Algorithm Theory

Chapter 1
Divide and Conquer

Part V:
Fast Polynomial Multiplication 2

Fabian Kuhn

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** in time $\boldsymbol{O(n \log n)}$

$2 \times 2n$ 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** in time $\boldsymbol{O(n)}$

$2n$ 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

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

→ 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:

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

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

DFT Matrix Inverse

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

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

- We need to show that
  - $(W^{-1}W)_{i,j} = 1$ for $i = j$
  - $(W^{-1}W)_{i,j} = 0$ for $i \neq j$

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 $\boldsymbol{i = j}$:

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

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 $\boldsymbol{i \neq j}$:

$\omega_n^{nk} = \omega_1^k = 1$

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

$\neq 1$

Geometric series: $\displaystyle\sum_{\ell=0}^{n-1} q^{\ell} = \frac{1 - q^n}{1 - q}$

Inverse DFT

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

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

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

- DFT and inverse DFT can both be computed using the 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** in time $\boldsymbol{O(n \log n)}$

$2 \times 2n$ 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** in time $\boldsymbol{O(n)}$

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

↓ **Interpolation** using **FFT** in time $\boldsymbol{O(n \log n)}$

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