研究ノート

大きな整数の除算アルゴリズム

野　呂　春　文
日本福祉大学　情報社会科学部

Division algorithm for BigInteger

Harufumi Noro
Faculty of Social and Information Sciences, Nihon Fukushi University

Keywords: division algorithm, Knuth' s algorithm D, Binary division algorithm, Big Integer

1．はじめに

はじめに以下で用いる言葉を整理しておこう．除算とは桁数制限の無い大きな整数a, bに対してaをbで割ったときの商を計算することを言い，割り算とは（除算を構成する部品となるかもしれない）レジスタ上で処理できる小さな整数u，vに対してuをvで割った商を計算することだとする．

基本的な演算であるにも関わらず，加算や乗算にくらべて，除算はむずかしい．アルゴリズムが複雑になりがちでプログラミングミスが混入しやすくデバッグに手間がかかる．Microsoft Excel上に大きな整数とその演算を実現する際に最も時間を要したのが除算であった[1]．

現代の強力なコンピュータのもとでは，桁数制限の無い大きな整数とその演算を実現するとき，古典的なコンピュータで常識とされた2を基数とする表現（2進数）ではなく，より大きな数Bを基数とするのが通例である．例えばJavaのBigIntegerクラスでは2^{15}を基数とし，和田[2]は2^7を基数として採用している．既報の「大きな整数」では$10^4 = 10000$を基数としている．このような基数のもとでは2進数を前提とした古典的でエレガントなアルゴリズムが有効でなくなることが多い．除算もその典型例である．

基数Bの整数xを実際に計算機上に表現するには次のようにする．簡単のために正の整数としよう．n+1桁の整数xを多項式で表わすと，$x = x(n)B^n + x(n-1)B^{(n-1)} + \ldots + x(2)B^2 + x(1)B + x(0)$と書くことができる．ここで，$B > x(n) > 0$，$B > x(k) \geqq 0$である．x(0), x(1), ..., x(n)をそれぞれ適当な配列の要素に格納すれば任意の大きな桁数の整数が実現できる．32ビットLong型整数の配列を使い，$B = 2^{15}$とするのがJavaのBigIntegerクラスである．和田は，16ビットShort型整数と$B = 2^7$を使い，筆者の大きな整数では32ビットLong型整数とB = 10000を使っている．ワークステーション上のC++では，もっと大きな64ビットLong型整数と$B = 2^{31}$が使われることがある．もちろん，実際にプログラミングシステムの上に実現するときは，適宜，構造体等を用いて付随的な情報を同時に取り扱えるようにするが，それらは以下の説明において本質的ではないので省略している．

以下の説明においては，問題を具体的に示すために10進表現された整数を取り扱っている．しかし，以下

のすべてのアルゴリズムは任意の基数を使えるように書かれている．基数はBと表記されている．

ここでは，除算アルゴリズムを実際に開発したプロセスを紹介する．初めに，最も単純で容易にプログラミングできるが，しかし計算効率が最低なアルゴリズムを説明する．そこから出発して，効率の良いアルゴリズムを導く．アルゴリズムはCあるいはJava類似の擬似コードで記述する．なお，割り算a / bは，a / bを越えない最大の整数を返す演算を表わしている．

2．古典的な2進除算アルゴリズムとその10進版

2進表現された整数に対して，本質的には割り算も掛け算も不要な除算アルゴリズムを構成できることは広く知られている．それをCrandall and Pmerance [3] やKnuth [4] は「古典的なアルゴリズム」と呼んでいる．きわめて魅力的なアルゴリズムなので，これをもとに10進版の開発を試みる．

アルゴリズム1　2進除算アルゴリズム

2個の大きな整数x，Nが$x \geqq N > 0$を満たすとする．そのとき整数$c = [x / N]$を計算する．cはx / Nを超えない最大の整数である．

(1) 初期化：$2^b \times N \leqq x < 2^{(b+1)} \times N$となる整数bを求め，$m = 2^b \times N$とする．$c = 0$とする．

(2) 除算ループ：

```
for (i = 0 to b) {
    c = 2 × c;
    a = x − m;
    if (a ≧ 0) {
            c = c + 1;
            x = a;
    }
    m = m/2;
}
return c;
```

このアルゴリズムに現れる掛け算と割り算は，ともにトリビアルである．cに2を掛けることはcを左に1ビットシフトすることであり，mを2で割ることはmを右に1ビットシフトすることにすぎないからである．このため，2進除算アルゴリズムは，ビット長がレジスタサイズ程度（32ビットあるいは64ビット）におさまる小さな整数の場合，きわめて効率的なアルゴリズムになっている．

そこで上のアルゴリズム1を改変して，10進表現（B進表現）された整数に適用できるように変形して試すことにする．

アルゴリズム2　10進除算アルゴリズム（2進除算アルゴリズム変形版）

2個の大きな整数x，Nが$x \geqq N > 0$を満たすとする．そのとき整数$c = [x / N]$を計算する．

(1) 初期化：$B = 10$とおく．$B^b \times N \leqq x < B^{(b+1)} \times N$となる整数bを求め，$m = B^b \times N$とする．$c = 0$とする．

(2) 除算ループ

```
for (i = 0 to b) {
    c = B × c;
    for (j = 1 to (B − 1)) {
            a = x − m;
            if (a < 0) {
                    exit innermost for loop;
            }
            c = c + 1;
            x = a;
    }
    m = m / B;
}
returun c;（商はc，余りはxである．）
```

これで本当に除算が可能なのか，わかり難いので，1つ例を見てみよう．x = 87654321, N = 2345とする．

xが8桁，Nが4桁だからb = 4である．m = 23450000，c = 0が初期値である．

```
i = 0：  c = 0, x = 87654321, m = 23450000
         j = 1, a = 64204321, c = 1
         j = 2, a = 40754321, c = 2
         j = 3, a = 17304321, c = 3
         j = 4, a = − 6145679
i = 1：  c = 30,  x = 17304321, m = 2345000
         j = 1, a = 14959321, c = 31
         途中省略
         j = 7, a = 889321, c = 37
```

j = 8, a = － 1455679

i = 2：　c = 370, x = 889321 m = 234500

j = 1, a = 654821, c = 371

j = 2, a = 420321, c = 372

j = 3, a = 185821, c = 373

j = 4, a = － 48679

i = 3：　c = 3730, x = 185821, m = 23450

j = 1, a = 162371, c = 3731

途中省略

j = 7, a = 21671, c = 3737

j = 8, a = － 1779

i = 4：　c = 37370, x = 21671, m = 2345

j = 1, a = 19326, c = 37371

途中省略

j = 9, a = 566, c = 37379

c = 37379 が得られた．

以上で見たように，初期化は，除数と被除数の「頭をそろえる」操作を意味している．つまり，整数 b は被除数 x の桁数から除数 N の桁数を引いた値であり，これを求めるには特別な計算を要しない．頭をそろえる操作も単なる桁シフトである．

除算ループ内における c = B × c は c を一桁左へシフトすることであり，掛け算は不要である．また，m = m / B は m を一桁右へシフトすることであり，これも割り算を必要としない．

一方，上の例でも明らかなように，アルゴリズム２の最大の欠点は内側のループにおいて複数回の引き算が実行されることである．商を一桁得るために平均して 5.5 回，最悪９回の引き算がありうる．この例では５桁の商を得るために 34 回の引き算が必要であった．c について何も予測せず，１から始めて総当りするのだから当然とも言える．

このアルゴリズムの中で「本当に」計算を要するのは，この引き算だけである．しかも，大きな桁数の整数の引き算は，通常の Long 型の整数のようにレジスタ上では実行できないので，多くの計算時間を要する．引き算の回数を減らすアルゴリズムを考えないと実用にならない．

２進除算アルゴリズムが簡潔で効率的なのは次の事実による：0 ≦ a < 2b なら a / b は０か１である．

この事実を 10 進（B 進）除算に適用すると，次のようになる：0 ≦ a < B × b なら a / b は０から B － １のいずれかである．これが効率の悪さの原因である．上記の例ではたかだか B=10 にすぎないため，正しい商を得るための繰り返しもかろうじて可能であるが，例えば B=2^15 ならまったく非実用的である．したがって，この方向での開発は断念せざるをえない．

３．小学校の割り算再訪

小学校で習った割り算（本報告で言う除算）にもどって検討してみる．87654321 割る 2345 は図１に示すように計算するのであった．手順を分解してみよう．

```
      37379
2345) 87654321
      70350000
      17304321
      16415000
        889321
        703500
        185821
        164150
         21671
         21105
           566
```

図１小学校の割り算

初めに除数の桁数を見て商の先頭を記入する位置を決める．次に，被除数と除数の先頭１桁の数字を見て仮の商を見積る．この場合 8/2 = ４である．４掛ける除数を被除数から引くと負になるので仮の商を１だけ減らし３として引き算を試す．正になったので成功であり，商の先頭として一桁の値３を記入し，３掛ける除数を被除数から引いた余りを記入する．

商の次の桁の数字は一桁下がった位置になる．ひとつ前に余りとして得られた値を新たな被除数とする．その先頭２桁と除数の先頭１桁を見て商の次の桁として７を記入し，７掛ける除数を被除数から引いて余りを記入する．この手続きを繰り返して結果を得る．ポイントは，商を予測することにある．

以上の観察から次のアルゴリズム３が得られる．

アルゴリズム３　小学校除算アルゴリズム

２個の大きな整数 x，N が x ≧ N > 0 を満たすとする．そのとき整数 c = [x / N] を計算する．

(1) 初期化：　B = 10 とおき，$B^b \times N$ の桁数が x の桁数に等しくなる整数 b を求め，$m = B^b \times N$ とする．m の最上位 1 桁の数を m_h とする．c = 0 とする．

(2) 初段（i = 0）の処理：

```
xh = (x の最上位 1 桁の数 )
q = min(xh / mh, B − 1);
if (q ≧ 0) {
    a = x − q × m;
    if (a ≧ 0) {
            c = c + q;
            x = a;
    }
    else if (q > 1) {
            while (a < 0 and q > 1) {
                    q = q − 1;
                    a = a + m;
            }
            if (q > 0) {
                    c = c + q;
                    x = a;
            }
    }
}
m = m / B;
```

(3) 除算ループ：

```
for (i = 1 to b) {
    c = B × c;
    xh = (x の先頭 2 桁の数 )
    q = min(xh / mh, B − 1)
    if (q ≧ 0) {
            a = x − q × m;
            if ( a ≧ 0) {
                    c = c + q;
                    x = a;
            }
            else if (q > 0) {
                    while ( a < 0 and q > 1) {
                            q = q − 1;
                            a = a + m;
                    }
                    if ( q > 0) {
                            c = c + q;
                            a = x;
                    }
            }
    }
    m = m / B;
}
return c;（商は c，余りは x である．）
```

先の例と同じ数で試して見る．

i = 0：　c = 0, x = 87654321, m = 23450000
　　q = 4, a = − 6145679
　　q = 3, a = 17304321, c = 3

i = 1：　c = 30, x = 17304321, m = 2345000
　　q = 8, a = − 1455679
　　q = 7, a = 889321, c = 37

i = 2：　c = 370, x = 889321, m = 234500
　　q = 4, a = − 48679
　　q = 3, a = 185821, c = 373

i = 3：　c = 3730　　, x = 185821, m = 23450
　　q = 9, a = − 25229
　　q = 8, a = − 1779
　　q = 7, a = 21671, c = 3737

i = 4：　c = 37370, x = 21671, m = 2345
　　q = 9, a = 566, c = 37379

c = 37379 が得られた．大きな引き算の回数は，10 回であった．アルゴリズム 2 では 34 回であったのと比べて大きく改善されている．「不純な」小さな数の割り算を導入して q の近似値を予測することにより大きな利益が得られた．なお，B × c および m / B が単なる桁シフトであることは言うまでも無い．

4．除算アルゴリズム

アルゴリズム 3　小学校除算アルゴリズムには，まだ無駄な引き算が多い．q の推定精度が低いためである．x = 9000, N = 199 という例を見てみよう．

i = 0：　c = 0, x = 9000, m = 1990
　　q = 9, a = − 8910
　　q = 8, a = − 6920
　　q = 7, a = − 4930
　　q = 6, a = − 2940
　　q = 5, a = − 950
　　q = 4, a = 1040, c = 4

i = 1：　c = 40, x = 1040, m = 199
　　q = 9, a = − 751
　　q = 8, a = − 552
　　q = 7, a = − 353
　　q = 6, a = − 154
　　q = 5, a = 45, c = 45

アルゴリズム２に匹敵する非効率さである．

q は被除数の先頭１桁あるいは２桁を除数の先頭１桁で割ることによって推定している．つまり，除数の先頭１桁が決定的な役割を担っている．q の推定精度が低くなるのは，この例に示されているように除数の先頭の数が小さいときである．

そこで，次のような前処理を考える．被除数 x と除数 N の両方に同じ数 d を掛ける．分子分母に同じ数を掛けることになるので，除算の結果は変化しない．d は，d × N の桁数が N と変わらず，N の先頭の１桁が B / 2 以上 B − 1 以下でなるべく大きくなるように決める．

アルゴリズム４　係数 d の決定

N の先頭２桁の数からなる値を n とする．N が１桁であればそれを n とする．n は正の整数である．

```
B = 10;
if (n < B) { d = B / (n+1);}
else { d = B*B / (n+1);}
return d;
```

この前処理は大きな整数 x と１桁の整数 d との掛け算である．そのためには，大きな整数同士の掛け算用の汎用ルーチンではなく，次の特化したアルゴリズムを使うことができる．

アルゴリズム５　１桁の整数による乗算

大きな整数 x > 0 は n 桁とする．整数 d > 0 は１桁とする．このとき w = x × d を計算する．x と w はそれぞれ x(1) x(2) … x(n)，w(0) w(1) w(2) … w(n) と B 進表記されるものとする．x(1)，w(0) が最上位桁である．w(0) = 0 になることもある．

(1) 初期化：　B = 10; r = 0;

(2) 乗算ループ：

```
for ( j = n to 1 step − 1) {
    w( j ) = (B × x( j ) + r) mod B;
    r = (B × x( j ) + r) / B
}
w(0) = r;
return w;
```

すべての道具がそろったので除算アルゴリズム６を書くことができる．

アルゴリズム６　除算アルゴリズム

２個の大きな整数 x，N が x ≧ N > 0 を満たすとする．そのとき整数 c = [x / N] を計算する．

(1) 初期化：　アルゴリズム４によって d を求め，アルゴリズム５を使って x = d×x, N = d×N とする．B = 10 とおき，$B^b \times N$ の桁数が x の桁数に等しくなる整数 b を求め，$m = B^b \times N$ とする．m の最上位１桁の数を m_h とする．c = 0 とする．

(2) 初段（i = 0）の処理：

```
x_h = (x の先頭 1 桁の数 )
if (x_h ≧ m_h){
    a = x − m;
    if(a ≧ 0) {
        c = c + 1;
        x = a;
}
m = m / B;
```

(3) 除算ループ：

```
for (i = 1 to b) {
    c = B × c;
    x_h = (x の先頭 2 桁の数 )
    q = min(x_h / m_h, B − 1)
    if (q ≧ 0) {
        a = x − q × m;
        if ( a ≧ 0) {
            c = c + q;
            x = a;
        }
        else if (q > 0) {
            while ( a < 0 and q > 1) {
                q = q ? 1;
                a = a + m;
            }
            if ( q > 0) {
                c = c + q;
```

```
                    a = x;
                }
            }
        }
        m = m / B;
    }
    return c; (商は c，余りは x/d である．1 桁の整数による除算は後述のアルゴリズム 7 を使う．)
```

初段（i = 0）の処理において，x の先頭は 1 以上 B − 1 以下，m の先頭は B / 2 以上 B − 1 以下なので，c の値は 0 あるいは 1 だけである．そのため上記のように簡単になる．

例を試して見る．x = 87654321, N = 2345 より，d = 4 となり，x = d × x = 350617284, N = d × N = 9380 である．x が 1 桁増えるので b = 5 である．

i = 0：　c = 0, x = 350617284, m = 938000000, xh = 3 < mh = 9
i = 1：　c = 0, x = 350617284, m = 93800000
　　q = 3, a = 69217284, c = 3
i = 2：　c = 30, x = 69217284, m = 9380000
　　q = 7, a = 3557284, c = 37
i = 3：　c = 370, x = 3557284, m = 938000
　　q = 3, a = 743284, c = 373
i = 4：　c = 3730, x = 743284, m = 93800
　　q = 8, a = − 7116
　　q = 7, a = 86684, c = 3737
i = 5：　c = 37370, x = 86684, m = 9380
　　q = 9, a = 2264, c = 37379

大きな引き算の回数は 6 回であった．では，アルゴリズム 3 小学校除算アルゴリズムが苦手であった x = 9000, N = 199 の例ではどうなるであろうか．d = 5 より x = 45000, N = 995 である．

i = 0　　c = 0, x = 45000, m = 99500, $x_h = 4 < m_h = 9$
i = 1　　c = 0, x = 45000, m = 9950
　　q = 5, a = − 4750
　　q = 4, a = 5200, c = 4
i = 2　　c = 40, x = 5200, m = 995
　　q = 5, a = 225, c = 45

11 回必要だった引き算が 3 回になる．

以上に示されたように，アルゴリズム 6 では，除数と被除数に対する前処理（大きな整数と 1 桁の整数の掛け算）という代償を支払うことによって大きな引き算の回数を少なくすることが可能になった．

アルゴリズム 6 の最後に，余りを計算するための x / d についての言及がある．1 桁の d による除算は特別なので，ここでアルゴリズムを示す．

アルゴリズム 7　1 桁の整数による除算

大きな整数 x > 0 は n 桁とする．整数 d > 0 は 1 桁とする．このとき w = [x / d] を計算する．x と w はそれぞれ x(1) x(2) ... x(n)，w(1) w(2) ... w(n) と B 進表記されるものとする．w(1) = 0 になることもある．

(1) 初期化：B = 10; r = 0;
(2) 除算ループ

```
for ( j = 1 to n ){
    w( j ) = (B × r + x( j )) / d;
    r = (B × r + x( j )) mod d;
}
return w;
```

5．他のアルゴリズムとの比較

大きな整数に対する除算アルゴリズムとしては，次に示す Knuth のアルゴリズム D が有名である．それを筆者のアルゴリズムと比較する．

Knuth による説明と証明

Knuth のアルゴリズムは，小学校除算アルゴリズムおよびアルゴリズム 6 と同様に，長い除算は，n+1 桁の数 u を n 桁の除数 v で割る，より簡単な段階に分かれる，という観察に基づいている．ただし，0 ≦ u/v < B である．

u = u(0)u(1)...u(n)，v = v(1)v(2)...v(n) を基数 B で表わした負でない整数であって u/v < B を満たすとする．q = [u/v] を決める算法が必要である．

条件 u/v < B は u/B < v という条件と同じである．すなわち，[u/B] < v である．これは，u(0)u(1)...u(n-1) < v(1)v(2)...v(n) と同じである．さらに r = u − qv と書くと，q は 0 ≦ r < v を満たす一意に決まる整数である．

q を u と v の最上位桁によって推定する．q' = minimum{[(u(0)B+u(1))/v(1)], B − 1} とおく．

定理 A：q' ≧ q である．

証明：q' = B − 1 なら正しい．そうでなければ，q' = [(u(0)B+u(1))/v(1)] だから，(u(0)B+u(1))/v(1) − 1 < q'．両辺に v(1) を掛けて，u(0)B+u(1) − v(1) < q' v(1)．両辺とも整数だから両者の差は 1 以上である（ここがポイント）．したがって，u(0)B+u(1) − v(1) + 1 ≦ q' v(1) が成り立つ．この結果，

u − q' v　≦ u − q' v(1)B^(n-1)　≦ u(0)B^n + ... + u(n) − (u(0)B^n + u(1)B^(n-1) − v(1)B^(n-1) + B^(n-1)) = u(2)B^(n-2) + … + u(n) − B^(n-1) + v(1)B^(n-1) < v(1)B^(n-1) ≦ v.

u − q'v < v であるから q' ≧ q でなければならない．証明終わり．

定理B：v(1) ≧ [B/2] なら q'−2 ≦ q ≦ q' + 2 である．

証明：q' ≧ q + 3 と仮定する．

q' = [(u(0)B+u(1))/v(1)]　≦ (u(0)B+u(1))/v(1) = (u(0)B^n+u(1)B^(n-1))/v(1)B^(n-1)　≦ u/v(1)B^(n-1) < u/(v − B^(n-1))．さらに，q > (u/v) − 1 より

3 ≦ q' − q < u/(v − B^(n-1)) − u/v + 1 = (u/v) × (B^(n-1)/(v − B^(n-1))) + 1.

したがって，

u/v > 2 × ((v − B^(n-1))/B^(n-1)) ≧ 2(v(1) − 1).

そして，B − 4 ≧ q' − 3 ≧ q = [u/v] ≧ 2(v(1) − 1) であるから，v(1) < [B/2] であり矛盾する．証明終わり．

定理 A および定理 B より，基数 B がどんなに大きくても試算の商 q' の誤差は 2 より大きくなることはない，という結論が得られる．

Knuth のアルゴリズム D

B 進表現された 2 つの整数 u = u(1)u(2)...u(m+n) と v = v(1)v(2)...v(n), ただし n ≧ 1，v(1) ≠ 0 より，基数 B の商 u / v = q(0)q(1)...q(m) と余り u mod v = r(1)r(2)...r(n) を計算する．n = 1 の場合は，上で説明したアルゴリズム 7 を使う．

(1) 正規化　d = B / (v(1)+1) を計算し，u = u × d = u(0)u(1)u(2)...u(m+n)，v = v × d = v(1)v(2)...v(n) とする．

(2) j の初期設定　j = 0 とおく．(2) から (7) までの繰り返しは，u(j)u(j+1)...u(j+n) を v(1)v(2)...v(n) で割った 1 桁の商 q(j) をもとめることである．

(3) q' の計算　if (u(j) = v(1)) { q' = B − 1 } else { q' = (u(j) × B+u(j+1)) / v(1) }．ここで，v(2) × q' > (u(j) × B+u(j+1) ? q' × v(1)) × B + u(j+2) なら q' = q' − 1 とする．この検査を繰り返して q' を決める．

(4) 乗算と減算 u(j)u(j+1)...u(j+n) = u(j)u(j+1)...u(j+n) − q' × v(1)v(2)...v(n) とする．結果が負であれば Bn+1 を加え，左からの借りを記録する．

(5) 剰余の検査　q(j) = q' とし，(4) の結果が負なら (6) へ，そうでなければ (7) へ行く．

(6) 加え戻し　q(j) を 1 減らし，u(j)u(j+1)...u(j+n) に v(1)v(2)...v(n) を加える．

(7) j に関する繰り返し　j = j + 1．ここで j ≦ m なら (3) へ戻る．

(8) 逆正規化 q(0)q(1)...q(m) が商であり，余りは u(m+1)u(m+2)...u(m+n) を d で割ってえられる．

Knuth のアルゴリズム疑似コードはアセンブラによる実現を強く意識しているため，現代の高級言語風の疑似コードを見慣れた目には構造がわかりにくい．このアルゴリズムの本体は (3) から (7) までのループなので，ループ構造を明示する疑似コードに書き直す．

Knuth のアルゴリズム D　高級言語風疑似コード

```
for (j = 0 to m) {
        if (u(j) = v(1)) { q' = B ? 1; } else { q' = {
        (u(j)*B+u(j+1)) / v(1); }
        while ( q' *v(2) > (u(j)*B+u(j+1)?q'
        *v(1))*B+u(j+2) ) { q' = q' ? 1; }
        u = u ? q' *v;
        q(j) = q' ;
        if ( u < 0) {
                q(j) = q(j) ? 1;
                u = u + v;
        }
}
```

Knuth の (3) は，q の推定をより精密におこなっている．その分だけ大きな整数の引き算の回数が減る．q の計算は，B があまり大きくないかぎりレジスタ上で可能である．一方，大きな整数の引き算はレジスタ上では不可能なのが普通である．レジスタ上で済む計算のほうが効率的なのは明らかなので，(3) によって効率が向上する．

(4),(5),(6) の順番は奇妙に見える．(4) で u が負なら，q' = q' − 1, u = u + v を先に実行すれば (5), (6) は不要

になる．これは，「(4) で u が負になる確率は 2/B を越えない」という事実に基づいているからである．まず，多く起こることについて処理し，Go To 命令を忌避しない，というアセンブラ・プログラマの立場に立てば不可解なコードではない．しかし，高級言語によるプログラムでは避けたほうが賢明であろう．

以上の解析から，やはり Knuth のアルゴリズムには一日の長無しとしない．そこで，ここで学んだことを織り込んだアルゴリズムが次である．これを筆者の完成版とする．

アルゴリズム８　除算アルゴリズム完成版

２個の大きな整数 x，N が x ≧ N > 0 を満たすとする．そのとき整数 c = [x / N] を計算する．

(1) 初期化：　アルゴリズム４によって d を求め，アルゴリズム５を使って x = d×x, N = d×N とする．B = 10 とおき，Bb × N の桁数が x の桁数に等しくなる整数 b を求め，m = Bb × N とする．m の最上位 1 桁の数を mh，先頭から２桁目の数を ms とする．c = 0 とする．

(2) 初段（i = 0）の処理：

```
xh = (x の先頭 1 桁の数 )
if (xh ≧ mh){
    a = x − m;
    if(a ≧ 0) {
            c = c + 1;
            x = a;
    }
    m = m / B;
```

(3) 除算ループ：

```
for (i = 1 to b) {
    c = B × c;
    xh = (x の先頭 2 桁の数 )
    xs = (x の先頭から 3 桁目の数 )
    q = min(xh / mh, B?1)
    while ( q × ms > (xh − q × mh) × B + xs ) {
            q = q − 1;
    }
    if (q ≧ 0) {
            a = x − q × m;
            if ( a ≧ 0) {
                    c = c + q;
                    x = a;
            }
            else if (q > 0) {
                    while ( a < 0 and q > 1) {
                            q = q − 1;
                            a = a + m;
                    }
                    if ( q > 0) {
                            c = c + q;
                            a = x;
                    }
            }
    }
    m = m / B;
}
return c;（商は c，余りは x/d である．1 桁の整数による除算はアルゴリズム７を使う．）
```

数値例を見てみよう．

i = 0：　c = 0, x = 350617284, m = 938000000, x_h = 3 < mh = 9, c = 0

i = 1：　c = 0, x = 350617284, m = 93800000, q = 3, 9 < 80, a = 69217284, c = 3

i = 2：　c = 30, x = 69217284, m = 9380000, q = 7, 21 < 62, a = 3557284, c = 37

i = 3：　c = 370, x = 3557284, m = 938000, q = 3, 9 < 85, a = 743284, c = 373

i = 4：　c = 3730, x = 743284, m = 93800, q = 8, 24 > 23
q = 7, 21 < 113, a = 86684,c = 3737

i = 5：　c = 37370, x = 86684, m = 9380, q = 9, 27 < 56, a = 2264, c = 37379

Knuth に習って追加した小さな計算によって，大きな整数の引き算を 1 回省くことができた．例とした小さな桁数の計算では，大きな整数の引き算の節約はたいしたことではない．しかし，数 10 桁から数 100 桁におよぶ数論や暗号学の計算においては，このような節約の効果が大きい．

６．おわりに

大きな整数の除算に的をしぼってアルゴリズムの改良プロセスを説明した．小さな整数計算の世界では２進除算アルゴリズム（アルゴリズム１）のようにエレガント

で簡潔なものが同時に効率的でありえた．しかし，桁数制限の無い大きな整数計算の世界では，Knuth のアルゴリズム D や筆者のアルゴリズム 8 のように無様で複雑なものが必要となっている．同様の現象は他の計算においても起こっているが，除算においてはそれが顕著に現れている．本報告では省略したが，割り算不要な除算として，ニュートン法の応用が可能である．多くの実験によれば，この方法は Knuth のアルゴリズム D や筆者のアルゴリズム 8 より高速である．

この報告では除算についてやや過剰と思えるほどくわしく説明した．その理由は，プログラミングやコンピュータの教育において，四則演算そのものをプログラミングの対象とすることは稀だと思われるからである．もし，教材の一部として利用していただければ，望外の光栄である．

参考文献

(報告の性質を考慮し，一例を除いて個々の原著論文は引用せず，まとまった著作のみ引用する．また，訳書がある場合はそちらを優先する.)

1）野呂春文：Microsoft Excel の上に大きな整数とその演算関数を実現する，日本福祉大学情報社会科学論集 ,9，pp.51-59 (2006)

2）和田秀雄：コンピュータと素因子分解（改定版），遊星社 (1999)

3）Crandall,R. and Pomerance,C.: PRIME NUMBERS: A Computational perspective, Springer-Verlag (2001)

4）Knuth, D.E., 渋谷政昭訳: 4．準数値算法 / 算術演算，サイエンス社 (1981) Knuth,D.E.: Seminumerical Algorithms (Second edition). Addison-Wesley, (1981)