Integer Multiplication

Cao Yusen

April 7, 2021

$$\underbrace{a}_{n-bits} \times \underbrace{b}_{n-bits} \longrightarrow O(?)$$

- $O(n^2)$
- $O(n^{\log_2 3})$
- faster

$O(n^2)$

Long multiplication

$$A \times B$$

$$\begin{array}{ccccc}
 & & 1 & 2 & 3 \\
\times & & 4 & 5 & 6 \\
\hline
 & & 7 & 3 & 8 \\
 & 6 & 1 & 5 & \\
4 & 9 & 2 & & \\
\hline
5 & 6 & 0 & 8 & 8
\end{array}$$

- $B = \sum_{i=1}^{n} b_i \cdot 10^{i-1}$

$$\implies A \times B = \sum_{i=1}^{n} A b_i \cdot 10^{i-1}$$

分治法

$$x \times y$$

- $x = a \cdot 10^{\frac{n}{2}} + b$
- $y = c \cdot 10^{\frac{n}{2}} + d$

$$\begin{aligned} xy &= (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) \\ &= ac \cdot 10^{n} + (ad + bc) \cdot 10^{\frac{n}{2}} + bd \end{aligned}$$

$O(n^2)$

分治法

$$x \times y$$

- $x = a \cdot 10^{\frac{n}{2}} + b$
- $y = c \cdot 10^{\frac{n}{2}} + d$

$$\begin{aligned} xy &= (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d) \\ &= ac \cdot 10^{n} + (ad + bc) \cdot 10^{\frac{n}{2}} + bd \end{aligned}$$

$$T(n) = 4T(\frac{n}{2}) + O(n)$$

$$\xrightarrow{\text{Master Theorem}}$$

$$T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$$

$O(n^2)$

分治法

$$xy = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$$

$O(n^2)$

分治法

$$xy = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$$

$$ad + bc = ac + bd - (a - b)(c - d)$$

$O(n^2) \to O(n^{\log_2 3})$

分治法 $\to$ Karatsuba 算法

$$xy = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$$

$$ad + bc = ac + bd - (a - b)(c - d)$$

$O(n^2) \to O(n^{\log_2 3})$

Karatsuba 算法

$$xy = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$$

$$ad + bc = ac + bd - (a - b)(c - d)$$

$$\begin{aligned} xy &= ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd \\ &= ac \cdot 10^n + (ac + bd - (a - b)(c - d)) \cdot 10^{\frac{n}{2}} + bd \end{aligned}$$

$O(n^2) \to O(n^{\log_2 3})$

Karatsuba 算法

$$xy = ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd$$

$$ad + bc = ac + bd - (a-b)(c-d)$$

$$\begin{aligned} xy &= ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd \\ &= ac \cdot 10^n + (ac + bd - (a-b)(c-d)) \cdot 10^{\frac{n}{2}} + bd \end{aligned}$$

$$\cancel{T(n) = 4T(\tfrac{n}{2}) + O(n)}$$

$$T(n) = 3T(\frac{n}{2}) + O(n)$$

$$\xrightarrow{\text{Master Theorem}}$$

$$T(n) = \Theta(n^{\log_2 3})$$

$O(n^{\log_2 3}) \to$ faster

Karatsuba 算法 $\xrightarrow{\text{Generalized}}$ Toom-Cook 算法

- Given two large integers, a and b, Toom-Cook splits up a and b into k smaller parts each of length l, and performs operations on the parts. As k grows, one may combine many of the multiplication sub-operations, thus reducing the overall complexity of the algorithm. The multiplication sub-operations can then be computed recursively using Toom-Cook multiplication again, and so on. Toom–Cook multiplication
- 例如 Toom-3 可以将本来需要的 9 次乘法优化成 5 次乘法, 时间复杂度为 $\Theta(n^{\frac{\log 5}{\log 3}}) \approx \Theta(n^{1.46})$

$O(n^{\log_2 3}) \rightarrow$ faster

$$A \times B$$

$$A = a_0 + a_1 \times 10 + a_2 \times 10^2 + \cdots + a_{n-1} \times 10^{n-1}$$

$$B = b_0 + b_1 \times 10 + b_2 \times 10^2 + \cdots + b_{n-1} \times 10^{n-1}$$

$O(n^{\log_2 3}) \rightarrow$ faster

$$A \times B$$

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

$$B(x) = b_0 + b_1 \times x + b_2 \times x^2 + \cdots + b_{n-1} \times x^{n-1}$$

$$A \times B = A(10) \times B(10)$$

$O(n^{\log_2 3}) \rightarrow$ faster

$$A \times B$$

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

$$B(x) = b_0 + b_1 \times x + b_2 \times x^2 + \cdots + b_{n-1} \times x^{n-1}$$

$$A \times B = A(10) \times B(10)$$

$$A(x) \times B(x)$$

多项式的表示

1. 系数表示法
   用多项式的系数序列来表示多项式

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \Leftrightarrow f(x) = \{a_0, a_1, \cdots, a_n\}$$

2. 点值表示法
   把多项式看成一个函数，从上面选取 $n+1$ 个点，利用这 $n+1$ 个点来唯一的表示这个函数

$$\begin{aligned} f(x_0) &= y_0 = a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 + \cdots + a_n x_0^n \\ f(x_1) &= y_1 = a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \cdots + a_n x_1^n \\ &\vdots \\ f(x_n) &= y_n = a_0 + a_1 x_n + a_2 x_n^2 + a_3 x_n^3 + \cdots + a_n x_n^n \end{aligned}$$

   那么用点值表示法表示 $f(x)$ 如下：

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \Leftrightarrow f(x) = \{(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)\}$$

Polynomial Multiplication

多项式的乘积

计算多项式乘积 $f(x) \times g(x)$

- 如果用系数表示法
  用多项式的系数序列来表示多项式，我们要枚举 f 的每一位的系数与 g 的每一位的系数相乘，时间复杂度 $O(n^2)$
- 如果用点值表示法

$$f(x) = \{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$$

$$g(x) = \{(x_0, g(x_0)), (x_1, g(x_1)), \ldots, (x_n, g(x_n))\}$$

$$f(x) \times g(x) = \{(x_0, f(x_0)\, g(x_0)), (x_1, f(x_1)\, g(x_1)), \ldots, (x_n, f(x_n)\, g(x_n))\}$$

  观察可知，如果两个多项式都取相同的 x，我们只要计算其对应的 y 的乘积即可得到点值形式的表示。这种情况下多项式乘法的时间复杂度只有枚举 x 的 $O(n)$。

如何快速解决这个问题呢?

DFT & FFT

Fourier Transform

- 离散傅里叶变换 (Discrete Fourier Transform), 是傅里叶变换在时域和频域上都呈离散的形式, 将信号的时域采样变换为其 DTFT 的频域采样。而 IDFT(Inverse Discrete Fourier Transform) 就是其对应的逆变换。
- 快速傅立叶变换 (Fast Fourier transform) 则是实现 DFT 的一种高效算法。

在多项式乘法中,

- DFT 是把多项式由系数表示法转为点值表示法的过程
- IDFT 是把多项式的点值表示法转化为系数表示法的过程
- FFT 则是通过取某些特殊的 x 的点值来加速 DFT 和 IDFT 的过程

Complex Roots of Unity

单位复根

定义

- $x^n = 1$ 在复数意义下的解称为 n 次复根。显然，这样的解有 n 个。
- 设 $\omega_n = e^{\frac{2\pi i}{n}}$，则 $x^n = 1$ 的解集表示为 $\{\omega_n^k | k = 0, 1, \cdots, n-1\}$，而我们称 ω_n 是 n 次单位复根

而根据欧拉公式：

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

我们就可以知道 n 次单位复根的算术表示。

性质

1. $\omega_n^n = 1$
2. $\omega_n^k = \omega_{2n}^{2k}$
3. $\omega_{2n}^{k+n} = -\omega_{2n}^k$

快速傅里叶变换

对于一个多项式

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

我们将高次项系数补零直到 $n = 2^m (m \in \mathbb{Z})$, 对于一个 $n-1$ 项的多项式
然后按次数的奇偶性分组, 从奇数组中提取一个 x

$$\begin{aligned} f(x) &= (a_0 + a_2 x^2 + \cdots a_{n-2} x^{n-2}) + (a_1 x + a_3 x^3 + \cdots + a_{n-1} x^{n-1}) \\ &= (a_0 + a_2 x^2 + \cdots a_{n-2} x^{n-2}) + x(a_1 + a_3 x^2 + \cdots + a_{n-1} x^{n-2}) \end{aligned}$$

分别用奇偶次项构建新函数:

- $G(x) = a_0 + a_2 x + \cdots + a_{n-2} x^{\frac{n-2}{2}}$
- $H(x) = a_1 + a_3 x + \cdots + a_{n-1} x^{\frac{n-2}{2}}$

则原函数 $f(x)$ 用新函数表示为:

$$F(x) = G(x^2) + x \times H(x^2)$$

FFT

$$F(x) = G(x^2) + x \times H(x^2)$$

取 $x = \omega_n^k$, 利用单位复根的性质可得

$$\begin{aligned}
F\left(\omega_n^k\right) &= G\left(\left(\omega_n^k\right)^2\right) + \omega_n^k \times H\left(\left(\omega_n^k\right)^2\right) \\
&= G\left(\omega_n^{2k}\right) + \omega_n^k \times H\left(\omega_n^{2k}\right) \\
&= G\left(\omega_{\frac{n}{2}}^k\right) + \omega_n^k \times H\left(\omega_{\frac{n}{2}}^k\right)
\end{aligned}$$

同理可得

$$\begin{aligned}
F\left(\omega_n^{k+\frac{n}{2}}\right) &= G\left(\left(\omega_n^{k+\frac{n}{2}}\right)^2\right) + \omega_n^{k+\frac{n}{2}} \times H\left(\left(\omega_n^{k+\frac{n}{2}}\right)^2\right) \\
&= G\left(\omega_n^{2k}\right) + \omega_n^{k+\frac{n}{2}} \times H\left(\omega_n^{2k}\right) \\
&= G\left(\omega_{\frac{n}{2}}^k\right) - \omega_n^k \times H\left(\omega_{\frac{n}{2}}^k\right)
\end{aligned}$$

FFT

$$F(x) = G(x^2) + x \times H(x^2)$$

$$F\left(\omega_n^k\right) = G\left(\omega_{\frac{n}{2}}^k\right) + \omega_n^k \times H\left(\omega_{\frac{n}{2}}^k\right)$$
$$F\left(\omega_n^{k+\frac{n}{2}}\right) = G\left(\omega_{\frac{n}{2}}^k\right) - \omega_n^k \times H\left(\omega_{\frac{n}{2}}^k\right)$$

我们发现上面的两个式子只有一个常数项不同，所以当我们求出第一个式子的值后可以 $O(1)$ 的时间复杂度求出第二个式子的值。所以我们将原来的问题缩小到了之前的一半，而缩小后的问题还能继续递归的缩小。

- 时间复杂度为 $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$

IFFT

快速傅里叶逆变换

- 我们现在已经求出了 $(F(\omega_n^0), F(\omega_n^1), \cdots, F(\omega_n^{n-1}))$ 这个点值表示，要将它还原为系数表示

这相当于求解解线性方程组

$$
\begin{cases}
a_0\left(\omega_n^0\right)^0 & +\cdots & + & a_{n-1}\left(\omega_n^0\right)^{n-1} & = & F\left(\omega_n^0\right) \\
a_0\left(\omega_n^1\right)^0 & +\cdots & + & a_{n-1}\left(\omega_n^1\right)^{n-1} & = & F\left(\omega_n^1\right) \\
\vdots & & \vdots & \vdots & & \\
a_0\left(\omega_n^{n-1}\right)^0 & +\cdots & + & a_{n-1}\left(\omega_n^{n-1}\right)^{n-1} & = & F\left(\omega_n^{n-1}\right)
\end{cases}
$$

表示为矩阵即

$$
\begin{bmatrix}
\left(\omega_n^0\right)^0 & \left(\omega_n^0\right)^1 & \cdots & \left(\omega_n^0\right)^{n-1} \\
\left(\omega_n^1\right)^0 & \left(\omega_n^1\right)^1 & \cdots & \left(\omega_n^1\right)^{n-1} \\
\vdots & \vdots & \ddots & \vdots \\
\left(\omega_n^{n-1}\right)^0 & \left(\omega_n^{n-1}\right)^1 & \cdots & \left(\omega_n^{n-1}\right)^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\ a_1 \\ \vdots \\ a_{n-1}
\end{bmatrix}
=
\begin{bmatrix}
F\left(\omega_n^0\right) \\ F\left(\omega_n^1\right) \\ \vdots \\ F\left(\omega_n^{n-1}\right)
\end{bmatrix}
$$

IFFT

快速傅里叶逆变换

变换得矩阵

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} \left(\omega_n^{-0}\right)^0 & \left(\omega_n^{-0}\right)^1 & \cdots & \left(\omega_n^{-0}\right)^{n-1} \\ \left(\omega_n^{-1}\right)^0 & \left(\omega_n^{-1}\right)^1 & \cdots & \left(\omega_n^{-1}\right)^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \left(\omega_n^{-(n-1)}\right)^0 & \left(\omega_n^{-(n-1)}\right)^1 & \cdots & \left(\omega_n^{-(n-1)}\right)^{n-1} \end{bmatrix} \begin{bmatrix} F\left(\omega_n^0\right) \\ F\left(\omega_n^1\right) \\ \vdots \\ F\left(\omega_n^{n-1}\right) \end{bmatrix}$$

所以，只要将 FFT 过程中的 ω_n^k 换为 ω_n^{-k}，然后再做一次 FFT，将所得结果乘以 $\frac{1}{n}$ 即可还原出乘积的系数表示

Faster-er

- Schonhage–Strassen algorithm : $O(n \log n \log \log n)$
- Fürer's algorithm (only for astronomically large values)
- Galactic algorithm (never used in practice)
- Integer multiplication in time $O(n \log n)$ (2019.3)

Acknowledgement

- [TC] Introduction to Algorithms
- Multiplication algorithm
- Karatsuba algorithm
- Toom–Cook multiplication
- OI wiki-FFT

Thanks for your listening!