Algorithm Theory

Chapter 1
Divide and Conquer

Part IV:
Fast Polynomial Multiplication 1

Fabian Kuhn

Representation of Polynomials

Coefficient Representation:

- Polynomial of degree $n-1$ defined by coefficients $a_0, \dots, a_{n-1}$:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

Point-value Representation:

- Polynomial $p(x)$ of degree $n-1$ is given by n point-value pairs:

$$p = \{(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{n-1}, p(x_{n-1}))\}$$

  where $x_i \neq x_j$ for $i \neq j$.

- Example: The polynomial

$$p(x) = 3x^3 - 15x^2 + 18x = 3x(x-2)(x-3)$$

  is uniquely defined by the four point-value pairs $(0,0), (1,6), (2,0), (3,0)$.

Operations: Coefficient Representation

$$\boldsymbol{p(x) = a_{n-1}x^{n-1} + \cdots + a_0}, \qquad \boldsymbol{q(x) = b_{n-1}x^{n-1} + \cdots + b_0}$$

Evaluation: Horner's method: Time $O(n)$

Addition:

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

- Time: $O(n)$

Multiplication:

$$p(x) \cdot q(x) = c_{2n-2}x^{2n-2} + \cdots + c_0, \qquad \text{where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

- Naïve solution: Need to compute product $a_i b_j$ for all $0 \leq i, j \leq n$
- Time: Naïve alg. $O(n^2)$ Karatsuba Alg. $O(n^{1.58496\ldots})$

Operations: Point-Value Representation

$$\boldsymbol{p} = \{(\boldsymbol{x_0}, \boldsymbol{p}(\boldsymbol{x_0})), \dots, (\boldsymbol{x_{n-1}}, \boldsymbol{p}(\boldsymbol{x_{n-1}}))\}$$
$$\boldsymbol{q} = \{(\boldsymbol{x_0}, \boldsymbol{q}(\boldsymbol{x_0})), \dots, (\boldsymbol{x_{n-1}}, \boldsymbol{q}(\boldsymbol{x_{n-1}}))\}$$

- Note: We use the **same points $\boldsymbol{x_0, \dots, x_{n-1}}$** for both polynomials.

Addition:

$$p + q = \{(x_0, p(x_0) + q(x_0)), \dots, (x_{n-1}, p(x_{n-1}) + q(x_{n-1}))\}$$

- Time: $O(n)$

Multiplication:

$$p \cdot q = \{(x_0, p(x_0) \cdot q(x_0)), \dots, (x_{2n-2}, p(x_{2n-2}) \cdot q(x_{2n-2}))\}$$

- Time: $O(n)$
- **Remark:** Need both polynomials at (the same) $2n - 1$ points.

Evaluation: Polynomial interpolation can be done in $O(n^2)$

Operations on Polynomials

Cost depending on representation:

	Coefficient	Point-Value
Evaluation	$O(n)$	$O(n^2)$
Addition	$O(n)$	$O(n)$
Multiplication	$O(n^{1.58})$	$O(n)$

default representation

Can we improve this?

Faster Polynomial Multiplication?

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 $x_0, x_1, \dots, x_{2n-2}$

> We will consider this first.
> ***Remark:*** *We can freely choose the points.*

$2 \times 2n$ point-value pairs $\big(x_i, p(x_i)\big)$ and $\big(x_i, q(x_i)\big)$

↓ **Point-wise multiplication**

> Time: $O(n)$

$2n$ point-value pairs $\big(x_i, p(x_i)q(x_i)\big)$

↓ **Interpolation**

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

Coefficients to Point-Value Representation

Given: Polynomial $p(x)$ by the coefficient vector $(a_0, a_1, \dots, a_{N-1})$

Goal: Compute $p(x)$ for all x in a given set $\boldsymbol{X}$

- Where X is of size $|X| = N$
- Assume that N is a power of 2

$N = 2n - 1$

We will fix X later.

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

We call the variable y because we will not plug in x into p_0 and p_1

Coefficients to Point-Value Representation

Goal: Compute $p(x)$ for all x in a given set X of size $|X| = N$

- Divide $p(x)$ of degr. $N-1$ into 2 polynomials of degr. $^N/_2 - 1$

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

Let's first look at the "combine" step:

- We need to compute $p(x)$ for all $x \in X$ after recursive calls for polynomials p_0 and p_1:
- Plug $y = x^2$ into $p_0(y)$ and $p_1(y)$:

$$p_0(x^2) = a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{N-2} x^{N-2}$$
$$p_1(x^2) = a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_{N-1} x^{N-2}$$

$$\boldsymbol{p(x) = p_0(x^2) + x \cdot p_1(x^2)}$$

Coefficients to Point-Value Representation

Goal: Compute $p(x)$ for all x in a given set X of size $|X| = N$

- Divide $p(x)$ of degr. $N-1$ into 2 polynomials of degr. $N/2 - 1$

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

Let's first look at the "combine" step:

$$\forall x \in X: \quad p(x) = p_0(x^2) + x \cdot p_1(x^2)$$

- Goal: *recursively compute* $p_0(y)$ and $p_1(y)$ for all $y \in X^2$
  - Where $X^2 := \{x^2 : x \in X\}$
- Generally, we have $|X^2| = |X|$

Analysis

Let's get a recurrence recurrence for the given algorithm:

Time for polynomial of degree N with set X: $T(N,|X|)$

$$T(N,|X|) = 2 \cdot T\left(N/_2, |X^2|\right) + O(N + |X|)$$

Assume that $|X^2| = |X| = N$:

$$T(N,N) = 2 \cdot T\left(N/_2, N\right) + O(N) = \cdots = N \cdot \left(T(1,N) + O(N)\right)$$

$$T(1,N) = O(N)$$

Therefore, we get $T(N,|X|) = O(N^2)$.

- We need $|X^2| < |X|$ to get a faster algorithm!

Faster Algorithm?

In order to have a faster algorithm, we need $|X^2| < |X|$:

- $|X^2| < |X|$ if X contains values x, x' such that $x^2 = x'^2$:

$$X = \{-1, +1\} \implies X^2 = \{+1\}$$

- We also need $|(X^2)^2| = |X^4| < |X^2|$:
  - Can we get a set Y of size 4 such that $Y^2 = \{-1, +1\}$?

- **Complex numbers $\mathbb{C}$:**
  - Define imaginary constant i s.t. $i^2 = -1$
  - Complex numbers: $\mathbb{C} = \{a + i \cdot b \mid a, b \in \mathbb{R}\}$

- $Y = \{-1, +1, -i, +i\} \implies Y^2 = \{-1, +1\}$

- $\forall x \in \mathbb{C} \setminus \{0\}$, there are 2 numbers $y, z \in \mathbb{C}$ s.t. $y^2 = z^2 = x$

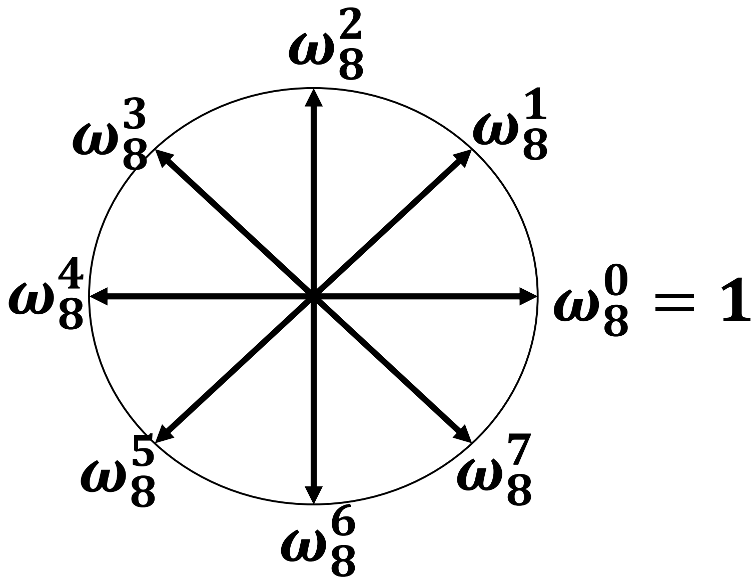
Choice of X

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

Consider the N complex roots of unity:

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

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



Powers of $\boldsymbol{\omega_N}$ (roots of unity):

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

Note: $\omega_N^k = e^{2\pi i k/N} = \cos\frac{2\pi k}{N} + i \cdot \sin\frac{2\pi k}{N}$

Properties of the Roots of Unity

Cancellation Lemma:

- For all integers $n > 0$, $k \geq 0$, and $d > 0$, we have:

$$\boldsymbol{\omega_{dn}^{dk} = \omega_n^k}\,, \qquad \boldsymbol{\omega_n^{k+n} = \omega_n^k}$$

Proof: Recall that $\omega_n = e^{2\pi i/n}$, $e^{2\pi i} = 1$

$$\omega_{dn}^{dk} = \left(e^{\frac{2\pi i}{dn}}\right)^{dk} = e^{\frac{2\pi i}{dn}\cdot dk} = e^{\frac{2\pi i}{n}\cdot k} = \omega_n^k$$

$$\omega_n^{k+n} = \left(e^{\frac{2\pi i}{n}}\right)^{k+n} = e^{\frac{2\pi i}{dn}\cdot(k+n)} = e^{\frac{2\pi i}{n}\cdot k} \cdot e^{2\pi i} = \omega_n^k$$

Properties of the Roots of Unity

Claim: If $X = \left\{\omega_{2k}^{j} : j \in \{0, \dots, 2k-1\}\right\}$, we have

$$\boldsymbol{X^2 = \left\{\omega_k^j : j \in \{0, \dots, k-1\}\right\}}, \qquad \boldsymbol{|X^2| = \frac{|X|}{2}}$$

Proof:

- We just showed: $\omega_{dn}^{dk} = \omega_n^k$, $\omega_n^{k+n} = \omega_n^k$

- Consider some $x = \omega_{2k}^{j} \in X$:

$$x^2 = \left(\omega_{2k}^{j}\right)^2 = \omega_{2k}^{2j} = \omega_k^j$$

$$\text{If } j \geq k : \omega_k^j = \omega_k^{j-k}$$

- Clearly, $|X^2| = |X|/2 \quad (|X| = 2k, |X^2| = k)$.

Analysis

New recurrence formula:

$$T(N, |X|) \leq 2 \cdot T\left({}^{N}\!/_{2}, {}^{|X|}\!/_{2}\right) + O(N + |X|)$$

- W.l.o.g., assume that N is a power of 2
  - We can just add additional coefficients that are equal to 0.

- To compute $p(x)$ for the N different points in X, we need to recursively compute $p_0(x^2)$ and $p_1(x^2)$ for all $x^2 \in X^2$
  - p has degree $N - 1$, p_0 and p_1 have degree ${}^{N}\!/_{2} - 1$, $|X^2| = {}^{|X|}\!/_{2}$
  - Combine step: compute $p(x) = p_0(x^2) + x \cdot p_1(x^2)$ for all $x \in X$

- $|X| = N \implies T(N) \leq 2 \cdot T({}^{N}\!/_{2}) + O(N)$

$$\boldsymbol{T(N) = O(N \cdot \log N)}$$

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 points $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$ — Time: $O(n \log n)$

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

Point-wise multiplication — Time: $O(n)$

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

Interpolation — **???**

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

Discrete Fourier Transform

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

Discrete Fourier Transform (DFT):

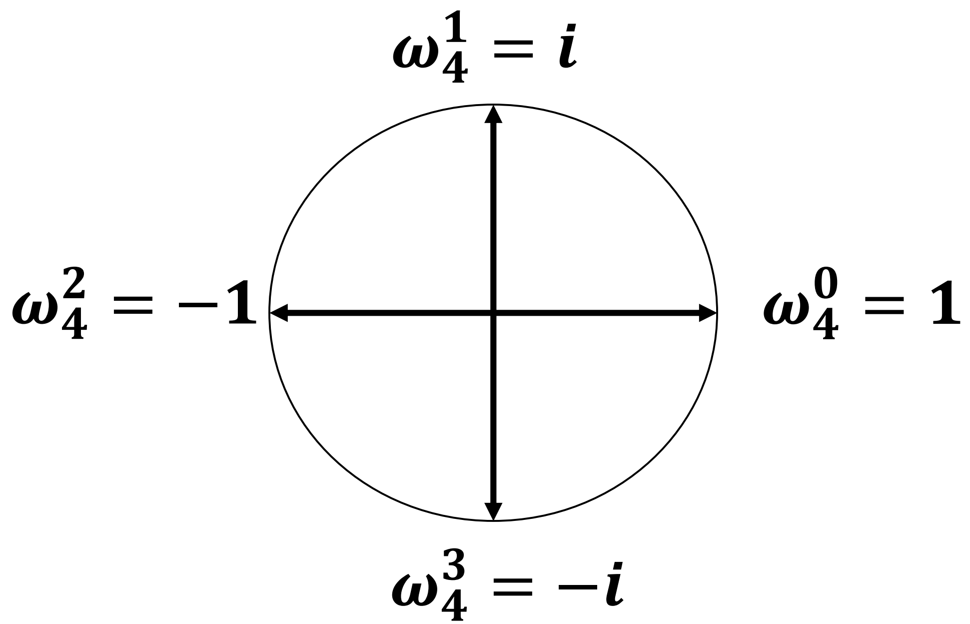
- Assume $a = (a_0, \dots, a_{N-1})$ is the coefficient vector of poly. p

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

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

Example

- Consider polynomial $p(x) = 3x^3 - 15x^2 + 18x$
- Choose $N = 4$
- Roots of unity:



Example

- Consider polynomial $p(x) = 3x^3 - 15x^2 + 18x$
- $N = 4$, roots of unity: $\omega_4^0 = 1, \omega_4^1 = i, \omega_4^2 = -1, \omega_4^3 = -i$
- Evaluate $p(x)$ at ω_4^k:

$$\left(\omega_4^0, p(\omega_4^0)\right) = \left(1, p(1)\right) = (1,6)$$
$$\left(\omega_4^1, p(\omega_4^1)\right) = \left(i, p(i)\right) = (i, 15 + 15i)$$
$$\left(\omega_4^2, p(\omega_4^2)\right) = \left(-1, p(-1)\right) = (-1, -36)$$
$$\left(\omega_4^3, p(\omega_4^3)\right) = \left(-i, p(-i)\right) = (-i, 15 - 15i)$$

- For $a = (0,18,-15,3)$:

$$\mathbf{DFT_4}(\boldsymbol{a}) = (\mathbf{6}, \mathbf{15} + \mathbf{15}\boldsymbol{i}, -\mathbf{36}, \mathbf{15} - \mathbf{15}\boldsymbol{i})$$

DFT: Recursive Structure

Evaluation for $k = 0, \dots, N-1$:

$$p(\omega_N^k) = p_0\big((\omega_N^k)^2\big) + \omega_N^k \cdot p_1\big((\omega_N^k)^2\big)$$

$$= \begin{cases} p_0\big(\omega_{N/2}^k\big) + \omega_N^k \cdot p_1\big(\omega_{N/2}^k\big) & \text{if } k < {}^N/_2 \\ p_0\left(\omega_{N/2}^{k-N/2}\right) + \omega_N^k \cdot p_1\left(\omega_{N/2}^{k-N/2}\right) & \text{if } k \geq {}^N/_2 \end{cases}$$

For the coefficient vector a of $p(x)$:

$$\mathrm{DFT}_N(a) = \left(p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right), p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right)\right)$$
$$+ \left(\omega_N^0 p_1(\omega_{N/2}^0), \dots, \omega_N^{N/2-1} p_1\left(\omega_{N/2}^{N/2-1}\right), \omega_N^{N/2} p_1(\omega_{N/2}^0), \dots, \omega_N^{N-1} p_1\left(\omega_{N/2}^{N/2-1}\right)\right)$$

Example

For the coefficient vector a of $p(x)$:

$$\mathrm{DFT}_N(a) = \left(p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right), p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right)\right)$$
$$+\left(\omega_N^0 p_1(\omega_{N/2}^0), \dots, \omega_N^{N/2-1} p_1\left(\omega_{N/2}^{N/2-1}\right), \omega_N^{N/2} p_1(\omega_{N/2}^0), \dots, \omega_N^{N-1} p_1\left(\omega_{N/2}^{N/2-1}\right)\right)$$

$N = 4$:

$$\begin{aligned} p(\omega_4^0) &= p_0(\omega_2^0) + \omega_4^0 p_1(\omega_2^0) \\ p(\omega_4^1) &= p_0(\omega_2^1) + \omega_4^1 p_1(\omega_2^1) \\ p(\omega_4^2) &= p_0(\omega_2^0) + \omega_4^2 p_1(\omega_2^0) \\ p(\omega_4^3) &= p_0(\omega_2^1) + \omega_4^3 p_1(\omega_2^1) \end{aligned}$$

Need: $\left(p_0(\omega_2^0), p_0(\omega_2^1)\right)$ and $\left(p_1(\omega_2^0), p_1(\omega_2^1)\right)$

(DFTs of coefficient vectors of p_0 and p_1)

Summary: Computation of DFT_N

- Divide-and-conquer algorithm for $\mathrm{DFT}_N(p)$:

1. Divide

$N \leq 1$: $\mathrm{DFT}_1(p) = a_0$

$N > 1$: Divide p into p_0 (even coeff.) and p_1 (odd coeff).

2. Conquer

Solve $\mathrm{DFT}_{N/2}(p_0)$ and $\mathrm{DFT}_{N/2}(p_1)$ recursively

3. Combine

Compute $\mathrm{DFT}_N(p)$ based on $\mathrm{DFT}_{N/2}(p_0)$ and $\mathrm{DFT}_{N/2}(p_1)$

Small Constant Improvement

Polynomial p of degree $N-1$:

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

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

- $\omega_N^{k-N/2} = e^{\frac{2\pi i}{N}\cdot(k-N/2)} = e^{\frac{2\pi i}{N}\cdot k} \cdot e^{-\frac{2\pi i}{N}\cdot\frac{N}{2}} = \omega_N^k \cdot e^{-\pi i} = -\omega_N^k$

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 $N = 8$

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

Fast Fourier Transform (FFT) Algorithm

Algorithm FFT(a)

- Input: Array a of length N, where N is a power of 2
- Output: $\mathrm{DFT}_N(a)$

if $n = 1$ **then return** a_0; // $a = [a_0]$

$d^{[0]} := \mathrm{FFT}([a_0, a_2, \dots, a_{N-2}])$;

$d^{[1]} := \mathrm{FFT}([a_1, a_3, \dots, a_{N-1}])$;

$\omega_N := e^{2\pi i/N}$; $\omega := 1$;

for $k = 0$ **to** $N/2 - 1$ **do** // $\omega = \omega_N^k$

  $x := \omega \cdot d_k^{[1]}$;

  $d_k := d_k^{[0]} + x$; $d_{k+N/2} := d_k^{[0]} - x$;

  $\omega := \omega \cdot \omega_N$

end;

return $d = [d_0, d_1, \dots, d_{N-1}]$;