基于快速傅里叶变换的大数乘法原理分析

2022 年 1 月 6 日

第一节　引言

对于十进制整数 $a_n*10^n+a_{n-1}*10^{n-1}+\cdots+a_0$，我们可以将数字 10 看作不定元，从而将十进制整数看作一个多项式，这样一来对于十进制数 $a=\sum_{i=0}^{n}a_i*10^i$ 与 $b=\sum_{i=0}^{n}b_i*10^i$（通过补充 0 我们总是可以假定两个多项式次数一样），我们有乘法 $ab=\sum_{k=0}^{2n}\sum_{i+j=k}a_ib_j*10^k$（注意需要处理十进制进位），容易看到这是一个时间复杂度为 $O(n^2)$ 的算法。对于 n 比较大的情况，这样的计算效率不够高，下面我们通过以快速傅里叶变换（FFT）计算多项式乘法的方法来计算大数乘法.

第二节　多项式不同表示方法下的乘法

设多项式 $f(x)=\sum_{i=0}^{n-1}a_ix^i$.

定义 2.1　*将序列 $(a_0,a_1,\cdots,a_n)$ 称为上述多项式的系数表示法.*

而事实上我们知道 n 次多项式上 $n+1$ 个点足够唯一确定这个多项式，所以我们又有另一种多项式的表示方法（此处我们假定取的点数量总是足够多的）.

定义 2.2　*取 n 个点 $x_0,x_1,\cdots,x_n$ 代入多项式，称 $\{(x_0,f(x_0),(x_1,f(x_1),\cdots,(x_n,f(x_n))\}$ 为多项式的点值表示法.*

两个多项式的乘积，在系数表示法下复杂度即为前面所讨论的 $O(n^2)$，而在点值表达法情况却截然不同

性质 2.1　*设 f 与 g 为两个多项式，*

$\{(x_0,f(x_0),(x_1,f(x_1),\cdots,(x_n,f(x_n))\}$ 与 $\{(x_0,g(x_0),(x_1,g(x_1),\cdots,(x_n,g(x_n))\}$ 为他们的点值表示法，则 $\{(x_0,f(x_0)g(x_0),(x_1,f(x_1)g(x_1),\cdots,(x_n,f(x_n)g(x_n))\}$ 为多项式 fg 的点值表示法.

证明　由多项式乘法的定义即可得到.　□

可以看到在新的表示法下，多项式乘法的时间复杂度由 $O(n^2)$ 转变为了 $O(n)$ 的线性复杂度而极大简化，于是大数乘法的问题转化为如何高效地将多项式变换为点值表示法以

及如何高效地将点值表示法逆变换为多项式. 为此我们先做一些准备工作，由于我们总是可以将多项式补充 0 使得次数为 2 的幂次，以下我们总是假定 n 为 2 的幂次.

第三节　单位根的部分性质

由于 FFT 利用了单位根的部分性质，我们在此重述一下.

性质 3.1　记 $\omega_n^1 = e^{\frac{2\pi i}{n}}$ 为 n 次单位根

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

证明　我们有

$\omega_{2n}^{2k} = e^{\frac{2\pi * 2i}{2n}} = e^{\frac{2\pi * i}{n}} = \omega_n^k$.

$\omega_n^{k+\frac{n}{2}} = \omega_n^k \omega_n^{\frac{n}{2}} = -\omega_n^k$. □

第四节　离散傅里叶变换

快速傅里叶变换（FFT）实际上是通过单位根性质以及分治思想对离散傅里叶变换（DFT）的优化，在此我们先讨论多项式取单位根时的离散傅里叶变换（DFT）与离散傅里叶逆变换 (IDFT).

设多项式 $f(x) = \sum_{i=0}^{n-1} a_i x^i$，记 $f_k = f(\omega_n^k) = \sum_{i=0}^{n-1} a_i w_n^{ki}, 0 \le k < n$ 为多项式的 DFT.

由单位根的循环性质，我们可以将上式表达为由单位根构成的 Vandermonde 矩阵的形式

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ 1 & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}.$$

最终我们还需要将点值表示法变换为多项式，所以我们还需要考虑其逆变换，事实上根据如下结论，我们知道 DFT 和 IDFT 的过程是极其相似的

定理 4.1　变换 $f_k = \sum_{i=0}^{n-1} a_i w_n^{ki}$ 的逆变换为 $f_k = \frac{1}{n} \sum_{i=0}^{n-1} a_i w_n^{-ki}$.

证明 实际上我们只需要求出上面 Vandermonde 矩阵的逆矩阵即可. 我们记

$$W=\begin{pmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1}\\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)}\\ 1 & \vdots & \vdots & & \vdots\\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)}\end{pmatrix}.$$

考虑其共轭转置矩阵

$$W^H=\begin{pmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)}\\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)}\\ 1 & \vdots & \vdots & & \vdots\\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)}\end{pmatrix}.$$

两者乘积我们有

$$\begin{aligned}(WW^H)_{ij}&=\begin{pmatrix}1 & \omega_n^{i-1} & \omega_n^{2(i-1)} & \cdots & \omega_n^{(n-1)(i-1)}\end{pmatrix}\begin{pmatrix}1\\ \omega_n^{-(j-1)}\\ \omega_n^{-2(j-1)}\\ \vdots\\ \omega_n^{-(n-1)(j-1)}\end{pmatrix}\\ &=1+\omega_n^{i-j}+\omega_n^{2(i-j)}+\cdots+\omega_n^{(n-1)(i-j)}=\begin{cases}n, & n\mid i-j \iff j=i\\ 0, & n\nmid i-j \iff j\neq i.\end{cases}\end{aligned}$$

可以看到 WW^H 是一个对角线全为 n 的对角矩阵，因此 $W^{-1}=\frac{1}{n}W^H$，立即得到我们所需的逆变换. □

第五节　快速离散傅里叶变换与大数乘法

按照 DFT 的计算方法,若是我们将单位根依次代入计算,对于多项式 $f(x)=\sum_{i=0}^{n-1}a_ix^i$,其时间复杂度依然是 $O(n^2)$，并没有使计算量降低，同时因为单位根计算带来的大量浮点数运算，在计算机上可能反而比直接的多项式乘法更慢.

但是我们可以考虑采用如下的分治（Divide & Conquer）策略计算

先将 f 按奇数下标和偶数下标分为两个多项式 f_1 和 f_2，其中

$$f_1(x)=a_0+a_2x+\cdots+a_{n-2}x^{\frac{n}{2}-1}$$

$$f_2(x)=a_1+a_3x+\cdots+a_{n-1}x^{\frac{n}{2}-1}.$$

则有 $f(x)=f_1(x^2)+xf_2(x^2)$.

为了实施分治策略，我们需要将问题规模缩小，设 $k < \frac{n}{2}$，

先计算指数部分于 $[0, \frac{n}{2} - 1]$ 的点值表示

$$f(\omega_n^k) = f_1(\omega_n^{2k}) + \omega_n^k f_2(\omega_n^{2k}) = f_1(\omega_{\frac{n}{2}}^k) + \omega_n^k f_2(\omega_{\frac{n}{2}}^k).$$

再计算指数部分于 $[\frac{n}{2}, n-1]$ 的点值表示

$$f(\omega_n^{k+\frac{n}{2}}) = f_1(\omega_n^{2k+n}) + \omega_n^{k+\frac{n}{2}} f_2(\omega_n^{2k+n}) = f_1(\omega_{\frac{n}{2}}^k) - \omega_n^k f_2(\omega_{\frac{n}{2}}^k).$$

根据上面两部分公式，若是 $(\omega_{\frac{n}{2}}^1 \omega_{\frac{n}{2}}^2 \cdots \omega_{\frac{n}{2}}^{\frac{n}{2}-1})$ 的点值表示已经求得，则在当前步骤只需要 $O(n)$ 的时间复杂度即可求得 $(\omega_n^1 \omega_n^2 \cdots \omega_n^{n-1})$ 处的点值表示.

注意到每一步骤问题规模已由 n 减为 $\frac{n}{2}$，通过同样的办法不断向下递归分治即可计算出结果. 我们有如下的时间复杂度递推式

$$T(n) = \begin{cases} O(n) + 2T(\frac{n}{2}), & n \geq 1 \\ O(1), & n = 1. \end{cases}$$

简单计算可知整体时间复杂度为 $O(n \log_2 n)$.

对于 FFT 的逆变换 IFFT 将多项式点值表示法变换为系数表示法，由 DFT 一节的分析，我们基本可以通过相同的 FFT 流程进行计算，所不同的地方只是在于我们取的是单位根的倒数并且结果乘以系数 $\frac{1}{n}$.

由于每个十进制整数总是对应于一个多项式，所以我们先通过 FFT 将两个大整数转化为点值表示法，复杂度 $O(n \log_2 n)$；再将两个点值表示法作乘法，复杂度 $O(n)$；最后将结果的点值表示法通过 IFFT 转化为系数表示法并处理进位，复杂度 $O(n \log_2 n)$. 因此整体大数乘法的时间复杂度就从原来的 $O(n^2)$ 降低为 $O(n \log_2 n)$.

注记 5.1 事实上我们有一个将多项式扩充为 2 的幂次的过程，此时我们设多项式是 k 次的，则我们将多项式通过补充 0 扩充为 $2^{\lceil \log_2 k \rceil}$ 次，并不会改变我们的时间复杂度.