universität freiburg

Algorithm Theory – WS 2024/25

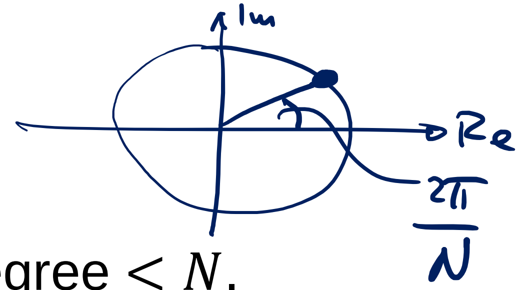
Chapter 1 : Divide and Conquer Algorithms
(Multiplication of Polynomials, remaining part)

Fabian Kuhn
Dept. of Computer Science
Algorithms and Complexity

Discrete Fourier Transform

$x^N = 1$

$\omega_N = e^{\frac{2\pi i}{N}}$



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

Discrete Fourier Transform (DFT):

- Assume $\boldsymbol{a} = (a_0, \dots, a_{N-1})$ is the coefficient vector of a polynomial p (of degree $\leq N-1$):

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

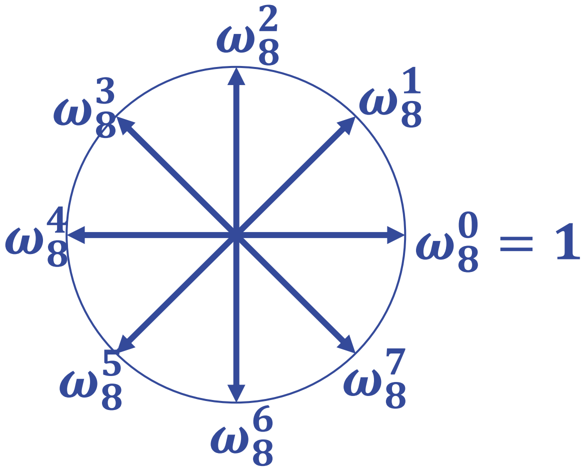
- Then, the Discrete Fourier Transform of the vector $\boldsymbol{a}$ is defined as

$$\mathbf{DFT}_N(\boldsymbol{a}) \coloneqq \left(\boldsymbol{p}(\boldsymbol{\omega}_N^0), \boldsymbol{p}(\boldsymbol{\omega}_N^1), \dots, \boldsymbol{p}(\boldsymbol{\omega}_N^{N-1})\right)$$

The values $\omega_N^0, \omega_N^1, \dots, \omega_N^{N-1}$ are the N complex solutions to the equation $x^N = 1$.

$$\omega_N^k = e^{2\pi i \cdot k/N}$$

We abuse notation and also write $\mathbf{DFT}_N(\boldsymbol{p})$



Faster Multiplication of Polynomials?

Observation: Multiplication is fast when using the point-value representation

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

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

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

Time: $O(n \cdot \log n)$

$2 \times 2n$ 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

Time: $O(n)$

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

Interpolation

???

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

Interpolation

Goal: 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

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

→ 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 $x_i = \omega_n^i$:

$$\begin{array}{cc} & \begin{array}{ccccc} & \phantom{\omega_n^{n-1}} & \phantom{\omega_n^{2(n-1)}} & j \downarrow & \phantom{\omega_n^{(n-1)(n-1)}} \end{array} \\ \begin{array}{c} \\ \\ i \longrightarrow \\ \\ \\ \end{array} & \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \end{array} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

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 $W^{-1}W = I_n$

Inverse Discrete Fourier Transform

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

$$g(z) = y_0 + y_1 z + y_2 z^2 + \ldots + y_{n-1} z^{n-1}$$

We get $\boldsymbol{a} = W^{-1} \cdot \boldsymbol{y}$ and therefore

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

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

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

$$\omega_n^{-k} = \omega_n^{n-k}$$

DFT and Inverse DFT

Inverse DFT:

$$a_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} y_j \cdot \left(\omega_n^{-k}\right)^j$$

- 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:

$$\begin{aligned}
&(a_0, a_1, \ldots, a_{n-1}) \\
&\quad = \frac{1}{n} \cdot \left(q(\omega_n^{-0}), q(\omega_n^{-1}), q(\omega_n^{-2}), \ldots, q\left(\omega_n^{-(n-1)}\right) \right) \\
&\quad = \frac{1}{n} \cdot \left(q(\omega_n^{0}), q(\omega_n^{n-1}), q(\omega_n^{n-2}), \ldots, q(\omega_n^{1}) \right)
\end{aligned}$$

Recall:

$$\begin{aligned}
\mathrm{DFT}_n(\boldsymbol{y}) &= \left(q(\omega_n^0), q(\omega_n^1), q(\omega_n^2), \ldots, q(\omega_n^{n-1}) \right) \\
&= n \cdot (a_0, a_{n-1}, a_{n-2}, \ldots, a_2, a_1)
\end{aligned}$$

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 the FFT algorithm in $O(n \log n)$ time.

- Hence, two polynomials of degree $< n$ can be multiplied in time $O(n \log n)$.

Faster Multiplication of Polynomials

Observation: Multiplication is fast when using the point-value representation

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

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

↓ **Evaluation** at points $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$ using **FFT** in time $O(n \log n)$

$2 \times 2n$ 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 $O(n)$

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

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

$p(x)q(x)$ 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 by itself has many other applications!

- e.g., in particular in signal processing when moving between time and frequency domain…