テクニカルノート

Fibonacci 数の高速計算法

高 橋 大 介[†,☆]

本論文では，Fibonacci 数を高速に計算する方法について述べる．Fibonacci 数 F_n を計算するには，Lucas 数の積に基づくアルゴリズムが，最もビット演算量が少ないことが知られている．このアルゴリズムにおいて，多倍長数の乗算を多倍長数の自乗計算に置き換えることで，さらに演算量を減らすことができることを示す．

A Fast Algorithm for Computing Fibonacci Numbers

Daisuke Takahashi[†,☆]

We present a fast algorithm for computing Fibonacci numbers. It is known that the product of Lucas numbers algorithm uses the fewest bit operations to compute the Fibonacci number F_n. We show that the number of bit operations in the conventional product of Lucas numbers algorithm can be reduced by replacing multiple-precision multiplication with the multiple-precision square operation.

1. はじめに

Fibonacci 数を計算するアルゴリズムが多数知られている[1)~7)]が，Lucas 数の積に基づくアルゴリズムが，最もビット演算量が少ないことが知られている[6)]．

本論文では，数万ビット以上の Fibonacci 数を高速に計算するアルゴリズムについて述べる．

Lucas 数の積を用いて Fibonacci 数を計算するアルゴリズムにおいて，多倍長数の乗算を多倍長数の自乗計算に置き換えることで，FFT（高速 Fourier 変換）による乗算を用いた場合，従来のアルゴリズムに比べて演算量を減らすことができることを示す．

以下，2 章で Fibonacci 数と Lucas 数について，3 章で多倍長数の乗算および自乗計算について述べる．4 章で従来の Lucas 数の積に基づくアルゴリズムについて，5 章で提案する Lucas 数の積に基づくアルゴリズムについて述べる．6 章で従来のアルゴリズムとの比較の結果を示す．最後の 7 章はまとめである．

2. Fibonacci 数と Lucas 数

Fibonacci 数は以下のように定義される．

$$F_0=0,\ F_1=1,\ F_{n+2}=F_{n+1}+F_n,\quad n\geq 0 \tag{1}$$

また，Lucas 数は以下のように定義される．

$$L_0=2,\ L_1=1,\ L_{n+2}=L_{n+1}+L_n,\quad n\geq 0 \tag{2}$$

Fibonacci 数と Lucas 数の一般項は，$n\geq 0$ に対して次のような式で表される．

$$F_n=\frac{\alpha^n-\beta^n}{\alpha-\beta} \tag{3}$$

$$L_n=\alpha^n+\beta^n \tag{4}$$

ここで，$\alpha=(1+\sqrt{5})/2$，$\beta=(1-\sqrt{5})/2$ である．

F_n は漸近的には $\alpha^n/\sqrt{5}$ で表されるので，$\gamma=\log_2\alpha\approx 0.69424$ とすると，F_n のビット数は γn で表すことができる[6)]．

式 (3)，(4) から，Fibonacci 数と Lucas 数には以下の関係が成り立つ．

$$F_{n+m}=\frac{1}{2}(F_nL_m+F_mL_n) \tag{5}$$

$$L_{n+m}=\frac{1}{2}(L_nL_m+5F_nF_m) \tag{6}$$

3. 多倍長数の乗算および自乗計算

本論文では，2 つの異なる n ビットの整数を掛ける際のビット演算量を $M(n)$ とし，n ビットの整数の自乗を計算する際のビット演算量を $S(n)$ と区別することにする．また，$M(n)$ および $S(n)$ は $a\geq 1$ に対して，$aM(n)\leq M(an)$，$aS(n)\leq S(an)$ の関係を満足するものとする．

n ビットの整数の積を求めるのに FFT を用いると，

† 東京大学情報基盤センター
Information Technology Center, University of Tokyo
☆ 現在，埼玉大学大学院理工学研究科
Presently with Graduate School of Science and Engineering, Saitama University

$O(n \cdot \log n \cdot \log\log n)$ のビット演算量で求められることが知られている[7),8)]．数千〜数万ビット以上の整数の積を求める際は，FFT を用いる方法が最も高速であることから，本論文では乗算に FFT を使うものとして議論する．

多倍長数 A と B の積を求める際には，A と B それぞれの順 FFT を求めることになるので，順 FFT が 2 回，Fourier 係数の要素ごとの積の後に逆 FFT が 1 回となり，合計 3 回の FFT が必要となる．ところが，多倍長数 A の自乗を計算するには，A の順 FFT が 1 回，Fourier 係数の要素ごとの積の後に逆 FFT が 1 回となり，合計 2 回の FFT で済む．

FFT を用いた多倍長数の乗算および自乗計算においては，FFT の計算量が大部分を占める[9)]ことから，本論文では $S(n) \approx (2/3)M(n)$ であると仮定する．また，n ビットの多倍長数の加減算および，n ビットの多倍長数と $O(1)$ ビットの単精度定数の乗算および除算は，$O(n)$ のオーダーで求まるので，本論文では考慮に入れないものとする．

本論文では，従来の Lucas 数の積に基づくアルゴリズムにおいて，多倍長数の乗算を自乗計算に置き換えることにより，演算量が減ることを示す．

4. 従来の Lucas 数の積に基づくアルゴリズム

式 (5) より以下の関係が導かれる[6),7)]．

$$F_{2k} = F_k L_k \tag{7}$$

また，式 (4) から以下の関係が成り立つ[6),7)]．

$$\begin{aligned} L_{2k} &= \alpha^{2k} + \beta^{2k} \\ &= (\alpha^k + \beta^k)^2 - 2(\alpha\beta)^k \\ &= L_k^2 - 2 \cdot (-1)^k \end{aligned} \tag{8}$$

式 (7) から，F_{2^i} は以下のようにして Lucas 数の積として計算することができる[6)]．

$$F_{2^i} = \prod_{j=0}^{i-1} L_{2^j} \tag{9}$$

図 1 に Lucas 数の積に基づく F_n $(n = 2^i,\ i \geq 2)$ を求めるアルゴリズム[6)]を示す．このアルゴリズムに基づき，任意の n に対して F_n を求めるアルゴリズムが文献 6) に示されている．

図 1 から分かるように，このアルゴリズムでは `for` ループ内において 1 回の反復につき，多倍長数の乗算 1 回と多倍長数の自乗計算 1 回が必要であり，`for` ループが終わった後に多倍長数の乗算 1 回が必要である．したがって，従来の Lucas 数の積に基づくアルゴリズムのビット演算量 $T(n)$ は，次のようになる．

```
fib(n)
    f ← 1
    l ← 3
    for i = 2 to log2 n − 1
        f ← f ∗ l
        l ← l ∗ l − 2
    f ← f ∗ l
    return f
```

図 1 従来の Lucas 数の積に基づくアルゴリズム[6)]
Fig. 1 Conventional product of Lucas numbers algorithm[6)].

$$\begin{aligned} T(n) &= \sum_{i=2}^{\lfloor \log_2 n \rfloor - 1} \left(M(\gamma \cdot 2^{i-1}) + S(\gamma \cdot 2^{i-1}) \right) \\ &\quad + M(\gamma \cdot n/2) \\ &\approx \frac{4}{3} M(\gamma n) \end{aligned} \tag{10}$$

5. 提案する Lucas 数の積に基づくアルゴリズム

従来の Lucas 数の積に基づくアルゴリズムでは，反復 1 回あたり多倍長数の乗算 1 回と多倍長数の自乗計算 1 回が必要である．本論文では，このアルゴリズムを変形して，多倍長数の乗算 1 回を多倍長数の自乗計算 1 回に置き換えることができることを示す．

式 (5)，(6) より以下の関係が導かれる[7)]．

$$F_{k+1} = \frac{1}{2}(F_k + L_k) \tag{11}$$

$$L_{k+1} = \frac{1}{2}(5F_k + L_k) \tag{12}$$

式 (11)，(12) では，多倍長数の乗算が不要であることに注意する．さらに式 (11)，(12) から以下の関係が成り立つ．

$$L_{k+1} = F_{k+1} + 2F_k \tag{13}$$

式 (13) では，式 (12) に比べて 2 で割る除算が不要になっていることが分かる．

また式 (3)，(4) から以下の関係が成り立つ．

$$\begin{aligned} L_k^2 &= (\alpha^k + \beta^k)^2 \\ &= 5\left(\frac{\alpha^k - \beta^k}{\alpha - \beta}\right)^2 + 4(\alpha\beta)^k \\ &= 5F_k^2 + 4 \cdot (-1)^k \end{aligned} \tag{14}$$

したがって，式 (7)，(11) と式 (14) より以下の関係が成り立つ．

$$\begin{aligned} F_{2k} &= F_k L_k \\ &= \frac{(F_k + L_k)^2 - (F_k^2 + L_k^2)}{2} \\ &= \frac{(F_k + L_k)^2 - (F_k^2 + 5F_k^2 + 4 \cdot (-1)^k)}{2} \end{aligned}$$

```
fib(n)
    f ← 1
    l ← 3
    for i = 2 to log_2 n − 1
        temp ← f ∗ f
        f ← (f + l)/2
        f ← 2 ∗ (f ∗ f) − 3 ∗ temp − 2
        l ← 5 ∗ temp + 2
    f ← f ∗ l
    return f
```

図 2 提案する Lucas 数の積に基づくアルゴリズム
Fig. 2 Presented product of Lucas numbers algorithm.

$$= 2F_{k+1}^2 - 3F_k^2 - 2 \cdot (-1)^k \qquad (15)$$

さらに，式 (8)，(14) より以下の関係が成り立つ．

$$L_{2k} = 5F_k^2 + 2 \cdot (-1)^k \qquad (16)$$

式 (15) において，F_{k+1}^2 と F_k^2 の 2 回の多倍長数の自乗計算が必要になるが，式 (15) を計算する際に F_k^2 の値を保存しておけば，式 (16) で F_k^2 を計算する必要はない．つまり，式 (7) の $F_{2k} = F_k L_k$ における 1 回の多倍長数の乗算が，式 (15) では F_{k+1}^2 の 1 回の多倍長数の自乗計算に置き換えられていることになる．

提案する Lucas 数の積に基づいて F_n $(n = 2^i,\ i \geq 2)$ を求めるアルゴリズムを**図 2** に示す．

図 2 から分かるように，このアルゴリズムでは `for` ループ内において 1 回の反復につき，多倍長数の自乗計算 2 回で済むことが分かる．

図 2 のアルゴリズムでは，`for` ループ内において Fibonacci 数と Lucas 数のペアを計算するが，`for` ループが終わった後は F_n だけが求まればよいために，式 (15) により多倍長数の自乗計算を 2 回用いるよりも，式 (7) により多倍長数の乗算を 1 回用いた方が有利である．

提案する Lucas 数の積に基づくアルゴリズムのビット演算量 $T(n)$ は，次のようになる．

$$T(n) = \sum_{i=2}^{\lfloor \log_2 n \rfloor - 1} 2S(\gamma \cdot 2^{i-1}) + M(\gamma \cdot n/2)$$
$$\approx \frac{7}{6} M(\gamma n) \qquad (17)$$

式 (10) と式 (17) を比べると，提案する Lucas 数の積に基づくアルゴリズムのビット演算量は，従来の Lucas 数の積に基づくアルゴリズムに比べて，約 13%少なくなっていることが分かる．

従来の Lucas 数の積に基づくアルゴリズムと提案する Lucas 数の積に基づくアルゴリズムの各反復における計算回数の比較を**表 1** に示す．表 1 から分かるように，提案する Lucas 数の積に基づくアルゴリズムでは，反復 1 回あたり 1 回の乗算が 1 回の自乗計算に置き換えられており，その結果として演算量が減っていることが分かる．

表 1 Lucas 数の積に基づくアルゴリズムの各反復における計算回数の比較
Table 1 Comparison between operation counts in each iteration of the product of Lucas numbers algorithms.

	従来のアルゴリズム	提案するアルゴリズム
乗算	1	0
自乗計算	1	2

```
fib(n)
    if n = 0 return 0
    else if n = 1 return 1
    else if n = 2 return 1
    else
        f ← 1
        l ← 1
        sign ← −1
        mask ← 2^(⌊log_2 n⌋−1)
        for i = 1 to ⌊log_2 n⌋ − 1
            temp ← f ∗ f
            f ← (f + l)/2
            f ← 2 ∗ (f ∗ f) − 3 ∗ temp − 2 ∗ sign
            l ← 5 ∗ temp + 2 ∗ sign
            sign ← 1
            if (n & mask) ≠ 0
                temp ← f
                f ← (f + l)/2
                l ← f + 2 ∗ temp
                sign ← −1
            mask ← mask/2
        if (n & mask) = 0
            f ← f ∗ l
        else
            f ← (f + l)/2
            f ← f ∗ l − sign
        return f
```

図 3 提案する Lucas 数の積に基づく任意の n に対する F_n を求めるアルゴリズム
Fig. 3 Presented product of Lucas numbers algorithm to compute F_n for arbitrary n.

提案する Lucas 数の積に基づくアルゴリズムに基づいて，任意の n に対して F_n を求めるには，べきの計算における 2 進計算法[7]を用いて F_n を計算する．

2 進計算法を用いて F_n を計算する際には，$k \geq 0$ に対して F_{k+1}，L_{k+1}，F_{2k}，L_{2k} が求まればよいことが分かる．これらは式 (11)，(13) と式 (15)，(16) より求めることができる．

図 2 のアルゴリズムと同様に，`for` ループ内においては Fibonacci 数と Lucas 数のペアを計算するが，

表 2 Fibonacci 数の計算時間（秒）
Table 2 Computation time of Fibonacci numbers (in seconds).

n	従来のアルゴリズム	提案するアルゴリズム	時間比
$2^{14}-1$	0.00223	0.00211	1.057
2^{14}	0.00213	0.00203	1.049
$2^{15}-1$	0.00485	0.00456	1.064
2^{15}	0.00464	0.00441	1.052
$2^{16}-1$	0.01310	0.01235	1.061
2^{16}	0.01266	0.01203	1.052
$2^{17}-1$	0.03293	0.03047	1.081
2^{17}	0.03202	0.02982	1.074
$2^{18}-1$	0.07569	0.06979	1.085
2^{18}	0.07364	0.06803	1.082
$2^{19}-1$	0.16930	0.15489	1.093
2^{19}	0.16469	0.15103	1.090
$2^{20}-1$	0.38196	0.34906	1.094
2^{20}	0.37317	0.34146	1.093
$2^{21}-1$	1.07153	0.98755	1.085
2^{21}	1.05273	0.96814	1.087
$2^{22}-1$	2.76208	2.50769	1.101
2^{22}	2.72864	2.46692	1.106
$2^{23}-1$	6.45605	5.81128	1.111
2^{23}	6.35107	5.74072	1.106

`for` ループが終わった後では F_n だけが求まればよい．F_n を求めるには，$k=\lfloor n/2 \rfloor$ とすると，n が偶数のときは F_{2k} として式 (7) より求まる．n が奇数のときは F_{2k+1} として式 (7)，(8) と式 (11) から以下のようにして求まる．

$$\begin{aligned} F_{2k+1} &= \frac{1}{2}(F_{2k}+L_{2k}) \\ &= \frac{1}{2}(F_kL_k + L_k^2 - 2\cdot(-1)^k) \\ &= F_{k+1}L_k - (-1)^k \end{aligned} \tag{18}$$

これらの式より，**図 3** に示すような任意の n に対する，提案する Lucas 数の積に基づくアルゴリズムが得られる．

6. 従来のアルゴリズムとの比較

提案するアルゴリズムの高速性を実証するために，n を $2^{14}-1$〜2^{23} まで変化させて Fibonacci 数 F_n を計算するのに要した CPU 時間を測定した．従来のアルゴリズムおよび提案するアルゴリズムを，FFT に基づく乗算ルーチンが組み込まれている多倍長計算プログラムを用いて計算した．計算機としては，ワークステーション DEC AlphaStation 600 5/333（Alpha 21164 333 MHz，メモリ容量 512 MB）を用いた．OS は Digital UNIX V3.2C，コンパイラは DEC OSF/1 C V3.1 を用い，最適化オプションとして `-O4` を指定した．

計算時間を**表 2** に示す．表 2 から，提案するアルゴリズムは従来のアルゴリズムに比べて約 5%〜11%程度高速であることが分かる．

7. ま と め

本論文では，Fibonacci 数を高速に計算する方法について述べた．Fibonacci 数 F_n を計算するには，Lucas 数の積に基づくアルゴリズムが最もビット演算量が少ないことが知られているが，多倍長数の乗算を多倍長数の自乗計算に置き換えることで，さらに演算量を減らすことができることを示した．

謝辞 本研究の一部は，文部省科学研究費補助金奨励研究（A）（課題番号 10780166）の支援を受けた．

参 考 文 献

1) Shortt, J.: An iterative program to calculate Fibonacci numbers in $O(\log n)$ arithmetic operations, *Inf. Process. Lett.*, Vol.7, pp.299–303 (1978).
2) Urbanek, F.J.: An $O(\log n)$ algorithm for computing the nth element of the solution of a difference equation, *Inf. Process. Lett.*, Vol.11, pp.66–67 (1980).
3) Gries, D. and Levin, G.: Computing Fibonacci numbers (and similarly defined functions) in log time, *Inf. Process. Lett.*, Vol.11, pp.68–69 (1980).
4) Martin, A.J. and Rem, M.: A presentation of the Fibonacci algorithm, *Inf. Process. Lett.*, Vol.19, pp.67–68 (1984).
5) Protasi, M. and Talamo, M.: On the number of arithmetical operations for finding Fibonacci numbers, *Theor. Comput. Sci.*, Vol.64, pp.119–124 (1989).
6) Cull, P. and Holloway, J.L.: Computing Fibonacci numbers quickly, *Inf. Process. Lett.*, Vol.32, pp.143–149 (1989).
7) Knuth, D.E.: *The Art of Computer Programming, Vol.2: Seminumerical Algorithms*, 3rd edition, Addison-Wesley, Reading, MA (1997).
8) Schönhage, A. and Strassen, V.: Schnelle Multiplikation grosser Zahlen, *Computing* (*Arch. Elektron. Rechnen*), Vol.7, pp.281–292 (1971).
9) 高橋大介，金田康正：多数桁の円周率を計算するための公式の改良：ガウス–ルジャンドルの公式とボールウェインの 4 次の収束の公式，情報処理学会論文誌，Vol.38, No.11, pp.2406–2409 (1997).

(平成 11 年 12 月 21 日受付)
(平成 12 年 4 月 6 日採録)