

# A linear algorithm for computing all the squares of a Fibonacci string

*Costas S. Iliopoulos*

Department of Computer Science  
King's College London  
London WC2R 2LS  
England &  
Curtin University of Technology  
*csi@dcs.kcl.ac.uk*

*Dennis Moore*

School of Computing  
Curtin University of Technology  
GPO Box U 1987  
Perth 6001  
Australia  
*moore@cs.curtin.edu.au*

*W. F. Smyth*

Department of Computer Science & Systems  
McMaster University  
Hamilton  
Canada L8S 4K1 &  
Curtin University of Technology  
*smyth@mcmaster.ca*

## Abstract

A (finite) Fibonacci string  $F_n$  is defined as follows:  $F_0 = b$ ,  $F_1 = a$ ; for every integer  $n \geq 2$ ,  $F_n = F_{n-1}F_{n-2}$ . For  $n \geq 1$ , the length of  $F_n$  is denoted by  $f_n = |F_n|$ , while it is convenient to define  $f_0 \equiv 0$ . The infinite Fibonacci string  $F$  is the string which contains every  $F_n$ ,  $n \geq 1$ , as a prefix. Apart from their general theoretical importance, Fibonacci strings are often cited as worst case examples for algorithms which compute all the repetitions or all the "Abelian squares" in a given string. In this paper we provide a characterization of all the squares in  $F$ , hence in every prefix  $F_n$ ; this characterization naturally gives rise to a  $\Theta(f_n)$  algorithm which specifies all the squares of  $F_n$  in an appropriate encoding. This encoding is made possible by the fact that the squares of  $F_n$  occur consecutively, in "runs", the number of which is  $\Theta(f_n)$ . By contrast, the known general algorithms for the computation of the repetitions in an arbitrary string require  $\Theta(f_n \log f_n)$  time (and produce  $\Theta(f_n \log f_n)$  outputs) when applied to a Fibonacci string  $F_n$ .

**Keywords** Combinatorial algorithms, string algorithms, lower bounds.

Proceedings of CATS'96 (Computing: The Australasian Theory Symposium), Melbourne, Australia, January 29–January 30 1996.

## 1 Introduction

Fibonacci numbers and strings have been studied extensively over the years (see [8,3]). Here we are interested in the Fibonacci numbers and strings as a tool used in the design and analysis of algorithms. There is a plethora of algorithms whose analysis makes use of Fibonacci numbers; e.g., searching, sorting, hashing, random number generation, Euclid's gcd computation, etc. [11]. Fibonacci strings are often cited as worst case examples for algorithms in string pattern matching (e.g., the Knuth-Morris-Pratt algorithm, the Boyer-Moore algorithm, the Aho-Corashic automaton [1]) and in string statistics (computing all the repetitions [4], computing all the "Abelian squares" [6]).

A (finite) Fibonacci string  $F_n$  is defined as follows:  $F_0 = b$ ,  $F_1 = a$ ; for every integer  $n \geq 2$ ,  $F_n = F_{n-1}F_{n-2}$  (see Figure 1). For  $n \geq 1$ , the length of  $F_n$  is denoted by  $f_n = |F_n|$ , while it is convenient to define  $f_0 \equiv 0$ . The infinite Fibonacci string  $F$  is the string which contains every  $F_n$ ,  $n \geq 1$ , as a prefix.

Furthermore, if a given string  $x$  defined on an arbitrary alphabet  $A$  can be written in the form  $x = yu^kz$  for some integer  $k \geq 2$ , some (possibly empty) strings  $y$  and  $z$ , and some nonempty string  $u$ , then  $u^k \equiv uu \cdots u$  ( $k$  times) is said to be a repetition in  $x$ . If in particular  $k = 2$ , the repetition is called a square. Thus, for example, the string  $x = ababcacacaab$  defined on  $A = \{a, b, c\}$  contains the repetition  $(ca)^3$  and the squares

$F_0 = b$   
 $F_1 = a$   
 $F_2 = ab$   
 $F_3 = aba$   
 $F_4 = abaab$   
 $F_5 = abaababa$   
 $F_6 = abaababaabaab$   
 $F_7 = abaababaabaababaabaab$   
 $F_8 = abaababaabaababaabaababaabaab$   
 $F_9 = abaababaabaababaabaababaabaababaabaababaabaab$

Figure 1: Fibonacci Strings

$(ab)^2$ ,  $(ac)^2$ , and  $a^2$ . The study of repetitions in strings is motivated by the equivalent problem, encountered by molecular biologists, of automatically detecting repeated regions (with errors) in DNA and protein sequences (see [17]).

There exist three well-known algorithms [2,4,13] for finding all the repetitions in a given string  $x = x[1..|x|]$ . Each of these algorithms is asymptotically optimal, executing in time  $\Theta(|x| \log |x|)$ , which is also the time required [13] merely to recognize whether or not  $x$  contains a repetition. Indeed, the execution time achieved by the three algorithms depends on an encoding of repetitions in triples  $(i, p, k)$  denoting

$$x[i..i + kp - 1] = x[i..i + p - 1]^k,$$

where it is required that the substring  $x[i..i + p - 1]$  be *primitive* — that is, not itself a repetition ;  $x[i..i + j]$  denotes the substring  $x$  that starts at position  $i$  and has length  $j$ . It is easy to see that, without this primitivity requirement, the straightforward reporting of distinct squares in a given string  $x$  might require as many as  $\Theta(n^2)$  outputs: for example,  $x = a^n$  contains  $\lfloor n^2/4 \rfloor$  distinct squares but it can be encoded in only one triple  $(1, 1, n)$ . Thus, the encoding of the output is of critical importance to the performance of the algorithms.

In [4] it is shown that, in terms of the  $(i, p, k)$ -encoding, Fibonacci strings  $F_n$  give rise to  $\Theta(f_n \log f_n)$  distinct repetitions, and so in some sense represent a worst case, both for these algorithms and for this encoding. It is not shown, however, whether or not there might exist some other encoding of repetitions which would, at least for Fibonacci strings, perhaps for all strings, make it possible to produce an asymptotically smaller amount of output.

Fibonacci strings also represent a worst case for other algorithms dealing with generalized

repetitions. Cummings and Smyth [6] have shown that  $F_n$  contains  $\Theta(f_n^2)$  “weak repetitions” (Abelian squares), and it turns out also that the “covers” [9,10,14] of a circular Fibonacci string are of cardinality  $\Theta(f_n^2)$ .

In Section 2 of this paper we provide a complete characterization of the squares in Fibonacci strings; in particular, we show that they occur in “runs” at adjacent positions of  $F$ , where each run consists of cyclic rotations of some Fibonacci string  $F_k$ ,  $k \geq 1$ . In Section 3 we show that the number of runs of squares in a finite Fibonacci string  $F_n$  is  $\Theta(f_n)$ , and so we are able to describe a simple linear (in terms of  $f_n$ ) algorithm, executing in  $\Theta(f_n)$  time, to compute all of them.

This algorithm depends on a slightly modified encoding of the output which reports runs of squares using only a single triple of integers. It thus remains an open question, of considerable theoretical interest, whether or not there exist classes of strings  $x$  that give rise to  $\Theta(|x| \log |x|)$  distinct repetitions over all “appropriate” encodings of the output.

## 2 Characterizing the squares

In this section we adopt a four-stage approach to characterizing all of the squares in  $F$ :

- (i) By expressing  $F$  only in terms of  $F_n$  and  $F_{n+1}$ , we identify the positions of squares of the form  $F_n^2$ ,  $n = 1, 2, \dots$  (Theorem 2.1);
- (ii) We show that in the first stage we have identified *all* the squares of the form  $F_n^2$  (Theorem 2.2);
- (iii) We quote a “folklore” result (a proof can be found in [16]) that for every square  $u^2$  in  $F$ ,  $u$  is necessarily a cyclic rotation of some  $F_n$  (Theorem 2.3);
- (iv) We exhibit all the squares in  $F$  as elements of “runs”, where the beginning

of each run is determined by one of the squares  $F_n^2$  (Theorem 2.4).

To begin the first stage, we generalize the definition of a Fibonacci string. Let  $F(x, y)$  denote the infinite Fibonacci string on the “alphabet”  $\{x, y\}$ , where now  $x$  and  $y$  denote arbitrary strings on any alphabet. (In fact, as we shall see below, we even allow  $x$  and  $y$  to be integers.) Then

$$F(x, y) = xyxyxyxyxyxyxyx \dots$$

and, in particular,  $F(a, b) = F$ , the “ordinary” infinite Fibonacci string defined above. The notation  $F_n(x, y)$  is similarly defined in the obvious way.

The following result is now easily proven by induction.

**Lemma 2.1** *For every integer  $n \geq 0$  and all strings  $x, y$ ,  $F_{n+1}(x, y) = F_n(xy, x)$ .  $\square$*

Repeated application of Lemma 2.1 leads to an important result (expressed in another form by Pansiot [15]):

**Lemma 2.2** *For all integers  $k, n$  satisfying  $0 \leq k \leq n - 1$ ,*

$$F_{n-k}(F_{k+1}, F_k) = F_n. \quad \square$$

Of course it is an immediate consequence of this result that, for every integer  $n \geq 0$ ,  
 $F = F(F_{n+1}, F_n) =$   
 $F_{n+1}F_nF_{n+1}F_{n+1}F_nF_{n+1}F_nF_{n+1}F_{n+1}F_n$   
 $\dots(2.1)$

Thus Lemma 2.2 enables us to express any Fibonacci string entirely in terms of selected Fibonacci substrings  $F_{n+1}$  and  $F_n$ ; we are able to focus only on the squares generated by those substrings. In order to prove our first main theorem, we need one more simple fact:

**Lemma 2.3** *For every integer  $n \geq 2$ ,  $F_n^2 = F_{n+1}F_{n-2}$ .*

**PROOF** Follows from rewriting  $F_{n+1} = F_nF_{n-1}$ .  $\square$

For all nonnegative integers  $i$  and  $n$ , let  $\Sigma_{i,n}$  denote the sum of the first  $i$  values in  $F(f_{n+1}, f_n)$ . Then  $\Sigma_{0,n} \equiv 0$  for all  $n$  and, for example, for  $n = 3$ ,

$$F(f_4, f_3) = 5355353553553553 \dots,$$

so that  $\Sigma_{1,3} = 5$ ,  $\Sigma_{2,3} = 8$ ,  $\Sigma_{5,3} = 21$ ,  $\Sigma_{9,3} = 46$ .

Based on this notation, it is now possible to specify all positions in  $F$  where  $F_{n+1}$  and  $F_n^2$  occur:

**Theorem 2.1** *For every integer  $i \geq 0$ ,*

- (a)  $F_{n+1} = F[\Sigma_{i,n} + 1, \Sigma_{i,n} + f_{n+1}]$ ,  $n \geq 2$ ;
- (b)  $F_n^2 = F[\Sigma_{i,n} + 1, \Sigma_{i,n} + 2f_n]$ ,  $n \geq 3$ .

**PROOF** Observe that since every occurrence of  $F_n$  in  $F$  is followed by an occurrence of  $F_{n+1} = F_{n-1}F_{n-2}F_{n-1}$ , therefore  $F_{n+1}$  occurs at every occurrence of  $F_n$  in  $F$ . Then (a) is an immediate consequence of Lemma 2.2: it merely says that  $F_{n+1}$  occurs at the beginning of  $F$  and then at displacements of  $f_{n+1}$  or  $f_n$ , depending on whether the current term in (2.1) is  $F_{n+1}$  or  $F_n$ , respectively. By Lemma 2.3 and the fact that, for  $n \geq 3$ ,  $F_{n-2}$  is a prefix of both  $F_{n+1}$  and  $F_n$ , we see that these positions also mark occurrences of  $F_n^2$ , and so (b) follows.  $\square$

The objection may be raised to Theorem 2.1(b) that it is restricted to the cases  $n \geq 3$  and so does not locate squares  $F_1^2 = a^2$  and  $F_2^2 = (ab)^2$ . However, as the next result shows, such squares are implicitly included:

**Lemma 2.4** *For  $n = 1, 2$ ,  $F_n^2$  occurs in  $F$  only as a substring of  $F_{n+2}^2$ .*

**PROOF** It is straightforward to verify that for  $n = 1, 2$ ,  $F_{n+2}^2$  contains  $F_n^2$  as a substring. Since  $a^3$  does not occur in  $F$ , each occurrence of  $a^2$  in  $F$  must always be surrounded by  $b$ 's, and so be a substring of  $baab$ . Also, since  $b^2$  does not occur in  $F$ , each occurrence of  $b$  in  $F$  must be surrounded by  $a$ 's, so that  $baab$  is a substring of  $abaaba = F_3^2$ . Similarly for  $n = 2$ , it follows from Lemma 2.2 that  $F = F(F_2, F_1) = F(ab, a)$ ; as above we have that  $(ab)^2$  occurs in  $F$  only as a substring of  $abaababaab = (F_3(ab, a))^2 = F_4^2$ .  $\square$

Thus Theorem 2.1 effectively allows us to locate occurrences of squares  $F_n^2$  for all  $n \geq 1$ ; we go on now to show that these occurrences are in fact the only ones in  $F$ . To do so, we introduce the idea of a *border* of a given string  $x$ ; that is, a substring of  $x$  which is both a prefix and a suffix of  $x$ . We introduce also an infinite *border array*  $B$  where, for every nonnegative integer  $n$ ,  $B[n]$  is the length of the longest border of  $F_n$ ; then  $B[n]$  is the “failure function” [12] of  $F_n$  and, for example,  $B[5] = 3$  corresponding to the longest border  $aba$  of  $F_5 = abaababa$ . The following result is proved also in [5].

**Lemma 2.5** *For every integer  $n \geq 2$ ,  $B[n] = f_{n-2}$ .*

**PROOF** By induction on  $n$ . Observe that the lemma holds for  $n = 2, 3$ , and suppose that it holds for every integer  $3 \leq n \leq N - 1$ , but not for  $n = N$ . Since  $F_N = F_{N-2}F_{N-3}F_{N-2}$ ,

it follows that  $B[N] > f_{N-2}$ . On the other hand, since by the inductive hypothesis  $B[N-1] = f_{N-3}$  and since for every integer  $k \geq 2$ ,  $B[k] \leq B[k-1] + f_{k-2}$ , we see that

$$B[N] \leq f_{N-2} + f_{N-3} = f_{N-1}.$$

Thus, setting  $B[N] = f_{N-2} + i$  for some integer  $i \in 1..f_{N-3}$ , we observe that, since  $F_{N-3}$  is a prefix of  $F_{N-2}$ ,

$$F[1..i] = F[f_{N-2} + 1..f_{N-2} + i] \quad (2.2a)$$

$$= F[f_{N-1} - i + 1..f_{N-1}] \quad (2.2b)$$

$$= F[f_{N-1} + 1..f_{N-1} + i] \quad (2.2c)$$

$$= F[f_N - i + 1..f_N]. \quad (2.2d)$$

From (2.2c) and (2.2d) we conclude that  $F[1..i]$  is a border of  $F_{N-2}$ . Hence, by the inductive hypothesis,  $i \leq N - 4$ . From (2.2a) and (2.2b), however, we conclude that  $F[1..i]$  is also a border of  $F_{N-3}$ . But this is impossible, since the inductive hypothesis implies that the borders of  $F_{N-2}$  and  $F_{N-3}$  are disjoint. Then there can exist no integer  $n = N \geq 2$  for which the lemma does not hold.  $\square$

The next lemma uses this result to establish an important property of Fibonacci strings. Suppose that an arbitrary string  $x$  is written in the form  $uv$ , where  $u$  (but not  $v$ ) is possibly empty; then  $x' = vu$  is said to be a *rotation* of  $x$  of *degree*  $|u|$ . For example, the rotations of  $x = F_4 = abaab$  are  $abaab$ ,  $baaba$ ,  $aabab$ ,  $ababa$  and  $babaa$ , of degrees 0 to 4 respectively. We show that a Fibonacci string cannot coincide with any rotation of itself, a result related to work of de Luca [7]:

**Lemma 2.6** *For every integer  $n \geq 2$ ,  $F_n$  is not equal to any rotation of  $F_n$  other than itself.*

**PROOF** The result holds for  $n = 2$ . Therefore, for some  $n \geq 3$  suppose the contrary, that there exists an integer  $k \in 1..f_n - 1$  such that

$$F_n = F[1..f_n] = F[k + 1..f_n]F[1..k].$$

But then  $F[1..k]$  and  $F[k + 1..f_n]$  are both borders of  $F_n$ , and one of these borders has length at least  $f_n/2 > f_{n-2}$ , in contradiction to Lemma 2.5.  $\square$

**Theorem 2.2** *For every integer  $n \geq 1$ , Theorem 2.1(b) specifies all the occurrences of  $F_n^2$  in  $F$ .*

**PROOF** By Lemma 2.4 the result holds for  $n = 1, 2$  if it holds for  $n \geq 3$ . But for  $n \geq 3$ , Theorem 2.1(b) makes clear that every position in  $F$  is included in some occurrence

of  $F_n^2$ . Hence any other occurrence of  $F_n^2$  not specified by Theorem 2.1(b) would require that  $F_n$  coincide with a rotation of itself, in contradiction to Lemma 2.6.  $\square$

As the third stage of our characterization of the squares in  $F$ , we quote a ‘‘folklore’’ result (Seebold provides a proof in [16]):

**Theorem 2.3** *For every nonempty substring  $u^2$  of  $F$ ,  $u$  is a rotation of  $F_n$  for some  $n \geq 1$ .*  $\square$

Finally, in order to complete the characterization of the squares of  $F$ , we need to define a new encoding. Suppose that  $F$  contains a set of  $h > 0$  consecutive squares

$$R(i, p, h) = \{(i, p, 2), \dots, (i + h - 1, p, 2)\},$$

all of length  $2p$ . We say that  $R(i, p, h)$  is a *run* of squares of *length*  $h$ . Then observe that for every integer  $j \in i + 1..i + h - 1$ , the square  $(j, p, 2)$  is a rotation of  $(j - 1, p, 2)$  of degree 1, by virtue of the fact that

$$F[j] = F[j + p] = F[j + 2p],$$

and so  $(j, p, 2)$  is a rotation of  $(i, p, 2)$  of degree  $(j - i) \bmod p$ . For example, the substring  $abaabaaba$  of  $F$  contains the run of squares

$$R(1, 3, 4) = \{(aba)^2, (baa)^2, (aab)^2, (aba)^2\},$$

where each square in the run is a rotation of degree 1 of the preceding one. More generally, observe that for  $n \geq 3$  every occurrence of

$$F_n^3 = (\Sigma_{i,n} + 1, f_n, 3)$$

in  $F$  gives rise to a run of squares  $R(\Sigma_{i,n} + 1, f_n, f_n)$  corresponding to the  $f_n$  rotations of  $F_n$ . Indeed, we have

**Lemma 2.7** *For all integers  $n \geq 3$  and  $i \geq 0$ ,  $R(\Sigma_{i,n} + 1, f_n, f_n)$  is a run of squares of  $F$  if and only if  $F[i] = b$ .*

**PROOF** Observe that  $F[i] = b$  if and only if  $F(F_{n+1}, F_n)[i] = F_n$  in the expansion (2.1). Then

$$F(F_{n+1}, F_n)[i..i + 2] = F_n F_{n+1} F_{n+1}$$

OR

$$F(F_{n+1}, F_n)[i..i + 2] = F_n F_{n+1} F_n.$$

In either of these cases,  $F_n F_{n+1} F_n$  is a prefix of  $F(F_{n+1}, F_n)[i..i + 2]$ , and it is straightforward to verify that  $F_n F_{n+1} F_n = F_n^3 F_{n-3} F_{n-2}$ . This proves sufficiency.

To prove necessity, observe that if  $i = 0$  or  $F[i] = a$ ,  $F(F_{n+1}, F_n)[i..i + 1]$  has prefix  $F_{n+1}F_n$ , which as we have just seen equals  $F_n^2F_{n-3}F_{n-2}$ . Since  $F_{n-3}F_{n-2}$  is a rotation of  $F_{n-1}$ , it follows from Lemma 2.6 that  $F_{n-3}F_{n-2} \neq F_{n-1}$ , hence that  $F_{n+1}F_n$  is not a prefix of  $F_n^3 = F_n^2F_{n-1}(F_{n-2})$ .  $\square$

More precise information about the runs of squares that occur when  $F[i] \neq b$  is provided by the following lemma:

**Lemma 2.8** *For every integer  $n \geq 2$ ,  $F_n = F_{n-2}F_{n-3} \cdots F_1u$ , where  $u = ab$  if  $n$  is even, and  $u = ba$  otherwise.*

**PROOF** By induction on  $n$ . Recalling that  $f_0 = 0$ , observe that the result holds when  $n = 2, 3$ , and suppose that it holds for some  $n = N - 2$ . But then, writing  $F_N = F_{N-2}F_{N-3}F_{N-2}$ , we see, using the inductive hypothesis, that it must hold also for  $n = N$ .  $\square$

From Lemma 2.8 it follows immediately that  $F_n$  and its rotation  $F_{n-2}F_{n-1}$  of degree  $f_{n-2}$  differ in precisely the last two positions. Thus, as we have seen in the proof of Lemma 2.7,

$$F(F_{n+1}, F_n)[i..i + 1] = F_n^2F_{n-3}F_{n-2}$$

differs from  $F_n^2F_{n-1}$  only in the last two positions whenever  $i = 0$  or  $F[i] = a$ . It follows that in this case  $R(\Sigma_{i,n} + 1, f_n, f_{n-1} - 1)$  is the maximal run of squares of  $F = F(F_{n+1}, F_n)$ . Furthermore, it follows also that none of the positions

$$\Sigma_{i,n} + f_{n-1} \cdots \Sigma_{i,n} + f_{n+1}$$

of  $F$  can mark the beginning of squares of rotations of  $F_n$ . Thus all the possible squares of length  $2f_n$  have been accounted for, and since by Theorem 2.3 there are no other possible squares, we have established the main result of this section:

**Theorem 2.4** *For every integer  $n \geq 3$ , the squares of  $F$  of length  $2f_n$  are*

(a)  $R(\Sigma_{i,n} + 1, f_n, f_n)$  if and only if  $F[i] = b$ ,

(b)  $R(\Sigma_{i,n} + 1, f_n, f_{n-1} - 1)$  if and only if  $i = 0$  or  $F[i] = a$ ;

where  $i$  assumes all nonnegative integral values.

For  $n = 1, 2$ , the squares of length  $2f_n$  occur as substrings of every occurrence of  $F_{n+2}^2$ , as specified by Lemma 2.4.  $\square$

As an example of this theorem, consider the case  $n = 4$ :

$$\begin{aligned} F(F_5, F_4) &= (abaababa)(abaab)(abaababa) \\ &\quad (abaababa)(abaab)(abaababa) \\ &\quad (abaab) \cdots \end{aligned}$$

For  $i = 0..5$ ,  $\Sigma_{i,4}$  takes the values 0, 8, 13, 21, 29, 34, respectively, and so the first six runs of squares of length  $2f_4 = 10$  are

$$R(1, 5, 2), R(9, 5, 5), R(14, 5, 2),$$

$$R(22, 5, 2), R(30, 5, 5), R(35, 5, 2).$$

The first square in each of these runs contains one occurrence of  $(ab)^2$  and one occurrence of  $(ba)^2$ .

Theorem 2.4 tells us how to locate a run, after we determine whether  $F[i] = a$  or not. The next lemma provides the basis of a constant time and space routine, for checking whether  $F[i] = a$ ; a proof can be found in [11] and an alternate approach in [3].

**Lemma 2.9** *If  $\phi = \frac{1}{2}(1 + \sqrt{5})$  (the golden mean), then*

$$F[i] = \begin{cases} a & \text{if } [(i + 1)/\phi] - [(i/\phi)] = 1; \\ b & \text{otherwise.} \quad \square \end{cases}$$

### 3 Computing the squares

Theorem 2.4 tells us how to locate all the squares in the infinite Fibonacci string  $F$ ; in practice, however, we will be asked to compute the squares in a given finite Fibonacci string  $F_n$ . The algorithm SQUARES, given in Pascal-like pseudocode in Figure 2, performs this computation for  $n \geq 6$  — it is convenient to treat the cases  $n \leq 5$  separately. The algorithm is based primarily on Theorem 2.4, with some minor complications related to the special conditions that arise when  $F_n$  terminates. We make the simplifying assumption that, for  $k = 3$  or 4, outputs of squares of length  $2f_k$  implicitly specify those of length  $2f_{k-2}$ , in accordance with Lemma 2.4.

Since each required value of a Fibonacci number  $f_k$ ,  $k = 1, 2, \dots, n$ , can be computed in constant time, it follows that the compound **if** statement inside the inner **for** loop in this algorithm also executes in constant time. This **if** statement will be executed exactly

$$\begin{aligned} Q_n &= \sum_{k=3}^{n-2} (f_{n-k} + 1 - 2) \\ &= f_n - f_{n-2} - 3 - (n - 4) \end{aligned}$$

times, by Lemma 2.8, a quantity which reduces to  $f_{n-1} - (n - 1)$ . Hence, excluding the squares  $a^2$ ,  $(ab)^2$ , and  $(ba)^2$ ,  $Q_n$

```

algorithm SQUARES ( $F_n$ );
{ This loop computes all the squares of length  $2f_k$ ,  $k = 3..n - 2$ .}
for  $k = 3..n - 2$  do
  { This loop considers all but the last term of  $F_{n-k}(F_{k+1}, F_k) = F_n$ 
  (Lemma 2.2).}
  for  $i = 0..f_{n-k} - 2$  do
    if  $i = 0$  then
       $\Sigma \leftarrow 1$ ;      { Maintain the invariant  $\Sigma = \Sigma_{i,n} + 1$ .}
      output ( $\Sigma, f_k, f_{k-1} - 1$ )  { Theorem 2.4(b).}
    else
       $\Sigma \leftarrow \Sigma + f_k$ ;
      if  $F_n[i] = a$  then  { Lemma 2.9 }
         $\Sigma \leftarrow \Sigma + f_{k-1}$ ;
        output ( $\Sigma, f_k, f_{k-1} - 1$ )  { Theorem 2.4(b).}
      else
        if  $i + 1 = f_{n-k} - 2$  then
          { A special case arises when  $F_n$  ends in  $F_k F_{k+1}$ .}
          output ( $\Sigma, f_k, f_{k-1} + 1$ )
        else
          output ( $\Sigma, f_k, f_k$ );      { Theorem 2.4(a).}
    if  $k = 3$  or  $4$  then
      find squares of length  $2f_{k-2}$  in the last term of  $F_{n-k}(F_{k+1}, F_k)$ .
      { The details are straightforward, and are omitted.}

```

Figure 2: The algorithm.

gives exactly the number of triples output by SQUARES( $F_n$ ); that is, exactly the number of runs of squares in  $F_n$ . If in addition the excluded squares are identified and printed separately by the routine **output**, we will have

$$\begin{aligned}
 Q_n &= f_{n-1} - (n-1) + f_{n-3} + f_{n-4} - 2 \\
 &= f_n - (n+1).
 \end{aligned}$$

In either case the time requirement of the algorithm is  $\Theta(f_n)$ . As for the space requirement, observe that there is only a single reference to  $F_n$  in the algorithm, when it becomes necessary to determine the value of  $F_n[i]$ , where  $i \leq f_{n-3} - 2$ .

Lemma 2.9 allows us to determine this value in constant time and space. We thus state formally our result:

**Theorem 3.1** *For every integer  $n \geq 6$ , the algorithm SQUARES computes all the runs of squares in  $F_n$  using  $\Theta(f_n)$  time and  $O(1)$  space.  $\square$*

We remark again that this algorithm can only be said to compute all the squares of  $F_n$  if the user is willing to accept the encoding of squares into runs  $R$  as an appropriate response to a query. Indeed, a slight modification to SQUARES could reduce the amount

of output still further, by reporting adjacent runs of squares (which occur whenever  $F_k$  occurs in  $F_n(F_{k+1}, F_k)$ ) as a single run. Such a modification would reduce the number of runs reported by a constant factor of  $\phi$ , but the total number of outputs would still be  $\Theta(f_n)$ . It appears that an asymptotic reduction (say to  $\Theta(\log f_n)$  outputs) could not be achieved in a manner consistent with an appropriate response to the user's request.

## Acknowledgements

The work of the first author was supported in part by SERC grants GR/F 00898 and GR/J 17844, NATO Grant No. CRG 900293, ESPRIT BRA Grant No. 7131 for ALCOM II, and MRC Grant No. G 9115730. The work of the third author was supported in part by Grant No. A8180 of the Natural Sciences & Engineering Research Council of Canada and by Grant No. GO-12778 of the Medical Research Council of Canada.

## References

- [1] A. V. Aho. Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science*, Elsevier, (1988).

- [2] A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, Volume 22, pages 297-315, (1983).
- [3] J. Berstel. Fibonacci words. *Book of L*, Springer-Verlag, pages 13-27, (1986).
- [4] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, Volume 12-5, pages 244-250, (1981).
- [5] L. J. Cummings, D. Moore and J. Karhumäki. Borders of Fibonacci strings. *Journal of Combinatorial Mathematics & Combinatorial Computing*, to appear.
- [6] L. J. Cummings & W. F. Smyth. Weak repetitions in strings. *Journal of Combinatorial Mathematics & Combinatorial Computing*, to appear.
- [7] A. de Luca. A combinatorial property of the Fibonacci words. *Information Processing Letters*, Volume 12-4, pages 193-195 (1981).
- [8] L. E. Dickson. *History of the Theory of Numbers*, Chelsea, (1971).
- [9] C. S. Iliopoulos, D. Moore and K. Park. Covering a string. In *Fourth Annual Symposium on Combinatorial Pattern Matching*, pages 54-62, (1993).
- [10] C. S. Iliopoulos and W. F. Smyth. *The covers of a circular Fibonacci String* Submitted.
- [11] D. E. Knuth. *The Art of Computer Programming*, Addison Wesley, (1971).
- [12] D. E. Knuth, J. H. Morris and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, Volume 6, pages 322-350, (1977).
- [13] M. G. Main and R. J. Lorentz. An  $O(n \log n)$  algorithm for finding all repetitions in a string. *Journal of Algorithms* Volume 5, pages 422-432, (1984).
- [14] D. Moore and W. F. Smyth. An optimal algorithm to compute all the covers of a string. *Information Processing Letters*, Volume 50-5, pages 239-246, (1994).
- [15] J. J. Pansiot. Mots infinis de Fibonacci et morphismes itérés. *RAIRO Informatique Théorique*, Volume 17-2, pages 131-135, (1983).
- [16] P. Seebold. Propriétés Combinatoires des Mots Infinites Engendrés par Certains Morphismes. *Report No. 85-16*, LITP, Paris, (1985).
- [17] T. F. Smith and M. S. Waterman. Identification of common molecular sequences. *Journal of Molecular Biology*, Volume 147, pages 195-197, (1981).