原著論文

受付：2008.10.30
受理：2009. 2. 2

フィボナッチ数とリュカ数の高速計算法

野　呂　春　文
日本福祉大学　健康科学部

Fast Algorithms of Fibonacci number and Lucas number

NORO, Harufumi
Faculty of Health Sciences, Nihon Fukushi University

Abstract:Recursive calculation formulae of Fibonacci series or Lucas series can be rewritten into algebraic forms of matrices and vectors. Using these representation, square ladder algorithms can be constructed.
Keywords: Fibonacci number, Lucas number, fast algorithm, matrix representation, square ladder algorithm

1　はじめに

整数添え字 n に対するフィボナッチ数 F_n は，

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

と定義される．リュカ数の定義には様々あるが，最も包括的かつ一般的なものは，a, b を $\Delta = b^2 - 4c$ が非平方数になる整数パラメータとするとき，リュカ数 U_n, V_n を，

$$U_0 = 0, U_1 = 1, U_n = bU_{n-1} - cU_{n-2} \tag{2}$$
$$V_0 = 2, V_1 = b, V_n = bV_{n-1} - cV_{n-2} \tag{3}$$

と定義するものである[1]．U_n は $b = 1, c = -1$ としたとき F_n に一致する．

フィボナッチ数列やリュカ数列にはさまざまな関係式が成り立ち，初等的な面倒な計算によって新たな関係式が見出されることがあるため，数学愛好家に大変人気がある．趣味的な場面はさておき，素数性証明問題や素因数分解問題において，しばしば数 10 桁以上の巨大な n に対する F_n やリュカ数 U_n, V_n が必要とされる．そのため高速計算法は興味を惹く話題となっている[2]．

この報告では，フィボナッチ数列やリュカ数列を線形代数で取り扱うことによって，見通しの良い高速計算法が構成できることを示す．そして，複数の計算法の計算量の比較をおこなう．

2　既存のアイデア

Dijkstra, E.W. は 1978 年頃に $\log_2 n$ 回のステップでフィボナッチ数を計算できるというアイデアを当時非公開のノートに記したが具体的な計算手順は示さなかった[3]．後年，公開されたそのアイデアを受けて Knott,R. は具体的な計算手順の一例を彼の web サイトに発表した[4]．高速計算法に関しては，Gosper and Salamin も 1972 年頃に異なるアプローチのアイデアをマサチューセッツ工科大学の人工知能研究室のノートブックに記録しており，それが 1995 年に公開された[5]．

2.1 Dijkstra のアイデアと Knott による計算手順

Dijkstra のアイデアは，次の 3 個の等式を使うというものである．

$$F_{2n} = F_n(F_n + 2F_{n-1}) \tag{4}$$
$$F_{2n-1} = F_n^2 + F_{n-1}^2 \tag{5}$$
$$F_{2n+1} = 2F_n^2 + 2F_nF_{n-1} + F_{n-1}^2 \tag{6}$$

Dijkstra のノートは，これらの等式を使えば $\log n$ 回のステップでフィボナッチ数を計算できる，と述べて終わっている．念のため，上の 3 個の等式の略式の証明を記す．

証明：準備としてまず，等式 (*) $F_k = F_{k-m}F_{m+1} + F_{k-m-1}F_m$ を証明する．
$F_k = F_{k-1}+F_{k-2}(= F_2F_{k-1}+F_1F_{k-2}) = 2F_{k-2}+F_{k-3}$ $(= F_3F_{k-2}+F_2F_{k-3}) = 3F_{k-3} + 2F_{k-4}(= F_4F_{k-3} + F_3F_{k-4}) = \cdots = F_{k-m}F_{m+1} + F_{k-m-1}F_m$ となり式 (*) が成り立つ．

この等式 (*) において，$k = 2n, m = n$ と置くと，$F_{2n} = F_nF_{n+1} + F_{n-1}F_n$ となる．$F_{n+1} = F_n + F_{n-1}$ を代入して整理すると式 (4) が得られる．$F_{2n} = F_{2n-1} + F_{2n-2}$ であるから，$F_{2n-1} = F_{2n} - F_{2(n-1)}$ である．この右辺の 2 個の項に式 (4) を適用すると，$F_{2n-1} = F_n(F_n + 2F_{n-1}) - F_{n-1}(F_{n-1} + 2F_{n-2}) = F_n(F_n+2F_{n-1}) - F_{n-1}(2F_{n-1}+2F_{n-2}-F_{n-1}) = F_n(F_n+2F_{n-1}) - F_{n-1}(2F_n - F_{n-1}) = {F_n}^2 + {F_{n-1}}^2$ となり，式 (5) が得られる．$F_{2n+1} = F_{2n} + F_{2n-1}$ より式 (6) が得られる．**証明おわり**．

Knott は，Dijkstra のノートにあるアイデアに基づくと断ったうえで，次の例を示している．

F_{1000} を計算するための手順

- F_{1000} は F_{500} と F_{499} を必要とする．式 (4)
- F_{500} と F_{499} は F_{250} と F_{249} を必要とする．式 (4)
- F_{250} と F_{249} は F_{125} と F_{124} を必要とする．式 (4)
- F_{125} は F_{63} と F_{62} を必要とする．式 (5)
- F_{124} は F_{62} と F_{61} を必要とする．式 (4)
- F_{63} は F_{32} と F_{31} を必要とする．式 (5)
- F_{62} と F_{61} は F_{31} と F_{30} を必要とする．式 (4)
- F_{32} と F_{31} は F_{16} と F_{15} を必要とする．式 (4)
- F_{30} は F_{15} と F_{14} を必要とする．式 (4)
- F_{16} と F_{15} は F_8 と F_7 を必要とする．式 (4)
- F_{14} は F_7 と F_6 を必要とする．式 (4)
- F_8 と F_7 は F_4 と F_3 を必要とする．式 (4)
- F_6 は F_3 と F_2 を必要とする．式 (4)
- F_4 と F_3 は F_2 と F_1 を必要とする．式 (4)
- F_3 は F_2 と F_1 を必要とする．式 (5)
- F_2 は F_1 と F_0 を必要とする．式 (4)
- F_1 と F_0 は定義により 1 と 0 である．

この計算手順によれば，ほぼ $\log_2 n$ 回のステップで F_n が得られる．しかし，残念ながらこの例に示された再帰的な計算手順は著しく見通しが悪いと言わざるをえない．そのため「高速」計算法と言いながら，それが高速であることを示すための計算コストの推定が難しい．

2.2 Gosper and Salamin のアイデア

順序対 (A, B) と (C, D) に対して乗法を

$$(A, B)(C, D) = (AC + AD + BC, AC + BD) \tag{7}$$

で定義する．A, B 等は何らかの数体の要素とする．なお，$(A, B)(C, D)$ を $(A, B) \times (C, D)$ と表わすこともある．

・この乗法は推移的かつ可換であり，結合律を満たすことが容易に示せる．

・$(0,1)(A,B)=(A,B)(0,1)$ より $(0, 1)$ は**単位元**とみなせる．

・$(A/(A^2 - AB - B^2), -(A + B)/(A^2 - AB - B^2))(A, B) = (A, B)(A/(A^2 - AB - B^2), -(A + B)/(A^2 - AB - B^2)) = (0, 1)$ より $(A, B)^{-1} = (A/(A^2 - AB - B^2), -(A + B)/(A^2 - AB - B^2))$ は (A, B) の**乗法逆元** $(A, B)^{-1}$ とみなせる．

これらにより，定義された乗法のもとで，順序対 (A, B) の集合は可換群 (GS 群と呼んでおく) を作ることがわかる．

さらに $(A, B)^m = \overbrace{(A, B)\ldots(A, B)}^{m\text{ 個}}$ とすると**指数法則** $(A, B)^m (A, B)^n = (A, B)^{m+n}$, $((A, B)^m)^n = (A, B)^{mn}$, $((A, B)(C, D))^n = (A, B)^n(C, D)^n$ が自然に成り立つ．

ここで，順序対 $(1, 0)$ に着目すると，$(A, B)(1, 0) = (1, 0)(A, B) = (A + B, A), (1, 0) = (F_1, F_0)$ はフィボナッチ数の定義に合致している．

$$(1, 0)^n = \overbrace{(1, 0)\ldots(1, 0)}^{n\text{ 個}} = (F_n, F_{n-1}) \tag{8}$$

が成り立つことは容易にわかるから，$\mathbf{f} \stackrel{\text{def}}{\equiv} (1, 0)$ とすると

$$(F_n, F_{n-1}) = \mathbf{f}^n \tag{9}$$

と表わすことができる．ここで，

$$\mathbf{f}^0 = (F_0, F_{-1}) = (0, 1) \tag{10}$$

$$\mathbf{f}^{-1} = (F_{-1}, F_{-2}) = (1, -1) \tag{11}$$

$$\tag{12}$$

であることに注意する．

この計算が，符号無し (あるいは符号付き)2 進展開繰り返し 2 乗法によってほぼ $\log_2 n$ 回のステップで実行できることを，Gosper and Salamin は指摘しているが具体的なアルゴリズムは示していない．

なお，Gosper and Salamin のアイデアを以下では GS アイデアと呼ぶことにする．

2.2.1 GS アイデアに基づく符号無し 2 進展開繰り返し 2 乗法による F_n の計算

F_nを計算するために，上で導入した GS 群を利用する．そうすると，問題は f^n の高速計算法になり，整数などに対するのと同様の各種計算法が使えるようになる．

f^n を単純に計算すると，n 個の f を掛け合わせる必要がある．一方，n の 2 進展開を用いると，その展開の桁数だけの回数で計算が終了する．

例として，f^{1000} を考える．1000の 2 進数表現は 1111101000 であるから，$f^{1000} = f^{2^9+2^8+2^7+2^6+2^5+2^3}$ であり，$f^{1000} = ((((((((f)^2 \times f)^2 \times f)^2 \times f)^2 \times f)^2)^2 \times f)^2)^2$ と計算すればよい．2 乗が 9 回，乗算が 5 回である．この計算法を「符号無し 2 進展開繰り返し 2 乗法」あるいは「符号無し 2 進連鎖法」という．

整数nの符号無し 2 進展開は，$n = n_k 2^{k-1} + n_{k-1} 2^{k-2} + \cdots + n_2 2^1 + n_1, (n_k = 1, n_i = 0, 1)$ という形の展開である．それを $n_k n_{k-1} \cdots n_2 n_1, (n_k = 1, n_i = 0, 1)$ と表わす．

符号無し 2 進展開繰り返し 2 乗法による F_n の計算アルゴリズム

以下の擬似コードにおける×は GS 群における乗法であることに注意．

入力：正整数 n.

準備：n の符号無し 2 進展開 $n_k n_{k-1} \cdots n_2 n_1, (n_k = 1, n_i = 0, 1)$を用意する．$n_k$ が最上位桁，n_1 が最下位桁である．$f = (1, 0), result = f$ とする．

ループ：

　$i = k - 1$から $i = 1$まで次を実行する．

　$result = result \times result$;

　$n_i = 1$なら $result = result \times f$;

終了：result を出力する．

例　F_{1000} を符号無し 2 進展開法で計算する．

1000 の符号無し 2 進展開は，$k = 10$ で $n_k = 1, 1, 1, 1, 1, 0, 1, 0, 0, 0 = n_1$である．

ループの 1 回ごとに $f \to f^2 \times f = f^3, f^3 \to f^6 \times f = f^7, f^7 \to f^{14} \times f = f^{15}, f^{15} \to f^{30} \times f = f^{31}, f^{31} \to f^{62}, f^{62} \to f^{124} \times f = f^{125}, f^{125} \to f^{250}, f^{250} \to f^{500}, f^{500} \to f^{1000}$ と進み終了する．2 乗が 9 回，乗算が 5 回である．

2.2.2 GS アイデアに基づく符号付き 2 進展開繰り返し 2 乗法による F_n の計算

例えば f^{15} を計算するのに $f^{8+4+2+1}$ の代わりに f^{16-1}を計算するというアイデアがある．つまり，$f^{15} = (((f)^2 \times f)^2 \times f)^2 \times f$ の代わりに $((((f)^2)^2)^2)^2/f$ と計算するのである．2 乗の回数は 1 回増すが，乗算が 3 回減り除算が 1 回増える．もちろん，普通の整数等であれば除算は乗算の数倍のコストがかかるために実用的な計算法ではない．しかし，GS 群においては，この計算が $f^{15} = ((((f)^2)^2)^2 \times f^{-1}$ となり，除算が乗算と同じコストであるため全体としての計算コストを下げられる可能性がある．この方法を「符号付き 2 進展開繰り返し 2 乗法」という．この方法は，楕円曲線上の多倍点の計算などに威力を発揮する [6]．

整数 n の符号付き 2 進展開は，$n = n_k 2^{k-1} + n_{k-1} 2^{k-2} + \cdots + n_2 2^1 + n_1, (n_k = 1, n_i = -1, 0, 1)$ という形の展開で $n_i \neq 0$ なら$n_{i+1} = 0$となるものである．当然，$n_{k-1} = 0$ である．この展開を $n_k n_{k-1} \ldots n_2 n_1$, $(n_k = 1, n_i = -1, 0, 1]$ と表わす．符号付き 2 進展開は，符号無し 2 進展開より 1 桁だけ大きくなることがある．

符号付き 2 進展開を求めるアルゴリズム

入力：正整数 n

準備：nの符号無し 2 進展開，$n_k n_{k-1} \cdots n_2 n_1, (n_k = 1, n_i = 0, 1)$ を用意する．n_k が最上位桁，n_1 が最下位桁である．

$n_{k+1} = 0$ とする．

ループ：

　$i = 1$から $i = k$ まで次を実行する．

　$n_i = 1$かつ $n_{i+1} = 1$ なら $n_i = -1$とし，$n^{k+1} n_k \ldots n^{i+1}$ を 2 進数と見て 1 を加える $(1 + 0 = 1, 1 + 1 = 10)$.

終了：

$n_{k+1} = 0$ なら $n_k n_{k-1} \ldots n_2 n_1$を，$n_{k+1} = 1$ なら $n_{k+1} n_k n_{k-1} \ldots n_2 n_1$ を出力する．

符号付き 2 進展開繰り返し 2 乗法による F_n の計算アルゴリズム

以下の擬似コードにおける× は GS 群における乗法であることに注意.

入力：正整数 n

準備：整数 n の符号付き 2 進展開，$n_k n_{k-1} \ldots n_2 n_1$，$(n_k = 1, n_i = -1, 0, 1)$ を用意する．n_k が最上位桁，n_1 が最下位桁である．$\mathbf{f} = (1,0), \mathrm{result} = \mathbf{f}, \mathbf{f}^{-1} = (1,-1)$ とする.

ループ：

$i = k-1$ から $i = 1$ まで次を実行する.
$\mathrm{result} = \mathrm{result} \times \mathrm{result}$;
$n_i = 1$ なら $\mathrm{result} = \mathrm{result} \times \mathbf{f}$;
そうでなくて $n_i = -1$ なら $\mathrm{result} = \mathrm{result} \times \mathbf{f}^{-1}$;

終了：result を出力する.

例　F_{1000} を符号付き 2 進展開繰り返し 2 乗法で計算する．1000 の符号付き 2 進展開は，$k = 11$ で $n_{11} = 1,0,0,0,0,-1,0,1,0,0,0 = n_1$ である.
ループの 1 回ごとに，$\mathbf{f} \to \mathbf{f}^2, \mathbf{f}^2 \to \mathbf{f}^4, \mathbf{f}^4 \to \mathbf{f}^8, \mathbf{f}^8 \to \mathbf{f}^{16}, \mathbf{f}^{16} \to \mathbf{f}^{32} \times \mathbf{f}^{-1} = \mathbf{f}^{31}, \mathbf{f}^{31} \to \mathbf{f}^{62}, \mathbf{f}^{62} \to \mathbf{f}^{124} \times \mathbf{f} = \mathbf{f}^{125}, \mathbf{f}^{125} \to \mathbf{f}^{250}, \mathbf{f}^{250} \to \mathbf{f}^{500}, \mathbf{f}^{500} \to \mathbf{f}^{1000}$ と進み終了する．2 乗が 10 回，乗算が 2 回である.

Gopsper and Salamin によって導入された GS 群の要素に対して，加法や減法，スカラー積が普通のベクトルと同じように定義できる．除法，すなわち，乗法逆元についても，基礎体として有理数体，実数体，複素数体や，あるいは，$\mathbf{F}_p$ のような有限体を取れば容易に定義できる．すなわち，ここで導入された順序対 (A, B) の集合は数体を作る.

この数体の上の計算は，高速計算以外にも，フィボナッチ数列にともなう数式の煩雑な計算を簡略化するのに有用である．例えば，前述の式 (4) 等の証明に関する煩雑なマヌーバは，$(F_{2n}, F_{2n-1}) = (1,0)^{2n} = (F_n, F_{n-1})(F_n, F_{n-1}) = ({F_n}^2 + 2F_nF_{n-1}, {F_n}^2 + {F_{n-1}}^2)$ だけで済んでしまう.

Gopsper and Salamin アイデアのすぐれているところは，独自の記法を導入したことによって，フィボナッチ数列に関する計算をきわめて簡潔に示せることにある．しかし，そのことは同時に，フィボナッチ数列に特化することによる見通しの悪さをも生み出している．例えば，式 (2),(3) への適用は自明ではない.

3　線形代数の応用

式 (1), (2), (3) のように再帰的に定義された関係式は，行列とベクトルを使って簡潔に再定義できる [7]．ここでは，それらをリュカ数列を含む一般的な場合と，フィボナッチ数列の特別な場合に分けて説明する.

3.1 リュカ数列を含む一般的な場合

リュカ数列等を一般化した，

$$X_n = \alpha X_{n-1} + \beta X_{n-2}, (n \geq 2) \tag{13}$$

は，

$$\begin{pmatrix} X_n \\ X_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X_{n-1} \\ X_{n-2} \end{pmatrix} \tag{14}$$

と表わすことができる．ここで初期パラメータを X_0, X_1 として，

$$A \stackrel{\mathrm{def}}{\equiv} \begin{pmatrix} \alpha & \beta \\ 1 & 0 \end{pmatrix}, \mathbf{X}_1 \stackrel{\mathrm{def}}{\equiv} \begin{pmatrix} X_1 \\ X_0 \end{pmatrix}, \mathbf{X}_n \stackrel{\mathrm{def}}{\equiv} \begin{pmatrix} X_n \\ X_{n-1} \end{pmatrix} \tag{15}$$

と置けば，

$$\mathbf{X}_n = A\mathbf{X}_{n-1} = \cdots = A^{n-1}\mathbf{X}_1 \tag{16}$$

であることが容易にわかる.

わき道にそれるが，数学の常識にしたがうなら，$A^{n-1}\mathbf{X}_1$ を計算するために以下のようなアプローチもありうる．実用的にはまったく価値が無いが，簡単に説明しておく.

行列 A の n 乗，A^n の計算を見通しよくするにはジョルダンの標準形の助けを借りるのが常識である．行列 A の特性方程式は，

$$\det(xI - A) = \begin{vmatrix} x-\alpha & -\beta \\ -1 & x \end{vmatrix} = x^2 - \alpha x - \beta = 0 \tag{17}$$

である．この特性方程式は $\alpha^2 + 4\beta \neq 0$ なら異なる 2 つの特性根 λ_1, λ_2 を持つ．そのとき，ある正則行列 P を用いて，

$$A = P^{-1} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} P \tag{18}$$

とすることができる (ジョルダンの標準形)．ここで，

$P\mathbf{X}_1 = \begin{pmatrix} X_1 \\ X_0 \end{pmatrix}$ とすれば，$A^{n-1}\mathbf{X}_n = P^{-1}\begin{pmatrix} {\lambda_1}^{n-1}Y_1 \\ {\lambda_2}^{n-1}Y_0 \end{pmatrix}$

である．さらに，$P^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ と置けば，

$X_n = {\lambda_1}^{n-1}aY_1 + {\lambda_2}^{n-1}bY_0$ であるから，複素数 ξ と η を使って，$X_n = {\lambda_1}^{n-1}\xi + {\lambda_2}^{n-1}\eta$ と表わすことができる．そこで，

$$X_1 = \xi + \eta \tag{19}$$
$$X_0 = {\lambda_1}^{-1}\xi + {\lambda_2}^{-1}\eta. \tag{20}$$

を解けば，

$$X_n = \frac{{\lambda_2}^{-1}X_1 - X_0}{{\lambda_2}^{-1} - {\lambda_1}^{-1}}{\lambda_1}^{n-1} - \frac{{\lambda_1}^{-1}X_1 - X_0}{{\lambda_2}^{-1} - {\lambda_1}^{-1}}{\lambda_2}^{n-1} \tag{21}$$

が得られる．よって，${\lambda_1}^{n-1}$ と ${\lambda_2}^{n-1}$ を計算することにより X_n が得られる，というのはあくまでも理屈の上だけである．一般に λ_1 と λ_2 は根号を含む複素数で表わされるため，大きな n に対する冪乗を高速に整数計算するのは困難である．このアプローチには**実用性が無い**．

$A^{n-1}\mathbf{X}_1$ を実際に計算するには，既に説明した符号無し (符号付き)2 進展開繰り返し 2 乗法が有効である．そこで，まず例を説明し，次いで計算法の擬似コードを示す．そのための例として，$\mathbf{X}_{1000}$ の計算手順を考える．$\mathbf{X}_{1000} = A^{999}\mathbf{X}_1$ である．A^{999} を計算し，最後に $\mathbf{X}_1$ をかけるものとする．

最初に符号無し 2 進展開繰り返し 2 乗法から見てみる．999 の符号無し 2 進展開は，$1,1,1,1,1,0,0,1,1,1$ である．したがって，$A^{999} = A^{2^9+2^8+2^7+2^6+2^5+2^2+2^1+2^0}$ であるから，

$A^{999} = ((((((((((A)^2A)^2A)^2A)^2A)^2)^2)A)^2A)^2A$

によって計算できる．2 乗が 9 回，乗算が 7 回である．2 乗の計算 1 回ごとに整数乗算が 7 回，A や A^{-1} をかけること 1 回ごとに整数乗算が 8 回必要である．

符号無し 2 進展開繰り返し 2 乗法による $A^{n-1}\mathbf{X}_1$ の計算アルゴリズム

以下の擬似コードにおける×は線形代数の乗法であることに注意．

入力：A, X_1, X_0, 正整数 n .

準備： $n-1$ の符号無し 2 進展開 $n_k n_{k-1}\cdots n_2 n_1$, $(n_k = 1, n_i = 0, 1)$ を用意する．n_k が最上位桁，n_1 が最下位桁である．

result $= A$ とする．

ループ：

　$i = k-1$ から $i = 1$ まで次を実行する．

　result $=$ result $\times$ result;

　$n_i = 1$ なら result $=$ result $\times A$;

終了：result $\times \begin{pmatrix} X_1 \\ X_0 \end{pmatrix}$ を出力する．

次に，符号付き 2 進展開繰り返し 2 乗法を見てみる．

そのためには $A = \begin{pmatrix} \alpha & \beta \\ 1 & 0 \end{pmatrix}$ に対する A^{-1} が必要なので，それを始めに示す．$\beta \neq 0$ とすると，

$$A^{-1} = \begin{pmatrix} 0 & 1 \\ \frac{1}{\beta} & -\frac{\alpha}{\beta} \end{pmatrix} \tag{22}$$

である．整数計算だけに限って考えると，$\frac{1}{\beta}$ および $\frac{\alpha}{\beta}$ がともに整数でなければならない．その条件が満たされた場合のみ，以下のアルゴリズムが適用可能である．

999 の符号付き 2 進展開は，-1 を $\bar{1}$ と表わして $1,0,0,0,0,\bar{1},0,1,0,0,\bar{1}$ となる．したがって，$A^{999} = A^{2^{10}-2^5+2^3-2^0}$ であるから，

$A^{999} = (((((((((A)^2)^2)^2)^2A^{-1})^2)^2A)^2)^2)^2A^{-1}$

によって計算できる．2 乗が 10 回，乗算が 3 回 (A による乗算が 1 回，A^{-1} による乗算が 2 回) である．

符号付き 2 進展開繰り返し 2 乗法による $A^{n-1}\mathbf{X}_1$ の計算アルゴリズム

以下の擬似コードにおける×は線形代数の乗法であることに注意．

入力：A, X_1, X_0, 正整数 n

準備：整数 $n-1$ の符号付き 2 進展開，$n_k n_{k-1}\cdots n_2 n_1, (n_k = 1, n_i = -1, 0, 1)$ を用意する．n_k が最上位桁，n_1 が最下位桁である．

A^{-1} を用意する．result $= A$ とする．

ループ：

　$i = k-1$ から $i = 1$ まで次を実行する．

　result $=$ result $\times$ result;

$n_i = 1$ なら result = result × A;
そうでなくて $n_i = -1$ なら result = result × A^{-1};

終了：result × $\begin{pmatrix} X_1 \\ X_0 \end{pmatrix}$ を出力する．

3.2 フィボナッチ数に関する特別な扱い

式 (1) で定義されるフィボナッチ数 F_n は，他と比べて構造が特殊かつ単純であるため，上で説明した一般的な場合よりも効率的な計算法が構成できる [8]．
フィボナッチ数 F_k からなる行列 $\mathcal{F}^n$ を

$$\mathcal{F}_n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \tag{23}$$

で定義する．ここで，

$$\mathcal{F}_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \tag{24}$$

$$\mathcal{F}_n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \mathcal{F}_{n-1} \tag{25}$$

であるから，特別に $\mathcal{F}_1$ を $\mathcal{F}$ と書いて，

$$\mathcal{F}_n = \mathcal{F}^n \tag{26}$$

が成り立つ．$\mathcal{F}^n$ は繰り返し 2 乗法を使って高速に計算できる．
繰り返し 2 乗法においては $(\mathcal{F}_k)^2$ の計算が多数回実行される．その計算は，

$$\begin{aligned} (\mathcal{F}_k)^2 &= \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} \\ &= \begin{pmatrix} {F_{k+1}}^2 + {F_k}^2 & F_{k+1}F_k + F_kF_{k-1} \\ F_{k+1}F_k + F_kF_{k-1} & {F_k}^2 + {F_{k-1}}^2 \end{pmatrix} \\ &= \begin{pmatrix} F_{2k+1} & F_{2k} \\ F_{2k} & F_{2k-1} \end{pmatrix} \end{aligned} \tag{27}$$

であるから，

$$F_{2k-1} = {F_k}^2 + {F_{k-1}}^2 \tag{28}$$

$$F_{2k} = F_{k+1}F_k + F_kF_{k-1} = {F_k}^2 + 2F_kF_{k-1} \tag{29}$$

$$F_{2k+1} = F_{2k} + F_{2k-1} \tag{30}$$

であることを利用すると，${F_k}^2, {F_{k-1}}^2, F_kF_{k-1}$ をそれぞれ 1 回づつ，つまり整数乗算 3 回で済むことがわかる．さらに，F_k と F_{k-1} があれば F_{2k} と F_{2k-1} が計算できるから，F_{k+1} と F_{2k+1} は計算と保存が不要であることもわかる．これは，Gopsper and Salamin のアイデアにもとづく方法と同じ計算負荷であることを意味している．

符号無し (符号付き)2 進展開繰り返し 2 乗法のアルゴリズムの擬似コードを以下に示す．それらにおいて，$\mathcal{F}$ や $\mathcal{F}^{-1}$ をかける処理は，実際は乗算を必要とせず，単純な足し算や引き算であることに注意する．

符号無し 2 進展開繰り返し 2 乗法による $\mathcal{F}^n$ の計算アルゴリズム

以下の擬似コードにおける 2 乗および×については上で述べられていることに注意．
入力：正整数 n .
準備：n の符号無し 2 進展開 $n_k n_{k-1} \cdots n_2 n_1, (n_k = 1, n_i = 0, 1)$ を用意する．n_k が最上位桁，n_1 が最下位桁である．
result = $\mathcal{F}$ とする．
ループ：

$i = k-1$ から $i = 1$ まで次を実行する．
result = result^2;
$n_i = 1$ なら result = result × f;

終了：result を出力する．

次の符号付き 2 進展開繰り返し 2 乗法で必要な $\mathcal{F}^{-1} = \mathcal{F}_{-1}$ は，

$$\mathcal{F}^{-1} = \begin{pmatrix} F_0 & F_{-1} \\ F_{-1} & F_{-2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \tag{31}$$

である．

符号付き 2 進展開繰り返し 2 乗法による $\mathcal{F}^n$ の計算アルゴリズム

以下の擬似コードにおける 2 乗および×については上で述べられていることに注意．
入力：正整数 n
準備：整数 n の符号付き 2 進展開，$n_k n_{k-1} \ldots n_2 n_1$, $\{n_k = 1, n_i = -1, 0, 1\}$ を用意する．n_k が最上位桁，n_1 が最下位桁である．
result = $\mathcal{F}$ とする．
ループ：

$i = k-1$ から $i = 1$ まで次を実行する．
result = result^2;
$n_i = 1$ なら result = result × $\mathcal{F}$;

そうでなくて $n_i=-1$ なら $\mathrm{result}=\mathrm{result}\times\mathcal{F}^{-1}$;

終了：result を出力する.

4　おわりに

フィボナッチ数 F_n とリュカ数 U_n,V_n の高速計算法について，簡単なレビューを行い，新たに線形代数にもとづく方法を提案した．符号付き (符号無し)2 進展開繰り返し 2 乗法を使えば $\log_2 n$ 回のステップを要することが示された.

一般的なリュカ数 U_n,V_n については，$\log_2 n$ 回のステップを要し，最悪でも $15\log_2 n$ 回の整数乗算によって計算できることが示された．リュカ数については符号付き 2 進展開繰り返し 2 乗法が必ずしも構成できるわけではないが，それを構成できる場合は，乗算の回数をさらに節約できる可能性がある.

フィボナッチ数 F_n は，それと比べて構造が特殊かつ単純であるため，より効率的な計算法が構成できる．また，符号付き 2 進展開繰り返し 2 乗法が必ず構成できるので，それを選ぶのが得策である．そうすると，$\log_2 n$ 回のステップによって，最悪でも $3\log_2 n$ 回の整数乗算によって計算できることが示された.

より一般的に $s_n=a_1s_{n-1}+a_2s_{n-2}+\cdots+a_ks_{n-k}$ によって再帰的に定義される数列 $\{s_n\}$ に対して，以上で述べた方法を適用すると，

$$\begin{pmatrix} s_n \\ s_{n-1} \\ \vdots \\ s_{n-k+1} \end{pmatrix}=\begin{pmatrix} a_1 & a_2 & \cdots & a_k \\ 1 & 0 & \cdots & 0 \\ 0 & \ddots & & 0 \\ 0 & \cdots & 1 & 0 \end{pmatrix}\begin{pmatrix} s_{n-1} \\ s_{n-2} \\ \vdots \\ s_{n-k} \end{pmatrix} \tag{32}$$

と表わすことができる．このような数列に対して符号なし 2 進展開繰り返し 2 乗法を適用することによって高速な計算法が構成できる.

特別に，$k=3$，$a_1=a_2=a_3=1$ とするとトリボナッチ数列 $\{T_n\}$ が得られる．このとき，

$$\begin{pmatrix} T_n \\ T_{n-1} \\ T_{n-2} \end{pmatrix}=\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} T_{n-1} \\ T_{n-2} \\ T_{n-3} \end{pmatrix} \tag{33}$$

であり，

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{-1}=\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & -1 \end{pmatrix}$$

であるから，より効率的な符号付き 2 進展開繰り返し 2 乗法が構成できる.

参考文献

1）Bressoud, D.M. ，玉井浩訳：素因数分解と素数判定，エスアイビー・アクセス発行，星雲社発売 (2004) Bressoud, D.M.: Factorization and primality testing, Springer－Verlag, (1989)

2）Crandall,R. andPomerance,C.: PRIME NUMBERS: A Computational perspective, Second edition, Springer－Verlag (2005)

3）Dijkstra,E.W.: note EWD654 “ In honor of Fibonacci ” ,http://www.cs.utexas.edu/users/EWD/

4）Knott,R.:http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/.b.html

5）Gosper and Salamin: The Fast Fibonacci Transform, http://www.inwap.com/pdp10/hbaker/hakmem/recurrence.html

6）Blake, I.F., Seroussi,G. and Smart,N.P. ，鈴木治郎訳：楕円曲線暗号，ピアソン・エデュケーション (2001) Blake,I.F.,Seroussi,G. and Smart,N.P.:Elliptic Curves in Cryptography, Cambridge UniversityPress (1999)

7）砂田利一：行列と行列式 1，岩波講座現代数学への入門，岩波書店 (1995)

8）Johnson, R.C. : Matrix methods for Fibonacci and related sequences,http://www.dur.ac.uk/bob.johnson/.bonacci/