ストラッセン-ショーンハーゲ法

梅谷 武

作成：2000-08-19 更新：2005-04-20

多倍長整数の高速乗算法であるストラッセン-ショーンハーゲ法の実装法について検討する。作譜に必要な個々の算法について見渡した後、32 ビット語長の場合について、その主要部分を算譜としてまとめる。
IMS:20000819001; NDC:412.1; keywords:ストラッセン-ショーンハーゲ法, 離散 Fourier 変換;

目　　次

1　ストラッセン-ショーンハーゲ法

1.1　離散 Fourier 変換の型

整数 $n, m > 0$ 及び、$R = \mathbf{Z}/m\mathbf{Z}$ 上の 1 の原始 n 乗根 ζ を適当に定めることによって、R 上の離散 Fourier 変換を定義することができる。([F1] 参照) ストラッセン-ショーンハーゲ法においては、$K = 2^k$ として、

$$(n, m, \zeta) = (K, 2^{2K} + 1, 2^4)$$

という離散 Fourier 変換を使用する。この変換は $K = 2^k$ であることから、Cooley-Tukey 型の高速算法が適用できることと、1 の原始 K 乗根 $\zeta = 2^4$ を乗ずることが、乗算命令ではなくて、2 進演算では非常に効率がいいビットシフト命令で実行することができることから効率の良い最適化が期待できる。まず、これが離散 Fourier 変換の必要条件を満たしていることを証明する。

補題 1.1 $K = 2^k (k > 0)$ として、

$$(n, m, \zeta) = (K, 2^{2K} + 1, 2^4)$$

とするとき、整数の剰余環 $R = \mathbf{Z}/m\mathbf{Z}$ は、次の離散 *Fourier* 変換の必要条件を満たす。

1. ζは 1 の原始 K 乗根
2. $\exists K^{-1} \in R = \mathbf{Z}/m\mathbf{Z}$
3. $K > l > 0 \rightarrow \sum_{i=0}^{K-1} \zeta^{il} = 0$

証明 *1.* は次の式による。

$$\zeta^{K/2} = 2^{2K} \equiv -1 (\mathrm{mod}\, 2^{2K} + 1),\ \zeta^K \equiv 1 (\mathrm{mod}\, 2^{2K} + 1)$$

2. は $(K, 2^{2K} + 1) = 1$ による。*3.* については、まず、

$$l \neq 0 (\mathrm{mod}\, K) \Rightarrow (\zeta^l - 1, 2^{2K} + 1) = 1$$

となることを示す。

$$\zeta^l \equiv 1 (\mathrm{mod}\, p), 2^{2K} \equiv -1 (\mathrm{mod}\, p)$$

なる素数 p が存在したと仮定すると

$$\zeta^l = 2^{4l} \equiv 1 (\mathrm{mod}\, p)$$

より、$4K | 4l$ から $K | l$ となり $l \equiv 0 (\mathrm{mod}\, K)$ が導かれる。

$$\begin{aligned} \zeta^l \sum_{i=0}^{K-1} \zeta^{il} &= \sum_{i=0}^{K-1} \zeta^{(i+1)l} \\ &\equiv \sum_{i=0}^{K-1} \zeta^{il} + \zeta^{Kl} - \zeta^{0l} (\mathrm{mod}\, 2^{2K} + 1) \\ &\equiv \sum_{i=0}^{K-1} \zeta^{il} (\mathrm{mod}\, 2^{2K} + 1) \end{aligned}$$

より、

$$(\zeta^l - 1) \sum_{i=0}^{K-1} \zeta^{il} \equiv 0 (\mathrm{mod}\, 2^{2K} + 1)$$

したがって $(\zeta^l - 1, 2^{2K} + 1) = 1$ より、

$$\sum_{i=0}^{K-1} \zeta^{il} \equiv 0 (\mathrm{mod}\, 2^{2K} + 1)$$

■

1.2 ストラッセン-ショーンハーゲ法の改良

ここではストラッセン-ショーンハーゲ法を少し改良した形で述べる。これにより計算量を少し減らすことができる。

仮定 1.2 (多倍長整数の高速乗算法)

$$K = 2^k (k > 0), N = K^2$$

とし、正の整数 a, b が、基数 $P = 2^K$ により次のように P 進表現されるものとする。

$$a = a_{K-1}P^{K-1} + \cdots + a_1 P + a_0, 0 \leqq a_i < P$$
$$b = b_{K-1}P^{K-1} + \cdots + b_1 P + b_0, 0 \leqq b_i < P$$

この場合、$0 \leqq a, b \leqq 2^N - 1$ であるが、さらに

$$0 \leqq ab \leqq 2^N - 1$$

と仮定する。

仮定より、ab は $2^N - 1$ を法として求めればよい。$P^K \equiv 1 (\mathrm{mod}\, 2^N - 1)$ より、

$$c = ab \equiv \sum_{r=0}^{K-1} \Big(\sum_{s+t \equiv r (mod K)} a_s b_t \Big) P^r (\mathrm{mod}\, 2^N - 1)$$

となる。この係数 $c_r = \sum_{s+t \equiv r (mod K)} a_s b_t$ の大きさを評価すると、

$$0 \leqq c_r \leqq K(2^K - 1)^2 < K(2^{2K} + 1)$$

となる。したがって c_r は $K(2^{2K} + 1)$ を法として求めればよい。$(K, 2^{2K} + 1) = 1$ であるから、孫子の剰余定理から次のことがわかる。

補題 1.3 c_r は、c'_r 及び c''_r が既知であるとして、連立合同式

$$c_r \equiv c'_r (\mathrm{mod}\, K)$$
$$c_r \equiv c''_r (\mathrm{mod}\, 2^{2K} + 1)$$

を解くことによって求めることができる。

K は $2^{2K} + 1$ に対して小さいので c'_r は直接計算する。

算法 1.4 c'_r は、次の手順で計算する。

1. $a'_r \equiv a_r (\mathrm{mod}\, K)$
2. $b'_r \equiv b_r (\mathrm{mod}\, K)$
3. $c'_r \equiv \sum_{s+t \equiv r (mod K)} a'_s b'_t (\mathrm{mod}\, K)$

c''_r は、$(K, 2^{2K} + 1, 2^4)$ 型の離散 Fourier 変換と多項式の高速乗算法を利用して計算する。([F1] 参照)

算法 1.5 c''_r は次のように計算する。

$$F(a)_i \equiv \sum_{s=0}^{K-1} a_s \zeta^{si} (\mathrm{mod}\, 2^{2K} + 1)$$
$$F(b)_i \equiv \sum_{t=0}^{K-1} b_t \zeta^{ti} (\mathrm{mod}\, 2^{2K} + 1)$$
$$c''_r \equiv K^{-1} \sum_{i=0}^{K-1} F(a)_i F(b)_i \zeta^{-ir} (\mathrm{mod}\, 2^{2K} + 1)$$

ここまでで $c_r, r = 0, \cdots, K-1$ を求めることができた。これに桁上げ処理を施すことで、$c = ab$ の P 進表現が得られる。

2 整数の算法

2.1 孫子の剰余定理

孫子の剰余定理は、3 世紀後半に著された「孫子算経」に連立合同式を満たす整数を求める問題があることから付けられた現代中国で使われている名称である。欧米では中国の剰余定理 (Chinese remainder thorem) と呼ばれている。([A2] 参照)

定理 2.1 (孫子の剰余定理) 正の整数 $m_1, m_2, \cdots, m_k$ はどの 2 つも互いに素であると仮定する。このとき、任意の整数 $u_1, u_2, \cdots, u_k$ に対して、連立合同式：

$$u \equiv u_1 (\mathrm{mod}\, m_1), u \equiv u_2 (\mathrm{mod}\, m_2), \cdots, u \equiv u_k (\mathrm{mod}\, m_k)$$

を満たす整数 u が存在し、$m_1 m_2 \cdots m_k$ を法として一意的に定まる。したがって、特に

$$0 \leqq u < m_1 m_2 \cdots m_k$$

を満たすものはただ一つ存在する。

証明 一意性をまず示す。整数 u, v が連立合同式を満たすとすると

$$u \equiv v (\mathrm{mod}\, m_j), 1 \leqq j \leqq k$$

となり、$u - v$ はすべての m_j の倍数であり、したがって $m = m_1 m_2 \cdots m_k$ の倍数となる。すなわち、

$$u \equiv v (\mathrm{mod}\, m_1 m_2 \cdots m_k)$$

である。存在することを構成的に証明する。

$$M_j = (m/m_j)^{\phi(m_j)}, 1 \leqq j \leqq k$$

とおく。ここで $\phi(m_j)$ はオイラーの関数である。そうするとオイラーの定理により

$$M_j \equiv 1 (\mathrm{mod}\, m_j)$$
$$M_j \equiv 0 (\mathrm{mod}\, m_k), k \neq j$$

が成り立つ。このとき、

$$\sum_{i=1}^{k} u_i M_i$$

はすべての合同式を満足する。■

上の証明は構成的ではあるが、実際に解を求める方法としてはあまり効率がよくない。この高速算法を H. L. Garner が与えている。([S2] 参照)

算法 2.2 (孫子の剰余定理の解の高速算法) まず、ユークリッドの互除法により、

$$c_{ij} m_i \equiv 1 (\mathrm{mod}\, m_j), 1 \leqq i < j \leqq k$$

なる c_{ij} を計算しておく。これは

$$a m_i + b m_j = 1$$

において $c_{ij} = a$ とすればよい。このとき、$v_r, 1 \leqq r \leqq k$ を

$$\begin{aligned}
v_1 &= u_1 \% m_1 \\
v_2 &= (u_2 - v_1)c_{12} \% m_2 \\
&\vdots \\
v_r &= (\cdots((u_r - v_1)c_{1r} - v_2)c_{2r} - \cdots - v_{r-1})c_{(r-1)r} \% m_r
\end{aligned}$$

とする。このとき、

$$u = v_k m_{k-1} \cdots m_1 + \cdots + v_3 m_2 m_1 + v_2 m_1 + v_1$$

が解である。

証明

$$\begin{aligned}
& v_j m_{j-1} \cdots m_1 \\
\equiv\ & m_{j-1} \cdots m_1 (\cdots((u_j - v_1)c_{1j} - v_2)c_{2j} - \cdots - v_{j-1})c_{(j-1)j} \\
\equiv\ & m_{j-2} \cdots m_1 (\cdots((u_j - v_1)c_{1j} - v_2)c_{2j} - \cdots - v_{j-2})c_{(j-2)j} - v_{j-1} m_{j-2} \cdots m_1 \\
&\vdots \\
\equiv\ & u_j - v_1 - v_2 m_1 - \cdots - v_{j-1} m_{j-2} \cdots m_1 (\mathrm{mod}\, m_j)
\end{aligned}$$

■

孫子の剰余定理の解法を算譜としてまとめておく。integer は無限精度整数型を表わしており、実際の作譜にあたっては桁あふれに対する考慮が必要である。

算譜 2.3 (孫子の剰余定理の解法)
```
integer c[k][k], m[k], u[k], v[k];
integer i, r, u;

// v[k] の計算
for (r = 0; r < k; r++)
{ i = 0;
  tmp = u[r];
  while (i < r)
  { tmp = (tmp - v[i])*c[i][r];
    i++;
  }
  v[r] = tmp % m[r];
}

// 解 u の計算
u = 0;
for (r = 0; r < k; r++)
{ i = 0;
  tmp = v[r];
  while (i < r)
  { tmp *= m[i];
    i++;
  }
  u += tmp;
}
```

2.2 ユークリッドの互除法

ユークリッドの互除法は最大公約数を求める算法であり、「原論」の　巻に定理として述べられている。近代の多くの初等整数論の教科書においても、そのまま最大公約数を求める算法として解説されているが、これ

が必ずしも最良の算法であるとはいえない。ここでは最大公約数を求める算法をある種の有限列を計算する方法として形式化し、その形式でユークリッドの互除法を表現した後、J. Stein により発見された除算命令を使わない算法について述べる。([S2] 参照)

定義 2.4 (最大公約数の算法のベクトル列表現) 2つの負でない整数 a, b について、

- 最大公約数 $d = (a, b)$ を求める
- 不定方程式 $d = ma + nb$ を満たす m, n を求める

算法について考える。

$$ax + by = z$$

を満たす整数ベクトル (x, y, z) の組の有限列 $(x_i, y_i, z_i), (s_i, t_i, u_i)_{i=0}^{n}$ が、次の条件を満たすものとする。

1. $z_0 = a, u_0 = b$
2. $z_i > u_i \geqq 0, i = 1, \cdots, n$
3. $max(z_i, u_i) > max(z_{i+1}, u_{i+1}), i = 0, \cdots, n-1$
4. $(z_i, u_i) = (z_{i+1}, u_{i+1}), i = 0, \cdots, n-1$
5. $z_n = d, u_n = 0$

もしこのような有限列を何らかの手段で生成することができれば最大公約数 $d = z_n$ 及び不定方程式の解 $m = x_n, y = z_n$ が求まる。これを最大公約数の算法のベクトル列表現と呼ぶ。

ユークリッドの互除法をベクトル列表現で述べる。

算法 2.5 (拡張されたユークリッドの互除法) 2つの負でない整数 a, b が与えられたとき、

1. $(x_0, y_0, z_0) = (1, 0, a), (s_0, t_0, u_0) = (0, 1, b)$
2. $u_i \neq 0$ ならば
   $(x_{i+1}, y_{i+1}, z_{i+1}) = (s_i, t_i, u_i)$
   $(s_{i+1}, t_{i+1}, u_{i+1}) = (x_i, y_i, z_i) - (s_i, t_i, u_i)z_i/u_i$

によって生成されるベクトル列は、最大公約数の算法のベクトル列表現である。

証明 $a = qb + r$ のとき $(a, b) = (b, r)$ であることからわかる。■

拡張されたユークリッドの互除法を算譜としてまとめておく。

算譜 2.6 (拡張されたユークリッドの互除法)
```
integer a, b;
integer x, y, z, s, t, u;
integer q, d, e, f;

x = 1; y = 0; z = a;
s = 0; t = 1; u = b;
while (u != 0)
{ q = z/u;
  d = s; e = t; f = u;
  s = x - s*q; t = y - t*q; u = z - u*q;
  x = d; y = e; z = f;
}
```

J. Stein による最大公約数の算法は、2進演算機で最適化できるように工夫されたものである。

算法 2.7 (J. Stein による最大公約数の算法) 二つの負でない整数 a, b が与えられたとき、もしともに 2 を素因数としてもつならば、必要なだけ 2 のべき乗で割ることによって、2 を共通の約数としてもたないようにしておく。このとき、

1. $(x_0, y_0, z_0) = (1, 0, a), (s_0, t_0, u_0) = (b, 1-a, b)$
2. z_i が偶数のとき、
   - x_i, y_i がともに偶数のとき、
     $(x_{i+1}, y_{i+1}, z_{i+1}) = (x_i, y_i, z_i)/2$
   - x_i, y_i のいずれかが奇数のとき、
     $(x_{i+1}, y_{i+1}, z_{i+1}) = (x_i + b, y_i - a, z_i)/2$
3. u_i が偶数のとき、
   - s_i, t_i がともに偶数のとき、
     $(s_{i+1}, t_{i+1}, u_{i+1}) = (s_i, t_i, u_i)/2$
   - s_i, t_i のいずれかが奇数のとき、
     $(s_{i+1}, t_{i+1}, u_{i+1}) = (s_i + b, t_i - a, u_i)/2$
4. z_i, u_i がともに奇数のとき、
   $(x_{i+1}, y_{i+1}, z_{i+1}) = (x_i, y_i, z_i) - (s_i, t_i, u_i)$
   $(s_{i+1}, t_{i+1}, u_{i+1}) = (s_i, t_i, u_i)$
5. もし *2.3.4.* の操作で、$z_{i+1} < u_{i+1}$ となればベクトルの順序を入れ替える。

によって生成されるベクトル列は、最大公約数の算法のベクトル列表現である。

証明 最大公約数が求まることは、a が偶数で b が奇数のとき $(a, b) = (a/2, b)$ であることと $(a, b) = (a-b, b)$ からわかる。*2.* と *3.* の操作で、$ax + by = z$ が偶数で、x, y がともに偶数でないとき $x + b$ と $y - a$ がともに偶数になることを示す。x が奇数であるとする。a が奇数のとき、b と y はともに奇数でなければならない。a が偶数のとき、仮定から b は奇数である。したがって y は偶数である。y が奇数である場合も同様である。■

J. Stein による最大公約数の算法を算譜としてまとめておく。

算譜 2.8 (J. Stein による最大公約数の算法)
```
integer a, b;
integer x, y, z, s, t, u;
integer d, e, f;
integer k;

k = 0;
while (a,b はともに偶数)
{ a >>= 1; b >>= 1;
  k++;
}

x = 1; y = 0; z = a;
s = 0; t = 1; u = b;
if (u > z)
{ d = s; e = t; f = u;
  s = x; t = y; u = z;
  x = d; y = e; z = f;
}
while (u != 0)
{ if (z は偶数)
  { if (x,y はともに偶数)
    { x >>= 1; y >>= 1; z >>= 1;
    } else {
      x += b; y -= a;
      x >>= 1; y >>= 1; z >>= 1;
    }
  }
  else if (u は偶数)
  { if (s,t はともに偶数)
    { s >>= 1; t >>= 1; u >>= 1;
    } else {
      s += b; t -= a;
      s >>= 1; t >>= 1; u >>= 1;
    }
  }
  else
  { x -= s; y -= t; z -= u;
  }
  if (u > z)
  { d = s; e = t; f = u;
    s = x; t = y; u = z;
    x = d; y = e; z = f;
  }
}
z <<= k;
```

2.3 法演算

ストラッセン-ショーンハーゲ法の中に $2^{2K}+1$ を法とする演算がでてくるが、ここでは一般に 2^e+1 を法とする加法と乗法を 2^e を法とする加法と乗法を利用して計算する方法についてまとめておく。証明は自明なので省略する。

命題 2.9 $0 \leqq u, v < 2^e+1$ とするとき、

$$\begin{aligned}(u+v)\%(2^e+1) &= u+v,\ u+v < 2^e+1\\ &= (u+v)\%2^e - 1,\ otherwise\\ uv\%(2^e+1) &= uv\%2^e - uv/2^e,\ uv\%2^e - uv/2^e \geqq 0\\ &= uv\%2^e - uv/2^e + 2^e + 1,\ otherwise\end{aligned}$$

3　32 ビット語長の場合

3.1　桁あふれの判定

仮定 3.1 (多倍長整数の高速乗算法 (32 ビット語長))

$$K = 2^5 = 32, N = K^2 = 1024$$

とし、正の整数 a, b が、基数 $P = 2^{32}$ により次のように P 進表現されるものとする。

$$a = a_{K-1}P^{K-1} + \cdots + a_1P + a_0, 0 \leqq a_i < P$$
$$b = b_{K-1}P^{K-1} + \cdots + b_1P + b_0, 0 \leqq b_i < P$$

この場合、$0 \leqq a, b \leqq 2^N - 1$ であるが、さらに

$$0 \leqq ab \leqq 2^N - 1$$

と仮定する。

a_u, b_v を 0 でない最大の係数としたとき、$0 \leqq ab \leqq 2^N - 1$ の判定を次の条件によって行なう。

$$(u + 1) + (v + 1) \leqq K - 1$$

この条件はきびしすぎるが、正確な判定を行なうにはかなりの計算量が必要になるので簡単のためにこうしておく。

3.2　c' の計算

(c'_r を計算する算譜を参照する。)

3.3　c'' の計算

(c''_r を計算する算譜を参照する。)

3.4　連立合同式の解法

c'_r, c''_r が与えられたときに、補題 1.3 の連立合同式を解いて c_r を求める部分をまとめておく。算法 2.2 を適用すると $m_1 = 2^5, m_2 = 2^{64} + 1$ である。まず、

$$c_{12}m_1 \equiv 1 (\mathrm{mod}\, m_2)$$

なる c_{12} を求める。

$$2^{59}2^5 \equiv -1 (\mathrm{mod}\, 2^{64} + 1)$$

の両辺を 2 乗することによって、

$$2^{123}2^5 \equiv 1 (\mathrm{mod}\, 2^{64} + 1)$$

c_{12}は $2^{64}+1$ を法として求めればよいので、

$$\begin{aligned} c_{12} &= 2^{123} = 2^{59}(2^{64}+1) - 2^{59} \\ &\equiv 2^{64} - 2^{59} + 1 (\mathrm{mod}\, 2^{64}+1) \\ &\equiv 2^{59}31 + 1 (\mathrm{mod}\, 2^{64}+1) \end{aligned}$$

となる。

参考文献

代数学

[A1] 松坂 和夫, “代数系入門”, 岩波書店, 1976
[A2] 上野 健爾, “代数入門”, 岩波書店, 2004

Fourier 解析

[F1] 梅谷 武, 離散 *Fourier* 変換
[F2] 大浦 拓哉, *FFT* の概略と設計法

数論

[N1] 和田 秀男, “コンピュータと素因子分解 改訂版”, 遊星社, 1999
[N2] 和田 秀男, “高速乗算法と素数判定法”, 上智大学数学教室, 1983

算法

[S1] 野下 浩平, 高岡 忠雄, 町田 元, “基本的算法”, 岩波書店, 1983
[S2] D. E. Knuth(中川 圭介訳), “準数値算法/算術演算”, サイエンス社, 1986

ディジタル信号処理

[D1] 電子情報通信学会, “ディジタル信号処理の基礎”, コロナ社, 1988