DETECTING LEFTMOST PERIODICITIES

Michael G. Main

CU-CS-357-87 January 1987
DETECTING LEFTMOST PERIODICITIES  
(January 1987)

Michael G. Main  
Department of Computer Science  
University of Colorado  
Boulder, CO 80309  USA

1. Introduction

Periodicities are nonempty strings of the form $p^n q$ with $n \geq 2$ and $q$ a substring of $p$. This note presents a new algorithm to find all leftmost occurrences of periodicities within a string. For a fixed alphabet, the worst-case time is linear in the length of the string.

The study of periodicities dates from the pioneering work of Axel Thue at the beginning of this century [11,12,5]. More recently, there has been a surge of interest in periodicities among researchers in formal language theory [2]. In 1978, an $O(n \log n)$ algorithm was presented to determine whether a string contains a periodicity of the form $pp$ (a "square") [6]. Later, this algorithm was improved to find all squares in the same asymptotic time bound. (In fact, this was done in at least three independent and simple ways [1,3,7].)

An algorithm was also presented by A.O. Slisenko [10], finding all periodicities in linear time. But, Slisenko’s algorithm was a difficult 100-page presentation, so research continued to either simplify Slisenko’s algorithm or find alternative methods of detecting periodicities. Most recently, there were two linear algorithms to determine whether a string contains a square [3,8]. Both of these papers left several open problems, which are addressed by the new algorithm presented here.

The principal open problem solved here is this: Let $x$ be a string. A periodicity (within $x$) is a substring of the form $p^n q$, with $n \geq 2$ and $q$ a prefix of $p$. The length of $p$ is called the period-length of the periodicity. If a periodicity $p^n q$ occurs several times within a string $x$, then the first time it occurs is called the leftmost occurrence. This note presents an algorithm to find all leftmost periodicities within a string $x$, in time proportional to the length of $x$, provided that the alphabet is a constant size.

Notation: The length of a string $x$ is denoted by $|x|$, and the $i^{th}$ character of $x$ is denoted by $x[i]$. The substring starting at character $i$ and ending at character $j$ is written $x[i..j]$. 
2. Main Theorem

The algorithm uses a decomposition of a string called the s-factorization, also used by Crochemore in his most recent algorithm [3]. This is a decomposition of a string $x$ into the concatenation of several strings $x = u_1 \cdots u_k$, defined recursively as follows: Suppose $u_1 \cdots u_{h-1}$ have already been defined, so that $u_1 \cdots u_{h-1}$ is a proper prefix of $x$. Now we want to define $u_h$. Here are the rules:

1. If the next character of $x$ (after $u_{h-1}$) has not yet appeared in $x$, then $u_h$ consists of this single character.
2. Otherwise, $u_h$ is the longest string such that $u_1 \cdots u_h$ is a prefix of $x$ and $u_h$ is a non-suffix substring of $u_1 \cdots u_h$.

The following theorem gives two properties of leftmost periodicities, in terms of the s-factorization of a string:

**Theorem 1:** Let $x$ be a string with s-factorization $x = u_1 \cdots u_k$ and let $r$ be a periodicity of $x$, with the leftmost occurrence of $r$ at $x[i]..x[j]$, and with $x[j]$ occurring within $u_h$. Then

1. $x[i]$ occurs before $u_h$, and
2. $|r| \leq 2|u_{h-1}u_h|.$

**Proof:** (1) Suppose condition 1 does not hold, so that $r$ is entirely within $u_h$. Then $u_h$ has at least two characters, and by the definition of the s-factorization, $u_h$ must occur as a non-suffix substring of $u_1 \cdots u_h$. But this means that $x[i]..x[j]$ is not the leftmost occurrence of $r$. By this contradiction, condition 1 must hold.

(2) Let $r=p^*q$, and suppose condition 2 does not hold. Then at least half of $x[i]..x[j]$ is before $u_{h-1}$. This means that at least one entire occurrence of $p$ has occurred at the beginning of $x[i]..x[j]$ before the start of $u_{h-1}$. Moreover, when we start $u_{h-1}$, we are in the middle of some later occurrence of $p$. This means that the substring beginning at the first character of $u_{h-1}$ and continuing until $x[j]$ also occurs earlier in $x$. But, since $u_{h-1}$ ends before $x[j]$, this violates the maximality condition in the definition of an s-factorization. Therefore, condition 2 must hold.
3. The Algorithm

Here is the algorithm to find all leftmost periodicities within a string $x$:

1. Compute the $s$-factorization $x = u_1 \cdots u_k$ of the input string $x$.

2. For each $h$ ($2 \leq h \leq k$), let $t_h$ be the substring of length $2|u_{h-1}| + |u_h|$ which immediately precedes $u_h$ in $x$ (or to the beginning of $x$ if there are not enough characters before $u_h$).

3. for $h := 2$ to $k$ do

   begin

   (3.1) Find all periodicities which start in $t_h$ and end in $u_h$.

   end.

From the theorem of the previous section, any leftmost occurrence of a periodicity $r$ (within $x$) which ends within $u_h$ will be found by step 3.1 of the algorithm.

For a finite alphabet, it is possible to compute the $s$-factorization of a string $x$ in $O(|x|)$ time, by adapting McCreight's suffix tree construction [9]. (This same adaptation is used by Crochemore [3].) Therefore, the first two steps of the algorithm require linear time (for a fixed alphabet). If the alphabet is infinite, then the $s$-factorization requires $O(|x| \log |x|)$ time, and the first two steps also require $O(|x| \log |x|)$ time.

Step 3.1 may be computed in time $O(|t_h u_h|)$, using a modification of an algorithm which finds all new squares that appear when two strings are concatenated. (This modification is given in the next section.) Since $\sum_{h=2}^{k} |t_h u_h| < 4|x|$, the total time spent in Step 3 is $O(|x|)$.

Therefore, the worst-case time of the entire algorithm is linear in the length of $x$ (for a fixed alphabet) or $O(|x| \log |x|)$ for an arbitrary alphabet.

4. Finding New Periodicities

Let $t$ and $u$ be two strings, with $|t| = m$ and $|u| = n$. This section shows how to find all new periodicities that are formed in the concatenation $tu$. (These are periodicities with the first character in $t$ and the last character in $u$.) The algorithm requires $O(m+n)$ time, and is a modification of an earlier algorithm which finds new squares [7].
The algorithm has two parts. The first part finds all new periodicities which have at least one full period in the string $u$. These are called right periodicities. The second part finds all new periodicities which have at least one full period in the string $t$ (left periodicities). Here we present only the first part, since the second part is symmetric. This first part makes use of two functions $LP$ and $LS$, defined as follows:

For $(2 \leq i \leq n+1)$: $LP(i)$ is the length of the longest prefix of $u$ which is also a prefix of $u[i]..u[n]$. ($LP[n+1]$ is defined as zero.)

For $(1 \leq i \leq n)$: $LS(i)$ is the length of the longest suffix of $t$ which is also a suffix of $tv$, where $v$ is $u[1]..u[i]$.

The following theorem characterizes new right periodicities which are formed in the concatenation of $tu$:

**Theorem 2:** Let $j$ ($1 \leq j \leq n$) be an integer. The new right periodicities (in $tu$) with period-length $j$ are precisely those substrings of $tu$ which:

1. Have length $2j$ or more, and
2. Begin at or before $t[m]$ and end at or after $u[j]$, and
3. Begin at or after $t[m-LS(j)+1]$ and end at or before $u[j+LP(j+1)]$.

**Proof:** The first two conditions are clearly necessary, so let $r$ be a substring of $tu$, which meets these two conditions, and let $i = |r|$. We will show that the remaining condition is necessary and sufficient for $r$ to be a periodicity with period-length $j$.

Let $a$ be the number of characters of $r$ in $t$, and let $b$ be the number of characters of $r$ in $u[j+1]..u[n]$. (So that $i = a+b+j$.) For $r$ to be a periodicity with period-length $j$, it is necessary and sufficient for the first $i-j$ characters of $r$ to match the last $i-j$ characters of $r$. Equivalently,

(A) $r[1]..r[a]$ matches $r[1+j]..r[a+j]$, and

(B) $r[a+1]..r[a+b]$ matches $r[a+j+1]..r[i]$. 
Condition (A) is equivalent to requiring $r[1+j] \ldots r[a+j]$ to be a suffix of $t$. Since $r[a+j]$ occurs at position $u[j]$, this is equivalent to requiring $r$ to begin at or after $t[m-LS(j)+1]$. Similarly, Condition (B) is equivalent to requiring $r$ to end at or before $u[j+LP(j+1)]$. Thus, the Conditions (A) and (B) together are equivalent to (3) in the statement of the theorem.

Theorem 2 is the basis of the following algorithm to find all new right periodicities in $tu$:

1. Calculate the values of $LP(2)$ through $LP(n+1)$, and the values of $LS(1)$ through $LS(n)$.

2. for $j := 1$ to $n$ do

   begin
   The new right periodicities (with period $j$) are all substrings of length $2j$ or
   more beginning in the range $t[m-LS(j)+1]$ through $t[m]$, and ending in the
   range $u[j]$ through $u[j+LP(j+1)]$.
   end.

The calculations of $LP$ and $LS$ in Step 1 require $O(m+n)$ time, using a variation of the Knuth-Morris-Pratt pattern matching algorithm [7, section 2]. The body of the loop in Step 2 requires constant time for each $j$, so the entire loop is $O(n)$. Therefore, the entire algorithm takes time proportional to $|uv|$.

6. Notes

The algorithm of Section 4 finds the leftmost occurrence of every periodicity within a string in linear time (for a fixed alphabet). The algorithm may also find some non-leftmost occurrences of periodicities (those that span boundaries in the s-factorization). This information can be used to solve the problem of determining whether a string has a periodicity of the form $p^n$ for different values of $n \geq 2$, solving a problem of Crochemore [3,4].

A further modification may allow the algorithm to find all periodicities in a simple manner. This seems likely since the periodicities that are not found are entirely within some $u_h$ in the s-factorization, and each such $u_h$ occurs previously in the string.
References

(1) A. Apostolico and F.P. Preparata, Optimal off-line detection of repetitions in a string, TCS 22 (1983) 297-315.

(2) J. Berstel and C. Reutenauer, Square-free words and Idempotent semigroups, in Combinatorics on Words (Lothaire, Ed.), Addison-Wesley, Reading, MA., Chapter 2.


(11) A. Thue, Uber unendliche Zeichenreihen, Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl. (Kristiania) (1906), Nr. 7, 1-22.

(12) A. Thue, Uber die gegenseitige Lage gleicher Teile gewisser Zeichenreihen, Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl. (Kristiania) (1912), Nr. 1, 1-67.