Euclidean Strings *

John Ellis[†], Frank Ruskey[†], Joe Sawada[⋆] and Jamie Simpson[‡]

[†]*Department of Computer Science, University of Victoria*
P.O. Box 3055, Victoria, British Columbia, V8W 3P6, Canada

[‡]*Department of Mathematics and Statistics, Curtin University,*
Perth, Western Australia, Australia

[⋆]*DIMATIA, Charles University, Prague, Czech Republic*

December 2, 2000

Abstract

A string $\mathbf{p} = p_0p_1 \cdots p_{n-1}$ of non-negative integers is a *Euclidean string* if the string $(p_0+1)p_1 \cdots (p_{n-1}-1)$ is rotationally equivalent to $\mathbf{p}$. We show that Euclidean strings exist if and only if n and $p_0 + p_1 + \cdots + p_{n-1}$ are relatively prime and that, if they exist, they are unique. We show how to construct them using an algorithm with the same structure as the Euclidean algorithm, hence the name. We show that Euclidean strings are Lyndon words and we describe relationships between Euclidean strings and the Stern-Brocot tree, Fibonacci strings and Beatty sequences. We also describe an application to a graph embedding problem.

Keywords: Euclidean, string, Lyndon, Stern-Brocot, Fibonacci, Beatty

*This work was supported by the Natural Sciences and Engineering Research Council of Canada and by Czech grant GACR 201/99/0242

1 Introduction

The string $aba\underline{ab}aba$ has a curious property. The string is not equal to any of its non-trivial rotations, but if we reverse the marked ab pair, we get the string $aba\underline{ba}aba$, which is rotationally equivalent to the original string. In this paper we will investigate this phenomenon in a slightly more general setting.

Let $\mathbf{p} = p_0 p_1 \cdots p_{n-1}$ denote a string of n non-negative integers. When the length of $\mathbf{p}$ is at least 2, we let $\rho(\mathbf{p})$ denote a right rotation of $\mathbf{p}$ by one position, i.e., $\rho(\mathbf{p}) = p_{n-1}p_0p_1 \cdots p_{n-2}$. Then let $\rho^d(\mathbf{p})$ denote a right rotation through d positions and $\rho^{-d}(\mathbf{p})$ denote a left rotation through d positions. Let $\tau(\mathbf{p})$ be the string obtained from $\mathbf{p}$ by replacing p_0 by $p_0 + 1$ and p_{n-1} by $p_{n-1} - 1$.

Definition 1
The string $\mathbf{p}$ *is a* Euclidean *string if it is of unit length or if there exists an integer* d *such that* $\tau(\mathbf{p}) = \rho^d(\mathbf{p})$.

We will refer to the parameter d in the definition as a *displacement* of the Euclidean string. For consistency we define the displacement of a unit length Euclidean string to be one.

Definition 2
The weight *of a Euclidean string is* $\sum_{i=0}^{n-1} p_i$, *i.e., the sum of all its elements.*

Definition 3
The cost *of a unit length Euclidean string is* $p_0 - 1$. *Otherwise, the* cost, *relative to a displacement* d, *is* $\sum_{i=0}^{d-1} p_i$, *i.e., the sum of the* d *elements to the right of and including* p_0.

Throughout the paper we use k to denote the weight and c to denote the cost of a Euclidean string. We denote a Euclidean string of length n and weight k by $E_{n,k}$. In arithmetic expressions, mod will denote the remainder after integer division. We adopt the convention that all arithmetic on the string indices is done modulo n, the length of the string, e.g., we write p_{i+j} for $p_{(i+j) \bmod n}$.

We show that a Euclidean string exists if and only if its length and weight are relatively prime and that, if it exists, it is unique, as are the cost and displacement. We show how to compute Euclidean strings together with their displacements and costs, we show that Euclidean strings are Lyndon words and we show their relationships to Stern-Brocot trees, Fibonacci strings and to Beatty sequences. We conclude with an application to a graph embedding problem. Some of these results were first presented in [6].

2 Characterization and generation

Lemma 1
Any Euclidean string exhibits the following properties:

a) *the length and the weight are relatively prime and*

b) *it is unique, for given length and weight, as are its displacement and cost and*

c) *for all* i *in* $\{jd \bmod n : 0 \leq j \leq n - (k \bmod n) - 1\}$, $p_i = \lfloor k/n \rfloor$ *and* *for all* i *in* $\{(n-1+jd) \bmod n : 0 \leq j \leq (k \bmod n) - 1\}$, $p_i = \lceil k/n \rceil$ *or, alternatively,* $p_i = |\{j : n-1+jd \equiv i \bmod n,\ 0 \leq j < k\}|$ *and*

d) $dk \equiv 1 \bmod n$.

Proof
Let $\mathbf{p}$ be a Euclidean string of length n and weight k and with a displacement d so that $\rho^d(\mathbf{p}) = \tau(\mathbf{p})$. We can deduce from this property that

$$\text{if } (0 \leq j \leq n-d-2) \text{ or } (n-d+1 \leq j \leq n-1) \text{ then } p_j = p_{j+d} \tag{1}$$

$$p_{n-d-1} = p_{n-1} - 1 \text{ and } p_{n-d} = p_0 + 1 \tag{2}$$

Consider the sequence of string elements

$$S = p_0, p_d, p_{2d}, \cdots, p_{jd}$$

where all the indices are taken modulo n and where j is the smallest positive integer such that $jd \equiv 0 \bmod n$. Such a j must exist. It must be that p_{n-d} is the penultimate item in S since it is the predecessor of p_0. By (2) $p_{n-d} = p_0 + 1$. But, by equations (1) and (2), every element in the sequence is equal to its predecessor in the sequence, unless the element is p_0 or p_{n-1}. Since the sequence starts and ends with p_0, it must then also contain p_{n-1}, because p_{n-d-1} is the only remaining element which can break the sequence of identical elements. So S is a sequence of string elements,

$$S = p_0, p_d, p_{2d}, \cdots, p_{n-d-1}, p_{n-1}, \cdots, p_{n-d}, p_0.$$

where every element is identical to its successor except for p_{n-d-1} and p_{n-d}.

It follows that there are only two values in a Euclidean string, differing by one. Let them be $x = p_0 = \lfloor k/n \rfloor$ and $x+1 = p_{n-1} = \lceil k/n \rceil$. Then S consists of a sequence of x's followed by a sequence of $(x+1)$'s. Let there be r of the x's. Then $rx + (n-r)(x+1) = k$ and hence $k + r \equiv 0 \bmod n$. Further, is must be that the first $(x+1)$ is p_{n-1} and that $n-1 \equiv rd \bmod n$, i.e., $rd \equiv -1 \bmod n$, which implies that $\gcd(r, n) = 1$. We can then derive the items in the lemma as follows.

a) From $\gcd(r, n) = 1$ and $k + r \equiv 0 \bmod n$ we derive $\gcd(n, k) = 1$.

b) We have that $rd \equiv -1 \bmod n$, where $r = n - (k \bmod n)$. This implies that d has a unique value in the reduced residue set modulo n. Hence the sequence S is unique and the corresponding Euclidean string is unique.

c) Since r, the number of smaller elements, is necessarily $n - (k \bmod n)$, the result follows.

d) From $k + r \equiv 0 \bmod n$ and $rd \equiv -1 \bmod n$ we derive $dk \equiv 1 \bmod n$.

$\square$

Lemma 2
If n and k are relatively prime, positive integers, then there exists a Euclidean string of length n and weight k.

Proof
The proof is constructive. The function E-string constructs a Euclidean string, of length n and weight k, when n and k are relatively prime. The procedure uses two string operations, increment and expand. Increment, denoted *inc*, adds one to every integer in the string. Expand, denoted *exp*, replaces every integer i in the string by 01^i, where 1^i denotes a string of i ones.

function E-string $(n,\ k : \mathbb{Z}) : \mathbb{Z}$;
  if $(n = k = 1)$ **then** return('1');
  if $k > n$ **then** return(inc(E-string $(n, k-n)$))
    else return(exp(E-string $(n-k, k)$));

We demonstrate the correctness of the procedure by induction on the number of invocations to the increment and expand procedures. For the base case, where $n = k = 1$, the procedure returns the string '1', which conforms to Definition 1.

It can not be the case that $n = k$ and $n > 1$ because n and k are relatively prime. Suppose $k > n$, in which case the increment operation is applied to the result of invoking the procedure with parameters $(n, k-n)$. Since n and k are relatively prime, so are n and $k - n$. Hence we may assume that a Euclidean string of length n and weight $k - n$, say $\mathbf{p}$, is returned. It is clear that since $\tau(\mathbf{p}) = \rho^d(\mathbf{p})$, where d is the displacement of $\mathbf{p}$, then $\tau(inc(\mathbf{p})) = \rho^d(inc(\mathbf{p}))$. Further, the incrementation increases the sum of the elements from $k - n$ to k. Hence $inc(\mathbf{p})$ is a Euclidean string of length n and weight k.

Now suppose $n > k$, in which case the expand operation is applied to the result of invoking the procedure with parameters $(n-k, k)$. Since n and k are relatively prime, so are $n - k$ and k. Hence we may assume that a Euclidean string of length $n - k$ and weight k, say $\mathbf{p}$, is returned and that $\tau(\mathbf{p}) = \rho^d(\mathbf{p})$ where d is the displacement of $\mathbf{p}$. Now:

$$exp(\mathbf{p}) = 01^{p_0}01^{p_1} \cdots 01^{p_{n-k-1}}$$

It follows that $\tau(exp(\mathbf{p})) = \rho^{d+c}(exp(\mathbf{p}))$, where c is the cost of $\mathbf{p}$, Definition 3. Further, the length of the expanded string is the number of 0's plus the number of 1's which is

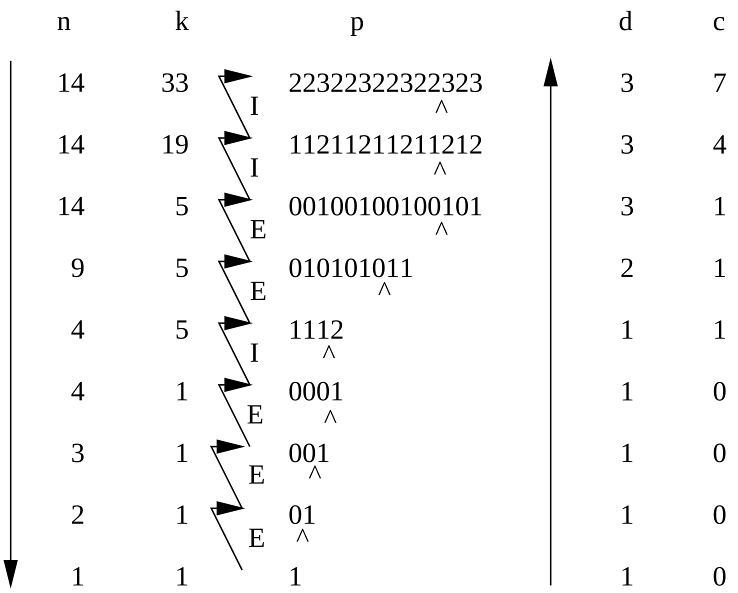

Figure 1: An Example of Computing a Euclidean String. The symbol I denotes an increment, E denotes an expansion and the caret denotes the exchange point (Lemma 5).

$(n-k)+k=n$, but the sum of the integers in the expanded string remains k. Hence $exp(\mathbf{p})$ is a Euclidean string of length n and weight k. □

An example of the computation of a string is given in Figure 1. In that figure the cost and dispacement are also shown. The computation of these parameters is discussed in the next section.

The unwinding of the recursion can be viewed as a walk from the root in an infinite binary tree $\mathcal{T}$. In $\mathcal{T}$ each node is labelled by an ordered pair (x, y). The left child of (x, y) is $(x, x+y)$ and the right child is $(x+y, y)$. The root is $(1, 1)$. See Figure 2. Every reduced fraction x/y occurs once in this tree and a breadth first traversal of this tree provides a proof that the rational numbers are countable [2]. Figure 3 shows the corresponding tree of strings.

From Lemma 2 and Lemma 1(a), we obtain:

Theorem 1
A string is Euclidean if and only if its length and weight are relatively prime.

If d is known, then we can use Lemma 1 (c), to generate the Euclidean string in time linear in the length of the string. Below is a procedure which computes the displacement and cost, d and c, assuming that all the variables are global. The correctness of the procedure follows immediately from the definition of each parameter, i.e., an incrementation requires

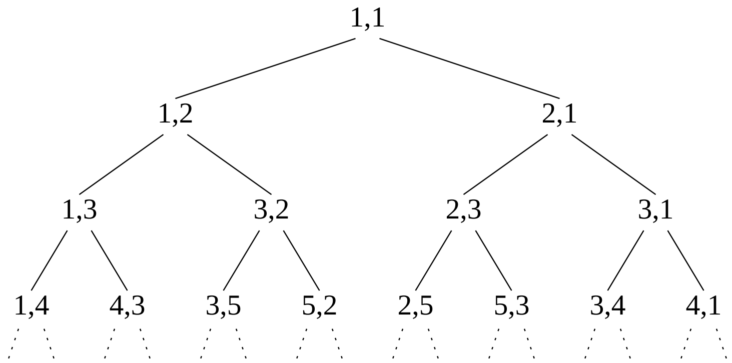

Figure 2: The tree of reduced fractions.

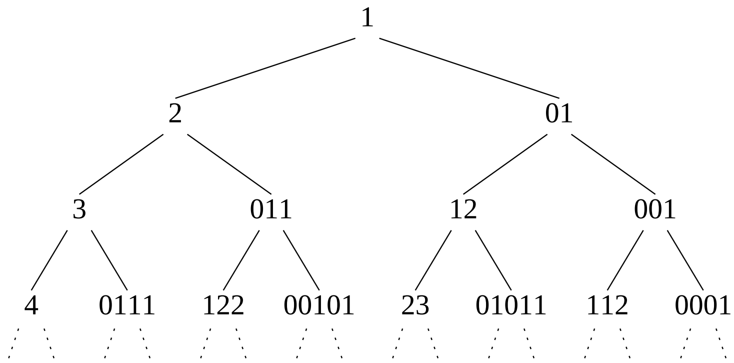

Figure 3: The corresponding tree of strings.

that the cost be increased by d, whereas an expansion requires that the displacement be increased by c. Note that for consistency we have defined the displacement and cost of a unit length Euclidean string to be 1 and $k-1$ respectively. Also note that one of the terminating conditions must be reached.

procedure disp-cost$(n, k : \mathbb{Z})$;
  if $n = 1$ and $k = 1$ **then** $c := 0$; $d := 1$
  else if $n < k$ **then** disp-cost$(n, k-n)$; $c := c + d$
    else disp-cost$(n-k, k)$; $d := c + d$;

The following lemma describes a relationship between cost and displacement.

Lemma 3
If $n + k \geq 2$ *then* $dk = cn + 1$

Proof
Let $d_{n,k}$ and $c_{n,k}$ denote the dispacement and cost respectively of a Euclidean string of length n and weight k. We argue by induction on the number of recursive invocations of the procedure. For the basis we note that if $n = 1$ and $k = 1$ the initialisation defined by the algorithm satisfies the lemma.

For the induction suppose first that $n < k$. By the inductive hypothesis: $d_{n,k-n}(k-n) = c_{n,k-n}n+1$. Hence $d_{n,k-n}(k-n)+d_{n,k-n}n = (c_{n,k-n}+d_{n,k-n})n+1$. Hence $d_{n,k-n}k = (c_{n,k-n}+d_{n,k-n})n+1$. But, when $n < k$, the algorithm sets $d_{n,k} = d_{n,k-n}$ and $c_{n,k} = c_{n,k-n} + d_{n,k-n}$. Hence $d_{n,k}k = c_{n,k}n + 1$.

Suppose $k < n$. By the inductive hypothesis: $d_{n-k,k}k = c_{n-k,k}(n-k) + 1$. Hence $(d_{n-k,k} + c_{n-k,k})k = c_{n-k,k}n + 1$. But, when $n < k$, the algorithm sets $d_{n,k} = d_{n-k,k} + c_{n-k,k}$ and $c_{n,k} = c_{n-k,k}$. Hence $d_{n,k}k = c_{n,k}n + 1$. □

So d and $-c$ are in fact the constants computed by the standard extended Euclidean algorithm, see for example [3, page 811]. We have already shown, in Lemma 1 (b), that $dk \equiv 1 \bmod n$, i.e., d is the multiplicative inverse of k modulo n. Lemma 3 immediately yields an analogous corollary, which is crucial to the application described is the final section of this paper and taken from [5].

Corollary 1
The number c is the multiplicative inverse of $(k-n)$ modulo k, i.e., $c(k-n) \equiv 1 \bmod k$.

3 Some properties of Euclidean strings

Let $R(\mathbf{p})$ denote the reversal of $\mathbf{p}$.

Lemma 4
If $\mathbf{p}$ *is a Euclidean string of length at least 2 and with displacement* d *then* $\mathbf{p} = R(\tau(\mathbf{p})) = R(\rho^d(\mathbf{p}))$.

Proof
We proceed by induction on the number of applications of the expand and increment operations in the construction of $\mathbf{p}$. The statement is true for strings of length 2 and for strings of the form 01^k. All strings are derived from some string in one of these forms, see Figure 3.

For the induction, suppose $\mathbf{p} = R(\tau(\mathbf{p}))$. The application of the increment operation obviously preserves the truth of the statement. Let $exp(\mathbf{p})$ denote the expansion of $\mathbf{p}$. We have asumed that $p_0p_1 \cdots p_{n-2}p_{n-1} = (p_{n-1} - 1)p_{n-2} \cdots p_1(p_0 + 1)$.

$$\begin{aligned}
\text{But } exp(\mathbf{p}) &= 01^{p_0}01^{p_1} \cdots 01^{p_{n-2}}01^{p_{n-1}}. \\
\text{Hence } \tau(exp(\mathbf{p})) &= 11^{p_0}01^{p_1} \cdots 01^{p_{n-2}}01^{p_{n-1}-1}0. \\
\text{Hence } R(\tau(exp(\mathbf{p}))) &= 01^{p_{n-1}-1}01^{p_{n-2}} \cdots 01^{p_1}01^{p_0+1} \\
&= 01^{p_0}01^{p_1} \cdots 01^{p_{n-2}}01^{p_{n-1}}
\end{aligned}$$

the last equation being obtained from the first. □

Lemma 4 implies that in any Euclidean string of length at least 3 there is another adjacent pair of elements, besides those on the ends, whose exchange results in a string rotationally equivalent to the original. We make this explicit in the following lemma. Let $swap(\mathbf{p})$ be the string obtained from $\mathbf{p}$ by exchanging elements p_{n-d-1} and p_{n-d}.

Lemma 5
If $\mathbf{p}$ *is a Euclidean string with displacement* d *then* $swap(\mathbf{p}) = \rho^{-d}(\mathbf{p})$

Proof
From the proof of Lemma 1 we know that $p_{n-d-1} + 1 = p_{n-d}$. Hence the exchange of those two elements is equivalent to a sequence of rotations, reversals and a τ operation as expressed in the following equation.

$$swap(\mathbf{p}) = \rho^{-d}(R(\tau(R(\rho^d(\mathbf{p})))))$$

Thus, by Lemma 4,

$$swap(\mathbf{p}) = \rho^{-d}(R(\tau(\mathbf{p}))).$$

By Lemma 4, $\tau(\mathbf{p}) = R(\mathbf{p})$, and thus

$$swap(\mathbf{p}) = \rho^{-d}(R(R(\mathbf{p}))) = \rho^{-d}(\mathbf{p})$$

□

In the example illustrated in Figure 1, the second exchangable pair is indicated by the caret symbol.

If $\mathbf{p}$ is a binary string then the complement of $\mathbf{p}$, denoted $C(\mathbf{p})$, is the string obtained by replacing every zero by a one and *vice versa*.

Lemma 6
If $n > k$ then $C(R(E_{n,k})) = E_{n,n-k}$.

Proof
Let $\mathbf{p}$ be $E_{n,k}$ where $n > k$. By Lemma 4 $R(\mathbf{p}) = \tau(\mathbf{p}) = \rho^d(\mathbf{p})$. Then the weight of $C(\mathbf{p})$ is $n-k$. We note that

$$\tau(C(R(\mathbf{p}))) = \tau(C(\tau(\mathbf{p}))) = C(\mathbf{p})$$

and that

$$\rho^{-d}(C(R(\mathbf{p}))) = \rho^{-d}(C(\rho^d(\mathbf{p}))) = C(\mathbf{p}).$$

Hence $\tau(C(R(\mathbf{p}))) = \rho^{-d}(C(R(\mathbf{p})))$, i.e., $C(R(\mathbf{p})) = E_{n,n-k}$. □

Definition 4
The function $\delta(\mathbf{p})$ is defined for all strings of positive integers $\mathbf{p}$. It is obtained by replacing every integer i in the string by the string $0^{i-1}1$.

The following theorem shows a relationship between $E_{n,k}$ and its "dual" $E_{k,n}$.

Theorem 2
If n and k are relatively prime and $n < k$, then $\delta(R(E_{n,k})) = E_{k,n}$.

Proof
We observe that for any string $\mathbf{p}$ of positive integers

$$\begin{aligned}
\delta(\mathbf{p}) &= \rho^{-1}(C(exp(inc^{-1}(\mathbf{p})))) \\
\text{Hence, } \delta(R(\mathbf{p})) &= \rho^{-1}(C(exp(inc^{-1}(R(\mathbf{p}))))) \\
&= \rho^{-1}(C(exp(R(inc^{-1}(\mathbf{p}))))) \\
&= \rho^{-1}(C(\rho(R(exp(R(inc^{-1}(\mathbf{p})))))) \text{ since } exp(R(\mathbf{p})) = \rho(R(exp(\mathbf{p}))) \\
&= C(R(exp(inc^{-1}(\mathbf{p}))))
\end{aligned}$$

Now, suppose $\mathbf{p} = E_{n,k}$ where $n < k$ implying that $E_{n.k}$ is a string of positive integers. From the discussion in Section 2, Figure 2, $exp(inc^{-1}(E_{n,k})) = E_{k,k-n}$. From Lemma 6 $C(R(E_{k,k-n})) = E_{k,n}$. Hence $\delta(R(E_{n,k})) = E_{k,n}$. □

A *Lyndon* word is one which is lexicographically less than all of its non-trivial rotations. We use the symbols $\prec$ and $\succ$ to denote "is lexicographically less than" and "greater than" respectively.

Lemma 7
Every Euclidean string is a Lyndon word.

Proof
We note that $\rho^i(\mathbf{p}) = p_{n-i}\cdots p_{n-1}p_0\cdots p_{n-i-1}$.

$$\begin{aligned} \text{Hence } \rho^{i+d}(\mathbf{p}) &= \rho^i(\rho^d(\mathbf{p})) \\ &= \rho^i(\tau(\mathbf{p})) \\ &= \rho^i(p_{n-1}p_1\cdots p_{n-2}p_0) \\ &= p_{n-i}\cdots p_0p_{n-1}\cdots p_{n-i-1}. \end{aligned}$$

Since $p_0 \prec p_{n-1}$ we have $\rho^{i+d}(\mathbf{p}) \prec \rho^i(\mathbf{p})$ for any i except $i = 0$. In that case $\rho^d(\mathbf{p}) \succ \mathbf{p}$. $\square$

Definition 5
The content *of a string is the multiset of characters that occur in the string. In other words, for the string* $\mathbf{p} = p_1p_2\cdots p_n$*, the content of* $\mathbf{p}$*, denoted content*$(\mathbf{p})$*, is the multiset* $\{p_1, \cdots, p_n\}$.

The following lemma is an immediate consequences of the definitions of the increment and expand functions, denoted *inc* and *exp*, respectively.

Lemma 8
If $\mathbf{p}$ *and* $\mathbf{q}$ *are strings, where* $\mathbf{p} \prec \mathbf{q}$*, then* $exp(\mathbf{p}) \prec exp(\mathbf{q})$ *and* $inc(\mathbf{p}) \prec inc(\mathbf{q})$.

Lemma 9
Among all numeric Lyndon words with the same length n *and weight* k*, where* $n > 1$*, the lexicographically largest has exactly two symbol types,* $\lfloor k/n \rfloor$ *and* $\lceil k/n \rceil$.

Proof
Clearly a Lyndon word starts with the smallest number in its content. Thus we wish to maximize the smallest numeric value. This maximum occurs at $\lfloor k/n \rfloor$. Since every Lyndon word with $n > 1$ has at least two symbols, there is at least one that is $\lceil k/n \rceil$. $\square$

Lemma 10
If $\mathbf{q}$ *is a binary Lyndon word different from 0, then* exp^{-1} *is well-defined and* $exp^{-1}(\mathbf{q})$ *is a Lyndon word.*

Proof
It is well-defined since a binary Lyndon word starts with a 0 and ends with a 1 (except for 1 itself, which is a fixed-point of exp^{-1}).

Now suppose that $exp^{-1}(\mathbf{q}) = \mathbf{uv}$, where $\mathbf{vu} \prec \mathbf{uv}$. By Lemma 8, we have $exp(\mathbf{vu}) \prec \mathbf{q}$. But this is a contradiction since $exp(\mathbf{vu}) = exp(\mathbf{v})exp(\mathbf{u})$ is a rotation of $\mathbf{q}$. $\square$

Theorem 3
If $\mathbf{p}$ *is a Euclidean string and* $\mathbf{q}$ *is a different Lyndon word with the same length and weight, then* $\mathbf{q} \prec \mathbf{p}$.

Proof
We argue by induction on the sum of the length and weight of the string. The theorem is obviously true for strings of length one or two. By the Lemma 9 we may assume that $content(\mathbf{p}) = content(\mathbf{q})$.

If $\mathbf{p}$ and $\mathbf{q}$ are not binary, then inductively $inc^{-1}(\mathbf{q}) \prec inc^{-1}(\mathbf{p})$, from which Lemma 8 gives $\mathbf{q} \prec \mathbf{p}$.

If $\mathbf{p}$ and $\mathbf{q}$ are both binary strings, then by Lemma 10 $exp^{-1}(\mathbf{q})$ exists and is a Lyndon word. If $content(exp^{-1}(\mathbf{q})) \neq content(exp^{-1}(\mathbf{p}))$, then by Lemma 9 $exp^{-1}(\mathbf{q}) \prec exp^{-1}(\mathbf{p})$. If $content(exp^{-1}(\mathbf{q})) = content(exp^{-1}(\mathbf{p}))$, then inductively, $exp^{-1}(\mathbf{q}) \prec exp^{-1}(\mathbf{p})$. Thus, in either case, by Lemma 8, $\mathbf{q} \prec \mathbf{p}$. □

Lyndon words are counted by length and weight in [9].

4 Stern-Brocot strings

In this section we demonstrate an interesting correspondence between the Stern-Brocot tree of reduced fractions and the set of all binary Euclidean strings.

The *Stern-Brocot tree* [12] is an infinite construction of the set of all non-negative fractions k/n between 0 and 1 where k and n are relatively prime. Each node in the tree is constructed by using its nearest left ancestor (L), and nearest right ancestor (R). In particular, if a node x has $L = k'/n'$ and $R = k''/n''$ then $x = (k' + k'')/(n' + n'')$. To start this recursive construction, L is initialized to $0/1$ and R is initialized to $1/1$. The Stern-Brocot tree is illustrated in Figure 4.

We now construct an equivalent tree composed of binary strings where each fraction k/n in the Stern-Brocot corresponds to a length n binary string with k ones. In this new tree, each string α is the concatenation of its nearest left ancestor L and its nearest right ancestor R. If L and R are initially assigned the characters 0 and 1 respectively, then we obtain the corresponding Stern-Brocot fraction by computing the number of ones in the string along with the length. This tree of strings, denoted $\mathcal{S}$, is illustrated in Figure 5. Interestingly, this tree includes all and only the binary Euclidean strings.

Theorem 4
The length n *binary string* α *is Euclidean if and only if* α *is in* $\mathcal{S}$. *Furthermore, if* $n > 1$ *where* $\alpha = LR$, *then* $\tau(LR) = \rho^{|L|}(LR)$.

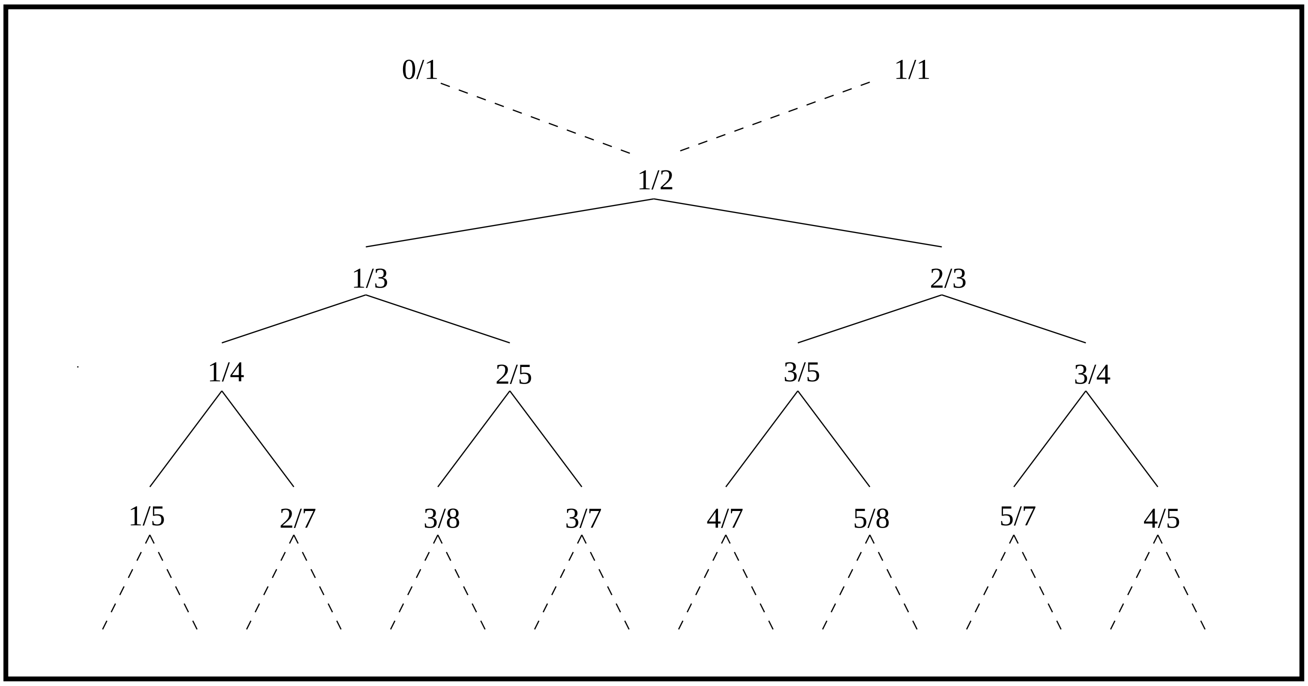

Figure 4: The Stern-Brocot tree of reduced fractions

Proof
We proceed by induction on the length of the string. Suppose the length n binary string α is in $\mathcal{S}$. In the base cases, strings 0, 1 and 01 are Euclidean and $\tau(01) = 10 = \rho(01)$.

Now consider two cases depending on whether α is a left-child or a right-child.

If α is a left-child then by construction $R = LR'$ where R' is the nearest right ancestor of R. By induction $\tau(LR') = \rho^{|L|}(LR')$ which implies that $\tau(LLR') = \rho^{|L|}(LLR')$. Thus by definition α is Euclidean and $\tau(LR) = \rho^{|L|}(LR)$.

If α is a right-child then by construction $L = L'R$ where L' is the nearest left ancestor of L. By induction $\tau(L'R) = \rho^{|L'|}(L'R)$ which implies that $\tau(L'RR) = \rho^{|L|}(L'RR)$. Thus by definition α is Euclidean and $\tau(LR) = \rho^{|L|}(LR)$.

Thus every string in $\mathcal{S}$ is Euclidean. Since this tree corresponds to the Stern-Brocot tree, there exists a length n string in $\mathcal{S}$ with k ones whenever k and n are relatively prime. By Theorem 1, this implies that if α is a binary Euclidean string, then it is in $\mathcal{S}$. $\square$

Theorem 4 implies that every Euclidean string of length greater than or equal to two is the concatenation of two shorter Euclidean strings.

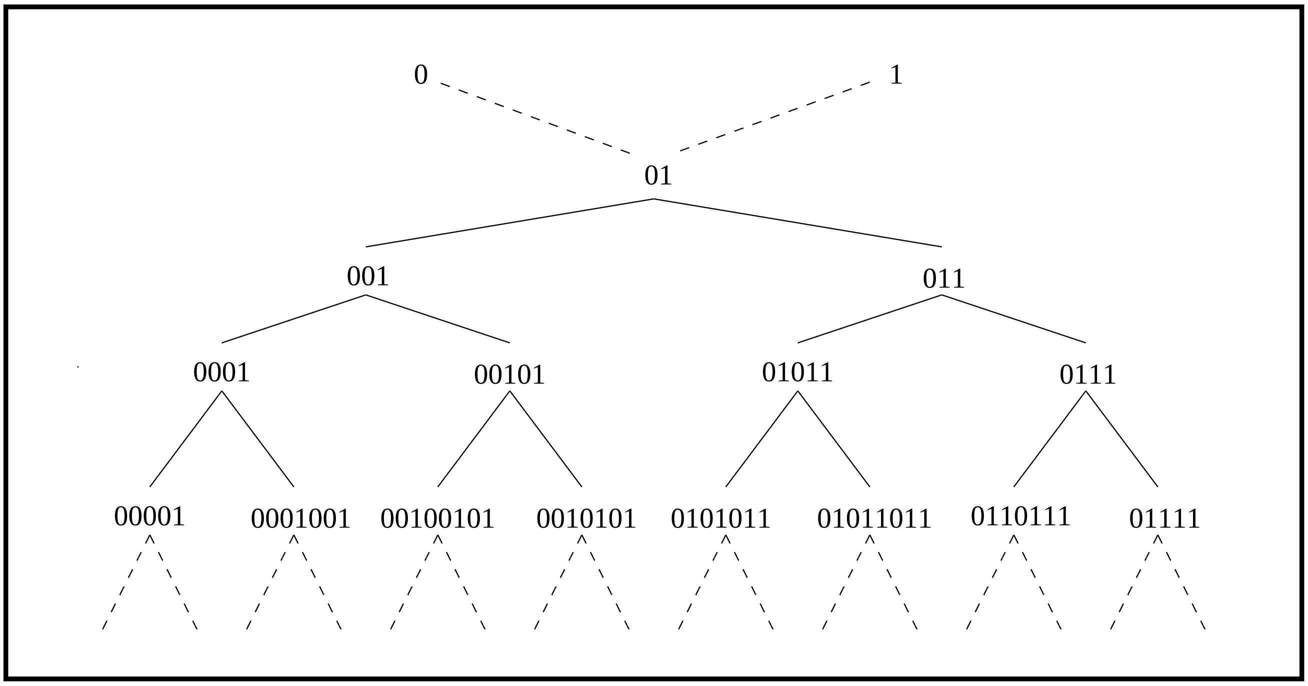

Figure 5: The Stern-Brocot tree of Euclidean strings.

Corollary 2
Every binary Euclidean string $E_{n,k}$ where $n \geq 2$ and with displacement d and cost c is the concatenation of the Euclidean strings $E_{d,c}$ and $E_{n-d,k-c}$.

Proof
By Theorem 4, the dispacement of the binary Euclidean string α is $|L|$ where α is the concatenation of L and R. Since L is binary, the cost of α is the weight of L. $\square$

The tree of binary Euclidean strings, Figure 5, suggests yet another way of generating any such string, i.e., by way of a sequence of string concatenations, defined by a path down the tree to the node corresponding to the fraction k/n. Corollary 2 implies that the diplacement and cost can be computed simultaneously.

The algorithm uses global variables for current left ancestor $L = E_{n_1,k_1}$, for the current right ancestor $R = E_{n_2,k_2}$ and for n and k. The variable L should be initialized to "0" and R to "1". The initial call to generate $E_{n,k}$ is Stern-Brocot(0,0,0,1).

procedure Stern-Brocot (n_1, k_1, n_2, k_2: $\mathbb{Z}$);

local $n_3, k_3 : \mathbb{Z}$;

$n_3 := n_1 + n_2; \quad k_3 := k_1 + k_2$;

if $k_3/n_3 = k/n$ **then** print(LR, n_1, k_1);

if $k_3/n_3 < k/n$ **then** $R := LR$; Stern-Brocot(n_1, k_1, n_3, k_3);

if $k_3/n_3 > k/n$ **then** $L := LR$; Stern-Brocot(n_3, k_3, n_2, k_2);

There are exactly $n-1$ concatenations. Hence, if the strings are represented by linked lists, the time complexity of the algorithm is linear.

All the ancestors of $E_{n,k}$ can be generated by removing the **if** condition from the print statement. As already observed, any Euclidean string $E_{n,k}$, where $n < k$, can be obtained from the binary string $E_{n,k \bmod n}$ by replacing the zeros by $\lfloor k/n \rfloor$ and the ones by $\lceil k/n \rceil$. The displacement is unchanged, and if the cost of $E_{n,k \bmod n}$ is c then the cost of $E_{n,k}$ is $d\lfloor k/n \rfloor + c$. See Figure 1.

5 Fibonacci strings

We go on to show a relationship between Fibonacci and Euclidean strings.

Definition 6
A Fibonacci string is defined by the recursive construction rules $b \to a$, $a \to ab$.

For example, the first seven Fibonacci strings are:

$$b,\ a,\ ab,\ aba,\ abaab,\ abaababa,\ abaababaabaab.$$

Let F_i denote i-th Fibonnacci string with length f_i. It is known that, $F_i = F_{i-1}F_{i-2}$ and hence that f_i is the i-th Fibonacci number. Fibonacci strings occur as the worst case inputs to certain algorithms and they possess many interesting properties [4], [10], [11], [13], [8].

Let $G = G_1, G_2, G_3, \cdots$ be an infinite sequence of Euclidean strings where each G_i is defined as follows:

$$G_i = \begin{cases} 1 & \text{if } i = 1, \\ 0 & \text{if } i = 2, \\ right(G_{i-1}) & \text{if } i > 2 \text{ and } i \text{ odd}, \\ left(G_{i-1}) & \text{if } i > 2 \text{ and } i \text{ even}, \end{cases}$$

where $right(\alpha)$ is the right-child of α in the Stern-Brocot tree of strings $\mathcal{S}$ and $left(\alpha)$ is the left-child of α in the tree $\mathcal{S}$. This construction implies that $G_i = G_{i-2}G_{i-1}$ when $i > 2$ and i is even; if $i > 2$ and i odd, then $G_i = G_{i-1}G_{i-2}$.

Lemma 11
Every Fibonacci string is rotationally equivalent to a Euclidean string. If a=0 and b=1 then,

$$F_i = \begin{cases} G_i & \text{if } i \leq 2, \\ \rho^{-(f_{i-2}+1)}(G_i) & \text{if } i > 2 \text{ and } i \text{ odd}, \\ \rho^{-1}(G_i) & \text{if } i > 2 \text{ and } i \text{ even}. \end{cases}$$

Proof
This result is easy to verify for $i < 5$. For $i \geq 5$,

$$\begin{aligned} F_i &= F_{i-1}F_{i-2} \\ &= F_{i-2}F_{i-3}F_{i-2} \\ &= F_{i-3}F_{i-4}F_{i-3}F_{i-3}F_{i-4}. \end{aligned}$$

We now consider two cases depending on the parity of i.

If i odd then since $i-3$ is even, $F_{i-3} = xa$ for some string x. If we let $F_{i-4} = y$ then we have:

$$F_i = xayxaxay.$$

Now, by induction $F_{i-1} = xayxa = \rho^{-1}(G_{i-1})$, which implies that $G_{i-1} = axayx$. Similarly, $F_{i-2} = xay = \rho^{-(f_{i-4}+1)}(G_{i-2})$, which implies that $G_{i-2} = ayx$. Since i is odd, $G_i = G_{i-1}G_{i-2} = axayxayx$ and $\rho^{-(f_{i-2}+1)}(G_i) = xayxaxay = F_i$.

If i is even then since $i-4$ is even, $F_{i-4} = xa$ for some string x. If we let $F_{i-3} = y$ then we have:

$$F_i = yxayyxa.$$

Now, by induction $F_{i-1} = yxay = \rho^{-(f_{i-3}+1)}(G_{i-1})$, which implies that $G_{i-1} = ayyx$. Similarly, $F_{i-2} = yxa = \rho^{-1}(G_{i-2})$, which implies that $G_{i-2} = ayx$. Since i is even, $G_i = G_{i-2}G_{i-1} = ayxayyx$ and $\rho(G_i) = yxayyxa = F_i$. □

From Lemmas 11 and 7 we see that we can construct the Lyndon word corresponding to any Fibonacci word.

6 Beatty sequences

In this section we show that Euclidean strings are related to Beatty sequences. A Beatty sequence is a sequence of the form $a_j = \lfloor \alpha j + \beta \rfloor$ where $\alpha > 1$ and $j \in \mathbb{Z}$. See for example [1] and [7]. Generally α and β can be rational or irrational but we will only be concerned with the case where α is rational and $\beta = 0$.

Lemma 12
Consider the Beatty sequence comprising the elements in the set $a_j = \left\lfloor \frac{n}{k} j \right\rfloor : j \in \mathbb{Z}$, *where* $\gcd(n,k) = 1$. *For each integer* j *there exists an* i *where* $0 \le i \le k-1$ *such that*

$$id + \left\lfloor \frac{n}{k} j \right\rfloor \equiv 0 \bmod n$$

where d *is the multiplicative inverse of* k, *modulo* n.

Proof
Since $dk \equiv 1 \bmod n$, for any z, $z \equiv zdk \bmod n$. Hence, for any j

$$-\left\lfloor \frac{n}{k} j \right\rfloor \equiv (nj - k \left\lfloor \frac{n}{k} j \right\rfloor) d \bmod n.$$

Consider the term in parentheses, which is a way of writing $nj \bmod k$. Because n and k are relatively prime, the term in parentheses can take on any integer value in the interval $[0, k-1]$. Thus for any j there exists $i \in [0, k-1]$ such that

$$id + \left\lfloor \frac{n}{k} j \right\rfloor \equiv 0 \bmod n.$$

Similarly for any $i \in [0, k-1]$ there exists j satisfying this congruence. $\square$

Corollary 3
The string $p_0 p_1 \cdots p_{n-1}$ *is* $E_{n,k}$ *where* $p_i = |\{j : n - \lfloor nj/k \rfloor - 1 = i,\ 0 \le j < k\}|$.

Proof
By Lemma 1(c), $p_i = |\{j : n-1+jd \equiv i \bmod n,\ 0 \le j < k\}|$, where d is the displacement of the string. By Lemma 1(d), $dk \equiv 1 \bmod n$. Hence, by Lemma 12, the set on the right hand side of the equation is identical to the set $\{j : (n - \left\lfloor \frac{n}{k} j \right\rfloor - 1) = i, 0 \le j < k\}$. $\square$

This gives us one more algorithm for computing a Euclidean string. First initialize $p_0 p_1 \cdots p_{n-1}$ to be all zeroes. Then execute the following line of code, which uses the C increment operator.

$$\textbf{for } j = 0..k-1 \textbf{ do } {+}{+}p_{n-\lfloor nj/k \rfloor -1}$$

That is, a Euclidean string is the reversal of the histogram of a rational Beatty sequence.

As previously noted, it is faster to reduce an n,k instance where $k > n$ to the instance $n, k \bmod n$. After the application of Corollary 3 we construct the correct string by replacing the ones by $\lceil k/n \rceil$ and the zeros by $\lfloor k/n \rfloor$.

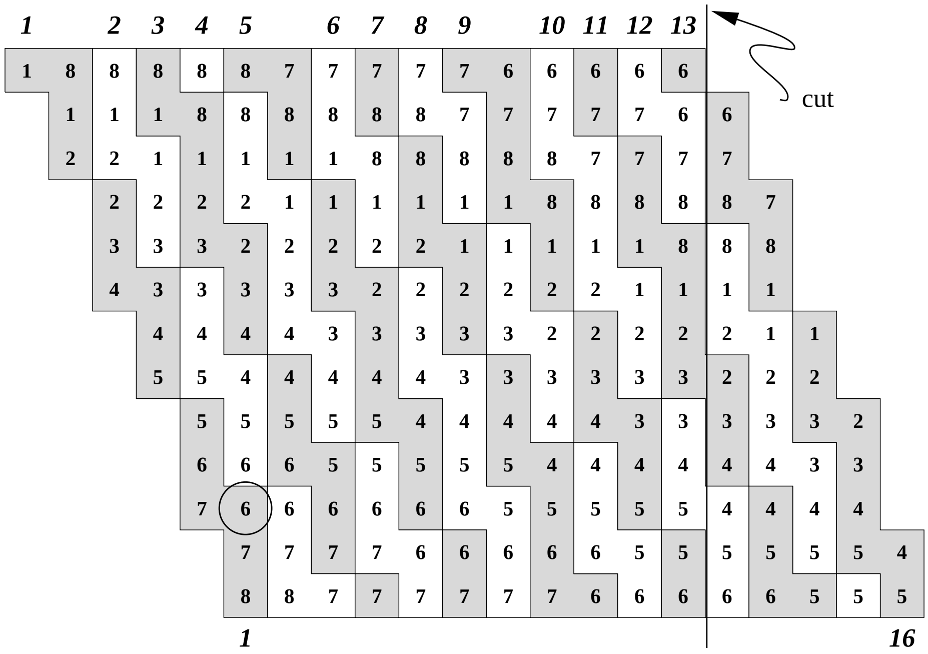

Figure 6: Grid to Torus Mapping

7 An application

The original motivation for studying these strings came from a graph embedding problem, in particular, the problem of many to one mappings from the nodes of a 2-dimensional grid onto the nodes of a hypercube. Here we require that the size of the grid is maximum with respect to the size of the hypercube and some specified "load", i.e., the maximum number of grid nodes that can be mapped onto a single hypercube node. We ask whether there exists a mapping with dilation one, i.e., in which any pair of grid nodes connected by an edge are either mapped to the same hypercube node, or to the ends of an edge in the hypercube. A solution to this problem which uses the analysis of Euclidean strings is given in [5], for loads ≥ 4.

We give a very informal description of why the Euclidean strings are useful. The mapping into the hypercube is established via a mapping into a torus which is a subgraph of the hypercube. Consider Figure 6. This diagram defines a partial mapping from part of a 13×21 grid onto an 8×16 torus with load 2. For each element in the grid, the number at each grid position specifies the torus row onto which the grid node is to be mapped. The shading distinguishes torus columns, numbered across the top. Thus for example, the second grid element in the top row of the grid is to be mapped to row 8, column 1 of the torus.

Suppose we chop off the "steps" at the right hand end, as indicated by the "cut" and insert the detached piece at the left end. Note that the shapes at the ends "match". If the numbers also matched (a separate problem) we would have a load 2, dilation 1 embedding of the 13×16 grid into the 8×16 torus with load 2. Because that torus is a subgraph of the degree 7 hypercube, this embedding is also an embedding into the hypercube.

We may define the left and right "profiles" of a hypercube column in the diagram to be the sequence of step heights on the left or right of the column taken circularly. For example the left profile of column 2 is (3, 2, 3, 2, 3). Note that each successive profile is a rotation of its predecessor. For the cut and paste method to work we want the left profile of the leftmost column to match the right profile of the rightmost column.

The Euclidean string analysis permits us to argue that it is always possible to find a profile such that left and right end profiles of the pattern match. The profile pattern in Figure 6 is periodic with period 13. In general the width of the grid is not a multiple of this period. Further, it is necessary to "drop" one of more torus nodes from the pattern because the number of nodes in the grid is usually strictly less than the number of nodes in the torus times the load.

Consider the effect of dropping one torus node from the profile. For example, let us drop the circled element from column 1. One element in the profile is incremented and one is decremented by one. The resulting profile is now (3, 2, 3, 3, 2), which is a rotation of the unmodified profile and which matches the left profile of column 6.

The analysis of Euclidean strings establishes that we can always find a profile with the properties we need, namely that the result of dropping an element is a profile that is rotationally equivalent to the original.

References

[1] S. Beatty, *Problem 3173*, Amer. Math. Monthly, 33 (1926), 159.

[2] N. Calkin and H.S. Wilf, *Recounting the Rationals*, American Mathematical Monthly 107 (2000) 360-363.

[3] T. H. Corman, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill (1990).

[4] X. Droubay, *Palindromes in the Fibonacci Word*, Information Processing Letters, 55 (1995) 217-221.

[5] J. Ellis, S. Chow and D. Manke *Embedding grids into cylinders, toruses and hypercubes*, submitted for publication.

[6] J. Ellis, F. Ruskey and J. Sawada *Euclidean Strings*, Proceedings of the Eleventh Australasian Workshop on Combinatorial Algorithms, L. Brankovic, J. Ryan (Eds), University of Newcastle, Australia (2000), 87-92.

[7] A. S. Fraenkel, *The bracket function and complementary sets of integers*, Canadian Jnl. Math., 21 (1967) 6-27.

[8] A. S. Fraenkel and J. Simpson, *The exact number of squares in Fibonacci words*, Theoretical Computer Science, 218, (1999), 95-106.

[9] F. Ruskey, A. Gulliver, C.R. Miers, and J. Sawada, *The Number of Lyndon Words and Irreducible Polynomials of Given Trace*, SIAM J. Discrete Mathematics, to appear.

[10] C.S. Iliopoulos, D. Moore, and W.F. Smyth, *The covers of a circular Fibonacci string*, J. Combin. Math. Combin. Comput. 26 (1998), 227-236.

[11] C.S. Iliopoulos, D. Moore, and W.F. Smyth, *A characterization of the squares in a Fibonacci string*, Theoretical Computer Science, 172 (1997), no. 1-2, 281-291.

[12] D.E. Knuth, R.L. Graham, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley (1989).

[13] W. Zhi-Xiong and W. Zhi-Ying, *Some properties of the singlular words of the Fibonacci word*, Séminaire Lotharingien de Combinatoire, 30 (1993).