RAIRO. INFORMATIQUE THÉORIQUE

A. APOSTOLICO

On context constrained squares and repetitions in a string

RAIRO. Informatique théorique, tome 18, nº 2 (1984), p. 147-159

<http://www.numdam.org/item?id=ITA_1984__18_2_147_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 18, n° 2, 1984, p. 147 à 159).

ON CONTEXT CONSTRAINED SQUARES AND REPETITIONS IN A STRING (*) (**)

by A. Apostolico ([1])

Communicated by J. Berstel

Abstract. — *Some combinatorial and computational problems concerning repetitions and repetition roots in a string x on a finite alphabet—that are characterized in general by an $O(n \log n)$ bound in terms of the length n of x—are shown to admit of a linear bound when approached in particular contexts.*

More precisely, it is shown that the number of distinct repetition roots u that are bond to the occurrence of their cube u^3 somewhere along the textstring is bounded by n, whence this same bound can be drawn for the number of distinct cube substrings appearing in a generic string. Constraints of similar nature are also discussed that guarantee linear time square-free recognition, and a linear time strategy is proposed to detect, in correspondence with each primitive root u that meets such conditions on x, and for all possible forms of u-rooted repetitions in x, the leftmost occurring repetition in this form.

Résumé. — *On montre que quelques problèmes combinatoires et de calcul concernant les répétitions et racines de répétition dans un mot x sur un alphabet fini — majorés en général par une borne en $O(n \log n)$ en fonction de la longueur n de x — admet une borne linéaire dans certaines situations.*

Plus précisément, on montre que le nombre de racines de répétitions u distincts dont au plus le cube u^3 apparaît dans le mot donné est borné par n, ce qui donne la même borne pour le nombre de cubes distincts apparaissant dans un mot générique. On discute de semblables contraintes pour obtenir un algorithme linéaire testant si un mot est sans carré. On propose une stratégie linéaire en temps pour détecter, avec chaque racine primitive vérifiant certaines conditions, la répétition la plus à gauche d'une certaine forme.

I. INTRODUCTION

Since the time of their discovery by *A.* Thue [1], square free strings have received increasing attention by workers in disparate fields.

Recently, some interest has focussed on detection and recognition problems for squares in a textstring: in this endeavor, an $O(n \log n)$ ([2]) algorithm has

(*) Received in February 1982, revised in April 1983.
(**) This work has been supported in part by the Italian Ministry of Education. Additional support was provided by N.A.T.O. under Research Grant No. 039.82 and by the Italian National Council for Research.

([1]) Istituto di Scienze dell'Informazione, University of Salerno, I-84100 Salerno, Italy. Current Address: Department of Computer Sciences, Purdue University, West Lafayette, In. 47907, U.S.A.

([2]) Here and hereafter, log denotes the logarithm to the base 2.

0399-0540/84/02 147 13/$ 3.30/

been proposed to test square-freeness of a string x of length n [2], and two strategies have been also set up that detect all distinct repetitions in x in time $O(n \log n)$ and space $O(n)$ [3, 4]. Since Fibonacci words [3] attain the $O(n \log n)$ upper bound [5] for distinct repetitions in a string, the two latter strategies above are also optimal. On the other hand, it is not known at present whether $O(n \log n)$ is optimal for plain square free testing.

Problems involving repetitions and repetition roots also arise in connection with the statistics "without overlap" of all substrings of a string [6]. Indeed, the cardinality of the set of distinct substrings that are root each of some repetition affects the time and space needed to perform such statistics. It is easy to check that infinitely many Fibonacci words also contain $O(n \log n)$ distinct substrings that are root each of some repetition.

In section 3 of this paper, a constraint (*cube constraint*) is introduced on the input string x that forces to within an $O(n)$ bound the variety of such repetition-root substrings of x. Constraints of similar nature (*rot constraints*) are presented in section 4, that guarantee linear time and space square free testing for a string. Let now the repetitions in x be partitioned into equivalence classes by grouping together those that are occurrences of the same substring of x, and let the leftmost (in x) repetition in each class be taken as the representative of the class. In the final part of the paper, it is shown that all root constrained representatives can be detected in linear time and space, on line with the construction of the suffix tree [7] associated with x.

II. NOTATIONS AND INTRODUCTORY REMARKS

Let I be a finite alphabet and I^+ the free semigroup generated by I. *A. string* $x \in I^+$ is fully specified by writing $x = a_0 a_1 \ldots a_{n-1}$, where $a_i \in I$ $(i = 0, 1, \ldots, n-1)$. We assume here that x is stored in an array x $(0, n-1)$, where $x\{i\} = a_i$ $(i = 0, 1, \ldots, n-1)$. Given $x = a_0 a_1 \ldots a_{n-1}$, w is a *substring* of x if there exist indices i, j $(0 \leqq i \leqq j \leqq n-1)$ such that $w = a_i a_{i+1} \ldots a_j$. We also say that $a_i a_{i+1} \ldots a_j$ is an *occurence* of w in this case, and we shortly denote it by $x(i, j)$. We use $|w|$ to denote the *length* of w. Occasionally in what follows, it will be assumed implicitly that $|x| = n$.

The set of all distinct nonempty substrings of x (*words*) is called the *vocabulary* of x and denoted by V_x. A *weighted vocabulary* for x is any pair (V_x, C), where $C: V_x \rightarrow N$ is a mapping that associates with each string $w \in V_x$ a natural number $k = C(w)$.

Let $\mathbb{S}$ be a special symbol not included in the alphabet I. A data structure suitable for organizing the words in V_x is the *suffix tree* [7] T_x for $x\mathbb{S}$. As is well known, such a tree T_x is rooted, has $O(n)$ nodes and for a string $x\mathbb{S}$

can be defined as the digital search tree associated with the set of all suffixes $suf_i = x(i, n)$ $(i=0, 1, \ldots, n)$ of $x\mathbb{S}$. The following definitions match to a good extent those found in reference 7.

A *partial path* in T_x is a connected sequence of tree arcs which starts at the root of T_x; a *path* is a partial path that terminates on a leaf of T_x; the *proper locus* of a string w is the node α, if it exists, of the (partial) path associated with w. Conversely, each node or leaf α is the proper locus of a string, hereafter denoted $W(\alpha)$. An *extension* (*prefix*) of w is any string y such that $wu=y$ ($yu=w$) with $u \in I^+$. The *extended* (*contracted*) *locus* of w is the locus of the shortest extension (longest prefix) of w admitting of a proper locus in T_x. When no confusion may arise, we shall refer to the proper or extended locus of w simply as to the *locus* of w. There exist clever algorithms for the construction of T_x in linear time [7, 9, 10].

We assume familiarity of the reader with the notions of *primitiveness* of a string, as well as with the related concepts (we defer the reader to ref. [4]). We adopt the following definition of repetition in a string: a *repetition* in x is a triplet $R(i, p, L)$ such that, letting $m=i+L-1$, there are indices $j, d\,(d-1 \leqq j \leqq m)$ such that: (*a*) $x(i, j)$ and $x(d, m)$ are occurrences of the same substring; (*b*) $x(i, d-1)$ corresponds to a primitive word, and (*c*) $x\{j+1\} \neq x\{m+1\}$. Thus, a repetition is a positioned periodic substring in the form $(st)^k s$, where it is $k > 1$, $s \in I^*$ *and* $t \in I^+$; The elements i, $p=d-i$ and $L=m-i+1$ shall be called its *starting position*, its *period* and its *length*, respectively.

Finally, we will make use of the "periodicity lemma" [8], which we report below for convenience of the reader.

LEMMA 1: *If w has periods p and q, and $|w| \geqq p+q$, then w has period g. c. d. (p, q).*

III. CUBE CONSTRAINTS

For a generic string x, let $U \in V_x$ denote the set of all substrings of x that are roots of some repetition in x. As mentioned, the cardinality of U is bounded by $O(n \log n)$; the following theorem shows that this bound is tight.

THEOREM 1: *Let the sequence of Fibonacci words be defined as follows: $F_0=b$, $F_1=a$ and $F_{m+1}=F_m F_{m-1}$ for $m > 1$, and let r_m denote the cardinality of U for F_m. Then r_m satisfies, for all $m \geqq 4$:*

$$r_m \geqq \frac{1}{12} |F_m| \log |F_m|.$$

Proof: Exercise for the reader. (Hint: use induction on m, in conjunction with the fact that the cyclic permutation of a primitive word yields a primitive word. See also the derivation of lemma 10 in ref. [3].) □

Let now u^2 be some square substring of x of (primitive) root u. We say that u^2 is a *cube constrained word* (CCW, for short) if $u^3 \in V_x$. In addition, x is a *cube constrained string* (CCS) if all of the square, primitive-rooted words in V_x are CCW's. In what follows, we exploit the structure of T_x in conjunction with the periodicity lemma to prove the following:

THEOREM 2: *The number of distinct CCW's in x is bounded by n.*

In proving the theorem, we shall need the two lemmas below.

LEMMA 2: *If u^{k+1} ($k \geqq 1$) is a substring of x, then u^k and u^{k+1} have distinct loci in T_x.*

Proof: Since $u^{k+1} \in V_x$, then there is some repetition in x in the form u^{k+1}, where u' is a (possibly empty) prefix of u. Let then $R(i, p, L)$ be one such repetition. By definition, there is a vertex ν in T_x whereby the suffixes $x(i, n)$ and $x(i+p, n)$ must bypart, and such that $W(\nu) = u^k u'$. Hence the locus of u^k is either ν or an ancestor of ν, whereas ν cannot be the locus of u^{k+1}, since it is $|u'| < |u|$. □

LEMMA 3: *If u^2 and v^2 are distinct CCW's in x, then they have distinct loci in T_x.*

Proof: The assertion is true if u^2 and v^2 have both proper loci in T_x. Assume now that at least one of the two has not a proper locus and let α be the common locus of u^2 and v^2. Let also, without loss of generality, $2|u| < 2|v| \leqq |W(\alpha)|$. Since u^2 is cube constrained, it must be $W(\alpha) = u^2 u'$, where in force of Lemma 2 u' is a nonempty prefix of u and α is an interior vertex of T_x. On the other hand, it is $|W(\alpha)| \geqq 2|v| > |v| + |u|$, whereas $W(\alpha)$ has periods $|u|$ and $|v|$. But then, by the periodicity lemma, $W(\alpha)$ has period $g.c.d.\,(|u|, |v|)$, so that v cannot be a primitive word, contrary to the assumption. □

Proof of theorem 2: The assertion follows at once from Lemmas 2 and 3 by recalling that, T_x being a multiway Patricia Tree [11], with $n+1$ leaves, the number of its interior vertices is bounded by n. □

We leave it as a simple exercise for the reader to show the following:

COROLLAIRE 1: *The number of distinct cube substring (whence, primitive roots of cubes) in a generic string x is bounded by n.*

And:

COROLLAIRE 2: *If x is a Cube Constrained String, then the number of distinct substrings of x that correspond to the root of some repetition in x is bounded by n.*

The cube constraint for a string x has consequences on the amount of storage needed to allocate the statistics without overlap of all its substrings [6]. To make this point more clear, consider the weighted vocabularies (V_x, C_1) and (V_x, C_2), defined as follows. C_1 simply associates, with each $w \in V_x$, the number of occurrences of w in x; on the other hand, C_2 associates with w the maximum number k of distinct occurrences of w such that it is possible to write $x = w_1 w w_2 w w_3 \ldots w w_{k+1}$, with $w_d \in I^*$ $(d = 1, 2, \ldots, k+1)$. It is almost straightforward to see that T_x itself can be weighted to store (V_x, C_1) in such a way that, for each $w \in V_x$, the weight of the locus α of w equals $C_1(w)$; moreover, an $O(n)$ time weighting procedure for T_x can be readily arranged. On the other hand, the exploitation of the Suffix Tree for the storage of (V_x, C_2) requires in general an augmentation of the tree [6] *via* the insertion of auxiliary nodes of degree 1, as being proper loci for substrings in the form u^k (u primitive and $k \geqq 1$) such that $u^{2k} \in V_x$. More accurately, let $\overline{T}_x$ denote the minimal (i. e., with the lowest possible number of vertices) weighted tree among those obtained by T_x by inserting auxiliary nodes in such a way that, for any word $w \in V_x$, the locus α of w is labeled with $C_2(w)$. The following theorem holds.

THEOREM 3 [6]: *If α is an auxiliary node of $\overline{T}_x$, then there are substrings u, v in x and an interger $k \geqq 1$ such that $W(\alpha) = u = v^k$ and there is a repetition in x in the form $v^m v'$, with v' a prefix of v and $m \geqq 2k$.*

We are now in the position to state the following:

THEOREM 4: *Let x be a Cube Constrained String. Then $O(n)$ locations suffice to store (V_x, C_2).*

Proof: By theorem 3, auxiliary nodes in $\overline{T}_x$ are due repetition roots in x and powers theoreof. By corollary 2, the cardinality of U for x is bounded by n. Let A be the set of auxiliary nodes of $\overline{T}_x$ that are loci of substrings in the form u^k, with $k > 1$, and let v and w be two distinct such substrings. By theorem 3, if it is, say, $v = s^i$ and $w = t^j$, then s^{i+1} and t^{j+1} must be substrings of x, too. By Lemma 1, the locus of s^{i+1} (t^{j+1}) in T_x differs from that of s^i (t^i). Hence the same argument used in deriving Lemma 2 above displays that v and w cannot have the same locus in $\overline{T}_x$, namely, at most one vertex from A can be used to split each original (nonterminal) arc of T_x. It follows that the cardinality of A is also bounded by n. □

IV. ROOT CONSTRAINTS

Let x be a non square-free string in I^+, and u^2 a square word of V_x of (primitive) root u. We say that u^2 is a *root constrained square word* (RCSW) of x if, letting $R(i, p, L)$ be the leftmost u-rooted repetition in x, and letting v be the longest prefix of u^2 occurring to the left of R, then if R has the form $u^k u'$, it is $|v| \leqq |uu'|$. On the other hand, a repetition $R(i, p, L)$ in x is a *root constrained repetition* (RCR) if $x(i, i+2p-1)$ is an occurrence of a RCSW u^2. For example, in the string:

$$\begin{array}{lcccccccccccccccccccccc} x= & a & c & a & b & b & c & b & c & b & a & c & a & c & b & c & b & c & b & c & b & b & b \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 \end{array}$$

bb, *bcbc*, *cbcb* are RSCW's whereas *acac* is not; $x(3, 4)$, $x(4, 8)$, $x(5, 8)$, $x(13, 19)$, $x(14, 19)$, $x(19, 21)$ are RCR's but $x(9, 12)$ is not.

In this section we propose an algorithmic criterion that supports the detection, in linear time and space, of all distinct RCSW's of V_x. We also present a more elaborated criterion supporting an optimal strategy that returns, for each RCSW u^2 and for all forms $u^k u'$ corresponding to a RCR $R(i_j, p, L)$ $(j=1, 2, \ldots, r)$, the leftmost such repetition in x. Let T_{i-1} denote the partial suffix tree that collects the suffixes suf_k $(k=0, 1, 2, \ldots, i-1)$. Since $\$ \notin I$, we can write $\mathrm{suf}_i = \mathrm{head}_i . \mathrm{tail}_i$ $(i=0, 1, \ldots, n)$ with tail_i non empty and head_i equal to the longest prefix of suf_i that is also a prefix of suf_j for some $j < i$.

LEMMA 4: *u^2 is a RCSW of x iff there are indices j and $i=j+|u|$ and a repetition $R(j, u, L)$ such that in T_{i-1}:*

(1) *$\alpha = locus\ (head)_i$ is either leaf j or the father node μ of leaf j.*

(2) *Letting $d=L\ (mod.\,|u|)$, it is $W(\mu) \leqq |u|+d$.*

Proof: If u^2 is a RCSW of x, then there must be a repetition $R(j, u, L)$ in x in the form $u^k u'$ such that no occurrence of uu'' with $|u''|>|u'|$ is found starting at some position of x smaller than j. Let $a \in I$ be the symbol following u' in u. Then the locus of $uu'a$ in T_j is leaf j. Considering now the father node μ of leaf j in T_j it is easily seen [4] that $i=j+u$ and j must be consecutive leaves in the subtree of T_x rooted at μ. Hence the locus of $u^{k-1}u'$ in T_{i-1} is either leaf j or node μ. Letting now $d=|u'|$, it is clearly $d=L(\mathrm{mod.}\,|u|)$, whence $W(\mu) \leqq |uu'|=|u|+d$. The converse portion of the proof is straightforward. □

If, in the above, we call the locus of head_i in T_i the *word detecting node* for u^2, then Lemma 4 can be rephrased by saying that all and only the RCSW's of x admit each of a distinct word detecting node. We now introduce augmented versions $\tilde{T}_i$ of T_i $(i=-1, 0, 1, \ldots, n)$ as follows: we let $\tilde{T}_{-1}=\text{Root}$. For $i \geqq 0$, $\tilde{T}_i$ is still produced by in inserting suf_i in $\tilde{T}_{i-1}$. However, if suf_i has a RCSW prefix u^2 which has an extended but not a proper locus in $\tilde{T}_{i-1}$, then an auxiliary unary node is also inserted in $\tilde{T}_i$ to be the proper locus of u^2.

LEMMA 5: *$\tilde{T}_x \triangleq \tilde{T}_n$ has $O(n)$ nodes.*

Proof: In view of Lemma 4 above, each auxiliary node can be charged to the corresponding word detecting node. □

For some $0 < i < n$, consider $\tilde{T}_i$ and assume that it be $\text{head}_i=(st)^k s$ with st primitive, $|st|=p$, $k \geqq 1$, $s \in I^*$, $t \in I^+$ and $(st)^2$ a RCSW of x. Let also $\nu=\text{locus}(\text{head}_i)$ in $\tilde{T}_i$, $\alpha=\text{locus}(\text{head}_i)$ in $\tilde{T}_{i-1}$ and $j=\max(\alpha) \triangleq$ the largest leaf in the subtree of $\tilde{T}_{i-1}$ rooted at α. Node ν is called a "p"-*node* if, in the above, it is $k \geqq 2$. Whatever the value of k, ν is a *detecting node* if it is in $\tilde{T}_i \, |W(\mu)| \geqq (i-j)$ and the locus of head_i in $\tilde{T}_{i-1}$ is not a "p"-node. Note that a word detecting node in T_i is also a detecting node in $\tilde{T}_i$. On the other hand, assume that $\nu \neq \alpha$ and α is a "p"-node in $\tilde{T}_i$. Then ν is an *extension node* or a *prefix node* according to whether or not $\text{suf}_i=\text{head}_i . vw$ where $\text{head}_i . v$ has period p and $v \in I^+$. The periodicity Lemma guarantees for the above definitions to be unambiguous. Figures 1 (a, b, c) display instances of such nodes.

LEMMA 6: *$R(j, |u|, L)$ is the leftmost u-rooted RCR in the form $u^k u'$ if and only if, letting $i=j+|u|$; one of the following holds:*

(1) *locus (head_i) in $\tilde{T}_i$ is a detecting node;*

(2) *locus (head_j) in $\tilde{T}_j$ is a detecting, a prefix or an extension node.*

Proof: By Lemma 4, if $R(j, |u|, L)$ is the leftmost u-rooted RCR in x, then locus (head_i) in $\tilde{T}_i$ is the word detecting node for u^2. Notice that, if it is $k > 2$, then $R(i, |u|, L-|u|)$ is also a RCR. Assume now that some u-rooted RCR $R(d, |u|, L')$ exist such that $d < j$ and $L' \neq L$. By construction, locus (head_j) in $\tilde{T}_j$ is a "$|u|$"-node. It is easy to check that such node must fulfill the definition of either a detecting, a prefix or an extension node. The converse portion of the proof is also straightforward. □

V. ALGORITHMIC IMPLEMENTATION

The two strategies presented in this section exploit the criteria conveyed by Lemmas 4 and 6. They shall be developed as consecutive upgrades of the suffix tree construction algorithm [7], a self-explanatory outline of which is listed below. For the sake of brevity, however, the discussion to follow

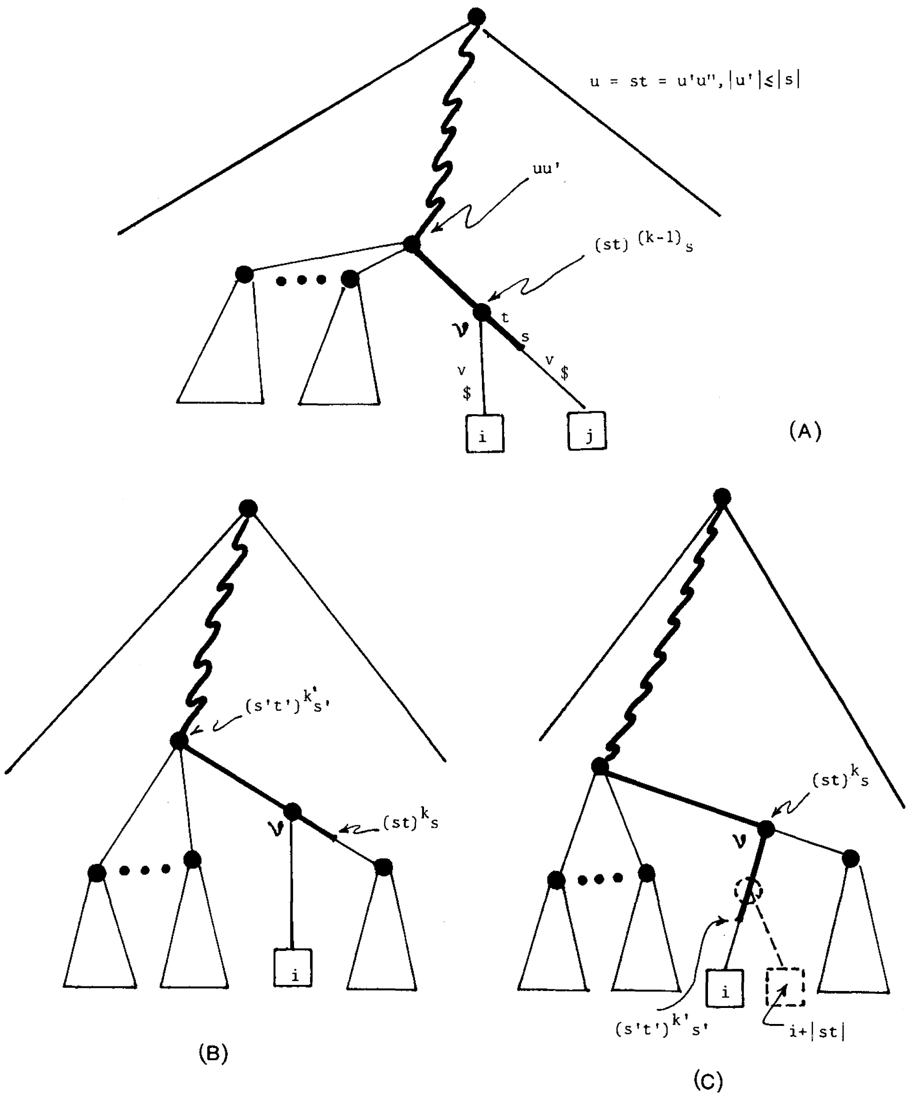

Figure 1. — Inserting the suffix $x(i, n)$ in T_x, a node might be created or found that is a detecting (*fig.* 1 *a*), a prefix (1 *b*) or an extension (1 *c*) node. Dashed lines in figure 1 *c* display the detecting node that will be issued at $j = 1 + |st|$ if $|\text{head}_j| > |\text{head}_i|$ (i. e., $|(s' t')^{k'} s'| - |(st)^k s| > |st|$).

also assumes good familiarity of the reader with the detailed version in reference [7].

1 *procedure S-Tree* (McCreight, 1976)
2 *begin* $T_{-1}=$ ROOT; $|W(\text{ROOT})|=0$
3 *for* $i=0$ *to* n *do* [* insert suf_i into T_{i-1} to produce T_i*]
4 *begin*
5 $\alpha=$ *locus* (head_i)
6 *if* $|W(\alpha)|>|\text{head}_i|$ *then* $\nu=$ *split* (Father (α), α)
[*Create a node to be the proper locus of head_i*]
else $\nu=\alpha$
8 *implant* (α, ν, i) [*Implant a terminal arc from ν to leaf labeled i*]
9 $\max(\nu)=i$
10 *end*
11 *end*

Line (9) is a special feature added for our purposes. The field *max* attached to each node is kept updated with the value of the leaf that was being inserted at the last time that the node was traversed. The algorithm derives its optimality from the ability to perform *locus* (head_i) $(i=0, 1, \ldots, n)$ in overall linear time (*see* ref. [7] for detailed constructions and proofs).

With reference to the listing of *S-tree* above, and in view of Lemma 4, all word detecting nodes, along with related RCSW's are readily recognized by the following simple procedure *squaresearch* inserted at the interstice between lines 7 and 8 in *S-tree* (the arguments to be passed to *squaresearch* are, in this order, α, ν, i and β, the contracted locus of head_i; for ease in cross-referencing, formal parameters are denoted by these same symbols).

procedure squaresearch (α, ν, i, β)
begin $p=i-\max(\alpha)$; $L=p+|W(\nu)|$; $m=L\ (\text{mod}\,.\,p)$
if $(p \leqq |W(\nu)|$ and $|W(\beta)| \leqq p+m)$ *then*
output $(i-p, i+p)$ [*$x(i-p, i+p)$ contains a RCSW*]
end

We remark that each call to squaresearch is executed in constant time. Thus, if we call *S′-tree* this upgrade of *S-tree* we can state the following:

THEOREM 5: *S′-tree recognizes all and only the RCSW's of a generic string x in $O(n)$ time and space.*

By further elaborating on *S-tree* we now set up still another version of it which shall be called *S″-tree*, that constructs $\tilde{T}_x$ and fully exploits Lemma 6. For simplicity, we shall think of *S″-tree* as obtained by *S′-tree* by substituting *squaresearch* with the more complex procedure *repsearch*. The task of *repsearch* embodies that of *squaresearch* and can be informally described as follows. First, we notice that one or two repetitions, namely, $R(i-p, p, p+|W(\nu)|)$ alone or along with $R(i, p, |W(\nu)|)$, may in fact be outputed at the site of a word detecting node, according to whether or not it results $|W(\nu)| \geqq 2p$. Moreover, auxiliary "p"-nodes of degree 1 can be easily created and marked soon after each RCSW detection, in care of *repsearch*. Based on the criterion of Lemma 6, it takes trivial, constant time checks at each iteration to spot a detecting, prefix or extension node and to mark appropriately "p"-nodes for each value of p. It should be remarked only that the recognition of a detecting node that is not a word detecting node always implies outputing two repetitions. The detection of RCR's that are the outcome of prefix nodes is also trivial. Less trivial is the management of an extension node: indeed, if ν is such a node then the value of the length L has to be appreciated before outputing $R(i, p, L)$. This entails some lookahead scanning of the symbols of tail_i, which might cause *S''-tree* to degenerate toward a quadratic worst case performance. To avoid such behavior we resort to the following:

LEMMA 7 [7]: *If* $head_{i-1}=aw$ *with* $a \in I$, *then* w *is a prefix of* $head_i$.

This lemma plays a crucial role in the constructions of T_x, where auxiliary links (*suffix links*) are always established from the proper locus of aw to that of w. Based on suffix links, the following function *s-ancestor* can be evaluated at each step i for the proper locus ν of head_i: *s*-ancestor (ν) is the node ν, if it exists, such that $W(\nu)=x\{i-1\}\ \text{head}_i$. The reader is urged to verify that the suffix tree construction in reference [7] can be amended so as to make at each iteration, *s*-ancestor available at the site of ν at no extra cost. We also attach a novel field scan(ν) with each extension node. We proceed to clarify the role of this field by showing how an extension "p"-node ν recognized by our second upgrade of *repsearch* at the i-th iteration of *S''-tree* is then handled (we assume that *s*-ancestor and p are global variables).

procedure repsearch (α, ν, *i*, β)
begin use Lemma 6 to test for ν to be a (word) detecting or a prefix node.
If ν is one such node *then* output the detected RCR (s) (and possibly RCSW) and insert a unary p-node if appropriate.
else if ν is an extension node, *then*
begin
if s-ancestor (ν) is defined *then* scan (ν) = scan (s-ancestor (ν) − 1)
else scan (ν) = 0
$L = W(\nu)$
case scan (ν) *of*
scan (ν) = 0: *begin while* (scan (ν) $\leqq p-1$) $\wedge$ ($x\{i+L\} = x\{i+L \,(\mathrm{mod}\,.\,p)\}$ *do*
begin scan (ν) = scan (ν) + 1; $L = L+1$ *end*
if scan (ν) $< p-1$ *then output* ($R(i, p, L)$)
end
scan (ν) $= p-2$: *begin* $L = L +$ scan (ν)
if $x\{i+L\} = x\{i+L\,(\mathrm{mod}\,.\,p)\}$ *then begin* $L = L+1$;
output $R(i, p, L)$ *end*
end
$0 <$ scan (ν) $< p-2$: *begin* $L = L +$ scan (ν); output $R(i, p, L)$ *end*
end
end

The *while* loop above will be referred to as the *look-ahead scanning*.

THEOREM 6: *For each RCSW u^2 and for each substring of x in the form $u^k u'$ ($k > 1$), S''-tree correctly detects the leftmost RCR in this form.*

Proof: We have already seen that, by using of Lemma 6, S''-*tree* identifies all the detecting, prefix and extension nodes. Let now $R(i, p, L)$ be the leftmost repetition in the form $(u)^k u'$. Simple inspection of *repsearch* shows that $R(i, p, L)$ is detected during step i whenever, by the end of this step, less than p matches are found. If on the other hand, it is scan (ν) $= p-1$ by the end of step i, then a detecting node will be issued at step $j = i+p$ (*see* also *fig*. 1 *c*) yielding both $R(i, p, L)$ and $R(j, p, L-p)$. □

We now prove the linearity of our strategy.

THEOREM 7: *S''-tree runs in $O(n)$ time and space.*

Proof: It will do to show that the work done the look ahead scanning through steps 0, 1, ..., n is bounded by n. We do this by exploiting the periodicity Lemma in conjunction with Lemma 7.

Let then scan_i be the work (i. e., the number of character comparisons) involved in step i ($i = 1, 2, \ldots, n$) and let also i_0 represent the first index value for which some scanning takes place. If $R(i_0, |u_0|, L_0)$ denotes the repetition that caused such lookahead scanning, we have that, by construction:

$$\text{scan}_{i_0} \leqq |u_0| i_0 - 1 + |u_0| + \text{scan}_0.$$

Moreover, if $\text{scan}^{(i)}$ denotes the sum of scan_k over all steps $k=0, 1, \ldots, i$, we also have that $\text{scan}_{i_0} = \text{scan}^{(i_0)}$. We now prove inductively that, if $\text{scan}^{(i_j)} < i_j + |u_{ij}| + \text{scan}_{ij}$, then it is also:

$$\text{scan}^{(i_j+1)} < i_{j+1} - 1 + |u_{i_{j+1}}| + \text{scan}_{i_{j+1}}.$$

The assertion is obviously true if:

$$i_{j+1} - 1 + |u_{i_{j+1}}| \leqq i_j - 1 + |u_{i_j}| + \text{scan}_{i_j}.$$

Assume then that:

$$i_{j+1} - 1 + |u_{i_{j+1}}| < i_j - 1 + |u_{i_j}| + \text{scan}_{i_j}.$$

Clearly, $|u_{i_{j+1}}| \neq |u_{i_j}|$, otherwise $u_{i_{j+1}}$ would be a cyclic permutation of u_{i_j} and no scanning would take place at this stage (*see* discussion of theorem 6).

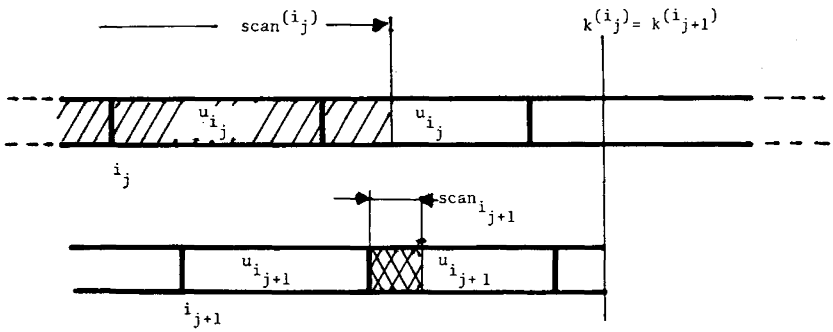

Figure 2. — Linearity of the look-ahead scanning.

With reference to figure 2 below, from the inequalities:

$$k^{(i_j)} - (i_j + |u_{i_j}| + \text{scan}_{i_j} - 1) \geqq |u_{i_j}|$$

and:

$$i_{j+1} > i_j,$$

we derive that $x(i_{j+1}, k^{(i_j)})$ has periods $|u_{i_j}|$ and $|u_{i_{j+1}}|$, whereas it is also:

$$k^{(i_j)} - i_{j+1} \geqq |u_{i_j}| + |u_{i_{j+1}}|,$$

whence, by the periodicity Lemma, u_{i_j} and $u_{i_{j+1}}$ cannot be primitive substrings of x at the same time, a contradiction. In conclusion, letting $i_{\max}$ denote the rightmost position in x where some look-ahead scanning takes place, it must be $i_{\max}-1+u_{i\max}+\mathrm{scan}_{i\max}<n$. The space is $O(n)$ in force of Lemma 5. □

ACKNOWLEDGEMENT

I am is indebted to M. Crochemore and D. Perrin for encouragement and discussions, and to the referee for helpful comments.

BIBLIOGRAPHIE

1. A. Thue, *Uber Die Gegenseitige Lage Gleicher Teile Gewisser Zeichenreichen*, Skr. Vid.-Kristiana I. Mat. Naturv. Klasse, Vol. 1, 1912, pp. 1-67.
2. M. Main and R. Lorentz, *An $O(n \log n)$ Algorithm for Finding Repetition in a String. T.R.* 79-056, Comp. Sc. Dept., Washington State Univ., Pullman, 1972.
3. M. Crochemore, *An Optimal Algorithm for Computing the Repetitions in a Word*, Information Processing Letters, Vol. 12, 1981, pp. 244-250.
4. A. Apostolico and F. P. Preparata, *Optimal Off-Line Detection of Repetitions in a String*, Theoretical Computer Science, Vol. 22, 1983, pp. 237-315.
5. A. Lentin and M. P. Schutzemberger, *A Combinatorial Problem in the Theory of Free Monoids*, in *Combinatorial Mathematics and its Applications*, University of North Carolina Press, N.C., 1969, pp. 128-144.
6. A. Apostolico and F. P. Preparata, *A Structure for the Statistics of All Substrings of a Textstring With or Without Overlap*. Proceedings of the 2nd World Conference on Mathematics at the Service of Man, Las Palmas (Canary Islands), 1982.
7. E. M. McCreight, *A Space Economical Suffix Tree Construction Algorithm*, J. of the ACM, Vol. 23, 1976, pp. 262-272.
8. R. C. Lyndon and M. P. Schutzemberger, *The Equation $a^M = b^N c^P$ in a Free Group*, Michigan Mathemat. Journal, Vol. 9, 1962, pp. 289-298.
9. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, MA, 1974.
10. P. Weiner, *Linear Pattern Matching Algorithms*, Proceedings of the 14th Annual Symposium on Switching and Automata Theory, 1973, pp. 1-11.
11. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, Addison-Wesley, MA, 1973.