EFFICIENT COMPUTATION OF TERMS OF LINEAR RECURRENCE SEQUENCES OF ANY ORDER

Dmitry I. Khomovsky
Lomonosov Moscow State University, Moscow, Russia
khomovskij@physics.msu.ru

Received: 10/21/16, Accepted: 4/15/18, Published: 4/20/18

Abstract

In this paper we give efficient algorithms for computing second-, third-, and fourth-order linear recurrences. We also present an algorithm scheme for computing terms with the indices $N, \ldots, N+n-1$ of an nth-order linear recurrence. Unlike Fiduccia's algorithm, our approach uses certain formulas for modular polynomial squarings.

1. Introduction

Let $\{W_k(a_0, \ldots, a_{n-1}; p_0 \ldots p_{n-1})\}$ be an nth-order linear recurrence defined by the relation

$$f_{k+n} = p_0 f_{k+n-1} + p_1 f_{k+n-2} + \ldots + p_{n-1} f_k, \tag{1}$$

with the initial values $W_i = a_i$ $(0 \le i \le n-1)$. The characteristic polynomial is

$$g(x) = x^n - (p_0 x^{n-1} + p_1 x^{n-2} + \ldots + p_{n-1}). \tag{2}$$

Widely known particular cases are the Lucas sequences $\{U_k(P,Q)\}$ and $\{V_k(P,Q)\}$. They are defined recursively by

$$f_{k+2} = P f_{k+1} - Q f_k, \tag{3}$$

with the initial values $U_0 = 0$, $U_1 = 1, V_0 = 2, V_1 = P$. The characteristic polynomial in this case is $x^2 - Px + Q$.

Computation of linear recurrences has been studied by many authors [5, 8, 12]. The most effective algorithm was proposed by Fiduccia in 1985. To obtain the Nth term of an nth-order linear recurrence using this method, we need to compute $r(x) = x^N \bmod g(x)$, where $g(x) = x^n - \sum_{i=0}^{n-1} p_i x^{n-1-i}$. Then we compute $r(C)$, where C is the $n \times n$ companion matrix of the linear recurrence:

$$C = \begin{pmatrix} 0 & & & & p_{n-1} \\ 1 & & & & p_{n-2} \\ & 1 & & & p_{n-3} \\ & & \ddots & & \vdots \\ & & & 1 & p_0 \end{pmatrix}. \tag{4}$$

Finally, we multiply the row vector of initial values $(a_0, \dots, a_{n-1})$ by the first column of $r(C)$ and obtain the Nth term. The computational complexity of this algorithm is $O(\mu(n) \log N + n^3)$. Here, $\mu(n)$ is the total number of operations required to multiply two polynomials of degree $n-1$ in the polynomial ring. Fiduccia actually manages to exploit the structure of the matrix C in order to decrease the complexity to $O(\mu(n) \log N)$, see Theorem 3.1 and Proposition 3.2 in [5].

2. Computation of Second-order Linear Recurrences

Let the second-order linear recurrence sequence $\{W_k\}$ be defined by the relation[1] $W_{k+2} = PW_{k+1} - QW_k$, with $W_0 = A, W_1 = B$. It was intensively studied by Horadam [9, 10].

For the Lucas sequences we have the following matrix formula:

$$\begin{pmatrix} U_{k+1} & V_{k+1} \\ U_k & V_k \end{pmatrix} = M \begin{pmatrix} U_k & V_k \\ U_{k-1} & V_{k-1} \end{pmatrix}, \quad \text{where } M = \begin{pmatrix} P & -Q \\ 1 & 0 \end{pmatrix}. \tag{5}$$

Then

$$\begin{pmatrix} U_{k+1} & V_{k+1} \\ U_k & V_k \end{pmatrix} = M^k \begin{pmatrix} 1 & P \\ 0 & 2 \end{pmatrix}. \tag{6}$$

Lemma 1. *For the sequence $\{W_k(A, B; P, Q)\}$ the following holds:*

$$W_k = BU_k - AQU_{k-1}, \tag{7}$$

$$W_k = (B - AP)U_k + AU_{k+1}. \tag{8}$$

Proof. We have:

$$\begin{pmatrix} W_{k+1} \\ W_k \end{pmatrix} = M^k \begin{pmatrix} B \\ A \end{pmatrix} = BM^k \begin{pmatrix} 1 \\ 0 \end{pmatrix} + AM^k \begin{pmatrix} 0 \\ 1 \end{pmatrix} = B \begin{pmatrix} U_{k+1} \\ U_k \end{pmatrix} + AM^{k-1} \begin{pmatrix} -Q \\ 0 \end{pmatrix}$$
$$= \begin{pmatrix} BU_{k+1} - AQU_k \\ BU_k - AQU_{k-1} \end{pmatrix}.$$

From this we get (7). By the definition of the Lucas sequence $QU_{k-1} = PU_k - U_{k+1}$. Using this, we obtain (8). □

[1] For recurrences of order greater than 2 we will use the relation (1).

The obtained result is known (for example, see [9]). We see that the computation of remote terms of $\{W_k(A,B;P,Q)\}$ can be done by the Lucas sequence $\{U_k(P,Q)\}$. In a sense, $\{U_k\}$ is a basis.

We note that the result given in the following theorem is known, moreover, there is a generalization [16]. But we still give the proof, since we will use a similar approach for higher-order linear recurrences.

Theorem 1. *Let $\{U_k(P,Q)\}$ be the Lucas sequence. Then*

$$\begin{pmatrix} U_{mk+1} \\ U_{mk} \end{pmatrix} = \begin{pmatrix} U_{k+1} & -QU_k \\ U_k & U_{k+1}-PU_k \end{pmatrix}^{m-1} \begin{pmatrix} U_{k+1} \\ U_k \end{pmatrix}. \tag{9}$$

Proof. We use the notations

$$S = \begin{pmatrix} 1 & P \\ 0 & 2 \end{pmatrix}, \; S^{-1} = \begin{pmatrix} 1 & -P/2 \\ 0 & 1/2 \end{pmatrix}. \tag{10}$$

We have

$$\begin{pmatrix} U_{mk+1} \\ U_{mk} \end{pmatrix} = M^{mk} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (M^k SS^{-1})^{m-1} \begin{pmatrix} U_{k+1} \\ U_k \end{pmatrix}. \tag{11}$$

By (6) and (10),

$$M^k SS^{-1} = \begin{pmatrix} U_{k+1} & (-PU_{k+1}+V_{k+1})/2 \\ U_k & (-PU_k+V_k)/2 \end{pmatrix}. \tag{12}$$

By Lemma 1 we can get the classical identity $V_k = PU_k - 2QU_{k-1}$, with the help of which we eliminate V_k, V_{k+1} from (12). Then

$$M^k SS^{-1} = \begin{pmatrix} U_{k+1} & -QU_k \\ U_k & -QU_{k-1} \end{pmatrix}. \tag{13}$$

Since $-QU_{k-1} = U_{k+1} - PU_k$, we get

$$M^k SS^{-1} = \begin{pmatrix} U_{k+1} & -QU_k \\ U_k & U_{k+1}-PU_k \end{pmatrix}. \tag{14}$$

Finally, we can modify (11) into (9). $\square$

If $m = 2$ in (9), then we obtain the following identities:

$$U_{2k} = U_k(2U_{k+1} - PU_k), \tag{15}$$

$$U_{2k+1} = U_{k+1}^2 - QU_k^2. \tag{16}$$

If we replace k by $k+1$ in (15) and use $U_{k+2} = PU_{k+1} - QU_k$, then we obtain

$$U_{2k+2} = U_{k+1}(PU_{k+1} - 2QU_k). \tag{17}$$

Now using (15), (16), and (17) we can present an algorithm for computing two terms of $\{U_k(P,Q)\}$ with the indices N and $N+1$. We need four temporary memories: u_1, u_2, U_1, U_2. In fact, the number of temporary memories can be reduced by eliminating u_2.

Algorithm 1 Computing the Lucas sequence $\{U_k(P,Q)\}$

Input: $N = \sum_{i=0}^{m-1} b_i 2^i$, $(b_{m-1} = 1)$
P, Q
Output: U_N, U_{N+1}
1: $U_1 \leftarrow 1$; $U_2 \leftarrow P$
2: **for** j from $m-2$ to 0 by -1 **do**
3: $u_1 \leftarrow U_1$; $u_2 \leftarrow U_2$
4: **if** $b_j = 1$ **then**
5: $U_1 \leftarrow u_2^2 - Qu_1^2$; $U_2 \leftarrow u_2(Pu_2 - 2Qu_1)$
6: **else if**
7: $U_1 \leftarrow u_1(2u_2 - Pu_1)$; $U_2 \leftarrow u_2^2 - Qu_1^2$
8: **end if**
9: **end for**
11: **return** U_1, U_2

Such a computational method was discussed by Reiter in [14]. Previously [4], it was proposed for the Fibonacci numbers.

Suppose we have computed U_N, U_{N+1} by Algorithm 1, then with the help of (8) we get W_N. Using $W_{N+1} = BU_{N+1} - AQU_N$ we get W_{N+1}. Thus, in the general case, to compute the terms U_N, U_{N+1} and W_N, W_{N+1} we need $3m$ multiplications[2], where $m = \lfloor \log_2 N \rfloor + 1$. But when $Q = 1$ or more generally $Q = a^2$, we can slightly transform Algorithm 1 so that we need only $2m$ multiplications. Indeed, when $Q = 1$, we replace the expression $u_2^2 - u_1^2$ by $(u_2 - u_1)(u_2 + u_1)$ at steps $5, 7$. When $Q = a^2$, we use the formula $u_2^2 - Qu_1^2 = (u_2 - au_1)(u_2 + au_1)$.

2.1. Comparison With Other Existing Algorithms

Currently, the main algorithm [11] for quick computation of the Lucas sequence terms U_N, V_N uses the following properties:

$$\begin{aligned} V_{2k+1} &= V_{k+1}V_k - PQ^k, \ V_{2k} = V_k^2 - 2Q^k, \\ U_{2k+1} &= U_{k+1}V_k - Q^k, \quad U_{2k} = U_kV_k. \end{aligned} \tag{18}$$

When $Q = \pm 1$, the computation of U_N and V_N requires $3m$ multiplications in the worst case (N is odd), and $2m$ in the best case (N is a power of two). When $Q \neq \pm 1$ and without any assumption about N, this algorithm needs less than $4m$ multiplications but not less than $3m$. We see that Algorithm 1 is more efficient

[2] We imply that P, Q are not large. So multiplications that involve them are similar to additions.

in most cases, but there is an important case when the algorithm offered in [11] is better. This is so when we need to compute the term $V_N(P,1)$ or $V_N(P,-1)$. For $N = 2^s(2d+1)$ the computation of $V_N(P,1)$ by the algorithm in [11] requires $2\lfloor \log_2(2d+1)\rfloor + s$ multiplications while Algorithm 1 needs $2\lfloor \log_2(2d+1)\rfloor + 2s$. So in applications such as Lucas-based cryptosystem [2] and Lucas-Lehmer-Risel primality test [15], it is preferable to use the algorithm offered in [11].

Now we compare Algorithm 1 with Fiduccia's algorithm. The characteristic polynomial is $g(x) = x^2 - Px + Q$. To compute $x^N \bmod g(x)$, Fiduccia's algorithm uses the classical method of repeating squaring. For an arbitrary linear polynomial $h(x) = -u_1 x + u_2$, we have $h^2(x) \bmod g(x) = -u_1(2u_2 - Pu_1)x + u_2^2 - Qu_1^2$. As is seen above, we can use formulas (15) and (16) for modular polynomial squarings. Therefore, Algorithm 1, together with the formula (8), is one way of implementing Fiduccia's algorithm for second-order linear recurrences, where the explicit formulas for modular polynomial squarings are used.

3. Computation of Third-order Linear Recurrences

We will follow the notation for third-order linear recurrences according to [13]. The sequences $\{X_k(p,q,r)\}$, $\{Y_k(p,q,r)\}$, and $\{Z_k(p,q,r)\}$ are defined recursively by

$$f_{k+3} = pf_{k+2} + qf_{k+1} + rf_k, \tag{19}$$

with the initial values $X_0 = 0$, $X_1 = 0$, $X_2 = 1$, $Y_0 = 0$, $Y_1 = 1$, $Y_2 = 0$, $Z_0 = 1$, $Z_1 = 0$, $Z_2 = 0$. Similar to (6), we have

$$\begin{pmatrix} X_{k+2} & Y_{k+2} & Z_{k+2} \\ X_{k+1} & Y_{k+1} & Z_{k+1} \\ X_k & Y_k & Z_k \end{pmatrix} = M^k S, \text{ where } M = \begin{pmatrix} p & q & r \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \ S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{20}$$

Lemma 2. *Let the sequence $\{W_k(a_0,a_1,a_2;p,q,r)\}$ be defined by the relation*

$$W_{k+3} = pW_{k+2} + qW_{k+1} + rW_k, \tag{21}$$

with the initial values $W_0 = a_0, W_1 = a_1, W_2 = a_2$. Then

$$W_k = a_2 X_k + (a_1 q + a_0 r)X_{k-1} + a_1 r X_{k-2}, \tag{22}$$

$$W_k = (a_2 - a_1 p - a_0 q)X_k + (a_1 - a_0 p)X_{k+1} + a_0 X_{k+2}. \tag{23}$$

Proof. We have:

$$
\begin{aligned}
\begin{pmatrix} W_{k+2} \\ W_{k+1} \\ W_k \end{pmatrix} = M^k \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} &= a_2 M^k \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + a_1 M^k \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + a_0 M^k \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
&= a_2 \begin{pmatrix} X_{k+2} \\ X_{k+1} \\ X_k \end{pmatrix} + a_1 M^{k-1} \begin{pmatrix} q \\ 0 \\ 1 \end{pmatrix} + a_0 M^{k-1} \begin{pmatrix} r \\ 0 \\ 0 \end{pmatrix} \\
&= a_2 \begin{pmatrix} X_{k+2} \\ X_{k+1} \\ X_k \end{pmatrix} + (a_1 q + a_0 r) \begin{pmatrix} X_{k+1} \\ X_k \\ X_{k-1} \end{pmatrix} + a_1 M^{k-2} \begin{pmatrix} r \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} a_2 X_{k+2} + (a_1 q + a_0 r) X_{k+1} + a_1 r X_k \\ a_2 X_{k+1} + (a_1 q + a_0 r) X_k + a_1 r X_{k-1} \\ a_2 X_k + (a_1 q + a_0 r) X_{k-1} + a_1 r X_{k-2} \end{pmatrix}. \qquad (24)
\end{aligned}
$$

So we obtain (22). With the help of $X_{k-2} = (X_{k+1} - pX_k - qX_{k-1})/r$ and $X_{k-1} = (X_{k+2} - pX_{k+1} - qX_k)/r$ we get (23). □

By Lemma 2 we get the following:

$$
\begin{aligned}
Y_k &= qX_{k-1} + rX_{k-2}, && (25) \\
Z_k &= rX_{k-1}, && (26) \\
Y_k &= X_{k+1} - pX_k, && (27) \\
Z_k &= X_{k+2} - pX_{k+1} - qX_k. && (28)
\end{aligned}
$$

Theorem 2. *Let $\{X_k(p, q, r)\}$ be the third-order linear recurrence sequence with the initial values $X_0 = 0, X_1 = 0, X_2 = 1$. Then*

$$
\begin{pmatrix} X_{mk+2} \\ X_{mk+1} \\ X_{mk} \end{pmatrix} = \begin{pmatrix} X_{k+2} & qX_{k+1} + rX_k & rX_{k+1} \\ X_{k+1} & X_{k+2} - pX_{k+1} & rX_k \\ X_k & X_{k+1} - pX_k & X_{k+2} - pX_{k+1} - qX_k \end{pmatrix}^{m-1} \begin{pmatrix} X_{k+2} \\ X_{k+1} \\ X_k \end{pmatrix}. \qquad (29)
$$

Proof. According to (20), we have

$$
\begin{pmatrix} X_{mk+2} \\ X_{mk+1} \\ X_{mk} \end{pmatrix} = M^{mk} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \left(M^k\right)^{m-1} \begin{pmatrix} X_{k+2} \\ X_{k+1} \\ X_k \end{pmatrix}. \qquad (30)
$$

Using (25) − (28) we eliminate $Y_k, Y_{k+1}, Y_{k+2}, Z_k, Z_{k+1}, Z_{k+2}$ from M^k. This may be done in such a way that M^k will contain only X_k, X_{k+1}, X_{k+2}. We obtain

$$
M^k = \begin{pmatrix} X_{k+2} & qX_{k+1} + rX_k & rX_{k+1} \\ X_{k+1} & X_{k+2} - pX_{k+1} & rX_k \\ X_k & X_{k+1} - pX_k & X_{k+2} - pX_{k+1} - qX_k \end{pmatrix}. \qquad (31)
$$

Finally, we can modify (30) into (29). □

If we put $m = 2$ in (29), then we get the following formulas:

$$X_{2k+2} = X_{k+2}^2 + X_{k+1}(qX_{k+1} + 2rX_k), \tag{32}$$

$$X_{2k+1} = rX_k^2 + X_{k+1}(2X_{k+2} - pX_{k+1}), \tag{33}$$

$$X_{2k} = X_{k+1}^2 + X_k(2X_{k+2} - 2pX_{k+1} - qX_k). \tag{34}$$

Remark. It is clear that $X_{k+1}(P, -Q, 0) = U_k(P, Q)$. Therefore, if we put $r = 0$ and $q = -Q$ in these formulas and subtract 1 from all indices, then up to the substitution of U for X we obtain the identities for second-order recurrences.
Remark. If we calculate the remainder

$$(c_1x^2 + c_2x + c_3)^2 \bmod x^3 - px^2 - qx - r, \tag{35}$$

then we obtain the formulas similar to (but not the same as) $(32) - (34)$ for squaring of quadratic polynomials modulo $g(x) = x^3 - px^2 - qx - r$. They can be used in Fiduccia's algorithm for computing third-order recurrences.

To get an algorithm for computing X_N, X_{N+1}, X_{N+2} similar to the binary exponentiation we need to be able to compute X_{2k}, X_{2k+1}, X_{2k+2}, X_{2k+3} using X_k, X_{k+1}, X_{k+2}. So we need another formula that helps us to compute X_{2k+3}. It can be obtained from (33) if we replace k by $k+1$ and use $X_{k+3} = pX_{k+2} + qX_{k+1} + rX_k$. It is as follows:

$$X_{2k+3} = rX_{k+1}^2 + X_{k+2}(pX_{k+2} + 2qX_{k+1} + 2rX_k). \tag{36}$$

Now we present an algorithm based on the formulas $(32) - (34)$, (36). We need to use six temporary memories.

Algorithm 2 Computing the third-order linear recurrence $\{X_k(p, q, r)\}$

Input: $N = \sum_{i=0}^{m-1} b_i 2^i$, $(b_{m-1} = 1)$
p, q, r
Output: X_N, X_{N+1}, X_{N+2}
$X_1 \leftarrow 0$; $X_2 \leftarrow 1$; $X_3 \leftarrow p$
for j from $m-2$ to 0 by -1 **do**
$x_1 \leftarrow X_1$; $x_2 \leftarrow X_2$; $x_3 \leftarrow X_3$
  if $b_j = 1$ **then**
    $X_1 \leftarrow rx_1^2 + x_2(2x_3 - px_2)$; $X_2 \leftarrow x_3^2 + x_2(qx_2 + 2rx_1)$;
    $X_3 \leftarrow rx_2^2 + x_3(px_3 + 2qx_2 + 2rx_1)$
  else if
    $X_1 \leftarrow x_2^2 + x_1(2x_3 - 2px_2 - qx_1)$; $X_2 \leftarrow rx_1^2 + x_2(2x_3 - px_2)$;
    $X_3 \leftarrow x_3^2 + x_2(qx_2 + 2rx_1)$
  end if
end for
return X_1, X_2, X_3

We will imply that multiplications by p, q, r can be simulated by additions. Then algorithm 2 needs $3m$ multiplications and $3m$ squarings. At the end of this section, we refer to some applications that use computation of remote terms of third-order linear recurrence sequences; see [1, 3, 6, 7].

4. Computation of Fourth-order Linear Recurrences

Since this section is similar to the previous one, we give only the main formulas and the final algorithm.

The fourth-order linear recurrence $\{W_k(a_0, a_1, a_2, a_3; p_0, p_1, p_2, p_3)\}$ is defined recursively by

$$f_{k+4} = p_0 f_{k+3} + p_1 f_{k+2} + p_2 f_{k+1} + p_3 f_k, \tag{37}$$

with the initial values $W_0 = a_0, W_1 = a_1$, $W_2 = a_2$, $W_3 = a_3$. Denote the sequence $\{W_k(0, 0, 0, 1; p_0, p_1, p_2, p_3)\}$ by $\{X_k(p_0, p_1, p_2, p_3)\}$. The formulas, which can be obtained by the matrix method, are:

$$W_k = a_3 X_k + (a_0 p_3 + a_1 p_2 + a_2 p_1) X_{k-1} + (a_1 p_3 + a_2 p_2) X_{k-2} + a_2 p_3 X_{k-3}, \tag{38}$$

$$\begin{aligned} W_k =& (a_3 - a_2 p_0 - a_1 p_1 - a_0 p_2) X_k + (a_2 - a_1 p_0 - a_0 p_1) X_{k+1} + \\ & (a_1 - a_0 p_0) X_{k+2} + a_0 X_{k+3}. \end{aligned} \tag{39}$$

We will use $W_k(a_0, a_1, a_2, a_3)$ instead of $W_k(a_0, a_1, a_2, a_3; p_0, p_1, p_2, p_3)$. By (38), (39), and (37) we obtain the following

$$W_k(0, 0, 1, 0) = p_1 X_{k-1} + p_2 X_{k-2} + p_3 X_{k-3}, \tag{40}$$

$$W_k(0, 1, 0, 0) = p_2 X_{k-1} + p_3 X_{k-2}, \tag{41}$$

$$W_k(1, 0, 0, 0) = p_3 X_{k-1}, \tag{42}$$

$$W_k(0, 0, 1, 0) = -p_0 X_k + X_{k+1}, \tag{43}$$

$$W_k(0, 1, 0, 0) = -p_1 X_k - p_0 X_{k+1} + X_{k+2}, \tag{44}$$

$$W_k(1, 0, 0, 0) = -p_2 X_k - p_1 X_{k+1} - p_0 X_{k+2} + X_{k+3}. \tag{45}$$

For convenience, we use the notation ${}^iW_k$ for $W_k(a_0, a_1, a_2, a_3)$ with only one nonzero $a_i = 1$. Then by the matrix method we get

$$\begin{pmatrix} X_{mk+3} \\ X_{mk+2} \\ X_{mk+1} \\ X_{mk} \end{pmatrix} = \begin{pmatrix} X_{k+3} & {}^2W_{k+3} & {}^1W_{k+3} & {}^0W_{k+3} \\ X_{k+2} & {}^2W_{k+2} & {}^1W_{k+2} & {}^0W_{k+2} \\ X_{k+1} & {}^2W_{k+1} & {}^1W_{k+1} & {}^0W_{k+1} \\ X_k & {}^2W_k & {}^1W_k & {}^0W_k \end{pmatrix}^{m-1} \begin{pmatrix} X_{k+3} \\ X_{k+2} \\ X_{k+1} \\ X_k \end{pmatrix}. \tag{46}$$

With the help of (40) – (45) we transform the matrix in (46) and obtain

$$\begin{pmatrix} X_{k+3} & p_1X_{k+2}+p_2X_{k+1}+p_3X_k & p_2X_{k+2}+p_3X_{k+1} & p_3X_{k+2} \\ X_{k+2} & X_{k+3}-p_0X_{k+2} & p_2X_{k+1}+p_3X_k & p_3X_{k+1} \\ X_{k+1} & X_{k+2}-p_0X_{k+1} & X_{k+3}-p_0X_{k+2}-p_1X_{k+1} & p_3X_k \\ X_k & X_{k+1}-p_0X_k & X_{k+2}-p_0X_{k+1}-p_1X_k & R_{4,4} \end{pmatrix}. \tag{47}$$

Here, $R_{4,4} = X_{k+3} - p_0X_{k+2} - p_1X_{k+1} - p_2X_k$. If we put $m = 2$ in (46), then after simplification we get the following formulas:

$$X_{2k+3} = X_{k+3}^2 + X_{k+2}(p_1X_{k+2} + 2p_2X_{k+1} + 2p_3X_k) + p_3X_{k+1}^2, \tag{48}$$

$$X_{2k+2} = X_{k+2}(2X_{k+3} - p_0X_{k+2}) + X_{k+1}(p_2X_{k+1} + 2p_3X_k), \tag{49}$$

$$X_{2k+1} = X_{k+2}^2 + X_{k+1}(2X_{k+3} - 2p_0X_{k+2} - p_1X_{k+1}) + p_3X_k^2, \tag{50}$$

$$X_{2k} = X_{k+1}(2X_{k+2} - p_0X_{k+1}) + X_k(2X_{k+3} - 2p_0X_{k+2} - 2p_1X_{k+1} - p_2X_k). \tag{51}$$

We also need the formula for X_{2k+4}. It can be obtained from (49) if we replace k by $k+1$ and use (37) for X_{k+4}. It is as follows:

$$X_{2k+4} = X_{k+3}(p_0X_{k+3} + 2p_1X_{k+2} + 2p_2X_{k+1} + 2p_3X_k) + X_{k+2}(p_2X_{k+2} + 2p_3X_{k+1}). \tag{52}$$

Algorithm 3 Computing the fourth-order linear recurrence $\{X_k(p_0, p_1, p_2, p_3)\}$

Input: $N = \sum_{i=0}^{m-1} b_i 2^i$, $(b_{m-1} = 1)$
p_0, p_1, p_2, p_3
Output: X_N, X_{N+1}, X_{N+2}, X_{N+3}
$X_1 \leftarrow 0$; $X_2 \leftarrow 0$; $X_3 \leftarrow 1$; $X_4 \leftarrow p_0$
for j from $m-2$ to 0 by -1 **do**
$x_1 \leftarrow X_1$; $x_2 \leftarrow X_2$; $x_3 \leftarrow X_3$; $x_4 \leftarrow X_4$
  if $b_j = 1$ **then**
    $X_1 \leftarrow x_3^2 + x_2(2x_4 - 2p_0x_3 - p_1x_2) + p_3x_1^2$;
    $X_2 \leftarrow x_3(2x_4 - p_0x_3) + x_2(p_2x_2 + 2p_3x_1)$;
    $X_3 \leftarrow x_4^2 + x_3(p_1x_3 + 2p_2x_2 + 2p_3x_1) + p_3x_2^2$;
    $X_4 \leftarrow x_4(p_0x_4 + 2p_1x_3 + 2p_2x_2 + 2p_3x_1) + x_3(p_2x_3 + 2p_3x_2)$
  else if
    $X_1 \leftarrow x_2(2x_3 - p_0x_2) + x_1(2x_4 - 2p_0x_3 - 2p_1x_2 - p_2x_1)$;
    $X_2 \leftarrow x_3^2 + x_2(2x_4 - 2p_0x_3 - p_1x_2) + p_3x_1^2$;
    $X_3 \leftarrow x_3(2x_4 - p_0x_3) + x_2(p_2x_2 + 2p_3x_1)$;
    $X_4 \leftarrow x_4^2 + x_3(p_1x_3 + 2p_2x_2 + 2p_3x_1) + p_3x_2^2$
  end if
end for
return X_1, X_2, X_3, X_4

As is seen from the algorithm, we need $6m$ multiplications and $4m$ squarings to compute the terms X_N, X_{N+1}, X_{N+2}, X_{N+3}. Here, as in the previous section, we count only "big" multiplications.

5. Computation of Linear Recurrence Sequences of Any Order

Let $\{W_k(a_0, \ldots, a_{n-1}; p_0 \ldots p_{n-1})\}$ be an nth-order linear recurrence defined by the relation $f_{k+n} = \sum_{i=0}^{n-1} p_i f_{k+n-1-i}$, with the initial values $W_i = a_i$ $(0 \le i \le n-1)$. Let $\{X_k(p_0, \ldots, p_{n-1})\}$ be the sequence that is derived from $\{W_k\}$ if $a_{n-1} = 1$ and the other $a_i = 0$. Using the matrix method as in Lemma 2 and by mathematical induction we get the following formulas

$$W_k = a_{n-1} X_k + \sum_{j=1}^{n-1} \left(p_j \sum_{i=0}^{j-1} a_{n-2-i} X_{k-j+i} \right), \tag{53}$$

$$W_k = \sum_{j=0}^{n-1} \left(a_{n-1-j} - \sum_{i=0}^{n-j-2} a_{n-j-2-i} p_i \right) X_{k+j}. \tag{54}$$

If we put $n = 4$ in these formulas, then we obtain (38) and (39).

Repeating the arguments of the previous section we get the matrix formula

$$\begin{pmatrix} X_{2k+n-1} \\ \vdots \\ X_{2k+1} \\ X_{2k} \end{pmatrix} = \begin{pmatrix} X_{k+n-1} & {}^{n-2}W_{k+n-1} & {}^{n-3}W_{k+n-1} & \ldots & {}^{0}W_{k+n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_{k+1} & {}^{n-2}W_{k+1} & {}^{n-3}W_{k+1} & \ldots & {}^{0}W_{k+1} \\ X_k & {}^{n-2}W_k & {}^{n-3}W_k & \ldots & {}^{0}W_k \end{pmatrix} \begin{pmatrix} X_{k+n-1} \\ \vdots \\ X_{k+1} \\ X_k \end{pmatrix}. \tag{55}$$

Here, as above, ${}^{i}W_k$ denotes $W_k(a_0, \ldots, a_{n-1})$ with only one nonzero $a_i = 1$. Let $R = (r_{i,j})$ be the matrix from (55). It has a special form, see (29) and (47). Note that if we know two rows of the matrix R which have numbers of different parity, then we can get the other rows. For example, we assume we know a formula which relates $X_{2k+\ell}$ to X_{k+i} $(0 \le i \le n-1)$, in other words we know the $(n-\ell)th$ row. If we replace k by $k+1$ and use $X_{k+n} = \sum_{i=0}^{n-1} p_i X_{k+n-1-i}$, then we get the formula for $X_{2k+\ell+2}$ that corresponds to the $(n-\ell-2)th$ row. Repeating this procedure we obtain all rows with numbers of the same parity as the parity of the $(n-\ell)th$ row. Thus, to get all formulas that will be used in the algorithm, we need to know formulas for X_{2k}, X_{2k+1}.

For the elements of R using (53), (54) we obtain

$$r_{i,j} = \begin{cases} X_{k+n-1-(i-j)} - \sum_{l=0}^{j-2} p_l X_{k+n-2-l-(i-j)}, & \text{if } i \ge j, \\ \sum_{l=j-1}^{n-1} p_l X_{k+n-2-l-(i-j)}, & \text{if } i < j. \end{cases} \tag{56}$$

Also, from the two last rows in (55) we obtain the formulas which relate X_{2k}, X_{2k+1} to X_{k+i} $(0 \le i \le n-1)$. These formulas are of the same form as (50), (51).

$$X_{2k} = eX^2_{k+(n-1)/2}+ \sum_{i=0}^{\lfloor v\rfloor} X_{k+\lfloor v\rfloor-i}\left(2X_{k+\lceil v\rceil+1+i} - p_{2i+e}X_{k+\lfloor v\rfloor-i} - 2\sum_{j=0}^{2i-1+e} p_j X_{k+\lceil v\rceil+i-j}\right), \tag{57}$$

$$X_{2k+1} = p_{n-1}X_k^2 + (1-e)X^2_{k+n/2}+ \sum_{i=0}^{\lceil v\rceil-1} X_{k+\lceil v\rceil-i}\left(2X_{k+\lfloor v\rfloor+2+i} - p_{2i+1-e}X_{k+\lceil v\rceil-i} - 2\sum_{j=0}^{2i-e} p_j X_{k+\lfloor v\rfloor+1+i-j}\right). \tag{58}$$

Here, $e = n \bmod 2$ and $v = n/2 - 1$.

The scheme for computing the terms of $\{W_k(a_0, \ldots, a_{n-1}; p_0, \ldots, p_{n-1})\}$ is:
(i) Using $X_{k+n} = \sum_{i=0}^{n-1} p_i X_{k+n-1-i}$ and repeating the replacement of k by $k+1$ in (57), (58) without removing brackets, we obtain the formulas for X_{2k+i} $(0 \le i \le n)$. These formulas determine the rules of transition from the terms X_{k+i} $(0 \le i \le n-1)$ to X_{2k+i} $(0 \le i \le n-1)$ and also to X_{2k+1+i} $(0 \le i \le n-1)$.
(ii) By using these formulas we obtain an algorithm for computing $\{X_k\}$ that is similar to Algorithm 3.
(iii) To get the value W_N, we need to use (54) after we have computed X_{N+i} $(0 \le i \le n-1)$ by the algorithm in (ii).
(iv) In order to obtain W_{N+1} we use the recurrence relation to get X_{N+n} from X_{N+i} $(0 \le i \le n-1)$ and use (54).

Remark. To compute the Nth term of an nth-order linear recurrence we need $n(n+1)/2\log_2 N$ multiplications[3]. Indeed, when n is even, the formulas for X_{2k+2i} $(0 \le i \le n/2)$ contain $n/2$ multiplications,[4] and for $X_{2k+2i+1}$ $(0 \le i \le n/2-1)$ contain $n/2+1$ multiplications. It is easy to see that each step of the algorithm needs $n/2$ formulas of the first type and $n/2$ formulas of the second type. Then, to compute X_{2k+i} $(0 \le i \le n-1)$ or X_{2k+1+i} $(0 \le i \le n-1)$ using X_{k+i} $(0 \le i \le n-1)$ we need $n(n+1)/2$ multiplications. Thus, computing X_{N+i} $(0 \le i \le n-1)$ needs $n(n+1)/2\log_2 N$ multiplications. Since (54) does not contain "big" multiplications, the above statement is proved for even n. The proof for odd n by analogous.

We implemented the above scheme in Wolfram Mathematica. An implementation is available at `http://community.wolfram.com/groups/-/m/t/940869`. The main function **AnyOrderRecurrence**$[a, p, N]$ returns $W_N(a_0, \ldots, a_{n-1}; p_0, \ldots, p_{n-1})$, where N is a positive integer and a, p are strings of length n.

[3]We use such a complexity model that multiplications involving p_i are similar to additions.
[4]Since they were derived from (57) without removing brackets.

Acknowledgments. The author is very grateful to A. Bostan for pointing to the reference [5] and for the evidence that our algorithm is one particular way of implementing Fiduccia's algorithm, where modular polynomial squarings are hard-coded.

References

[1] W.W. Adams, Characterizing pseudoprimes for third-order linear recurrences, *Math. Comp.* **48**.177 (1987), 1-15.

[2] D. Bleichenbacher, W. Bosma, A.K. Lenstra, Some remarks on Lucas-based cryptosystems, in: *Annual International Cryptology Conference*, Springer, Berlin, Heidelberg, 1995, 386-396.

[3] G.H. Cho, N. Koo, E. Ha, S. Kwon, New cube root algorithm based on the third order linear recurrence relations in finite fields, *Des. Codes Cryptogr.* **75**.3 (2015), 483-495.

[4] E.W. Dijkstra, In honour of Fibonacci, in: *Program Construction. Lecture Notes in Computer Science 69*, ed. F.L. Bauer et al., Springer, Berlin, Heidelberg, 1979, 49-50.

[5] C.M. Fiduccia, An efficient formula for linear recurrences, *SIAM J. Comput.* **14**.1 (1985), 106-112.

[6] G. Gong, A. Hassan, H. Wu, and A. Yousse, *An Efficient Algorithm for Exponentiation in DH key Exchange and DSA in Cubic Extension Fields*, Faculty of Mathematics, University of Waterloo, 2002.

[7] G. Gong, and L. Harn, Public-key cryptosystems based on cubic finite field extensions, *IEEE Trans. Inform. Theory* **45**.7 (1999), 2601-2605.

[8] D. Gries, G. Levin, Computing Fibonacci numbers (and similarly defined functions) in log time, *Inform. Process. Lett.* **11**.2 (1980), 68-69.

[9] A.F. Horadam, Basic properties of a certain generalized sequence of numbers, *Fibonacci Quart.* (1965) **3**.3, 161-176.

[10] A.F. Horadam, Special properties of the sequence $W_n(a, b; p, q)$, *Fibonacci Quart.* **5**.4 (1967), 424-434.

[11] M. Joye and J.-J. Quisquater, Efficient computation of full Lucas sequences, *Electronics Letters* **32**.6 (1996), 537-538.

[12] J.C.P. Miller, D.S. Brown, An algorithm for evaluation of remote terms in a linear recurrence sequence, *Comput. J.* **9**.2 (1966), 188-190.

[13] S. Rabinowitz, Algorithmic manipulation of third-order linear recurrences, *Fibonacci Quart.* **34** (1996), 447-463.

[14] C.A. Reiter, Exact Horadam numbers with a Chebyshevish accent, *Vector* **16** (1999), 122-131.

[15] E.L. Roettger, H.C. Williams, and R.K. Guy, Some primality tests that eluded Lucas, *Des. Codes Cryptogr.* **77**.2-3 (2015), 515-539.

[16] Z.H. Sun, Linear recursive sequences and the powers of matrices, *Fibonacci Quart.* **39**.4 (2001), 339-351.