SIAM J. COMPUT.
Vol. 14, No. 1, February 1985

AN EFFICIENT FORMULA FOR LINEAR RECURRENCES*

CHARLES M. FIDUCCIA†

Abstract. The solutions to a scalar, homogeneous, constant-coefficient, linear recurrence are expressible in terms of the powers of a companion matrix. We show how to compute these powers efficiently via polynomial multiplication. The result is a simple expression for the solution, which does not involve the characteristic roots and which is valid for any module over any commutative ring. The formula yields the nth term of the solution to a kth order recurrence with $O(\mu(k)\cdot\log n)$ arithmetic operations, where $\mu(k)$ is the total number of arithmetic operations required to multiply two polynomials of degree $k-1$. Thus if the ring supports a fast Fourier transform, then $O(k\cdot\log k\cdot\log n)$ operations are sufficient to compute the nth term.

Key words. linear recurrences, difference equations, companion matrices, algebraic complexity, polynomial arithmetic, fast algorithms, parallel algorithms

1. Introduction.

Let the infinite sequence $F_0, F_1, F_2, \cdots$ be a solution to the kth order linear recurrence

$$F_{n+k} = c_0F_n + c_1F_{n+1} + \cdots + c_{k-1}F_{n+k-1}. \tag{1.1}$$

Given the coefficients, the initial values and an arbitrary natural number n, we wish to compute efficiently the nth term F_n without computing all terms which precede it. Toward this end, let $F[i:j]$ denote the row vector $[F_i, F_{i+1}, \cdots, F_j]$. Equation (1.1) can then be written in the vector-matrix form

$$F[n+1:n+k] = F[n:n+k-1]\cdot C, \tag{1.2}$$

where C is the $k\times k$ companion matrix

$$C = \begin{bmatrix} 0 & & & c_0 \\ 1 & & & c_1 \\ & 1 & & c_2 \\ & & \ddots & \vdots \\ & & 1 & c_{k-1} \end{bmatrix}.$$

From (1.2) we see that, in terms of the initial values $F[0:k-1]$, we have

$$F[n:n+k-1] = F[0:k-1]\cdot C^n. \tag{1.3}$$

If in (1.3) we let n range over all multiples of k, the entire solution $F[0:\infty]$ can be written as the infinite matrix equation

$$F[0:\infty] = F[0:k-1]\cdot C^*, \tag{1.4}$$

where, by definition, C^* is the $k\times\infty$ matrix

$$C^* = [I \quad C^k \quad C^{2k} \quad C^{3k} \quad \cdots]. \tag{1.5}$$

Equation (1.4) shows that any solution to (1.1) is a linear combination

$$F_0\beta_1 + \cdots + F_{k-1}\beta_k, \tag{1.6}$$

* Received by the editors February 3, 1983, and in revised form September 19, 1983. A preliminary version of this paper was presented at the Twentieth Annual Allerton Conference on Communication, Control and Computing, Monticello, Illinois, October 1982.

† General Electric R & D Center, Schenectady, New York 12345.

where β_i, henceforth called the ith *basic solution*, is the ith infinite row of C^*. This terminology is suggested by the fact, evident from (1.6), that β_i is the solution to (1.1), when $F[0:k-1]$ is taken as the ith vector of the standard basis.

We also see from (1.3) that the nth term F_n of a solution is the inner product of the vector of its initial values times the first column of C^n. Because the inner product is easy, we will confine our attention to computing the first column of the matrix power C^n.

In any semigroup, the nth power x^n of an element x can be computed with at most $2 \cdot \log n$ semigroup multiplications by the well-known method of repeated squaring

$$x^{2n} = (x^n)^2, \quad x^{2n+1} = x(x^n)^2, \quad x^1 = x.$$

Urbanek [4] suggests this approach for computing F_n with $O(k^3 \cdot \log n)$ arithmetic operations, by using the classical $O(k^3)$ algorithm to multiply $k \times k$ matrices. Gries and Levin [3] give more efficient recursive formulas for computing the required entries on C^n with $O(k^2 \cdot \log n)$ operations; unfortunately, their method sheds no light on what these entries are.

We show that, for the purpose of computing the nth power of a companion matrix C, indeed any polynomial $p(C)$, matrix multiplication can be replaced with modular polynomial multiplication. This not only reveals what the entries of $p(C)$ are, but also yields a simple expression for the nth term F_n. Unlike existing formulas, such as the well-known one for the nth Fibonacci number, no characteristic roots are required. The new expression immediately explains the result of Gries and Levin, gives a more efficient $O(k^{1.59} \log n)$ algorithm, and yields an $O(k \cdot \log k \cdot \log n)$ algorithm over rings that support an FFT (fast Fourier transform).

2. Arithmetic with a companion matrix. In the sequel, the underlying scalar domain, from which the coefficients of the recurrence are taken, is assumed to be an arbitrary commutative ring K with 1. As usual, $K[X]$ denotes the ring of polynomials over K, while $K[X]/(f(X))$ denotes the ring of polynomials modulo the monic polynomial

$$f(X) = X^k - (c_0 + c_1 X + \cdots + c_{k-1} X^{k-1}). \tag{2.1}$$

For concreteness, we view $K[X]/(f(X))$ as the set of all polynomials of degree less than k in which arithmetic is done modulo $f(X)$. Doing arithmetic in $K[X]/(f(X))$ is then equivalent to doing it in $K[X]$ and taking the result modulo $f(X)$. To avoid the repeated use of the operator $\bmod f(X)$, the congruence class in $K[X]/(f(X))$ containing X will be denoted by ξ. This means that for any $p(X)$ in $K[X]$, the element $p(\xi)$ in $K[X]/(f(X))$ "is" $p(X) \bmod f(X)$. In particular, $f(\xi) = 0$; so that

$$\xi^k = c_0 + c_1 \xi + \cdots + c_{k-1} \xi^{k-1}. \tag{2.2}$$

It is well known that $f(X)$ is the characteristic polynomial of the companion matrix C; so that $f(C) = 0$. Since the characteristic polynomial of a companion matrix "comes free", from its last column, one obvious way to compute C^n is first to use the method of repeated squaring to compute $X^n \bmod f(X) = \xi^n = r(\xi)$, say, and then to compute $r(C) = C^n$.

The problem thus reduces to doing fast multiplication in $K[X]/(f(X))$. This can always be done with $O(k^2)$ operations, by doing one multiplication and one division. Modular multiplication can, in fact, be done much faster. Indeed, it has the same complexity as polynomial multiplication, because division is reducible to multiplication [1]. We will henceforth let $\mu(k) = \mu(k, K)$ denote the total number of arithmetic operations required to multiply two polynomials of degree $k-1$ in $K[X]$. We note

[1], [2] that $\mu(k) = O(k^{1.59})$ for any ring K, and that $\mu(k) = O(k \cdot \log k)$ if K supports an FFT or any Vandermonde transform [2]. For our purposes, the most relevant result is the following:

Fact. $X^n \bmod f(X) = \xi^n$ can be computed with $O(\mu(k) \cdot \log n)$ operations.

Returning to the computation of C^n, we see that once $r(\xi)$ has been computed, we can then compute $C^n = r(C)$, with an additional $O(k^3)$ operations, by using Horner's rule and the fact that, owing to its sparseness, C can be multiplied by any $k \times k$ matrix with $O(k^2)$ operations. This yields an $O(\mu(k) \cdot \log n + k^3)$ algorithm for computing C^n. A slight improvement is possible because polynomials $p(C)$ of low degree are sparse; we need not pursue this, however, for we will show that C^n can be computed with only $O(\mu(k) \cdot \log n + k^2)$ operations. In fact, if our only interest is to compute the nth term F_n, we need not compute C^n at all, as we will shortly derive an efficient polynomial expression for any desired column of C^n.

This more efficient method is based on a simple lemma from Fiduccia [2]. Consider the equivalence between $K[X]/(f(X))$ and K^k (obtained by choosing the basis $1, \xi, \xi^2, \cdots, \xi^{k-1}$ for the former and the standard basis for the latter) that identifies $p(\xi) = p_0 + p_1\xi + \cdots + p_{k-1}\xi^{k-1}$ with its *column vector* of coefficients $\tilde{p} = [p_0\, p_1 \cdots p_{k-1}]$. Formally, this equivalence is given by

$$p(\xi) = [1 \quad \xi \quad \xi^2 \quad \cdots \quad \xi^{k-1}] \cdot \tilde{p}. \tag{2.3}$$

We shall henceforth make no distinction between $p(\xi)$ and $\tilde{p}$, calling them equal. This gives us the benefits of both matrix notation and polynomial notation.

We are immediately confronted with the question of which polynomial in $K[X]/(f(X))$ is equal to the vector $C \cdot \tilde{p}$ in K^k. The following lemma shows that pre-multiplication by the matrix C is equivalent to multiplication by $X \bmod f(X)$.

LEMMA 2.1 [2]. *For any $p(\xi)$ in $K[X]/(f(X))$, $C \cdot \tilde{p} = \xi p(\xi)$.*

Proof. The proof is by direct computation of $C \cdot \tilde{p}$ and $\xi p(\xi) = Xp(X) \bmod f(X)$. Both computations are trivial, since C is sparse and since the second computation requires only one step of the long division process. Alternatively, we may derive it formally from (2.3), using the identity $f(\xi) = 0$. More elegantly, and a hint of things to come, note that the ith column of the companion matrix C is the element $p(\xi) = \xi^i$; so that $C \cdot \tilde{p} = p_0\xi + \cdots + p_{k-1}\xi^k = \xi p(\xi)$. □

COROLLARY 2.2. *For any $q(X)$ in $K[X]$ and any $p(\xi)$ in $K[X]/(f(X))$, $q(C) \cdot \tilde{p} = q(\xi)p(\xi)$.*

Proof. We use induction on the degree of $q(X)$. Say $q(X) = s(X)X + q_0$; so that $q(C) \cdot \tilde{p} = s(C)(C \cdot \tilde{p}) + q_0\tilde{p}$. Using Lemma 2.1 and induction, this is equal to $s(\xi)(\xi p(\xi)) + q_0 p(\xi) = (s(\xi)\xi + q_0)p(\xi) = q(\xi)p(\xi)$. □

The reader may have noticed that we appear to be working on the wrong problem! For if we were to take $q(X) = X^n$ in Corollary 2.2, we would obtain an efficient method for pre-multiplying by C^n, not post-multiplying by it as required by (1.3). We resolve this problem by using Corollary 2.2 to compute the columns of C^n rather than the product itself.

COROLLARY 2.3. *For any $q(X)$ in $K[X]$, the ith column of $q(C)$ is $q(\xi)\xi^{i-1}$.*

Proof. Choose $p(\xi) = \xi^{i-1}$ in Corollary 2.2; so that $\tilde{p}$ is the ith column of the $k \times k$ identity matrix I, i.e., consider the matrix identity $q(C) = q(C)I$. Since the ith column of I is $p(\xi) = \xi^{i-1}$, the ith column of $q(C)$, on the left, is $q(C) \cdot \tilde{p} = q(\xi)p(\xi) = q(\xi)\xi^{i-1}$. □

PROPOSITION 2.4. *For any $q(X)$ in $K[X]$, $q(C)$ can be computed with $N(q) + (k-1)(2k-1)$ arithmetic operations, where $N(q)$ is the number of operations needed to compute $q(\xi)$. In particular, C^n can be computed with $O(\mu(k) \cdot \log n + k^2)$ operations.*

Proof. By Corollary 2.3, the first column of $q(C)$ is $q(\xi)$; this can be done with $N(q)$ operations by hypothesis. Using Corollary 2.3 again, along with Lemma 2.1, the $(i+1)$th column of $q(C)$ is C times its ith column. Each of these $k-1$ multiplications by C takes at most k multiplications and $k-1$ additions; hence, all the remaining entries of $q(C)$ can be done with $(k-1)(2k-1)=O(k^2)$ operations. We know that the first column ξ^n of C^n can be computed with $N(q)=O(\mu(k)\cdot\log n)$ operations; so the total number of operations to compute all the entries of C^n is $O(\mu(k)\cdot\log n+k^2)$. □

3. An expression for F_n. As previously noted, (1.3) shows that F_n is the inner product of the vector $F[0:k-1]$ of initial values times the first column of C^n. By Corollary 2.3, this column is $X^n \bmod f(X)=\xi^n$. Thus if $[\gamma_0,\cdots,\gamma_{k-1}]$, say, is the coefficient vector of ξ^n, then

$$F_n=\gamma_0F_0+\cdots+\gamma_{k-1}F_{k-1}. \tag{3.1}$$

Using the inner product operator $\langle\cdot,\cdot\rangle$ we obtain the simple expression:

THEOREM 3.1.

$$F_n=\langle[F_0\ \cdots\ F_{k-1}],\xi^n\rangle=\langle[F_0\ \cdots\ F_{k-1}],X^n \bmod f(X)\rangle.$$

Since the inner product can be computed with $2k-1=O(\mu(k))$ operations, we obtain, as a corollary, our main complexity result:

PROPOSITION 3.2. *The nth term F_n of a kth order linear recurrence can be computed with $O(\mu(k)\cdot\log n)$ arithmetic operations, where $\mu(k)=\mu(k,K)$ is the total number of operations required to multiply two polynomials of degree $k-1$ in $K[X]$.*

The bulk of the work is the computation of $\xi^n=X^n \bmod f(X)$. We may of course use any method at our disposal for this computation, making use of any special knowledge about the polynomial $f(X)$ and the base ring K. It is well-known that solutions exist based on the roots of $f(X)$; those solutions are, of course, equivalent to the expression given by Theorem 3.1 after an appropriate change of basis for $K[X]/(f(X))$ based on the factorization properties of $f(X)$ over K or its extensions.

Equivalence, however, has nothing to do with complexity. Consider for example the case when K is a field and $f(X)$ is irreducible; so that $K[X]/(f(X))$ is a field of degree k over K. The classical expression for F_n is a linear combination of the nth power of each of the k roots of $f(X)$. Since these roots are in $K[X]/(f(X))$, the nth power of each root can be computed with $O(\mu(k)\cdot\log n)$ operations; however, as this must be done for each of the k roots, the total cost will be $O(k\cdot\mu(k)\cdot\log n)$. So, even if we ignore the cost of finding the roots of $f(X)$, the classical method is k times less efficient than our solution; this is because we compete the nth power of only one root of $f(X)$—the "symbolic" root ξ.

4. The universal solution. Let us delve further into other possible solutions to (1.1) by noting that its right-hand side is simply a K-linear combination valid over any K-module M (essentially any set closed under linear combinations). Thus, given initial values $F_0,F_1,\cdots,F_{k-1}$ in M, the recurrence will generate a solution $F_0,F_1,F_2,\cdots$ in M. This suggests that Theorem 3.1 has a more general abstract form:

THEOREM 4.1. *For any K-linear mapping L, from $K[X]/(f(X))$ into any K-module M, the infinite sequence $L(1),L(\xi),L(\xi^2),L(\xi^3),\cdots$ is a solution to* (1.1) *in M. All solutions to* (1.1) *in all K-modules M are of this form.*

Proof. It is clear from (2.2) that the infinite sequence of powers

$$1,\xi,\xi^2,\xi^3,\cdots \tag{4.1}$$

is a solution to (1.1) in $K[X]/(f(X))$, since for all n, we have

$$\xi^{n+k} = c_0\xi^n + c_1\xi^{n+1} + \cdots + c_{k-1}\xi^{n+k-1}. \tag{4.2}$$

Apply L to both sides and invoke its linearity to establish the first part of Theorem 4.1. To establish that all solutions are as claimed, let $F_0, F_1, F_2, \cdots$ be a solution to (1.1) in M. Since $1, \xi, \xi^2, \cdots, \xi^{k-1}$ is a basis for $K[X]/(f(X))$, there is always a linear mapping L such that

$$L(1) = F_0, \qquad L(\xi) = F_1, \cdots, L(\xi^{k-1}) = F_{k-1}. \tag{4.3}$$

For any element $p(\xi) = p_0 + \cdots + p_{k-1}\xi^{k-1}$ in $K[X]$, we then have $L(p(\xi)) = p_0F_0 + \cdots + p_{k-1}F_{k-1}$. We establish that $F_n = L(\xi^n)$ for all n by induction. It is true for $n = 0, \cdots, k-1$ by definition of L. Assume that $F_i = L(\xi^i)$ for all $i < n+k$. Since, by hypothesis, $F_0, F_1, F_2, \cdots$ is a solution to (1.1), we can substitute (4.3) into (1.1) to get

$$F_{n+k} = c_0L(\xi^n) + \cdots + c_{k-1}L(\xi^{n+k-1}). \tag{4.4}$$

The linearity of L and (4.2) then yield $F_{n+k} = L(\xi^{n+k})$ for all n. □

We again see that the nth term F_n of any solution in any K-module M is

$$F_n = L(\xi^n) = \gamma_0F_0 + \gamma_1F_1 + \cdots + \gamma_{k-1}F_{k-1}, \tag{4.5}$$

where $\xi^n = \gamma_0 + \gamma_1\xi + \cdots + \gamma_{k-1}\xi^{k-1}$, say. This expression is identical to (3.1), except that the F_i are now arbitrarily chosen initial values in M. Note that Theorem 4.1 is an independent abstract reaffirmation of our previous results. No appeal was made to (basis dependent) companion matrices.

Theorem 4.1 shows that the sequence (4.1) in $K(X)/(f(X))$ is, in some sense, the *universal solution* to (1.1), since all other solutions in all other K-modules are linear images of it. It is interesting to pursue the reason for this universality to get an intuitive feeling for it.

For notational simplicity, we confine our attention to solutions in K and consider the infinite matrix equation $F[0:\infty] = F[0:k-1] \cdot C^*$ given by (1.4), where by definition

$$C^* = [I \quad C^k \quad C^{2k} \quad C^{3k} \quad \cdots].$$

As previously noted in (1.6), $F[0, \infty]$ is the linear combination

$$F_0\beta_1 + \cdots + F_{k-1}\beta_k,$$

where β_i is the ith infinite row of C^*. This row is the ith basic solution to (1.1) generated by choosing the ith element of the standard basis as the vector $F[0:k-1]$ of initial values. Hence, the rows of C^* are precisely the k basic solutions from which all other solutions can be obtained. The universality of solution (4.1) is now evident from Corollary 2.3; for the powers of ξ are precisely the column vectors of C^*, i.e.,

$$C^* = [1 \quad \xi \quad \xi^2 \quad \xi^3 \quad \cdots]. \tag{4.6}$$

Consequently, as we generate the next element in the power sequence (4.1), in $K[X]/(f(X))$, we are computing the next column of C^*, and thus *simultaneously* computing the next term of every one of the k basic solutions $\beta_1, \cdots, \beta_k$.

As an interesting application of these observations, consider the obvious shift-register implementation of (1.1). This is shown in Fig. 1, and is valid over any K-module M. The boxes represent delay elements (to store the current state), whose internal labels comprise the initial state-vector of the shift-register. In this implementation, the

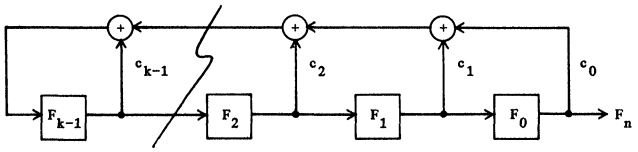

FIG. 1. *The obvious implementation of the recurrence* (1.1).

initial state consists of the initial values of the recurrence. At each step, the circuit performs k scalar multiplications and $k-1$ additions in M to compute the next term of the single *specific solution* generated by the given initial values $F_0, \cdots, F_{k-1}$.

Now consider the shift-register shown in Fig. 2, which operates strictly over the base ring K. If its current state is a column k-vector $\tilde{p}=[p_0, \cdots, p_{k-1}]$, say, then its next state will clearly be $C \cdot \tilde{p}$. Hence, by Lemma 2.1, the shift-register's next-state function is multiplication by ξ. It follows that if its initial state is any $p(\xi)$ in $K[X]/(f(X))$, its state after the nth iteration will be $\xi^n p(\xi)$. If we start it with the initial state $p(\xi)=[1, 0, \cdots, 0]=1$, in $K[X]/(f(X))$, it will generate the power sequence (4.1). In particular, the infinite sequence produced at the output of the ith delay element will be precisely the basic solution β_i.

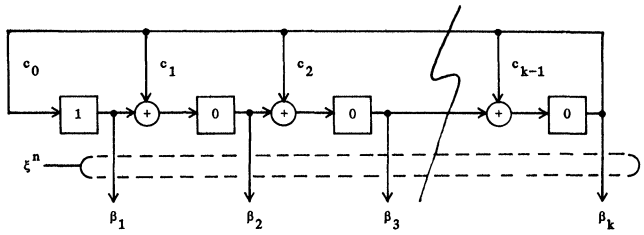

FIG. 2. *The generator of the powers* $1, \xi, \xi^2, \xi^3, \cdots$.

This is interesting from a complexity viewpoint; for this "power generator" simultaneously computes all k basic solutions to (1.1) using only k multiplications and $k-1$ additions (both in K) per iteration. Moreover, this shift-register is inherently faster than the one in Fig. 1, since that one requires at least $\log k$ time, per iteration, to compute the term F_{n+k}. This speed-up is accomplished at the expense of a high "fan-out" from the last stage of the shift-register in Fig. 2.

Considering the fact that (1.1) uses $2k-1$ operations to generate each term of a solution, the above observations establish the following rather surprising complexity result:

PROPOSITION 4.2. *The first n terms of all basic solutions* $\beta_1, \cdots, \beta_k$ *to the kth order linear recurrence* (1.1) *can be computed with* $(2k-1)(n-k-1)=O(k \cdot n)$ *arithmetic operations.*

Proof. Follows directly from (1.5) and (1.6), because these $k \cdot n$ terms comprise the first n columns of the matrix C^*. Since its first k columns form the identity matrix, start with column $k+1$ (it comes free from column k of C). To generate each of the remaining $n-(k+1)$ columns, pre-multiply the most current column by the matrix C; this uses at most $2k-1$ operations per column. □

Since the first n columns of C^* contain $O(k \cdot n)$ entries, this is (within a factor of 2) an optimal algorithm for computing these terms. Note that each new term of each basic solution β_i is being computed with only one multiplication and one addition per term. By contrast, (1.1) always uses $2k-1$ operations per term, even for a basic solution.

Proposition 4.2 remains valid over any algebraic structure in which equations (1.1) and (1.5) make sense; for example, the coefficient ring K may be replaced by an arbitrary semiring. An even more general valid setting is obtained by replacing the K-module structure by an (additive) monoid with endomorphisms.

Another interesting property of the second shift-register is that it operates independently of the initial values $F_0, \cdots, F_{k-1}$ of the recurrence. These values may be introduced, at any time, by performing a K-linear transformation on the current state-vector ξ^n of the shift-register. One potentially practical bnefit of this is that at any stage, having done the bulk of the computation, we can experiment with various initial values, of the recurrence, without having to restart the computation. Indeed, as a consequence of the universality of the power sequence $1, \xi, \xi^2, \xi^3, \cdots$ we may "fan-out" the state-vector to as many linear transformations as desired and *simultaneously* compute as many solutions, each with its own initial values, in as many different K-modules, as desired.

5. Conclusions. We have shown that the power sequence $X^n \bmod f(X)$, $n \geqq 0$, plays a fundamental role in the efficient solution to a linear recurrence. This was done both concretely, using companion matrices, and abstractly, using K-modules. In the process, we learned how to do efficient arithmetic with a companion matrix and derived a simple generic formula for the solution. The formula does not involve characteristic roots and yields the nth term of a kth order recurrence with $O(k \cdot \log k \cdot \log n)$ arithmetic operations over any ring which supports an FFT. This is a considerable improvement over the $O(k^3 \cdot \log n)$ method and the previously best known $O(k^2 \cdot \log n)$ algorithm.

We have also shown that, unlike the obvious method suggested by (1.1), which uses $2k-1$ operations to compute each term of a single particular solution, all k basic solutions can be simultaneously computed with no more than two operations per term. This k-fold speed-up is not merely an asymptotic improvement, but an honest-to-goodness gain valid for any k.

This speed-up begs the issue of whether an algorithm exists to generate the first n terms of a single solution with fewer than the obvious $O(k \cdot n)$ operations. The author has recently established that this computation can be done with only $O(n \cdot \log k)$ operations, via the FFT, even for the nonhomogeneous case. Because the methods are substantially different from those presented here, we leave it for a future paper.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] C. M. FIDUCCIA, *On the algebraic complexity of matrix multiplication*, Ph.D. Thesis, Brown University, Providence, RI, 1973.

[3] D. GRIES AND G. LEVIN, *Computing Fibonacci numbers (and similarly defined functions) in* log *time*, Inform. Proc. Lett., 11 (1980), pp. 68–69.

[4] F. J. URBANEK, *An* $O(\log n)$ *algorithm for computing the nth element of the solution of a difference equation*, Inform. Proc. Lett., 11 (1980), pp. 66–67.