- Might need to take care of boundary points…

More Applications of Convolutions

Combining Histograms:

- Vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ represent two histograms
- E.g., annual income of all men & annual income of all women

- Goal: Get new histogram $\boldsymbol{c}$ representing combined income of all possible pairs of men and women:

$$\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$$

Also, the DFT (and thus the FFT alg.) has many other applications!