Linear searching for a square in a word.

Bul. Euro. Assoc. Theor. Comput. Sci. 24:66-72, 1984

Max CROCHEMORE
Laboratoire d'Informatique
Université de Haute-Normandie
BP 67
76130 MONT-SAINT-AIGNAN
FRANCE

Abstract

Searching for a square in a word may be implemented in time proportional to the length of the word on a randomm access machine provided the alphabet is fixed.

Several methods have been developped to find a square in a word x[AP83, Cr81, ML83] . They share in common an $O(|x|.\log|x|)$ time complexity. In fact the three algorithms are able to produce all the repetitions inside the given word with the same complexity. On an infinite alphabet or if the alphabet is not known this complexity cannot be improved even when only one square is searched. We show here that the situation changes when the alphabet is fixed. In this case the search for a square can be implemented in linear time. Our algorithm makes use of a result on concatenation of squarefree words and the fact that suffixes of a word, on a fixed alphabet, may be lexicographically ordered in linear time [Mc76, We73] .

1 Definitions

The words that are considered are elements of the free monoid A^* generated by some finite alphabet A. The empty word is denoted by 1, A^+ is $A^*-\{1\}$ and $|x|$ means the length of a word x in A^*.

A word x of A^* is said to <u>contain a square</u> yy if $y \in A^+$ and x may be written x_1yyx_2 for some x_1 and x_2 in A^*. Otherwise x is said to be <u>square-free</u>. For example, words of length $\leqslant 1$ are square-free.

When $x = yz$, y is called a prefix of x and z a suffix of x. To each such decomposition (y,z) of x is associated its separator denoted by $sep(y,z)$: it is the greatest couple of words (v,w) which satisfies : $|v| < |y|$ if $w \neq 1$, vw is a prefix of yz and w is a prefix of z. Couples of words are ordered by the relation :

$$(v',w') \leqslant (v,w) \text{ iff } |w'| < |w| \text{ or } (|w'|=|w| \text{ and } |v'| \geqslant |v|).$$

When (v,w) is the separator of (y,z), w is then the longest prefix of z which appears at least twice in the word yw. The word v marks the first occurrence of w in yw.

Example 1. If a is a letter which does not occur in y then $sep(y,az) = (1,1)$. For any word $x \in A^*$, $sep(1,x) = sep(x,1) = (1,1)$.■

Example 2. On $A = \{a,b,c,d\}$, for the word

$x_0 = abcdbabdcadbcbdbabcabcb$, we have

$sep(abcdbabdcadbcbdbabc,abcb) = (1,abc)$.■

Based on the separators of the decompositions of a word $x \in A^*$ in prefix and suffix is defined the s-factorization of x which is a sequence $(u_1,u_2,.....,u_k)$ of non-empty words such that

$$x = u_1u_2....u_k$$

and which is built iteratively as follows : if $u_1,...,u_{i-1}$ are already built and $x = u_1...u_{i-1}az$ for $a \in A$ and $z \in A^*$ and if $sep(u_1...u_{i-1}, az) = (v,w)$, then u_i is a or w whether w is empty or not.

The s-factorization of a word is unique and the empty sequence is associated to the empty word. We may note that u_1 the first term of the s-factorization of a non-empty word is its first letter.

Example 2 (continued). The s-factorization of x_0 is

$(a,b,c,d,b,ab,d,c,a,db,c,bd,bab,ca,bcb)$.■

We finally introduce two properties on couples of square-free words which deals with squares in their concatenation.

When $u,u' \in A^*$, we say that (u,u') satisfies the property right, denoted by $r(u,u')$ if u and u' are square-free and

$\exists\, u_1,u_2,u_3,u_4 \in A^*$ $u_2u_3 \neq 1$, $u = u_1u_2$ and $u' = u_3u_2u_3u_4$.

The property left, denoted by l(u,u'), is true for (u,u') exactely when r(ū',ū) is true (ū and ū' are the minor images of u and u' respectively).

Then, r(u,u') = true means that u and u' are squarefree but their concatenation uu' contains a square centered at the right of u.

Example 2 (continued)

r(abcdbabdcadbcbdbab,cabcb) = true

with u_1 = abcdbabdcadbcbdb, u_2=ab, u_3=c, u_4=b. The word x_0 contains the square abcabc.■

2 Main Theorem

Separators alone are not sufficient to caracterize words which contain a square ; nevertheless they provide a sufficient condition which is given in the following lemma. With the help of s-factorizations we are able to give an 'if and only if' condition for words that contain a square. The result in the next theorem is the base of our linear test for squarefreeness of words.

Lemma. Let $a_1,\ldots,a_n$ be letters of A. If the word $a_1\ldots a_n$ satisfies

(1) $\exists\, i \in (1,2,\ldots,n)$ $\exists\, v,w \in A^*$ $(v,w) = sep(a_1\ldots a_{i-1}, a_i\ldots a_n)$, $w \neq 1$ and $a_1\ldots a_{i-1}$ prefix of vw

then it contains a square.

Proof : $a_1\ldots a_n$ contains two occurrences of w which overlap or are consecutive and then $a_1\ldots a_n$ is not square-free.■

Theorem. Let $a_1,\ldots,a_n$ be letters of A and $(u_1,\ldots,u_k)$ be the s-factorization of $a_1\ldots a_n$. The word $a_1\ldots a_n$ contains a square if and only if it satisfies (1) or

(2) $\exists\, j \in (1,\ldots,k-1)$ $l(u_i,u_{i+1})$ or $r(u_i,u_{i+1})$ or $r(u_1\ldots u_{i-1}, u_iu_{i+1})$

Proof : We have only to prove that if $a_1\ldots a_n$ contains a square and does not satisfy(1), it satisfies (2).

Let x_1yy be the shortest prefix of $a_1...a_n$ which contains a square ($x_1 \in A^*$, $y \in A^+$). Let i be the smallest integer such that x_1yy is prefixe of $u_1u_2....u_iu_{i+1}$. Note that $i \geqslant 1$, since $|u_1| = 1$ and then u_1 is square-free.

The definition of i and the fact that u_i's are non empty words ensure that $u_1...u_i$ is square-free and is a proper prefix of x_1yy.

From this, we deduce that $u_1...u_{i-1}$ (which is empty if i=1) is a proper prefix of x_1y : the contrary wouldcontradict the definition of u_i and particulary the maximality of its length.

It then follows that the second occurrence of y is a factor (neither prefix nor suffix) of $u_i\ u_{i+1}$. We conclude, noting that $l(u_i,u_{i+1}) = r(u_i,u_{i+1})$ = false implies u_iu_{i+1} is squarefree and then $d(u_1...u_{i-1}, u_iu_{i+1})$ is true.□

Example 2 (continued)
The word x_0 does not satisfy the condition (1) of theorem but (2) holds.□

3 Square-freeness test

Following the conditions in the previous theorem, we get a square-freeness test that we write as a function named SQUARE which value is true exactely when the input word contains a square.

```
function SQUARE (a_1...a_n) ;
    i ← 1 ;
    while  i ⩽ n do
           (v,w) ← sep(a_1...a_{i-1},a_i...a_n) ;
           if w = 1  then u' ← a_i  ; j ← i ; i ← i+1
           else u ← u' ; u' ← w ;
                if (|vw| ⩾ i-1 or l(u,u') or r(u,u') or r(a_1..a_{j-1},uu'))
                then return (true)
                end if ;
                j ← i ; i ← i+|w|
           end if
    end while
```

return (false)
end function

The function SQUARE follows the theorem in 2 and takes into account that fact : if $a_1 \ldots a_{i-1}$ is square-free and $sep(a_1 \ldots a_{i-1}, a_i \ldots a_n) = (1,1)$, the letter a_i does not appear in $a_1 \ldots a_{i-1}$ and then $a_1 \ldots a_i$ is also square-free.

4 Time complexity

The time complexity of the function SQUARE depends on the time needed to compute the values of functions sep, l and r.

On any finite alphabet, it is possible to find the separators of all the decompositions (y,z) of a word x(=yz) in linear time adapting the construction of the suffix tree of x Mc 76 . The suffix tree may be precomputed, or built during the execution of the function SQUARE. In the latter case the algorithme becomes 'almost on-line '.

It remains to see that all the evaluations of the functions l and r is also linear in the length of the input word.The Morris and Pratt's algorithm [KMP 77] includes the computation of a function called f which is useful for this purpose.

For any word u in A^+, f(u) is defined to be the longest word distinct from u which is both a prefix and a suffix of u. Futhermore, the computations of the f(u)'s, for the non empty prefixes u of a word x, may be done in time proportionnal to |x|.

The next proposition gives a base to built algorithms which realize the functions l and r.

Proposition. Let $u, u' \in A^*$ which satisfy r(u,u'), and let psp be the shortest prefix of u' such that s is a suffix of u ($p \in A^*$, $s \in A^+$). Then

i- p = f(psp)

ii- if p'vs'p' is a prefix of u'($p' \in A^*$, v, $s' \in A^+$) such that |psp|<|p'vs'p'| ans s' is the longest common suffix of both u and p'vs', then |ps|<max(|p'v|,|p'vs'|/2).

To evaluate d(u,u') when u and u' are square-free we proceed as follows : reading u' from left to right f is first computed ; then from right to left for a prefix psp of u' such that p = f(psp) it is checked if s is a suffix of u. The total number of comparisons between letters during these tries is bounded by 2|u'| as a consequence of part ii of the proposition which allows to make jumps.

Corollary. When u and u' are square-free words in A*, r(u,u') (resp. l(u,u')) may be evaluated in time proportionnal to |u'| (resp. |u|).

Turning back to the total cost of calls to functions l and r in the function SQUARE, the above corollary gives a complexity proportionnal to $\sum_{i=1}^{k-1} |u_i u_{i+1}|$ if $(u_1,\ldots,u_k)$ is the s-factorization of the input word $a_1 \ldots a_n$, and this is majorate by 2n.

Theorem. On any finite alphabet, searching a word for a square may be realized in time linear in the length of the input word.

5 Conclusion

The above algorithm gives a positive answer to a question of Main & Lorentz which were the first to publish an 0(|x|log|x|) algorithm to test whether a word x is square-free or not. On arbitrary (or infinite) alphabets our algorithm is still useful : but, in that case, the implementation could be different using hash tables or search trees to deal with the distinct letters of the word. The worst case complexity becomes 0(|x|log m) where m is the number of distinct letters of x.

The method used here to find a square in a word cannot be directly applied to cubes or higher power but we conjecture that recognition of these other patterns may also be realized in linear time.

6 References

AP 83 **A. Apostolico & F.P. Préparata.**
Optimal off-line detection of repetitions in a string, Theor-Compt. Sci. 22 (1983) 297-315.

Cr81 **M. Crochemore**
An optimal algorithm for computing the repetitions in a word, Information Processing Letters, 12 (1981) 244-250.

KMP77 **D.E. Knuth, J.H. Morris & V.R. Pratt.**
Fast pattern-matching in strings, SIAM J. Comput. 6 (1977) 323-350.

ML83 **M. Main & R. Lorentz.**
An O(nlogn) algorithm for finding all repetitions in a string, J. Of algorithms (1983) to appear.

Mc76 **E.M. Mc Creight.**
A space-economical suffix tree construction algorithm, J. of the ACM, 23, 2(1976) 262-272.

We73 **P. Weiner.**
Linear pattern matching algorithms, in : (proceedings of the 14th annual symposium on switching and automata theory, 1973) 1-11.

```
@ARTICLE{MCrochemore84eatcs,
  author =       {Maxime Crochemore},
  title =        {Linear searching for a square in a word},
  journal =      {Bul. Euro. Assoc. Theor. Comput. Sci.},
  volume =       {24},
  year =         {1984},
  pages =        {66--72},
  note =         {Presented at ICALP'84.
                  Abstract in \textit{Automata, Languages and Programming},
                  LNCS 172, pp. 137. Springer, Berlin, 1984},
}

See also:

@ARTICLE{Cro83cras,
  author =       {Maxime Crochemore},
  title =        {Recherche lin{\'e}aire d'un carr{\'e} dans un mot},
  journal =      {C. R. Acad. Sc. Paris S\'er. I Math.},
  volume =       {296},
  number =       {18},
  year =         {1983},
  pages =        {781--784},
}
```