$O(n^{\log 3})$ 的整数乘法

Fancy

April 15, 2019

$\Theta(n^2)$

- 直接乘。
- $\Theta(n^2)$。

$\Theta(n^2)$

- **考虑分治：令** $m = n/2$，**令** $x = 10^m a + b$, $y = 10^m c + d$，**则** $xy = (10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$。

$\Theta(n^2)$

- **考虑分治**：**令** $m = n/2$，**令** $x = 10^m a + b$, $y = 10^m c + d$，**则** $xy = (10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$。
- **计算复杂度**：$T(n) = 4T(n/2) + O(n)$，**根据主定理，得** $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$。

$\Theta(n^2)$

- **考虑分治**：**令** $m = n/2$，**令** $x = 10^m a + b$, $y = 10^m c + d$，**则** $xy = (10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$。
- **计算复杂度**：$T(n) = 4T(n/2) + O(n)$，**根据主定理，得** $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$。
- 

$\Theta(n^{\log 3})$

- **观察式子** $xy = 10^{2m}ac + 10^m(bc + ad) + bd$，**发现** $bc + ad = ac + bd - (a - b)(c - d)$。

$\Theta(n^{\log 3})$

- 观察式子 $xy = 10^{2m}ac + 10^m(bc + ad) + bd$，发现 $bc + ad = ac + bd - (a - b)(c - d)$。
- 于是 $T(n) = 3T(n/2) + O(n)$，根据主定理，得 $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{\log 3})$。

$\Theta(n^{\log 3})$

- 观察式子 $xy = 10^{2m}ac + 10^m(bc + ad) + bd$，发现 $bc + ad = ac + bd - (a - b)(c - d)$。
- 于是 $T(n) = 3T(n/2) + O(n)$，根据主定理，得 $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{\log 3})$。
- Karatsuba 算法。

$O(n \log n \log \log n)$

- ~~众所周知，有一种毒瘤~~众所周知，快速傅里叶变换算法 FFT 可以计算多项式卷积。
- Schonhage–Strassen 算法。
- 复杂度 $O(n \log n \log \log n)$。

$O(n \log n)$

- 最近 (2019.03) 有两个人 (David Harvey, Joris van der Hoeven) 发了篇论文叫 “Integer multiplication in time $O(n \log n)$”。
- http://www.texmacs.org/joris/nlogn/nlogn.pdf
  https://web.maths.unsw.edu.au/~davidharvey/papers/nlogn

多项式乘法

给定两个多项式 $A(x), B(x)$，其中

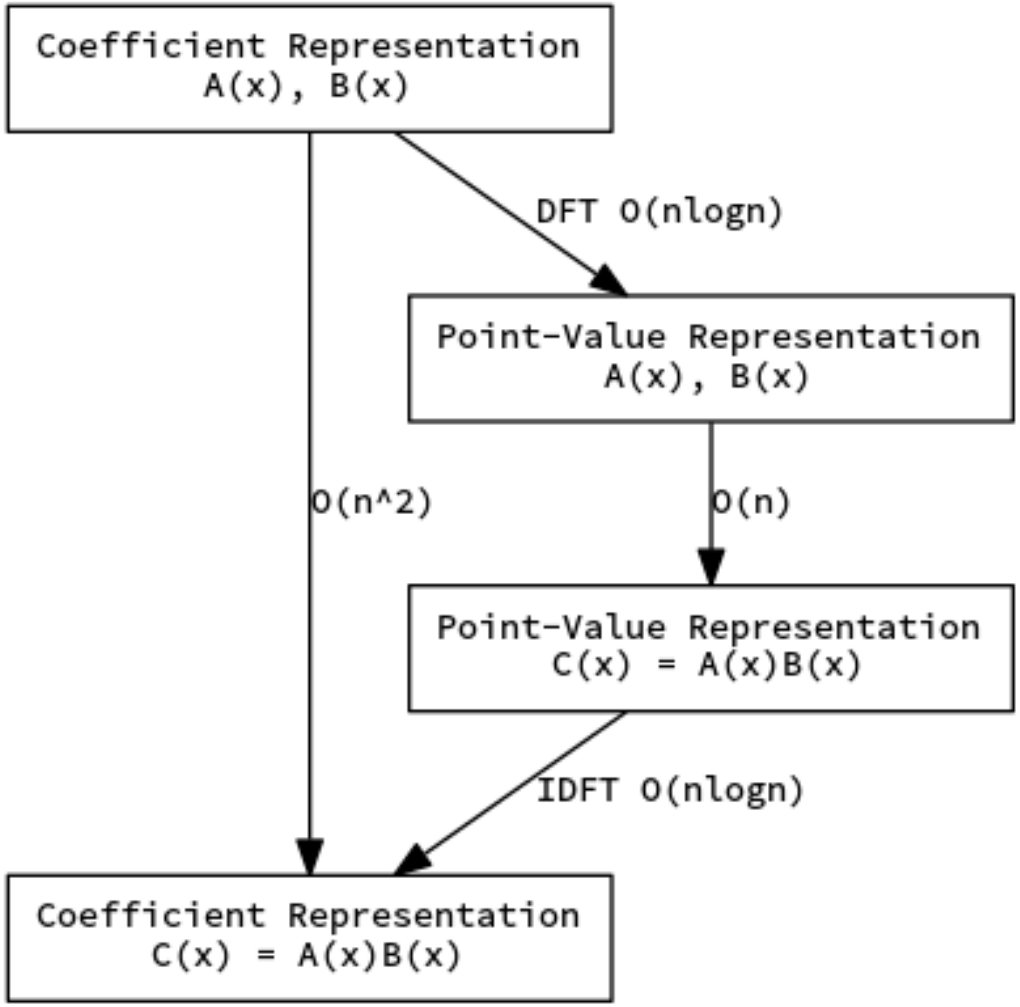
$$A(x) = \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

$$B(x) = \sum_{i=0}^{n} b_i x^i = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_1 x + b_0.$$

将这两个多项式相乘得到 $C(x) = \sum_{i=0}^{2n} c_i x^i$，其中

$$c_i = \sum_{j+k=i, 0 \le j,k \le n} a_j b_k.$$

快速傅里叶变换 (Fast Fourier Transform)



复数单位根

- n 次单位根是指能够满足方程 $z^n = 1$ 的复数，它们把单位圆等分成 n 个部分。

$$e^{\frac{2\pi ki}{n}}, k = 0, 1, 2, \cdots, n-1.$$

- 根据欧拉公式

$$e^{\theta i} = \cos\theta + i\sin\theta$$

就可以知道 n 次单位根的算术表示。

- 如果记 $\omega_n = e^{\frac{2\pi i}{n}}$，那么 n 次单位根就是 $\omega_n^0, \omega_n^1, \cdots, \omega_n^{n-1}$。

DFT

- 假设现在有一个 $n-1$ 次多项式 $A(x)=\sum_{i=0}^{n-1} a_i x^i$。为了方便，假设 $n=2^m, m\in\mathbb{Z}$，如果不足可以在高次项系数补成 0。
- 将 n 个 n 次单位根 $\omega_n^0, \omega_n^1, \cdots, \omega_n^{n-1}$ 带入 $A(x)$ 将其转换成点值表达

$$A(\omega_n^k)=\sum_{i=0}^{n-1} a_i \omega^{ki}, k=0,1,\cdots,n-1.$$

DFT

将每一项按照指数奇偶分类

$$A(\omega_n^k) = \sum_{i=0}^{n-1} a_i \omega_n^{ki} = \sum_{i=0}^{\frac{n}{2}-1} a_{2i} \omega_n^{2ki} + \omega_n^k \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} \omega_n^{2ki}$$

$$= \sum_{i=0}^{\frac{n}{2}-1} a_{2i} \omega_{\frac{n}{2}}^{ki} + \omega_n^k \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} \omega_{\frac{n}{2}}^{ki}$$

$$A(\omega_n^{k+\frac{n}{2}}) = \sum_{i=0}^{\frac{n}{2}-1} a_{2i} \omega_{\frac{n}{2}}^{ki} + \omega_n^{k+\frac{n}{2}} \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} \omega_{\frac{n}{2}}^{ki}$$

$$= \sum_{i=0}^{\frac{n}{2}-1} a_{2i} \omega_{\frac{n}{2}}^{ki} - \omega_n^k \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} \omega_{\frac{n}{2}}^{ki}$$

DFT

- 递归计算 $\sum_{i=0}^{\frac{n}{2}-1} a_{2i}\omega_{\frac{n}{2}}^{ki}$ 和 $\sum_{i=0}^{\frac{n}{2}-1} a_{2i+1}\omega_{\frac{n}{2}}^{ki}$。
- 复杂度 $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$。

IDFT

相当于解线性方程组

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

写成矩阵

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

IDFT

得

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

这样，IDFT 就相当于把 DFT 过程中的 ω_n^i 换成 ω_n^{-i}，然后做一次 DFT，之后结果除以 n 就可以了。

快速数论变换

- Fast Number-Theoretic Transform(FNT / NTT).
- **整数取模意义下，大致理解为把 ω 换成原根。**

- 从多项式乘法到快速傅里叶变换
  http://blog.miskcoo.com/2015/04/polynomial-multiplication-and-fast-fourier-transform
- FFT and Schonhage–Strassen
  https://pdfs.semanticscholar.org/198e/7a6c7bcd2348c4e516857a287
- 乘法算法
  https://en.wikipedia.org/wiki/Multiplication_algorithm
- 谢谢。