有限体上での
離散フーリエ変換による
高速乗算

芝浦工業大学　数理科学研究会

平成 25 年 11 月 2 日

※何か不明な点や計算ミス等がありましたら加筆修正しますので指摘をお願いします

制作：数理科学科2年 深谷 徹

目 次

1　研究背景

N 桁の多倍長乗算を行う際, 古典的乗算法の計算量は $O(N^2)$ であるが, 高速フーリエ変換を用いると, 計算量が $O(N \log N)$ で済む. 高速フーリエ変換は通常複素数で計算することが多いが, 有限体 $\mathbb{F}_p$ 上でも計算できると知り, 興味が湧いたので研究するに至った.

2　他の多倍長乗算法

2.1　古典的算法

古典的乗算法は, いわゆる掛け算の筆算のことで, 最も基本的な多倍長乗算法である. しかし, n 桁同士の乗算をするときの計算量は $O(n^2)$ である.

例.

$$
\begin{array}{rr}
 & 1234 \\
\times) & 5678 \\
\hline
 & 9872 \\
 & 8638 \\
 & 7404 \\
 & 6170 \\
\hline
 & 7006652
\end{array}
$$

2.2　Karatsuba 法

$A = a_1 r + a_0,\ B = b_1 r + b_0$ (r は基数) と表わされる $A,\ B$ の積を求める際, 古典的乗算法では $a_1 b_1 r^2 + (a_0 b_1 + a_1 b_0) r + a_0 b_0$ と計算する. ここで掛け算を 4 回していることに注意する.
これを Karatsuba 法では $a_1 b_1 r^2 + (a_1 b_1 + a_0 b_0 - (a_1 - a_0)(b_1 - b_0)) r + a_0 b_0$ と計算する. 一見複雑になったように見えるが, $a_1 b_1,\ a_0 b_0$ が 2 回使われているので実質掛け算は 3 回で済む.
よって, $N * N$ の掛け算を $(N/2) * (N/2)$ の掛け算に分割すると通常 4 回になるが, Karatsuba 法を使うと 3 回で済ませられる事が分かる.

サイズが 2^K の掛け算を古典的乗算法で計算すると, $\left(2^K\right)^2 = 2^{2K}$ 回程度の計算回数が必要となるが, Karatsuba 法を 1 回使うと $3 \cdot \left(2^{K-1}\right)^2 = 3 \cdot 2^{2(K-1)}$ 回程度の計算回数で済む. もう一回使うと, $3 \cdot (3 \cdot (2^{K-2})^2) = 3^2 \cdot 2^{2(K-2)}$ 回程度になり, さらにもう一回使うと, $3^2 \cdot (3 \cdot (2^{K-3})^2) = 3^3 \cdot 2^{2(K-3)}$ 回程度になる. 従って, 合計 K 回使うと, $3^K \cdot 2^{2(K-K)} = 3^K$ 回程度の計算回数で済む.
よって, Karatsuba 法を繰り返し適用すると計算量は $O(n^{\log_2 3})$ となる.

例. $A = 12 * 100 + 34,\ B = 56 * 100 + 78.$
$12 * 56 = 672,\ 34 * 78 = 2652,\ (12 - 34) * (56 - 78) = 484,$
$672 + 2652 - 484 = 2840,$
$672 * 10000 + 2840 * 100 + 2652 = 7006652.$

3　離散フーリエ変換

3.1　離散フーリエ変換

定義 3.1 (離散フーリエ変換)
N 次の変換元ベクトル $\boldsymbol{x}$, 変換後のベクトル $\boldsymbol{y}$ に対しての離散フーリエ変換は 1 の原始 N 乗根[1] ζ_N, N 次正方行列 A $(a_{ij} = \zeta_N^{ij},\ 0 \leq i \leq N-1,\ 0 \leq j \leq N-1)$, を用いて以下のように表せる.

$$\boldsymbol{y} = A\boldsymbol{x}.$$

また, 逆離散フーリエ変換も N 次正方行列 B $(b_{ij} = \zeta_N^{-ij})$, を用いて以下のように表せる.

$$\boldsymbol{x} = \frac{1}{N}B\boldsymbol{y}.$$

例として $N = 8$ のときの A は,

$$A = \begin{pmatrix} \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 \\ \zeta_N^0 & \zeta_N^1 & \zeta_N^2 & \zeta_N^3 & \zeta_N^4 & \zeta_N^5 & \zeta_N^6 & \zeta_N^7 \\ \zeta_N^0 & \zeta_N^2 & \zeta_N^4 & \zeta_N^6 & \zeta_N^8 & \zeta_N^{10} & \zeta_N^{12} & \zeta_N^{14} \\ \zeta_N^0 & \zeta_N^3 & \zeta_N^6 & \zeta_N^9 & \zeta_N^{12} & \zeta_N^{15} & \zeta_N^{18} & \zeta_N^{21} \\ \zeta_N^0 & \zeta_N^4 & \zeta_N^8 & \zeta_N^{12} & \zeta_N^{16} & \zeta_N^{20} & \zeta_N^{24} & \zeta_N^{28} \\ \zeta_N^0 & \zeta_N^5 & \zeta_N^{10} & \zeta_N^{15} & \zeta_N^{20} & \zeta_N^{25} & \zeta_N^{30} & \zeta_N^{35} \\ \zeta_N^0 & \zeta_N^6 & \zeta_N^{12} & \zeta_N^{18} & \zeta_N^{24} & \zeta_N^{30} & \zeta_N^{36} & \zeta_N^{42} \\ \zeta_N^0 & \zeta_N^7 & \zeta_N^{14} & \zeta_N^{21} & \zeta_N^{28} & \zeta_N^{35} & \zeta_N^{42} & \zeta_N^{49} \end{pmatrix} = \begin{pmatrix} \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 & \zeta_N^0 \\ \zeta_N^0 & \zeta_N^1 & \zeta_N^2 & \zeta_N^3 & \zeta_N^4 & \zeta_N^5 & \zeta_N^6 & \zeta_N^7 \\ \zeta_N^0 & \zeta_N^2 & \zeta_N^4 & \zeta_N^6 & \zeta_N^0 & \zeta_N^2 & \zeta_N^4 & \zeta_N^6 \\ \zeta_N^0 & \zeta_N^3 & \zeta_N^6 & \zeta_N^1 & \zeta_N^4 & \zeta_N^7 & \zeta_N^2 & \zeta_N^5 \\ \zeta_N^0 & \zeta_N^4 & \zeta_N^0 & \zeta_N^4 & \zeta_N^0 & \zeta_N^4 & \zeta_N^0 & \zeta_N^4 \\ \zeta_N^0 & \zeta_N^5 & \zeta_N^2 & \zeta_N^7 & \zeta_N^4 & \zeta_N^1 & \zeta_N^6 & \zeta_N^3 \\ \zeta_N^0 & \zeta_N^6 & \zeta_N^4 & \zeta_N^2 & \zeta_N^0 & \zeta_N^6 & \zeta_N^4 & \zeta_N^2 \\ \zeta_N^0 & \zeta_N^7 & \zeta_N^6 & \zeta_N^5 & \zeta_N^4 & \zeta_N^3 & \zeta_N^2 & \zeta_N^1 \end{pmatrix}$$

となる.

ここで離散フーリエ変換をしてさらに逆離散フーリエ変換をすると元に戻ることを示す. つまり, $\frac{1}{N}BA = I$ を示せば良い.

証明. まず,

$$\sum_{k=0}^{N-1} \zeta_N^{ak} = \begin{cases} 0 & (a \not\equiv 0 \pmod N) \\ N & (a \equiv 0 \pmod N) \end{cases}$$

を示す.
$a \not\equiv 0 \pmod N$ のとき
$X = \sum_{k=0}^{N-1} \zeta_N^{ak}$ とおく.

$$\begin{aligned} X = \sum_{k=0}^{N-1} \zeta_N^{ak} &= \zeta_N^{a\cdot 0} + \zeta_N^{a\cdot 1} + \cdots + \zeta_N^{a\cdot(N-2)} + \zeta_N^{a\cdot(N-1)} \\ &= \zeta_N^{a\cdot N} + \zeta_N^{a\cdot 1} + \cdots + \zeta_N^{a\cdot(N-2)} + \zeta_N^{a\cdot(N-1)} \ (\because \zeta_N^N = \zeta_N^0 = 1.) \\ &= \zeta_N^{a\cdot 1} + \cdots + \zeta_N^{a\cdot(N-2)} + \zeta_N^{a\cdot(N-1)} + \zeta_N^{a\cdot N} \\ &= \sum_{k=1}^{N} \zeta_N^{ak} = \zeta_N^a \sum_{k=1}^{N} \zeta_N^{a(k-1)} = \zeta_N^a \sum_{k=0}^{N-1} \zeta_N^{ak} = \zeta_N^a X. \end{aligned}$$

以上より, $X = \zeta_N^a X$ となる. また, $a \not\equiv 0 \pmod N$ より $\zeta_N^a \neq 1$ なので $X = 0$. よって, $\sum_{k=0}^{N-1} \zeta_N^{ak} = 0$ ($a \not\equiv 0 \pmod N$).

$a \equiv 0 \pmod N$ のとき

$\zeta_N^a = 1$ なので, $\sum_{k=0}^{N-1} \zeta_N^{ak} = \sum_{k=0}^{N-1} 1 = N$.

以上より,

$$\sum_{k=0}^{N-1} \zeta_N^{ak} = \begin{cases} 0 & (a \not\equiv 0 \pmod N) \\ N & (a \equiv 0 \pmod N) \end{cases}$$

$C = \frac{1}{N}BA$ とおくと,

$$c_{ij} = \frac{1}{N}\sum_{k=0}^{N-1} b_{ik}a_{kj} = \frac{1}{N}\sum_{k=0}^{N-1} \zeta_N^{-ik}\zeta_N^{kj} = \frac{1}{N}\sum_{k=0}^{N-1} \zeta_N^{(j-i)k}.$$

$i \neq j$ のとき, $j - i \not\equiv 0 \pmod N$ なので, 先に示したことより $c_{ij} = 0$.

$i = j$ のとき, $j - i \equiv 0 \pmod N$ なので, 先に示したことより $c_{ij} = 1$.

よって C は単位行列. 以上より $\frac{1}{N}BA = I$. □

ここで A^2 について考える. $C = A^2$ とおくと,

$$c_{ij} = \sum_{k=0}^{N-1} a_{ik}a_{kj} = \sum_{k=0}^{N-1} \zeta_N^{ik}\zeta_N^{kj} = \sum_{k=0}^{N-1} \zeta_N^{(i+j)k}.$$

$i + j \equiv 0 \pmod N$ のとき $c_{ij} = N$. $i + j \not\equiv 0 \pmod N$ のとき $c_{ij} = 0$. つまり, 以下のような行列となる.

$$\begin{pmatrix} N & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & N \\ \vdots & & \iddots & \iddots & 0 \\ \vdots & \iddots & \iddots & \iddots & \vdots \\ 0 & N & 0 & \cdots & 0 \end{pmatrix}$$

$\frac{1}{N}A^2$ をベクトルにかけた時のことを考える.

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \\ \vdots & & \iddots & \iddots & 0 \\ \vdots & \iddots & \iddots & \iddots & \vdots \\ 0 & 1 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} x_0 \\ x_{n-1} \\ x_{n-2} \\ \vdots \\ x_1 \end{pmatrix}$$

となり, $k \geq 1$ では x_k と x_{n-k} が入れ替わっている. また, $\left(\frac{1}{N}A^2\right)^2 = \frac{1}{N^2}A^4$ が単位行列になることも $\frac{1}{N}A^2$ の行列の意味より容易に分かる.

3.2 離散フーリエ変換と多倍長乗算

ある数 X, Y, Z が $\{x_n\}$, $\{y_n\}$, $\{z_n\}$, $r \geq 2$ ($r \in \mathbb{N}$) で

$$X = \sum_{k=0}^{2N-1} x_k r^k,\ Y = \sum_{k=0}^{2N-1} y_k r^k,\ Z = \sum_{k=0}^{2N-1} z_k r^k$$

と表されたとする. ここで, $x_n = 0,\ y_n = 0\ (n \geq N)$ かつ $Z = X \cdot Y$ とすると,

$$
\begin{aligned}
Z &= X \cdot Y \\
&= \left(\sum_{k=0}^{2N-1} x_k r^k \right) \left(\sum_{k=0}^{2N-1} y_k r^k \right) \\
&= \left(\sum_{k=0}^{N-1} x_k r^k \right) \left(\sum_{k=0}^{N-1} y_k r^k \right) \ (\because x_n = 0,\ y_n = 0\ (n \geq N)) \\
&= x_0 y_0 \cdot r^0 + (x_0 y_1 + x_1 y_0) r^1 + \cdots + (x_{N-1} y_0 + x_{N-2} y_1 + \cdots + x_1 y_{N-2} + x_0 y_{N-1}) r^{N-1} \\
&\quad + \cdots + (x_{N-2} y_{N-1} + x_{N-1} y_{N-2}) r^{2(N-1)-1} + x_{N-1} y_{N-1} \cdot r^{2(N-1)} \\
&= \sum_{k=0}^{2(N-1)} \left(\sum_{m=\max(0,\ k-(N-1))}^{\min(k,\ N-1)} x_m y_{k-m} \right) r^k
\end{aligned}
$$

となる. また,

$$
Z = \sum_{k=0}^{2N-1} z_k r^k
$$

より,

$$
\sum_{k=0}^{2N-1} z_k r^k = \sum_{k=0}^{2(N-1)} \left(\sum_{m=\max(0,\ k-(N-1))}^{\min(k,\ N-1)} x_m y_{k-m} \right) r^k
$$

となる. ここで, r^k の係数を比較すると,

$$
\begin{aligned}
z_k &= \sum_{m=\max(0,\ k-(N-1))}^{\min(k,\ N-1)} x_m y_{k-m},\ (0 \leq k \leq 2(N-1)) \\
z_{2N-1} &= 0.
\end{aligned}
$$

ここで, 同じ $X,\ Y,\ Z$ に対して,

$$
F_x = \begin{pmatrix} x_1^{(f)} \\ x_2^{(f)} \\ \vdots \\ x_{2N-1}^{(f)} \end{pmatrix} = A \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{2N-1} \end{pmatrix},\ F_y = \begin{pmatrix} y_1^{(f)} \\ y_2^{(f)} \\ \vdots \\ y_{2N-1}^{(f)} \end{pmatrix} = A \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{2N-1} \end{pmatrix},
$$

$$
F_w = \begin{pmatrix} w_1^{(f)} \\ w_2^{(f)} \\ \vdots \\ w_{2N-1}^{(f)} \end{pmatrix} = \begin{pmatrix} x_1^{(f)} y_1^{(f)} \\ x_2^{(f)} y_2^{(f)} \\ \vdots \\ x_{2N-1}^{(f)} y_{2N-1}^{(f)} \end{pmatrix},\ \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_{2N-1} \end{pmatrix} = \frac{1}{2N} B \begin{pmatrix} w_1^{(f)} \\ w_2^{(f)} \\ \vdots \\ w_{2N-1}^{(f)} \end{pmatrix}
$$

($A,\ B$ は定義 3.1 で定義したもの)
とおくと, $w_k = z_k$ となっていることつまり, フーリエ変換した $X,\ Y$ の係数を要素ごとにかけて逆フーリエ変換をすると普通に掛け算したのと同じ結果になることを示す.

証明. 仮定より,

$$x_k^{(f)} = \sum_{m=0}^{2N-1} \zeta_{2N}^{mk} x_m = \sum_{m=0}^{N-1} \zeta_{2N}^{mk} x_m,$$
$$y_k^{(f)} = \sum_{m=0}^{2N-1} \zeta_{2N}^{mk} y_m = \sum_{m=0}^{N-1} \zeta_{2N}^{mk} y_m.$$
$$(\because x_n = 0,\ y_n = 0\ (n \geq N))$$

$w_k^{(f)} = x_k^{(f)} y_k^{(f)}$ より,

$$\begin{aligned}
w_k^{(f)} &= x_k^{(f)} y_k^{(f)} \\
&= \left(\sum_{m=0}^{N-1} \zeta_{2N}^{mk} x_k \right) \left(\sum_{m=0}^{N-1} \zeta_{2N}^{mk} y_k \right) \\
&= x_0 y_0 \zeta_{2N}^{0} + (x_0 y_1 + x_1 y_0) \zeta_{2N}^{k} + \cdots + (x_0 y_{N-1} + \cdots + x_{N-1} y_0) \zeta_{2N}^{(N-1)k} \\
&\quad + \cdots + (x_{N-2} y_{N-1} + x_{N-1} y_{N-2}) \zeta_{2N}^{(2N-3)k} + x_{N-1} y_{N-1} \zeta_{2N}^{(2N-2)k} \\
&= \sum_{m=0}^{2N-2} \zeta_{2N}^{mk} \sum_{i=\max(0,\ m-(N-1))}^{\min(m,\ N-1)} x_i y_{m-i}
\end{aligned}$$

となる. ここで,

$$z_k = \sum_{m=\max(0,\ k-(N-1))}^{\min(k,\ N-1)} x_m y_{k-m},\ (0 \leq k \leq 2(N-1))$$

より

$$w_k^{(f)} = \sum_{m=0}^{2N-2} \zeta_{2N}^{mk} z_m$$

となる.

$$\begin{aligned}
w_j &= \frac{1}{2N} \sum_{k=0}^{2N-1} \zeta_{2N}^{-jk} w_k^{(f)} \\
&= \frac{1}{2N} \sum_{k=0}^{2N-1} \zeta_{2N}^{-jk} \sum_{m=0}^{2N-2} \zeta_{2N}^{mk} z_m \\
&= \frac{1}{2N} \sum_{k=0}^{2N-1} \sum_{m=0}^{2N-2} \zeta_{2N}^{(m-j)k} z_m \\
&= \frac{1}{2N} \sum_{m=0}^{2N-2} \sum_{k=0}^{2N-1} \zeta_{2N}^{(m-j)k} z_m \ (\because \text{有限和なので順序交換可能}) \\
&= \frac{1}{2N} \sum_{m=0}^{2N-2} z_m \sum_{k=0}^{2N-1} \zeta_{2N}^{(m-j)k} \\
&= \frac{1}{2N} \sum_{m=0}^{2N-2} z_m \cdot 2N \delta_{mj} \quad \left(\because \sum_{k=0}^{N-1} \zeta_{2N}^{ak} = 0\ (a \not\equiv 0 \pmod{2N}) \right) \\
&= \sum_{m=0}^{2N-2} z_m \cdot \delta_{mj} \\
&= z_j\ (z_{2N-1} = 0 \text{ より } j = 2N-1 \text{ でも成立.})
\end{aligned}$$

よって, $w_j = z_j$. □

以上より, フーリエ変換した X, Y の係数を要素ごとにかけて逆フーリエ変換をすると普通に掛け算したのと同じ結果になることが示された.

例. 1234×5678 を計算する.

$X = 12 \cdot 100^2 + 34 \cdot 10^0$, $Y = 56 \cdot 100^2 + 78 \cdot 10^0$ と表せるので,

$N = 2$, $\zeta_4 = -i$ とすると,

$$F_x = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \begin{pmatrix} 34 \\ 12 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 46 \\ 34 - 12i \\ 22 \\ 34 + 12i \end{pmatrix}$$

$$F_y = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \begin{pmatrix} 78 \\ 56 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 134 \\ 78 - 56i \\ 22 \\ 78 + 56i \end{pmatrix}$$

$$F_w = \begin{pmatrix} 46 \cdot 134 \\ (34 - 12i) \cdot (78 - 56i) \\ 22 \cdot 22 \\ (34 + 12i) \cdot (78 + 56i) \end{pmatrix} = \begin{pmatrix} 6164 \\ 1980 - 2840i \\ 484 \\ 1980 + 2840i \end{pmatrix}$$

$$W = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 6164 \\ 1980 - 2840i \\ 484 \\ 1980 + 2840i \end{pmatrix} = \begin{pmatrix} 2652 \\ 2840 \\ 672 \\ 0 \end{pmatrix}$$

よって, $672 \cdot 100^2 + 2840 \cdot 100^1 + 2652 \cdot 100^0 = 7006652$ と計算できる.

3.3 離散フーリエ変換から高速フーリエ変換へ

前の節で離散フーリエ変換を用いて多倍長乗算が出来ることを示した. しかし, 古典的乗算法の何倍もの計算を必要とするのでこのままでは実用的でない. そこで, 離散フーリエ変換をする行列の規則性を利用して離散フーリエ変換の計算量を大幅に減らす方法を用いる. この離散フーリエ変換を高速に計算するアルゴリズムを高速フーリエ変換という. 高速フーリエ変換のアルゴリズムには様々なものがあるが, ここでは Cooley-Tukey のアルゴリズムを用いる. 以下, 特に断らない限り $N = 2^K$ $(K \in \mathbb{N})$ とする.

数列 $\{x_n\}$ $(0 \leq n \leq N-1)$ をフーリエ変換したものを $\{x_n^{(f)}\}$ $(0 \leq n \leq N-1)$ とすると,

$$\begin{aligned} x_i^{(f)} &= \sum_{k=0}^{N-1} x_k \zeta_N^{ik} = \sum_{k=0}^{N/2-1} x_{2k} \zeta_N^{2ki} + \sum_{k=0}^{N/2-1} x_{2k+1} \zeta_N^{(2k+1)i} \\ &= \sum_{k=0}^{N/2-1} x_{2k} \zeta_N^{2ki} + \zeta_N^i \sum_{k=0}^{N/2-1} x_{2k+1} \zeta_N^{2ki} \\ &= \left(\sum_{k=0}^{N/2-1} x_{2k} \zeta_{N/2}^{ki} \right) + \zeta_N^i \left(\sum_{k=0}^{N/2-1} x_{2k+1} \zeta_{N/2}^{ki} \right) \end{aligned}$$

上の括弧の中身は長さが半分の離散フーリエ変換になっている. また, 前半部と後半部の足し合わせは N 回程度の計算で済む. 定義通りに計算すると N^2 回程度の計算回数が必要となるが, 長さが半分の離散フーリエ変換は $(N/2)^2 = N^2/4$ 回程度の計算回数で済む. これが 2 つあるが, 合わせても元の半分程度の計算

量で計算できる. さらに分割することを考えると

$$
\begin{aligned}
x_i^{(f)} &= \left(\sum_{k=0}^{N/2-1} x_{2k}\zeta_{N/2}^{ki}\right) + \zeta_N^i \left(\sum_{k=0}^{N/2-1} x_{2k+1}\zeta_{N/2}^{ki}\right) \\
&= \left(\sum_{k=0}^{N/4-1} x_{4k}\zeta_{N/2}^{2ki} + \sum_{k=0}^{N/4-1} x_{4k+2}\zeta_{N/2}^{(2k+1)i}\right) + \zeta_N^i \left(\sum_{k=0}^{N/4-1} x_{4k+1}\zeta_{N/2}^{2ki} + \sum_{k=0}^{N/4-1} x_{4k+3}\zeta_{N/2}^{(2k+1)i}\right) \\
&= \left(\sum_{k=0}^{N/4-1} x_{4k}\zeta_{N/2}^{2ki} + \zeta_{N/2}^i \sum_{k=0}^{N/4-1} x_{4k+2}\zeta_{N/2}^{2ki}\right) + \zeta_N^i \left(\sum_{k=0}^{N/4-1} x_{4k+1}\zeta_{N/2}^{2ki} + \zeta_{N/2}^i \sum_{k=0}^{N/4-1} x_{4k+3}\zeta_{N/2}^{2ki}\right) \\
&= \left(\sum_{k=0}^{N/4-1} x_{4k}\zeta_{N/4}^{ki} + \zeta_{N/2}^i \sum_{k=0}^{N/4-1} x_{4k+2}\zeta_{N/4}^{ki}\right) + \zeta_N^i \left(\sum_{k=0}^{N/4-1} x_{4k+1}\zeta_{N/4}^{ki} + \zeta_{N/2}^i \sum_{k=0}^{N/4-1} x_{4k+3}\zeta_{N/4}^{ki}\right) \\
&= \left(\left(\sum_{k=0}^{N/4-1} x_{4k}\zeta_{N/4}^{ki}\right) + \zeta_{N/2}^i \left(\sum_{k=0}^{N/4-1} x_{4k+2}\zeta_{N/4}^{ki}\right)\right) \\
&\quad + \zeta_N^i \left(\left(\sum_{k=0}^{N/4-1} x_{4k+1}\zeta_{N/4}^{ki}\right) + \zeta_{N/2}^i \left(\sum_{k=0}^{N/4-1} x_{4k+3}\zeta_{N/4}^{ki}\right)\right)
\end{aligned}
$$

となり, 偶数番目をさらに分割する際 4 で割った余りが 0 のものと 2 のものに分かれることが分かる. また, 奇数番目も同様に 4 で割った余りがが 1 のものと 3 のものに分かれる. 先ほどと同様に計算量を考えると $4\cdot(N/4)^2+2N = N^2/4+2N$ とさらに約半分になっていることが分かる. これを K 回繰り返すと長さ N の離散フーリエ変換が $2^K\cdot(N/2^K)^2+KN = N\cdot 1^2+KN = (K+1)N$ 回程度, つまり $N(\log_2 N+1)$ 回程度の計算回数で行える. 長さ N の離散フーリエ変換を計算する際, N^2 回から $N(\log_2 N+1)$ 回になると実際どの程度計算回数が減るのかというと,
$N = 2^{10}$ では通常の方法約 1 百万回に対して, 約 1 万回.
$N = 2^{20}$ では通常の方法約 1 兆回に対して, 約 2 千万回.
というように N が大きいほど高速フーリエ変換は有利になる.

高速フーリエ変換と先ほど示した離散フーリエ変換を用いた多倍長乗算を組み合わせると, 高速フーリエ変換 3 回と要素ごとの積を計算すれば多倍長乗算が出来ることが分かる. このときの計算量は, $3\cdot N(\log_2 N+1)+N = N(3\log_2 N+4)$ となる.

方法	計算量	$N=2^{10}$ のときの計算回数	$N=2^{20}$ のときの計算回数
古典的乗算法	N^2	約 104 万回	約 1 兆回
Karatsuba 法	$N^{\log_2 3}$	約 6 万回	約 35 億回
離散フーリエ変換	$3N^2+N$	約 315 万回	約 3 兆回
高速フーリエ変換	$N(3\log_2 N+4)$	約 3 万回	約 6711 万回

表 1: 計算量まとめ

今まで紹介した方法の計算量をまとめると上の表 1 のようになる. これより, 高速フーリエ変換が最も計算量が少ないことが分かる. また, 計算しているのはどれも同じ桁数の長い掛け算であるが, 計算方法によってここまで計算量の差が出てくるということについても是非覚えていて欲しいと思う.

4　複素数でのフーリエ変換

複素数の範囲での 1 の原始 N 乗根には $e^{-2\pi i/N} = \cos\left(-\frac{2\pi}{N}\right) + i\sin\left(-\frac{2\pi}{N}\right)$ などが主に用いられる．また, 複素数は倍精度浮動小数を 2 つ組み合わせることで実現しているため．計算誤差の問題がある．よって, 最大でどの程度誤差が出るのかを調べる必要があるが, 今回は省略した．

5　有限体でのフーリエ変換

整数を素数 p で割った余りを考えると, 整数を p 個の集合に分類することが出来る．ある p に対して $\overline{a}$ を p で割った余りが a と等しいものと定義する．つまり, $\overline{a} := \{x \in \mathbb{Z} \mid x \equiv a \pmod{p}\}$ とする．また, 分類されたもの同士の和や積はその元に対して (属していればどの元をとってきても良い) の和や積で定義すると, 矛盾なく定義できる．また, 1 次合同式の理論より $a \equiv 0 \pmod{p}$ でない任意の整数 $a,\ b$ に対して, $ax \equiv b \pmod{p}$ となるような整数 x が存在する．つまり 0 以外の数での “割り算” が出来る事を意味する．また, 初等整数論の定理によれば $p-1$ 乗して初めて $\overline{1}$ になる $\overline{a}$ が存在する．

ここで, $\mathbb{F}_p$ を $\mathbb{F}_p = \{\overline{0},\ \overline{1}, \ldots,\ \overline{p-1}\}$ と定義する．ここで, 初等整数論の定理を紹介する．

定理 5.1
$\omega \in \mathbb{Z}$ が法 p において原始 n 乗根であることと次は同値である．

$$\forall k \in \mathbb{Z},\ \omega^k \equiv 1 \pmod{p} \Leftrightarrow k \equiv 0 \pmod{n}.$$

さて, 離散フーリエ変換をするにあたり 1 の原始 2^k 乗根を求める必要がある．また, p は r を基数として $2^k(r-1)^2$ より大きい必要がある．なぜなら, 基数 r の 1 桁同士の積は高々$(r-1)^2$ であり, 最大で 2^k 個足されるから桁上げの処理をする前では高々$2^k(r-1)^2$ の値が出てくる．なので p がこれ以下の数であると, 係数が a のとき本当に a なのか $a+p$ という値なのかが分からなくなってしまう．よって, $p > 2^k(r-1)^2$ である必要がある．そこで, $p = n2^k + 1\ (n \geq (r-1)^2)$ とする．(このような素数は無限に存在する) この p における原始 2^k 乗根 ω を求めたい．しかし, 定義通りに $\omega^1,\ \omega^2, \ldots, \omega^{2^k-1}$ が 1 に合同でなく, $\omega^{2^k} \equiv 1$ であることを確認すると, 多大な労力を要する．また, ω が原始 2^k 乗根ならば $\omega^{2^{k-1}} \equiv -1 \pmod{N}$ ということは簡単に分かるが, 逆は自明ではない．なぜなら, $\omega^m \not\equiv 1 \pmod{p}\ (1 \leq m \leq 2^k - 1)$ を示さなければならないからである．ここで次の命題について考える．

命題 5.2
素数 $p = n2^k + 1\ (n,\ k \in \mathbb{N})$ と $\omega \in \mathbb{Z}$ について,

$$\omega \text{ が法 } p \text{ において原始 } 2^k \text{ 乗根である} \Leftrightarrow \omega^{2^{k-1}} \equiv -1 \pmod{p}$$

これが正しいことを示す．

証明.
(十分条件)
仮定より ω は原始 2^k 乗根である．従って, 原始根の定義より

$$\forall m \in \mathbb{Z},\ \omega^m \equiv 1 \pmod{p} \Leftrightarrow m \equiv 0 \pmod{2^k}$$

となる．よって, $\omega^{2^k} \equiv 1 \pmod{p}$．従って, $\omega^{2^{k-1}} \equiv \pm 1 \pmod{p}$ となるが, $2^{k-1} \not\equiv 0 \pmod{2^k}$ より, $\omega^{2^{k-1}} \not\equiv 1 \pmod{p}$．よって, $\omega^{2^{k-1}} \equiv -1 \pmod{p}$．
(必要条件)
$\omega^{2^{k-1}} \equiv -1 \pmod{p}$ かつ, ω が原始 2^k 乗根でないと仮定する．

$\omega^{2^{k-1}} \equiv -1 \pmod{p}$ より, $\omega^{2^k} \equiv 1 \pmod{p}$. よって, ω は 2^k 乗根である. ここで, ω は原始 2^k 乗根ではないので, $\exists m < 2^k,\ \omega^m \equiv 1 \pmod{p}$. $m < 2^k$ と $\omega^{2^k} \equiv 1 \pmod{p}$ より, $\exists q \geq 2,\ qm = 2^k$. ここで, $q \geq 2$ と素因数分解の一意性より q は偶数. 従って $q = 2q'$ とおく. $qm = 2^k$ より, $q'm = 2^{k-1}$ となるが, $\omega^{q'm} \equiv (\omega^m)^{q'} \equiv 1^{q'} \equiv 1 \pmod{p}$ となる一方で, $\omega^{2^{k-1}} \equiv -1 \pmod{p}$ となる. ここで, $p = n2^k + 1$ より $p > 2$ なので $-1 \not\equiv 1 \pmod{p}$. よって矛盾. これより, $\omega^{2^{k-1}} \equiv -1 \pmod{p} \Rightarrow \omega$ が原始 2^k 乗根.

以上より, ω が法 p において原始 2^k 乗根である $\Leftrightarrow \omega^{2^{k-1}} \equiv -1 \pmod{p}$. □

よって, $\omega^{2^{k-1}}$ だけを調べれば良いと分かる.

6 プログラム

6.1 コード

GCC インラインアセンブラを用いているため, GCC でコンパイルすること.

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <time.h>
#include "other.c"
#include "old.c"
#include "ComplexFourier.c"
#include "RemainderFourier.c"
#define TIME(STR,HOGE) fprintf(stdout, "%s : %.3fs\n",STR,((st=clock()),(HOGE),\
(et=clock()),((double)(et-st)/CLOCKS_PER_SEC)))
#ifndef K
#define K 16//要素数=2^K
#endif
#define N (1<<K)
#define BASE 2//基数

int main(int argc, char *argv[]){
    time_t st,et;
    static short a[N], b[N], c[N];
    srand((unsigned)time(NULL));
    int logn, num, base;
    if(argc<3){
        logn=K;
        num=N;
        base=BASE;
    }else{
        logn=strtol(argv[1], NULL, 0);
        num=1<<logn;
```

```
        base=strtol(argv[2], NULL, 0);
    }
    printf("START num=%d, base=%d\n",num,base);
    Setting(a, b, num, base);Clear(c, num);

    TIME("      筆算", MULn(a, b, c, num, base));
    check(c, num, base);
    Clear(c, num);

    TIME("カラツバ法", KMUL(a, b, c, num, base));
    check(c, num, base);
    Clear(c, num);

    TIME("   DFTMULT", DFTMULT(a, b, c, num, base));
    check(c, num, base);
    Clear(c, num);

    TIME("   FFTMULT", FFTMULT(a, b, c, num, base));
    check(c, num, base);
    Clear(c, num);

    TIME("   NTTMULT", NTTMULT(a, b, c, num, base));
    check(c, num, base);
    Clear(c, num);
    return 0;
}
```

other.c

```
#include <stdio.h>
int Setting(short a[], short b[], int num, int base);
int check(short a[], int num, int base);
int Clear(short a[], int num);

int Setting(short a[], short b[], int num, int base){
    int i;
    for(i=0;i<num;i++){
        a[i]=base-1;
        b[i]=base-1;
    }
    for(i=num/2;i<num;i++){
        a[i]=0;
        b[i]=0;
    }
    return 0;
```

```
}

int check(short a[], int num, int base){
    int i;
    for(i=0;i<num;i++){
        if((i>=num/2 && a[i]!=base-1)||(i<num/2 && a[i]!=0)){
            if(!((i==num/2 && a[i]==base-2)||(i==0 && a[i]==1))){
                fprintf(stderr, "ERROR@%d, %d\n",i,a[i]);
        return -1;
            }
        }
    }
    return 0;
}

int Clear(short a[], int num){
    int i;
    for(i=0;i<num;i++){
        a[i]=0;
    }
    return 0;
}
```

old.c

```
#define SIZE 64
int MULn(short a[], short b[], short c[], int n, int base);
void ADDn(short a[],short b[],short c[],int n, int base);
void SUBn(short a[],short b[],short c[],int n, int base);
void KMUL(short a[],short b[],short c[],int n, int base);

int MULn(short a[], short b[], short c[], int n, int base){
    int i,j;
    long t;
    for(i=0;i<n;i++)c[i]=0;
    for(i=0;i<n/2;i++){
        t=0;
        for(j=0;j<n/2;j++){
            t+=a[i]*b[j]+c[i+j];
            c[i+j]=t%base;
            t/=base;
        }
        c[i+n/2]+=t;
    }
    return 0;
```

```
}

void ADDn(short a[],short b[],short c[],int n, int base){
    int i;
    int t=0;
    for(i=0;i<n;i++){
        t+=a[i]+b[i];
        c[i]=t%base;
        t/=base;
    }
    return;
}

void SUBn(short a[],short b[],short c[],int n, int base){
    int i;
    int t=0;
    for(i=0;i<n;i++){
        t+=a[i]-b[i]+base;
        c[i]=t%base;
        t/=base;
        t--;
    }
    return;
}

void KMUL(short a[],short b[],short c[],int n, int base){
    int i;
    short *f,*g,*t,*u;
    if(n>SIZE){
        for(i=0;i<n;i++)c[i]=0;
        if((t=(short*)calloc(n/2,sizeof(short)))==NULL)exit(EXIT_FAILURE);
        if((u=(short*)calloc(n/2,sizeof(short)))==NULL)exit(EXIT_FAILURE);
        SUBn(a+n/4,a,t,n/4,base);
        SUBn(b,b+n/4,u,n/4,base);
        if((g=(short*)calloc(n+1,sizeof(short)))==NULL)exit(EXIT_FAILURE);
        KMUL(t,u,g,n/2,base);
        free(t);
        free(u);
        KMUL(a    ,b    ,c,n/2,base);
        ADDn(g,c,g,n/2+1,base);
        if((f=(short*)calloc(n+1,sizeof(short)))==NULL)exit(EXIT_FAILURE);
        KMUL(a+n/4,b+n/4,f,n/2,base);
        ADDn(g,f,g,n/2+1,base);
        ADDn(c+n/4,g,c+n/4,n/2+1,base);
```

```
        free(g);
        ADDn(c+n/2,f,c+n/2,n/2,base);
        free(f);
    }else{
        MULn(a,b,c,n,base);
    }
    return;
}
```

Number.c

```
#ifndef NUNBER
#define NUNBER
inline int tib(int a, int k);
int IsPrime(int a);
int powmod(int a,int n,int m);
int modpinv(int a,int m);
inline int addmod(int a,int b,int m);
inline int submod(int a,int b,int m);
inline int mulmod(int a,int b,int m);
inline int mulmodasm(int a,int b,int m);

inline int mulmodasm(int a,int b,int m);

inline int tib(int a, int k){
    int i=k,z=0;
    while(i--){
        z=(z<<1)+(a&1);
        a>>=1;
    }
    return z;
}

int IsPrime(int a){
    int i;
    if(a<2 || (a&1)==0){
        return a==2?1:0;
    }else{
        for(i=3;i*i<=a;i+=2)if(a%i==0)return 0;
        return 1;
    }
}

int powmod(int a,int n,int m){
    long long x,y;
```

```
    if(n<0 || m<=1)return -1;//ERROR
    if(n==0)return 1;
    y=a%m;
    x=1;
    while(n){
        if(n&1)x=x*y%m;
        y=y*y%m;
        n>>=1;
    }
    return (int)x;
}

int modpinv(int a,int m){
    int temp,y=1,oy=0,om=m;
    while(a!=0){
        temp=oy-m/a*y;oy=y;y=temp;
        temp=m%a;m=a;a=temp;
    }
    temp=oy%om;
    return temp<0?temp+om:temp;
}

inline int addmod(int a,int b,int m){
    int t=a+b;
    return (t>=m)?t-m:t;
}

inline int submod(int a,int b,int m){
    int t=a-b;
    return (t<0)?t+m:t;
}

inline int mulmod(int a,int b,int m){
    return (int)((long long)a*(long long)b%(long long)m);
}

inline int mulmodasm(int a,int b,int m){
    int c;
    __asm__("movl %1, %%eax\n\t"
            "mull %2\n\t"
            "divl %3\n\t"
            "movl %%edx, %0"
            :"=g"(c)
            :"g"(a),"g"(b),"g"(m)
```

```
            :"%eax","%edx"
    );
    return c;
}
#endif
```

ComplexFourie.c

```
#include "Number.c"
int I2C(short *a, double complex *b, int n);
int C2I(double complex *a, short *b, int n);
int DFT(double complex *a, int n);
int MULT(double complex *a, double complex *b, double complex *c, int n);
int DIV(double complex *a, int n);
int CARRY(double complex *fa, short *a, int n, int base);
int FFT(double complex *a, int n);
int FFTMULT(short *a, short *b, short *c, int num, int base);
int DFTMULT(short *a, short *b, short *c, int num, int base);
int checkFFT(int num);

int I2C(short *a, double complex *b, int n){
    int i;
    for(i=0;i<n;i++){
        b[i]=(double complex)a[i];
    }
    return 0;
}

int C2I(double complex *a, short *b, int n){
    int i;
    for(i=0;i<n;i++){
        b[i]=(short)lround(a[i]);
    }
    return 0;
}

int DFT(double complex *a, int n){
    int i,j,r;
    double complex *t,*w;
    if((t=(double complex*)
        calloc(n,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    if((w=(double complex*)
        calloc(n,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    for(i=0;i<n;i++){
        w[i]=cexp(2*M_PI*i/n*I);
```

```
    }
    for(i=0;i<n;i++){
        r=0;
        for(j=0;j<n;j++){
            t[i]+=a[j]*w[r];
            r=addmod(r,i,n);
        }
    }
    for(i=0;i<n;i++){
        a[i]=t[i];
    }
    free(w);
    free(t);
    return 0;
}

int MULT(double complex *a, double complex *b, double complex *c, int n){
    int i;
    for(i=0;i<n;i++){
        c[i]=a[i]*b[i];
    }
    return 0;
}

int DIV(double complex *a, int n){
    int i;
    for(i=0;i<n;i++){
        a[i]/=(double)n;
    }
    return 0;
}

int CARRY(double complex *fa, short *a, int n, int base){
    int i;
    long long t=0;
    for(i=0;i<n;i++){
        t+=llround(fa[addmod(n,-i,n)]);
        a[i]=t%base;
        t/=base;
    }
    return 0;
}

int FFT(double complex *a, int n){
```

```
    int i,j,k=lround(log2(n)),m,s;
    double complex t,*b,*w;
    if((b=(double complex*)
        calloc(n,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    for(i=0;i<n;i++){
        b[i]=a[tib(i,k)];
    }
    for(i=0;i<n;i++){
        a[i]=b[i];
    }
    free(b);
    if((w=(double complex*)
        calloc(n,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    for(i=0;i<n;i++){
        w[i]=cexp(2*M_PI*i/n*I);
    }
    for(m=0;m<k;m++){
        s=1<<m;
        for(i=0;i<n;i+=s){
            for(j=0;j<s;j++){
                t=w[(1<<(k-m-1))*j%n]*a[i+s];
                a[i+s]=a[i]-t;
                a[i  ]=a[i]+t;
                i++;
            }
        }
    }
    free(w);
    return 0;
}

int FFTMULT(short *a, short *b, short *c, int num, int base){
    double complex *fa, *fb, *fc;
    if((fa=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    if((fb=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    if((fc=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    I2C(a,fa,num);
    FFT(fa,num);
    I2C(b,fb,num);
    FFT(fb,num);
    MULT(fa,fb,fc,num);
```

```
    FFT(fc,num);
    DIV(fc,num);
    CARRY(fc,c,num,base);
    free(fa);
    free(fb);
    free(fc);
    return 0;
}

int DFTMULT(short *a, short *b, short *c, int num, int base){
    double complex *fa, *fb, *fc;
    if((fa=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    if((fb=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    if((fc=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    I2C(a,fa,num);
    DFT(fa,num);
    I2C(b,fb,num);
    DFT(fb,num);
    MULT(fa,fb,fc,num);
    DFT(fc,num);
    DIV(fc,num);
    CARRY(fc,c,num,base);
    free(fa);
    free(fb);
    free(fc);
    return 0;
}

int checkFFT(int num){
    int i;
    double complex *fa, *fb, *fc;
    if((fa=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    if((fb=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    if((fc=(double complex*)
        calloc(num,sizeof(double complex)))==NULL)exit(EXIT_FAILURE);
    for(i=0;i<num;i++){
        fa[i]=(rand()%100)+(rand()%100)*I;
        fb[i]=fa[i];
    }
```

```
    DFT(fa,num);
    FFT(fb,num);
    for(i=0;i<num;i++){
        if(cabs(fa[i]-fb[i])>0.001){
            printf("%d %f+%fi\n",i,creal(fa[i]),cimag(fa[i]));
            printf("%d %f+%fi\n",i,creal(fb[i]),cimag(fb[i]));
        }
    }
    return 0;
}
```

RemainderFourie.c

```
#include "Number.c"
int findprime(int num, int base);
int findomega(int prime, int num);
int MakeTable(long w[], int prime, int num);
int S2L(short *a,long *fa,int n);
int FNTT(long *a,int prime,int n,long w[]);
int MUL(long *a,long *b,long *c,int n,int prime);
int DIVN(long *a, int n, int prime);
int NCARRY(long *fa, short *a, int num, int base);
int NTTMULT(short *a, short *b, short *c, int num, int base);

int findprime(int num, int base){
    int prime;
    //適当な素数を見つける
    for(prime=(base-1)*(base-1)*num+1;IsPrime(prime)==0;prime+=num);
    return prime;
}

int findomega(int prime, int num){
    int omega;
    //原始根を見つける
    for(omega=2;powmod(omega,num/2,prime)!=(prime-1);omega++);
    return omega;
}

int MakeTable(long w[], int prime, int num){
    int i,temp;
    int omega=findomega(prime,num);
    for(i=0,temp=1;i<num;i++){
        w[i]=temp;
        temp=mulmod(temp,omega,prime);
    }
```

```
    return 0;
}

int S2L(short *a,long *fa,int n){
    int i;
    for(i=0;i<n;i++){
        fa[i]=a[i];
    }
    return 0;
}

int FNTT(long *a,int prime,int n,long w[]){
    int i,j,k2,s,cnt,temp,k=lround(log2(n));
    long *t;
    if((t=(long*)calloc(n,sizeof(long)))==NULL)exit(EXIT_FAILURE);
    for(i=0;i<n;i++)t[tib(i,k)]=a[i];
    for(i=0;i<n;i++)a[i]=t[i];
    free(t);
    for(i=0;i<k;i++){
        k2=(1<<i);
        s=(1<<(k-i-1));
        for(j=0;j<n;j+=k2){
            for(cnt=0;cnt<k2;cnt++,j++){
                temp=mulmodasm(w[s*cnt],a[j+k2],prime);
                a[j+k2]=submod(a[j],temp,prime);
                a[j   ]=addmod(a[j],temp,prime);
            }
        }
        //for(j=0;j<n;j++)printf("%d, ",a[j]);printf("\n");
    }
    return 0;
}

int MUL(long *a,long *b,long *c,int n,int prime){
    int i;
    for(i=0;i<n;i++){
        c[i]=mulmod(a[i],b[i],prime);
    }
    return 0;
}

int DIVN(long *a, int n, int prime){
    int i,inv=modpinv(n,prime);
    for(i=0;i<n;i++){
```

```
        a[i]=mulmod(a[i], inv, prime);
    }
    return 0;
}

int NCARRY(long *fa, short *a, int num, int base){
    int i;
    long long t=0;
    for(i=0;i<num;i++){
        t+=fa[addmod(num,-i,num)];
        a[i]=t%base;
        t/=base;
    }
    return 0;
}

int NTTMULT(short *a, short *b, short *c, int num, int base){
    int prime;
    long *fa, *fb, *fc, *w;
    if((fa=(long*)calloc(num,sizeof(long)))==NULL)exit(EXIT_FAILURE);
    if((fb=(long*)calloc(num,sizeof(long)))==NULL)exit(EXIT_FAILURE);
    if((fc=(long*)calloc(num,sizeof(long)))==NULL)exit(EXIT_FAILURE);
    if((w =(long*)calloc(num,sizeof(long)))==NULL)exit(EXIT_FAILURE);
    prime=findprime(num,base);
    MakeTable(w,prime,num);
    S2L(a,fa,num);
    FNTT(fa,prime,num,w);
    S2L(b,fb,num);
    FNTT(fb,prime,num,w);
    MUL(fa,fb,fc,num,prime);
    FNTT(fc,prime,num,w);
    DIVN(fc,num,prime);
    NCARRY(fc,c,num,base);
    free(fa);
    free(fb);
    free(fc);
    free(w);
    return 0;
}
```

6.2 計算時間

実行したところ以下のような結果となった. また, 実行時間は 15 回平均である. 表の FFT とは高速フーリエ変換のことで, DFT とは離散フーリエ変換のことである. 「-」は計算時間が非常に長くなることが予

想されるので省略した.

N	古典的乗算法	Karatsuba 法	複素数上の DFT	複素数上の FFT	有限体上の FFT
2^{10}	0.001s	0.001s	0.039s	0.000s	0.000s
2^{11}	0.015s	0.000s	0.111s	0.001s	0.000s
2^{12}	0.036s	0.008s	0.444s	0.001s	0.001s
2^{13}	0.117s	0.015s	1.804s	0.006s	0.000s
2^{14}	0.429s	0.053s	7.934s	0.009s	0.004s
2^{15}	1.680s	0.147s	34.682s	0.024s	0.006s
2^{16}	6.669s	0.437s	142.197s	0.049s	0.014s
2^{17}	26.617s	1.310s	577.547s	0.101s	0.030s
2^{18}	106.434s	3.931s	-	0.216s	0.068s
2^{19}	425.703s	11.798s	-	0.495s	0.140s
2^{20}	-	35.425s	-	1.165s	0.304s
2^{21}	-	106.190s	-	2.834s	0.692s
2^{22}	-	318.526s	-	6.736s	1.807s
2^{23}	-	955.688s	-	15.070s	5.192s
2^{24}	-	-	-	32.253s	12.415s
2^{25}	-	-	-	65.027s	28.488s

表 2: 実行時間

コンパイルは “gcc main.c -o main -pipe -W -Wall -O3 -lm” で適宜 “-D K=(num)” を加えた. gcc のバージョンは 4.5.3, 実行環境は OS は Windows 7 SP1 (64bit), CPU は Corei7-3770 (3.40GHz), RAM は 32.0GB となっている.

表より, 有限体 FFT が最も速いことが分かる. 複素 FFT が有限体 FFT の 3 倍程度の時間がかかっている理由は, 複素数の四則演算は実数と比べて数倍になることと, 複素数は 1 つの値を保持するのに 16 バイト必要だが整数は 4 バイトで済むなどの要因が考えられる. また, 古典的乗算法より複素 DFT が遅いのも同様の理由に加え, 表 1 を見ると分かるように計算量が 3 倍程度あるからだと思われる.

参考文献

[1] D.E.KNUTH, 中川圭介訳, KNUTH The Art of Computer Programing=4 準数値算法/算術演算, サイエンス社, 1986.

[2] 高木貞治, 初等整数論講義 第 2 版, 共立出版, 1971.

[3] T.W.KÖRNER, 高橋陽一郎訳, フーリエ解析大全 (上), 朝倉書店, 1996.

[4] T.W.KÖRNER, 高橋陽一郎訳, フーリエ解析大全 (下), 朝倉書店, 1996.