A-038

多倍精度整数のための小さな定数の剰余計算

Reminder of a division by a small constant for large integer

松井 祥悟*
Shogo Matsui

1 はじめに

多倍精度整数は，通常 β 進数 (ただし β は 2^{32} や 2^{64}) の整数として保存されている．β より小さな整数 c でこの整数を割ったときの剰余は，各桁の基数 β^i の c の剰余が既知である場合，各桁に対する乗算とその総和から求められる．

本論文では，この方法の詳細を説明し，基数変換への応用を示した．また，Java を用いた実験を行い，有効性を示した．

2 数の表記と解決すべき問題

多倍精度数 A は，数 a_i とその桁の基数である β のべきを用いて，次のような n 桁の β 進数の数として表現できる:

$$A = a_{n-1}\beta^{n-1} + \cdots + a_2\beta^2 + a_1\beta + a_0.$$

ただし，$0 \leq a_i \leq \beta - 1$ であり，$a_{n-1} \neq 0$ と仮定する．

本論文において解決すべき問題は，β より小さな整数 c で A を割る場合の剰余を求めることである．ただし，β は 2 のべき乗 (具体的には 2^{32} や 2^{64}) と考え，$3 \leq c \leq \beta - 1$ である場合を考える．

3 基本的な考え方

基本は，A の各桁ごとに c の剰余を求めることである．r_m を A の各基数 β^m の c の剰余，すなわち $r_m = \beta^m \bmod c$ とおくと，A の各桁は，

$$a_m\beta^m \equiv r_m \cdot a_m \pmod{c}$$

が成り立つ．これより，

$$\begin{aligned} A &= a_{n-1}\beta^{n-1} + \cdots + a_2\beta^2 + a_1\beta + a_0 \\ &\equiv r_{n-1} \cdot a_{n-1} + \cdots r_2 \cdot a_2 + r_1 \cdot a_1 + r_0 \cdot a_0 \\ &\quad \pmod{c}. \end{aligned}$$

したがって，A の剰余 R は，

$$R = \left(\sum_{i=0}^{n-1} r_i \cdot a_i \right) \bmod c$$

と計算できる．

$0 \leq m \leq n-1$ に対する $r_m \equiv \beta^m \pmod{c}$ が既知の場合には，A の剰余は，n 回の乗算とそれらの総和，一度の剰余計算で得られることになる．一般的な剰余計算が，n 回の除算を必要とすることを考えると，計算時間の短縮が期待できる．

4 基数の剰余

実際的な例として，$\beta = 2^{32}$ の場合を考える．

$c = 5$ や $c = 10$ の場合，基数の剰余 r_m は単純な数となる．

$c = 5$ の場合，$\beta \equiv 1 \pmod{5}$ なので，$1 \leq m \leq n-1$ に対して $\beta^m \equiv 1 \pmod{5}$ である．したがって，A の剰余 R は，

$$R = (a_{n-1} + \cdots + a_2 + a_1 + a_0) \bmod 5$$

で計算できる．

$c = 10$ の場合も同様に，$\beta \equiv 6 \pmod{10}$ なので，$1 \leq m \leq n-1$ に対して $\beta^m \equiv 6 \pmod{10}$ である．A の剰余 R は，

$$R = \{6(a_{n-1} + \cdots + a_2 + a_1) + a_0\} \bmod 10$$

で計算できる．

r_m が単純な数でない場合は，巡回する数列となる．

$c = 7$ の場合は $\beta \equiv 4 \pmod{7}$ なので，$\beta^2 \equiv 2, \beta^3 \equiv 1, \beta^4 \equiv 4, \beta^5 \equiv 2, \beta^6 \equiv 1, \beta^7 \equiv 4, \beta^8 \equiv 2, \beta^9 \equiv 1, \beta^{10} \equiv 4, \cdots \pmod{7}$ と，$4 \to 2 \to 1$ のサイクルを繰り返す．たとえば $n - 1 = 3k + 2$ の場合，A の剰余 R は，

$$\begin{aligned} R = \{ & (a_{3k} + \cdots + a_6 + a_3 + a_0) \\ & + 4(a_{3k+1} + \cdots + a_7 + a_4 + a_1) \\ & + 2(a_{3k+2} + \cdots + a_8 + a_5 + a_2) \quad \} \bmod 7 \end{aligned}$$

*神奈川大学理学部情報科学科

入力:$\sum_{i=0}^{n-1} a_i\beta^i, 0 < c < \beta$，出力:$\sum_{i=0}^{n-1} q_i\beta^i$

1: $d \leftarrow 1/c \pmod{\beta}$
2: $b \leftarrow 0$
3: for i from 0 to $n-1$ do
4: if $b \le a_i$ then $(x, b') \leftarrow (a_i - b, 0)$
5: else $(x, b') \leftarrow (\beta + a_i - b, 1)$
6: $q_i \leftarrow dx \bmod \beta$
7: $b'' \leftarrow \frac{q_i c - x}{\beta}$
8: $b \leftarrow b' + b''$
9: return $\sum_0^{n-1} q_i\beta^i$

図 1: 厳密除算アルゴリズム (algorithm1)

で計算できる．

$c = 11$ の場合，r_m は $5 \to 3 \to 4 \to 9 \to 1$ の 5 つの数のサイクルを繰り返す．剰余計算は $c = 7$ の場合と同様である．

$c = 10^2 = 100$ の場合は $96 \to 16 \to 36 \to 56 \to 76$ の 5 つの数のサイクル，$c = 10^3 = 1000$ の場合は $296 \to 616 \to 336 \to \cdots \to 856 \to 376$ の 25 の数のサイクル，$c = 10^4 = 10000$ の場合は $7296 \to 1616 \to 336 \to 1456 \to \cdots \to 9856 \to 9376$ の 125 の数のサイクルを繰り返す (付録 A)．A の剰余は，掛ける数や繰り返すサイクル数が違うだけで，$c = 7$ の場合と同様に計算できる．

5 基数変換への応用

数 A の c による剰余 $R = A \bmod c$ が得られている場合，$A - R$ は c で割り切れることが保証される．商 Q は $(A - R)/c$ の厳密除算 (必ず割り切れる除算) で得られる．このような場合，Toom-Cook 法[1]などで用いられている厳密除算アルゴリズム [1][2] が利用できる．この方法を図 1algorithm1 に示す．

この方法では，各桁の c による除算を行う代わりに，乗法逆元 $1/c \pmod{\beta}$ による乗算を行う．乗算の計算速度が除算の計算速度を大きく上回る場合には有効な方法である．

一般に，数 A の N 進数への基数変換は，A の N による商と余りを求め，その商の N による商と余りを求め，$\cdots$ という操作を商が 0 になるまで繰り返す．i 回めで得られる余りを r_i とし，m 個の余りが得られた場合，$\sum_{i=0}^{m-1} r_i \cdot N^i$ が変換された N 進数の数である．

[1]多倍精度数の乗算を高速に行うアルゴリズムの一つ．アルゴリズム中の 6 での厳密除算に使用される

入力:$\sum_0^{n-1} a_i\beta^i, 0 < c < \beta$，出力:$\sum_0^{n-1} q_i\beta^i$, b

1: $b \leftarrow 0$
2: for i from $n-1$ downto 0 do
3: $l \leftarrow b\beta + a_i$
4: $q_i \leftarrow l/c$
5: $b \leftarrow l - q_i c$
6: return $\sum_0^{n-1} q_i\beta^i$, b

図 2: 従来の除算アルゴリズム (algorithm2)

この手順の商と余りの計算に本論文の方法を用いることを考える．たとえば 10 進数への変換 ($N = 10$) の場合，10 の剰余を求め，厳密除算で商を求める．ただし，$\beta = 2^{32}$ の場合，公約数 2 をもつため，$1/c \pmod{\beta}$ が直接求まらない．このため，2 の商 Q' を求め，Q' に対する 5 の商 Q を求める．具体的には次の手順となる:

1. $R \leftarrow A \bmod 10$
2. $Q' \leftarrow A/2$
3. $Q \leftarrow (Q' - R/2)/5$

手順 1 の R は先に述べたように $(a_0 + 6\sum_{i=1}^{n-1} a_i) \bmod 10$ で得られる．手順 2 は多倍精度数 A に対してシフト演算を行うことで得られる．手順 3 は “Algorithm1” を使う．

大きな数の場合，一度 10^i で割り，その剰余を 10 進変換する方が速い．たとえば 10000 で割る場合も上の手順と同様に求められる．手順 1 の R は，これも先に述べたように，付録 A の数列の数を順に a_i に掛け，それらの総和の 10000 の剰余を求める。手順 2 は 4 ビットのシフト演算を使って 2^4 で割る．手順 3 は “Algorithm1” を使って 5^4 で割る．

6 実験および評価

Java の BigInteger 数に対する 10 進変換プログラムを作成し，実行速度を計測した．

BigInteger は内部データを直接参照できないので，数値データを byte 配列として取り出し，int 配列に詰め直したものを数値データとして使用した．このデータは A と同じ構造で，$\beta = 2^{32}$ である．これを 10 進変換し，文字列 String として返す．

プログラムは 3 種類作成した．1 つ目は前節で説明した本論文の方法で 10 で割るもの (Rem10)，2 つ目は同じ方法で 10000 で割るもの (Rem10000)，最後は

表 1: 実験環境

項目	内容
コンピュータ名	Apple PowerBook
CPU	1.67GHz PowerPC G4
メモリ	1G バイト
OS	MacOSX10.5.8
Java	バージョン 1.5.0_30

割り算命令を使った従来の方法 (図 2 の algorithm2) で 10000 で割るもの (Sch10000) である．これら 3 つと，BigInteger にある標準の 10 進変換メソッド (toString) の合計 4 つの実行時間を比較した．評価に使用した機器およびソフトウェアは表 1 の通りである．

各プログラムの実行時間を表 2 に示す．bit 数は変換する BigInteger 数のビット数を表し，その横に各方法での実行時間 (mS) が示されている．

Java オリジナルの toString の実行時間に対して，本論文の Rem10 は 4 倍近いが，Rem10000 は 2/3 程度である．基本的に 10 で割る方法は実用的でないことがわかる．Rem10000 については，配列の詰め直しなどの余分な操作を考えると，toString を上回ったという結果には注目できる．

また，従来の除算命令を使った方法である Sch10000 は，toString とほとんど同じ実行時間である．Rem10000 とは除算と剰余の計算部分だけが違うので，本論文の方法が従来の除算を用いた方法 (algorithm2) より速いことが示されている．

Sch10000 の algorithm2 の除算 (4 行目) と剰余 (5 行目) の β^2 長の演算が，Rem10000 の剰余処理と厳密除算では複数の β^2 長の乗算に置き換えられている．したがって，上の結果は，この実験環境において β^2 長の乗算が β^2 長除算に対して十分速いことを示している．

また，10 進変換自体は $O(N^2)$ である．bit 数に 50000 を超える部分は明らかにこれを示している．本論文の方法は実行時間を短くはするが，オーダを抜本的に改善するものではない．

7 まとめ

小さい定数に対する剰余計算は，あらかじめ基数に対する剰余を求めることで，除算をほとんど用いずに簡潔に計算できる．

この方法と厳密計算を組み合わせることで，基数変換にも応用できることを示した．

表 2: 10 進変換の実行時間 (mS)

bit 数	toString	Rem10	Rem10000	Sch10000
3143	16	22	29	7
6366	30	33	27	17
12823	35	87	37	36
25725	87	288	77	91
51548	304	1202	233	330
103219	1151	4566	913	1283
206549	4545	17637	3387	5074
413187	18494	70794	13330	20480
826496	84178	285440	53598	82264
1653165	366762	1176996	215908	333416

参考文献

[1] Tudor Jebelean, “An algorithm for exact division”, Journal of Symbolic Computation, volume 15, 1993, pp. 169-180.
ftp://ftp.risc.uni-linz.ac.at/pub/techreports/1992/92-35.ps.gz

[2] Richard Brent and Paul Zimmermann, “Modern Computer Arithmetic”, version 0.4, pp.40, November 2009,
http://www.loria.fr/~zimmerma/mca/mca-0.4.pdf

付録

A β^i の 10^4 の剰余の数列

$\beta = 2^{32}$ において，$i \geq 1$ に対する β^i の 10^j の剰余数列は巡回する数列となる．その個数 m は $m = 5^{j-1}$ となる．$j = 4$ の例を示す．

$10^4 = 10000$ **の数列** ($m = 125$)

```
7296,1616,336,1456,2976,2896,9216,9936,
3056,6576,8496,6816,9536,4656,176,4096,
4416,9136,6256,3776,9696,2016,8736,7856,
7376,5296,9616,8336,9456,976,896,7216,
7936,1056,4576,6496,4816,7536,2656,8176,
2096,2416,7136,4256,1776,7696,16,6736,
5856,5376,3296,7616,6336,7456,8976,8896,
5216,5936,9056,2576,4496,2816,5536,656,
6176,96,416,5136,2256,9776,5696,8016,
4736,3856,3376,1296,5616,4336,5456,6976,
6896,3216,3936,7056,576,2496,816,3536,
8656,4176,8096,8416,3136,256,7776,3696,
6016,2736,1856,1376,9296,3616,2336,3456,
4976,4896,1216,1936,5056,8576,496,8816,
1536,6656,2176,6096,6416,1136,8256,5776,
1696,4016,736,9856,9376
```