Advanced Core in Algorithm Design #6
算法設計要論 第 6 回

Yasushi Kawase
河瀬 康志

Nov. 9th, 2021

last update: 2:21pm, November 9, 2021

Schedule

Lec. #	Date	Topics
1	10/5	Introduction, Stable matching
2	10/12	Basics of Algorithm Analysis, Graphs
3	10/19	Greedy Algorithms (1/2)
4	10/26	Greedy Algorithms (2/2)
5	11/2	Divide and Conquer (1/2)
6	11/9	Divide and Conquer (2/2)
7	11/16	Dynamic Programming (1/2)
8	11/30	Dynamic Programming (2/2)
9	12/7	Network Flow (1/2)
10	12/14	Network Flow (2/2)
11	12/21	NP and Computational Intractability
12	1/4	Approximation Algorithms (1/2)
13	1/11	Approximation Algorithms (2/2)
14	1/18	Final Examination

Reprint: reccurence relations

Recurrence relations	Computational time
$T(n) = T(n/2) + \mathrm{O}(1)$	$T(n) = \mathrm{O}(\log n)$
$T(n) = 2 \cdot T(n/2) + \mathrm{O}(1)$	$T(n) = \mathrm{O}(n)$
$T(n) = 2 \cdot T(n/2) + \mathrm{O}(n)$	$T(n) = \mathrm{O}(n \log n)$
$T(n) = 3 \cdot T(n/2) + \mathrm{O}(n)$	$T(n) = \mathrm{O}(n^{\log_2 3})$
$T(n) = aT(n/b) + \mathrm{O}(n^d)$	$T(n) = \begin{cases} \mathrm{O}(n^d) & \text{if } d > \log_b a \\ \mathrm{O}(n^d \log n) & \text{if } d = \log_b a \\ \mathrm{O}(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$

$a > 0, b > 1, d \geq 0$

Outline

Integer multiplication

Problem

Input n-bit positive integers x and y

Goal output their product $x \cdot y$

"grade school" algorithm: $\Theta(n^2)$ time

→ we can improve it to $\mathrm{O}(n^{1.59})$ by the divide-and-conquer technique

Example $x = 12 = 1100_{(2)}$, $y = 13 = 1101_{(2)}$

$$
\begin{array}{rr}
 & 12 \\
\times) & 13 \\
\hline
 & 36 \\
 & 12 \\
\hline
 & 156
\end{array}
\qquad\qquad
\begin{array}{rr}
 & 1100 \\
\times) & 1101 \\
\hline
 & 1100 \\
 & 0000 \\
 & 1100 \\
 & 1100 \\
\hline
 & 10011100
\end{array}
$$

Idea

- split x and y into their left and right halves

$$x = \boxed{\quad x_L \quad}\boxed{\quad x_R \quad} = 2^{n/2} x_L + x_R$$

$$y = \boxed{\quad y_L \quad}\boxed{\quad y_R \quad} = 2^{n/2} y_L + y_R$$

- the product of x and y is

$$xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

- straightforward application of divide-and-conquer

$T(n) = 4T(n/2) + \mathrm{O}(n) \longrightarrow T(n) = \mathrm{O}(n^2)$ (not improved)

Idea

- split x and y into their left and right halves

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$
$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R$$

- the product of x and y is

$$xy = 2^n x_L y_L + 2^{n/2}(\underbrace{x_L y_R + x_R y_L}_{x_L y_L + x_R y_R - (x_L - x_R)(y_L - y_R)}) + x_R y_R$$

- straightforward application of divide-and-conquer

$T(n) = 4T(n/2) + \mathrm{O}(n) \longrightarrow T(n) = \mathrm{O}(n^2)$ (not improved)

- xy can be computed by three $n/2$-bit multiplications (Karatsuba algorithm)

$T(n) = 3T(n/2) + \mathrm{O}(n) \longrightarrow T(n) = \mathrm{O}(n^{\log_2 3}) = \mathrm{O}(n^{1.59})$

$\log_2 3 \approx 1.58496$

Karatsuba algorithm

multiply(n, x, y) (assume n is a power of 2)

if $n = 1$ **then Return** xy;
Let x_L and x_R be leftmost and rightmost $n/2$ bits of x, respectively;
Let y_L and y_R be leftmost and rightmost $n/2$ bits of y, respectively;
$p \leftarrow \texttt{multiply}(n/2, x_L, y_L)$;
$q \leftarrow \texttt{multiply}(n/2, x_R, y_R)$;
$r \leftarrow \texttt{multiply}(n/2, x_L - x_R, y_L - y_R)$;
Return $p \cdot 2^n + (p + q - r)2^{n/2} + q$;

Theorem

The running time of the algorithm is $\mathrm{O}(n^{\log_2 3}) = \mathrm{O}(n^{1.59})$

Exercise

base 100

Compute 2021×1024 by grade school algorithm and Karatsuba algorithm

Outline

Polynomial multiplication

Problem

Input $A(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ and $B(x) = b_0 + b_1 x + \cdots b_{n-1}x^{n-1}$

Goal output $C(x) = A(x) \cdot B(x) = c_0 + c_1 x + \cdots + c_{2(n-1)}x^{2(n-1)}$

Essentially contains integer multiplication

naive algorithm: $\Theta(n^2)$ time $\quad (\because c_k = \sum_{i=0}^{k} a_i b_{k-i})$

Karatsuba algorithm: $\mathrm{O}(n^{1.59})$

$\longrightarrow$ we can improve it to $\mathrm{O}(n \log n)$

Example

$$(1 + 2x + 4x^2) \cdot (3 - x + 2x^2) = 3 + 5x + 12x^2 + 8x^4$$

Basic operations

Two polynomials:

$$A(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$$
$$B(x) = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}$$

- Addition: $\mathrm{O}(n)$ time

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

- Evaluation: $\mathrm{O}(n)$ time

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

Value representation of polynomials

Fundamental theorem of algebra

A degree n polynomial with complex coefficients has exactly n complex roots

Corollary

A degree-$(n-1)$ polynomial is characterized by its values at distinct n points

Example $(n = 3)$

- Coefficient representation: $A(x) = 1 + 2x + 4x^2$
- Value representation: $A(0) = 1$, $A(1) = 7$, $A(-1) = 3$

Basic operations (value representation)

Two polynomials with points $x_0, x_1, \dots, x_{n-1}$:

$$A(x)\colon (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$$
$$B(x)\colon (x_0, z_0), \dots, (x_{n-1}, z_{n-1})$$

- Addition: $\mathrm{O}(n)$ time

$$A(x) + B(x)\colon (x_0, y_0 + z_0), \dots, (x_{n-1}, y_{n-1} + z_{n-1})$$

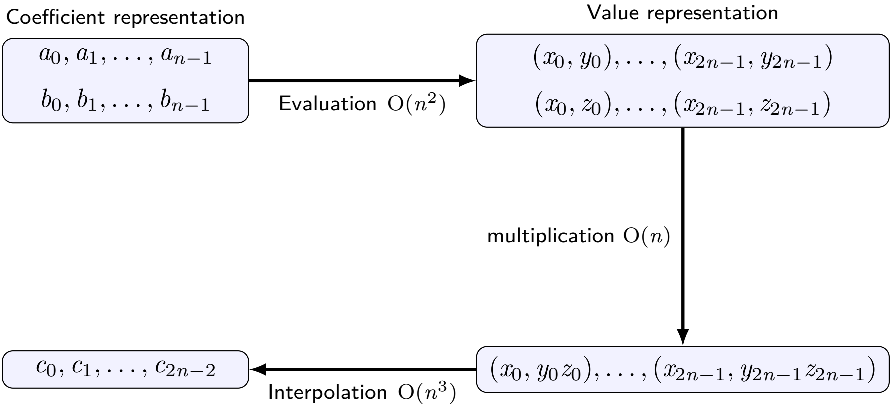
- Multiplication: $\mathrm{O}(n)$ time (if $A(x) \cdot B(x)$ is of degree at most $n-1$)

$$A(x) \cdot B(x)\colon (x_0, y_0 \cdot z_0), \dots, (x_{n-1}, y_{n-1} \cdot z_{n-1})$$

- Interpolation: $\mathrm{O}(n^3)$ time by Lagrange's formula (too slow)

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

Compute the multiplication in value representation



Idea

Compute the multiplication in value representation

Coefficient representation

Value representation

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\begin{array}{c} a_0, a_1, \ldots, a_{n-1} \\ b_0, b_1, \ldots, b_{n-1} \end{array} \arrow[r, "\cancel{\text{Evaluation } \mathrm{O}(n^2)}"', "\text{FFT } \mathrm{O}(n \log n)"' {yshift=-12pt}] & \begin{array}{c} (x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1}) \\ (x_0, z_0), \ldots, (x_{2n-1}, z_{2n-1}) \end{array} \arrow[d, "\text{multiplication } \mathrm{O}(n)"'] \\
c_0, c_1, \ldots, c_{2n-2} & (x_0, y_0 z_0), \ldots, (x_{2n-1}, y_{2n-1} z_{2n-1}) \arrow[l, "\cancel{\text{Interpolation } \mathrm{O}(n^3)}", "\text{inverse FFT } \mathrm{O}(n \log n)" {yshift=-12pt}]
\end{tikzcd}$$

Evaluation by divide-and-conquer

- Break up polynomial into even- and odd-degree terms

$$A(x) = \underbrace{A_e(x^2)}_{\text{even-degree terms}} + x\underbrace{A_o(x^2)}_{\text{odd-degree terms}}$$

Example: $3 + 4x + 6x^2 + 5x^3 + x^4 + 2x^5 = (3 + 6x^2 + x^4) + x(4 + 5x^2 + 2x^4)$

- Calculations needed for $A(x_i)$ can be recycled toward computing $A(-x_i)$

$$A(x_i) = A_e(x_i^2) + x_i A_o(x_i^2)$$
$$A(-x_i) = A_e(x_i^2) - x_i A_o(x_i^2)$$

- What is a good way to choose points that can be recycled well?

Discrete Fourier Transform

Key idea: Choose $x_k = \omega^k$ where $\omega = e^{2\pi i/n}$ is principle nth root of unity

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Fourier matrix F_n

- $A(x) = \sum_{j=0}^{n-1} a_j x^j = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- $y_k = A(\omega^k) = \sum_{j=0}^{n-1} a_j \omega^{jk}$ $(k = 0, 1, \ldots, n-1)$

Fast Fourier Transform

- $A(\omega^k) = A_e(\omega^{2k}) + \omega^k A_o(\omega^{2k})$
- $A(\omega^{k+n/2}) = A(-\omega^k) = A_e(\omega^{2k}) - \omega^k A_o(\omega^{2k})$

$\texttt{FFT}(n, a_0, a_1, a_2, \dots, a_{n-1})$

1. **if** $n = 1$ **then Return** a_0;
2. $(e_0, e_1, \dots, e_{n/2-1}) \leftarrow \texttt{FFT}(n/2, a_0, a_2, a_4, \dots, a_{n-2})$;
3. $(d_0, d_1, \dots, d_{n/2-1}) \leftarrow \texttt{FFT}(n/2, a_1, a_3, a_5, \dots, a_{n-1})$;
4. **for** $k \leftarrow 0, 1, 2, \dots, n/2-1$ **do**
5. $\quad \omega^k \leftarrow e^{2\pi i k/n}$, $y_k \leftarrow e_k + \omega^k d_k$, $y_{k+n/2} \leftarrow e_k - \omega^k d_k$
6. **Return** $(y_0, y_1, y_2, \dots, y_{n-1})$;

The total computational time is $T(n) = 2T(n/2) + \mathrm{O}(n)$

Theorem

The running time of FFT is $\mathrm{O}(n \log n)$

Inverse Discrete Fourier Transform

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Inverse Fourier matrix F_n^{-1}

$$F_n^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{pmatrix}$$

$(F_n F_n^{-1})_{jk} = \frac{1}{n} \sum_{\ell=0}^{n-1} \omega^{\ell(j-k)} = \delta_{jk}$ by $x^n - 1 = (x-1)(1 + x + x^2 + \cdots + x^{n-1})$

Inverse Fast Fourier Transform

InverseFFT$(n, y_0, y_1, y_2, \ldots, y_{n-1})$

```
if n = 1 then Return y_0;
(e_0, e_1, ..., e_{n/2-1}) ← InverseFFT(n/2, y_0, y_2, y_4, ..., y_{n-2});
(d_0, d_1, ..., d_{n/2-1}) ← InverseFFT(n/2, y_1, y_3, y_5, ..., y_{n-1});
for k ← 0, 1, 2, ..., n/2 - 1 do
    ω^k ← e^{-2πik/n}, a_k ← e_k + ω^k d_k, a_{k+n/2} ← e_k - ω^k d_k
Return (a_0, a_1, a_2, ..., a_{n-1});
```

To be precise, we need to divide the result by n to get the Inverse FFT

The total running time is $T(n) = 2\,T(n/2) + \mathrm{O}(n)$

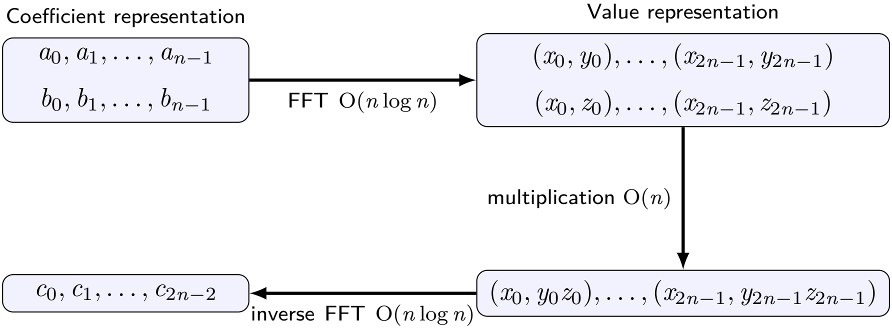
Theorem

The running time of Inverse FFT is $\mathrm{O}(n \log n)$

Conclusion

Theorem

The multiplication of $A(x) = \sum_{k=0}^{n-1} a_k x^k$ and $B(x) = \sum_{k=0}^{n-1} b_k x^{k-1}$ can be computed in $\mathrm{O}(n \log n)$ time



Integer multiplication revisited

Problem

Input n-bit positive integers a and b

Goal output their product $a \cdot b$

- $A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ $(a = A(2))$
- $B(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$ $(b = B(2))$
- $C(x) = A(x) \cdot B(x)$ $(ab = C(2))$

→ $a \cdot b$ can be computed in $\mathrm{O}(n \log n)$ arithmetic operations

require $\mathrm{O}(\log n)$ bits of precision

Integer multiplication revisited

Problem

Input n-bit positive integers a and b

Goal output their product $a \cdot b$

- $A(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ $(a = A(2))$
- $B(x) = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}$ $(b = B(2))$
- $C(x) = A(x) \cdot B(x)$ $(ab = C(2))$

→ $a \cdot b$ can be computed in $\mathrm{O}(n \log n)$ arithmetic operations

require $\mathrm{O}(\log n)$ bits of precision

- $\mathrm{O}(n \log^2 n)$ bit operations as we need to use $\mathrm{O}(\log n)$ bits of precision
- $\mathrm{O}(n \log n \cdot \log \log n)$ bit operations by computing FFT over a ring
- cf. simple divide-and-conquer: $\mathrm{O}(n^{1.59})$ bit operations