数理解析研究所講究録 1295 巻 2002 年 80-86

大整数に対する整数·有理数変換について

佐々木建昭 (Tateaki Sasaki) *
筑波大学 数学系
INSTITUTE OF MATHEMATICS, UNIVERSITY OF TSUKUBA

高橋善徳 (Yoshinori Takahashi) †
筑波大学 教育研究科
MASTER'S PROGRAM IN EDUCATION, UNIVERSITY OF TSUKUBA

杉本卓也 (Takuya Sugimoto) ‡
筑波大学 理工学研究科
MASTER'S PROGRAM IN SCIENCE AND ENGINEERING, UNIVERSITY OF TSUKUBA

1 はじめに

M と S は与えられた整数で、$M > |S| > 0$ を満たすとする。M と S に対して

$$DS \equiv N \pmod{M}, \quad 0 < D < \sqrt{M/2},\ 0 < |N| < \sqrt{M/2}, \tag{1}$$

を満たす整数 D と N が存在するとき、S は M を法として有理数 N/D に等しいとし、S から D, N への変換を整数·有理数変換という [5] [6] [2]。(M が小さいと D と N が存在しないこともあるが、M が十分大きければ大抵存在する)。

整数·有理数変換は数式処理において基本的かつ重要な演算である。たとえば、計算代数における最も重要な算法である Gröbner 基底計算法は、そのまま実行すると猛烈な中間係数膨張を引き起こすことがほとんどで、結果として膨大な計算時間を必要とする。しかし、1 語長程度の多数の素数を選び、それらの素数を法として計算を行えば計算の大部分の過程で中間係数膨張は現れず、計算は非常に効率化される [3]。ただし、個々の素数での計算を中国剰余定理でまとめると通常、巨大な整数が現れ、最後の段階で巨大な整数を有理数に変換する必要がある。計算が大規模になればなるほど、計算時間に占める整数·有理数変換の割合が大きくなるので、整数·有理数変換を効率化することは実際面で非常に重要である。

整数·有理数変換は通常、拡張互除法で行われる。整数 GCD(最大公約数) の計算が今でも互除法で行われているように、この方法は古典的にもかかわらず極めて有効である。しかしながら、整数 GCD に対して、Binary 法や Lehmer 法が考案され ([1] 参照)、拡張互除法に対しても Schönhage の分割征服法が考案されたように [4]、整数·有理数変換の効率化も追求すべきである。Lehmer 法はそのまま拡張互除法にも使える。Schönhage の分割征服法は、互いに素な 2 整数 A, B に対し $UA + VB = 1$ を満たす整数 U, V を

*sasaki@math.tsukuba.ac.jp
†suunistus00@aol.com
‡sugimoto@math.tsukuba.ac.jp

計算するものであり、$2m$ 桁の二つの整数の上位約半分の m 桁を "消去" する操作を再帰的に実行する。一方、**2** で見るように整数・有理数変換では M と S の上位桁の約半分を消去するので、分割征服法がぴったり当てはまると言える。

本稿は、整数・有理数変換の効率化が Lehmer 法と分割征服法により、どの程度達成されるかを実際的に調べる。なお、本稿では紙面が限られているため Lehemer 法の説明は割愛する (文献 [1] に分かりやすい説明がある)。

2 拡張互除法と整数・有理数変換

もしも $N/D \equiv S \pmod{M}$ かつ $S > 0$ ならば $-N/D \equiv -S \pmod{M}$ であり、もしも $S \geq M/2$ ならば $N/D \equiv (S-M) \equiv -(M-S) \pmod{M}$ である。したがって、以下では一般性を失うことなく、S は次式を満たすとする。

$$M/2 > S > 0. \tag{2}$$

(1) を満たす整数は (もしあれば) 有名な拡張互除法で計算できる。$S_0 = M$, $S_1 = S$ とおき、剰余列 $(S_0, S_1, S_2, \cdots, S_n)$ と余因子列 $(C_0, C_1, C_2, \cdots, C_n)$, $(D_0, D_1, D_2, \cdots, D_n)$ を次の算法で計算する (拡張互除法):

$$\begin{cases} Q_i := \mathrm{quo}(S_{i-1}, S_i), & i = 1 \Rightarrow 2 \Rightarrow \cdots \Rightarrow n-1, \\ S_{i+1} := S_{i-1} - Q_i S_i & (\text{i.e.}, S_{i+1} = \mathrm{rem}(S_{i-1}, S_i)), \\ C_{i+1} := C_{i-1} - Q_i C_i, & \text{with } C_0 := 1 \text{ and } C_1 := 0, \\ D_{i+1} := D_{i-1} - Q_i D_i, & \text{with } D_0 := 0 \text{ and } D_1 := 1. \end{cases} \tag{3}$$

ここで quo と rem はそれぞれ quotient と remainder を表す。このとき次式が成立する。

$$C_i M + D_i S = S_i \quad (i = 1, 2, \ldots, n). \tag{4}$$

条件 (2) のため各剰余 S_i は正である。$(S_1, S_2, \cdots, S_n)$ は降下列で $(|D_1|, |D_2|, \cdots, |D_n|)$ は上昇列である。したがって、剰余列を計算していき、$S_n < \sqrt{M/2}$ かつ $|D_n| < \sqrt{M/2}$ になったならば、$D = |D_n|$ かつ $N = \mathrm{sign}(D_n)\, S_n$ ある。この条件が満たされなければ、求める D と N は存在しない。

$i \geq 2$ のとき、S_i と C_i, D_i は次の各式を満たす。

$$S_i > 0, \quad \mathrm{sign}(C_i) = (-1)^i, \quad \mathrm{sign}(D_i) = (-1)^{i+1}, \tag{5}$$

$$|C_i| < |D_i|, \quad |C_i| < |C_{i+1}|, \quad |C_i| < |C_{i+1}|, \tag{6}$$

$$\text{with exceptions } \; C_2 \leq |D_2|, \;\; C_2 \leq |C_3|,$$

$$\begin{cases} S_0 = |D_i| S_{i-1} + |D_{i-1}| S_i & \Longrightarrow \quad |D_i| < S_0/S_{i-1}, \\ S_1 = |C_i| S_{i-1} + |C_{i-1}| S_i & \Longrightarrow \quad |C_i| < S_1/S_{i-1}. \end{cases} \tag{7}$$

3 拡張互除法に対する分割征服法

剰余列計算は二つの整数を組み合わせて最上位桁から順に "消去" していく算法であり、Lehmer の方法も基本的にそうである。巨大な整数の GCD 計算では GCD 自体も大きいことが多く、上位桁から順に消去するのは実際的である。一方、整数・有理数変換では $\gcd(M, S) \ll S$ であることが多く、S の上位桁の約半分を消去すれば十分である。そのことを考慮すれば効率的な算法を構成できる。本節では分割征服に基づくアイデアを述べる。

整数 S は 底を B として次のように k 桁で表現されているとする（通常の計算機では大整数は、$B=2^{24}$ あるいは $B=10^8$ などとして、そのように表現される）。

$$S = s_kB^k + s_{k-1}B^{k-1} + \cdots + s_1B + s_0, \quad B > s_i \geq 0 \quad (i = k, \ldots, 1, 0). \tag{8}$$

さて、U と V は $U > V \gg 1$ を満たす巨大な整数で、M と S から生成される剰余列の 相続く 2要素であるとし、次のように表現されているとする（$u_m \neq 0$）。

$$\begin{aligned} U &= u_mB^m + u_{m-1}B^{m-1} + \cdots + u_1B + u_0, \qquad B > u_i \geq 0 \quad (i = m, \ldots, 1, 0), \\ V &= v_mB^m + v_{m-1}B^{m-1} + \cdots + v_1B + v_0, \qquad B > v_i \geq 0 \quad (i = m, \ldots, 1, 0). \end{aligned} \tag{9}$$

まず、U と V を次のように上位 $m-m'$ 桁と下位 m' 桁に分割する。

$$\begin{aligned} &U = U'B^{m'} + U'', \quad U' = u_mB^{m-m'} + \cdots + u_{m'}, \quad U'' = u_{m'-1}B^{m'-1} + \cdots + u_0, \\ &V = V'B^{m'} + V'', \quad V' = v_mB^{m-m'} + \cdots + v_{m'}, \quad V'' = v_{m'-1}B^{m'-1} + \cdots + v_0. \end{aligned} \tag{10}$$

つぎに、U と V の上位約 $m/2$ 桁の消去を三つのステップに分けて実行する。

Step 1 : $m' \simeq m/2$ とし、U' と V' の上位約 $m/4$ 桁を消去する、
Step 2 : その消去結果と U'', V'' から約 $3m/4$ 桁の整数 $\tilde{U}, \tilde{V}$ をつくる、
Step 3 : $m' \simeq m/4$ とし、$\tilde{U}$ と $\tilde{V}$ の上位約 $m/4$ 桁を消去する。

Step 1 で U' と V' の上位約 $m/4$ 桁を消去するならば、剰余列の 相続く 2要素として、約 $m/4$ 桁の次なる整数 $\hat{U}'$ と $\hat{V}'$ が得られる。

$$\hat{U}' = c'_uU' + d'_uV', \quad \hat{V}' = c'_vU' + d'_vV'. \tag{11}$$

整数 c'_u, d'_u, c'_v, d'_v は約 $m/4$ 桁である。Step 2 では、$\tilde{U}$ と $\tilde{V}$ を次式で計算する。

$$\begin{aligned} \tilde{U} = c'_uU + d'_uV &= \hat{U}'B^{m'} + c'_uU'' + d'_uV'', \\ \tilde{V} = c'_vU + d'_vV &= \hat{V}'B^{m'} + c'_vU'' + d'_vV''. \end{aligned} \tag{12}$$

このとき $\tilde{U}$ と $\tilde{V}$ は約 $3m/4$ 桁 (以下) となるので、Step 2 の主張は正しい。

さて、Step 3 で $\hat{U}''$ と $\hat{V}''$ の上位約 $m/2$ 桁を取り出して上位約 $m/4$ 桁を消去すれば、剰余列の 相続く 2要素として、約 $m/2$ 桁の次なる整数 $\hat{U}, \hat{V}$ が得られる。

$$\hat{U} = c''_u\tilde{U} + d''_u\tilde{V}, \quad \hat{V} = c''_v\tilde{U} + d''_v\tilde{V}. \tag{13}$$

整数 $c''_u, d''_u, c''_v, d''_v$ は約 $m/4$ 桁である。$\hat{U}$ と $\hat{V}$ を U と V で表すと次のようになる。

$$\begin{aligned} &\hat{U} = \hat{c}_uU + \hat{d}_uV, \qquad \hat{c}_u = c''_uc'_u + d''_uc'_v, \quad \hat{d}_u = c''_ud'_u + d''_ud'_v, \\ &\hat{V} = \hat{c}_vU + \hat{d}_vV, \qquad \hat{c}_v = c''_vc'_u + d''_vc'_v, \quad \hat{d}_v = c''_vd'_u + d''_vd'_v. \end{aligned} \tag{14}$$

初期設定を $U=M$, $V=S$ とすれば、k 桁の整数 M と S の上位約 $k/2$ 桁の消去が、$k/2$ 桁の整数の上位約 $k/4$ 桁の二つの消去に帰着されるので、この消去手順を再帰的に適用することにより、問題が分割征服されることになる。

4 方法の詳細と正しさの証明

剰余列の各要素 S_i は 余因子 C_i と D_i で (4) のように表され、余因子は商列 $(Q_1, Q_2, \cdots, Q_{n-1})$ のみから計算される。したがって、前節で述べた算法で検討すべきは次の2点である。第1点:$U \gg V$ のとき U と V を上位桁と下位桁へ分割する方法、第2点:U', V' から計算される商列が U, V から計算される商列と等しくなるための条件。第一点は、$v_m \neq 0$ であれば問題はなく、V の上位の桁のいくつかが0の場合が問題となる。そこで、計算は次の二つの場合、正常な場合と非正常な場合、に分けて行われる。

正常な場合： (U' の桁数) $\simeq$ (V' の桁数) $\gg 1$ の場合。
この場合は前節で述べた分割征服の方針どおりに分割 (10) を行い、消去を実行する。

2整数 $U_0 = U$, $U_1 = V$ から生成される剰余列を $(U_0, U_1, \cdots, U_{\nu-1}, U_\nu, U_{\nu+1})$ とし、付随する商列を $(q_1, \cdots, q_{\nu-1}, q_\nu)$ とする。また、$U'_0 = U'$, $U'_1 = V'$ から生成される剰余列を $(U'_0, U'_1, \cdots, U'_{\nu-1}, U'_\nu, U'_{\nu+1})$ とし、付随する商列を $(q'_1, \cdots, q'_{\nu-1}, q'_\nu)$、余因子列を $(c'_0, c'_1, \cdots, c'_{\nu-1}, c'_\nu, c'_{\nu+1})$, $(d'_0, d'_1, \cdots, d'_{\nu-1}, d'_\nu, d'_{\nu+1})$ とする。実際に生成されるのは後者の列であって、U_0, U_1 からは生成されないことに注意されたい。商列 $(q_1, q_2, \cdots)$ と $(q'_1, q'_2, \cdots)$ は、対応する要素同士を比べると、最初の方は等しいが、ある段階から異なってくるはずである。そこで、次のようになる条件を求める。

$$q_i = q'_i \quad (i = 1, \cdots, \nu-1), \qquad q_\nu \neq q'_\nu. \tag{15}$$

以下では、U と V を (10) の U', V' を用いて次のように表す。

$$U = (U' + \varepsilon_u) \times B^{m'}, \quad V = (V' + \varepsilon_v) \times B^{m'}, \quad 1 > \varepsilon_u, \varepsilon_v \geq 0. \tag{16}$$

実際のプログラムでは ε_u と ε_v は $\varepsilon_u = (u_{m'-1} + 1)/B$, $\varepsilon_v = (v_{m'-1} + 1)/B$ と計算する。

命題 1
剰余列 $(U'_0, U'_1, \cdots, U'_{\nu-1}, U'_\nu, U'_{\nu+1})$ と対応する商列および余因子列に

$$\begin{aligned} &\text{for } j < \nu: \quad -(c'_{j+1}\varepsilon_u + d'_{j+1}\varepsilon_v) \leq U'_{j+1} < U'_j - (c'_{j+1} - c'_j)\varepsilon_u - (d'_{j+1} - d'_j)\varepsilon_v, \\ &-(c'_{\nu+1}\varepsilon_u + d'_{\nu+1}\varepsilon_v) > U'_{\nu+1} \quad \text{or} \quad U'_{\nu+1} \geq U'_\nu - (c'_{\nu+1} - c'_\nu)\varepsilon_u - (d'_{\nu+1} - d'_\nu)\varepsilon_v. \end{aligned} \tag{17}$$

が成立すれば、(15) が成立する。

証明 $i = 1, \ldots, j-1$ の場合に $q_i = q'_i$ を仮定し、$i = j$ の場合を考える。仮定より、

$$c'_i U' + d'_i V' = U'_i, \quad c'_i U + d'_i V = U_i \qquad (i = 1, 2, \ldots, j)$$

である。したがって、U_{j+1} は次のように表すことができる。

$$\begin{aligned} U_{j+1} &= \mathrm{rem}(U_{j-1}, U_j) = (c'_{j-1} - q_j c'_j)\, U + (d'_{j-1} - q_j d'_j)\, V \\ &= [(c'_{j-1} - q_j c'_j)\,(U' + \varepsilon_u) + (d'_{j-1} - q_j d'_j)\,(V' + \varepsilon_v)] \times B^{m'} \\ &= [U'_{j-1} - q_j U'_j + (c'_{j-1} - q_j c'_j)\,\varepsilon_u + (d'_{j-1} - q_j d'_j)\,\varepsilon_v] \times B^{m'} \\ &= [U'_{j-1} - q'_j U'_j + (c'_{j-1} - q'_j c'_j)\,\varepsilon_u + (d'_{j-1} - q'_j d'_j)\,\varepsilon_v] \times B^{m'} \\ &\qquad + (q'_j - q_j)\,(U'_j + c'_j\varepsilon_u + d'_j\varepsilon_v) \times B^{m'}. \end{aligned}$$

ここで、$U'_{j-1} - q'_j U'_j = U'_{j+1}$ であり、また $U'_j + c'_j\varepsilon_u + d'_j\varepsilon_v = c'_j(U' + \varepsilon_u) + d'_j(V' + \varepsilon_v) = (c'_j U + d'_j V)/B^{m'}$ なので、上式の最後の行は $(q'_j - q_j)\,U_j$ に等しい。

もしも $q_j = q'_j$ ならば、$0 \le \mathrm{rem}(U_{j-1}, U_j) < U_j$ より次式が成立しなければならない。

$$0 \ \le \ U'_{j+1} + c'_{j+1}\varepsilon_u + d'_{j+1}\varepsilon_v \ < \ U'_j + c_j\varepsilon_u + d_j\varepsilon_v. \tag{18}$$

同様に、もしも $q_j \neq q'_j$ ならば次式が成立しなければならない。

$$\text{if } q_j > q'_j: \quad U'_{j+1} + c'_{j+1}\varepsilon_u + d'_{j+1}\varepsilon_v \ge U'_j + c'_j\varepsilon_u + d'_j\varepsilon_v,$$
$$\text{if } q_j < q'_j: \quad U'_{j+1} + c'_{j+1}\varepsilon_u + d'_{j+1}\varepsilon_v < 0.$$

これらは (18) の条件下では成立しないので、$q_j = q'_j$ となるには条件 (18) で十分である。$j < \nu$ では (18) が成立し $j = \nu$ では成立しないことより、(17) を得る。 ∎

非正常な場合： V' の桁数が U' の桁数よりかなり小さい場合。
この場合、前節に述べた分割では V' の精度が不足して、商が正確に計算できないことがある。このような非正常な場合には、V' の桁数を大きくとって（m' の値を小さくして）一度だけ剰余を計算する。こうすることにより、多くの場合、剰余列は正常に戻るが、戻らなければ再び非正常処理をすればよい。

命題 2
次式を満たすように m' を決めれば、$\mathrm{quo}(U,V) = \mathrm{quo}(U',V')$ となる。

$$\mathrm{quo}(U',V')\,\varepsilon_v \le \mathrm{rem}(U',V') < V' + \mathrm{quo}(U',V')\,\varepsilon_v - 1. \tag{19}$$

証明 $q = \mathrm{quo}(U,V)$, $q' = \mathrm{quo}(U',V')$ とおき、U と V を次のように表す。

$$U = (U' + \varepsilon_u) \times B^{m'}, \quad V = (V' + \varepsilon_v) \times B^{m'}, \quad 1 > \varepsilon_u, \varepsilon_v \ge 0. \tag{20}$$

このとき、$\mathrm{rem}(U,V) = U - qV = [(U' - qV') + \varepsilon_u - q\varepsilon_v]B^{m'} = [(U' - q'V') + \varepsilon_u - q'\varepsilon_v + (q'-q)(V' + \varepsilon_v)]B^{m'}$ であり、$U' - q'V' = \mathrm{rem}(U',V')$ である。
もしも $q = q'$ ならば、$0 \le \mathrm{rem}(U,V) < V$ より次式が成立しなければならない。

$$0 \le \mathrm{rem}(U',V') + \varepsilon_u - q'\varepsilon_v < V' + \varepsilon_v. \tag{21}$$

同様に、もしも $q \neq q'$ ならば次式が成立しなければならない。

$$\text{if } q > q': \quad \mathrm{rem}(U',V') + \varepsilon_u - q'\varepsilon_v \ge V' + \varepsilon_v,$$
$$\text{if } q < q': \quad \mathrm{rem}(U',V') + \varepsilon_u - q'\varepsilon_v < 0.$$

これらは (21) の条件下では成立せず、条件 (21) は (19) で満たされる。 ∎

m_1 桁と m_2 桁の大きな整数の乗算の計算量を $M(m_1, m_2)$ とする。簡単のため剰余列は常に正常であると仮定して、分割征服法の計算量 $T(k)$ を求めよう。Step I の計算量は $T(k/2)$、Step II の計算量は $4M(k/4, k/2)$、Step 3 では (14) も計算されるので計算量は $T(k/2) + 4M(k/4, 3k/4) + 8M(k/4, k/4)$ となり、次の漸化式を得る.

$$T(k) = 2T(k/2) + 4M(k/4, k/2) + 8M(k/4, k/4) + 4M(k/4, 3k/4). \tag{22}$$

上式の右辺で $M(k/4, 3k/4) \le 3/2M(k/4, k/2) \le 3M(k/4, k/4)$ を使って簡単化する。

$$T(k) \le 2^{\log_2 k}T(1) + 7\int_1^{\log_2 k} 2^{x+1}M(k/2^{x+1}, k/2^{x+1})dx$$

たとえば、Karatsuba の算法では $M(m,m) \approx 9m^{1.59}$ であるから、$T(k) \approx 45k^{1.59}$ を得る。一方、拡張互除法を効率化した Lehmer 法の計算量は $2.5k^2$ なので、分割征服法は巨大整数に対して Lehmer 法より有効なはずである。

5 計算機への実装と実験

整数や有理数に関する演算は最も基本的なものなので、アルゴリズムの実装はシステムの一番下のレベルまで立ち入って、アセンブリ語でプログラムすべきである。しかしながら、上述のアルゴリズムについては、有効性をテストしている段階なので、我々は次のようにインプリメンテーションを行い、実験した。

1. Big-integer の内部表現を Lisp のリストでシミュレートする。
2. すべてのプログラムを Lisp で書く。
3. 底 B としては $B = 10^8$ あるいは $B = 2^{24}$ とする。

表は Euclid 法と Lehmer 法および分割征服法を上記のようにプログラムし、性能を比較したものである。データはランダムに生成した異なる 10 個のサンプルに対する平均である。

k の値が比較的小さくても Lehmer 法は効率的であることが分かる。同時に、k が大きくなっても Lehmer 法は Euclid 法に比べて一定倍しか速くないことも分かる。このことは、Euclid 法に比べて Lehmer 法は「巨大整数 × 小整数」の計算回数が約 1/10 ~ 1/15 であることから、当然のことである。Lehmer 法と分割征服法については、k の値が比較的小さくても分割征服法が効率的であり、桁数が増加するにつれてその効率が向上するという結果を得た。分割を再帰的に行っているので、扱うデータ量が小分けされてキャッシュメモリに十分収まることも高効率に寄与していると考えられる。

実験環境 OS : Linux 2.2.17 (Vine Linux)、総メモリ : 320MB、CPU : Pentium II 300MHz.

実験結果 (U, V それぞれ k 桁 ($100 \leq k \leq 1000$) の整数について上位約 $k/2$ 桁を消去)

	$B = 10^8$ の場合 (単位 : sec)			$B = 2^{24}$ の場合 (単位 : sec)		
桁数	Euclid	Lehmer	分割征服	Euclid	Lehmer	分割征服
100	0.896	0.212	0.0530	0.778	0.213	0.0540
200	3.41	0.778	0.129	2.94	0.783	0.124
300	7.66	1.70	0.220	6.56	1.71	0.21
400	13.5	2.98	0.332	11.6	3.01	0.311
500	21.1	4.62	0.465	17.6	4.65	0.438
600	30.0	6.64	0.595	26.0	6.69	0.561
700	40.8	8.99	0.773	35.3	9.02	0.724
800	53.5	11.7	0.927	46.0	11.8	0.863
900	67.3	14.8	1.12	58.5	14.9	1.04
1000	83.2	18.2	1.34	71.7	18.3	1.26

参 考 文 献

[1] D. E. Knuth: *The Art of Coputer Programming*, Vol. 2 (Seminumerical Algorithm). 1969, Section 4.5.

[2] T. Sasaki and M. Sasaki: On Integer-to-Rational Conversion Algorithm. SIGASAM Bulletin, Vol. 26, ACM, 1992, pp. 19–21.

[3] T. Sasaki and T.Takeshima: A Modular Method for Gröbner-basis Computation over **Q** and Solving System of Algebraic Equation. J. Inf. Proces., Vol. 12, 1989, pp. 371–379.

[4] A. Schönhage: Schnelle Berechnung von Kettenbruchentwicklungen. Acta Informatica 1, 1971, pp. 139–144.

[5] P. S. Wang: A p-adic Algorithm for Univariate Partial Fractions. Proc. ACM Symp. on Symbolic and Algebraic Computation, 1981, pp. 212–217.

[6] P. S. Wang, M. J. T. Guy and J. H. Davemport: P-adic Reconstruction of Rational Numbers. SIGSAM Bulletin, Vol. 16, ACM, 1982, pp. 2–3.