Information and Computation 240 (2015) 74–89

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Detecting regularities on grammar-compressed strings

Tomohiro I [a,b,*], Wataru Matsubara [c], Kouji Shimohira [a], Shunsuke Inenaga [a], Hideo Bannai [a], Masayuki Takeda [a], Kazuyuki Narisawa [c], Ayumi Shinohara [c]

[a] Department of Informatics, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka 819-0395, Japan
[b] Japan Society for the Promotion of Science (JSPS), Japan
[c] Graduate School of Information Sciences, Tohoku University, 6-3-09 Aoba, Aramaki, Aoba-ku, Sendai, Miyagi 980-8579, Japan

ARTICLE INFO

Article history:
Received 1 October 2013
Available online 7 October 2014

Keywords:
Straight-line programs (SLPs)
Runs
Squares
Gapped palindromes
Compressed string processing algorithms

ABSTRACT

We address the problems of detecting and counting various forms of regularities in a string represented as a straight-line program (SLP) which is essentially a context free grammar in the Chomsky normal form. Given an SLP of size n that represents a string s of length N, our algorithm computes all runs and squares in s in $O(n^3h)$ time and $O(n^2)$ space, where h is the height of the derivation tree of the SLP. We also show an algorithm to compute all gapped-palindromes in $O(n^3h + gnh \log N)$ time and $O(n^2)$ space, where g is the length of the gap. As one of the main components of the above solution, we propose a new technique called approximate doubling which seems to be a useful tool for a wide range of algorithms on SLPs. Indeed, we show that the technique can be used to compute the periods and covers of the string in $O(n^2h)$ time and $O(nh(n + \log^2 N))$ time, respectively.

1. Introduction

Finding regularities such as squares, runs, and palindromes in strings, is a fundamental and important problem in stringology with various applications, and many efficient algorithms have been proposed (e.g., [14,6,1,8,15,2,11,10]). See also [5] for a survey.

In this paper, we consider the problem of detecting regularities in a string s of length N that is given in a compressed form, namely, as a straight-line program (SLP), which is essentially a context free grammar in the Chomsky normal form that derives only s. Our model of computation is the word RAM: We shall assume that the computer word size is at least $\lceil \log_2 N \rceil$, and hence, standard operations on values representing lengths and positions of string s can be performed in constant time. Space complexities will be determined by the number of computer words (not bits).

Given an SLP whose size is n and the height of its derivation tree is h, Bannai et al. [3] showed how to test whether the string s is square-free or not, in $O(n^3h \log N)$ time and $O(n^2)$ space. Independently, Khvorost [9] presented an algorithm for computing a compact representation of all squares in s in $O(n^3h \log^2 N)$ time and $O(n^2)$ space. Matsubara et al. [16] showed that a compact representation of all maximal palindromes occurring in the string s can be computed in $O(n^3h)$ time and $O(n^2)$ space. Note that the length N of the decompressed string s can be as large as $O(2^n)$ in the worst case. Therefore, in such cases these algorithms are more efficient than *any* algorithm that works on uncompressed strings.

* Corresponding author at: Department of Computer Science, Technische Universität Dortmund, 44221 Dortmund, Germany.
E-mail addresses: tomohiro.i@cs.tu-dortmund.ge (T. I), tomohiro.i@inf.kyushu-u.ac.jp (T. I), inenaga@inf.kyushu-u.ac.jp (S. Inenaga), bannai@inf.kyushu-u.ac.jp (H. Bannai), takeda@inf.kyushu-u.ac.jp (M. Takeda), narisawa@ecei.tohoku.ac.jp (K. Narisawa), ayumi@ecei.tohoku.ac.jp (A. Shinohara).

http://dx.doi.org/10.1016/j.ic.2014.09.009
0890-5401/

In this paper we present the following extension and improvements to the above work, namely,

1. an $O(n^3h)$-time $O(n^2)$-space algorithm for computing a compact representation of squares and runs;
2. an $O(n^3h + gnh \log N)$-time $O(n^2)$-space algorithm for computing a compact representation of palindromes with a gap (spacer) of length g.

We remark that our algorithms can easily be extended to count the number of squares, runs, and gapped palindromes in the same time and space complexities.

Note that Result 1 improves on the work by Khvorost [9] which requires $O(n^3h \log^2 N)$ time and $O(n^2)$ space. The key to the improvement is our new technique of Section 3.3 called *approximate doubling*, which we believe is of independent interest. An integer p $(1 \leq p < |s|)$ is called a period of string s if $s = x^k x'$ such that x is of length p and x' is a proper prefix of p. Also, an integer c $(1 \leq c < |s|)$ is called a cover of string s if every position of s is covered by an occurrence of the prefix y of s of length c. Using the approximate doubling technique, one can improve the time complexity of the algorithms of Lifshits [12] to compute the periods and covers of a string given as an SLP from $O(n^2h \log N)$ and $O(n^2h \log^2 N)$ to $O(n^2h)$ and $O(nh(n + \log^2 N))$, respectively.

If we allow no gaps in palindromes (i.e., if we set $g = 0$), then Result 2 implies that we can compute a compact representation of all maximal palindromes in $O(n^3h)$ time and $O(n^2)$ space. Hence, Result 2 can be seen as a generalization of the work by Matsubara et al. [16] with the same efficiency.

2. Preliminaries

2.1. Strings

Let Σ be the *alphabet*. An element of Σ^* is called a *string*. For string $s = xyz$, x is called a *prefix*, y is called a *substring*, and z is called a *suffix* of s, respectively. The length of string s is denoted by $|s|$. The empty string ε is a string of length 0, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. For $1 \leq i \leq |s|$, $s[i]$ denotes the i-th character of s. For $1 \leq i \leq j \leq |s|$, $s[i..j]$ denotes the substring of s that begins at position i and ends at position j. For any string s, let s^R denote the reversed string of s, that is, $s^R = s[|s|] \cdots s[2]s[1]$. For any strings s and u, let $lcp(s, u)$ (resp. $lcs(s, u)$) denote the length of the longest common prefix (resp. suffix) of s and u. For a pair of integers $b \leq e$, $[b, e] = \{b, b + 1, \ldots, e\}$ is called an *interval*.

We say that string s has a *period* c $(0 < c \leq |s|)$ if $s[i] = s[i + c]$ for any $1 \leq i \leq |s| - c$. For a period c of s, we denote $s = u^q$, where u is the prefix of s of length c and $q = \frac{|s|}{c}$. If $q \geq 2$, $s = u^q$ is called a *repetition* with root u and period $|u|$. Also, we say that s is *primitive* if there is no string u and integer $k > 1$ such that $s = u^k$. If s is primitive, then s^2 is called a *square*.

We denote a repetition in a string s by a triple $\langle b, e, c\rangle$ such that $s[b..e]$ is a repetition with period c. A repetition $\langle b, e, c\rangle$ in s is called a *run* (or *maximal periodicity* in [13]) if c is the smallest period of $s[b..e]$ and the substring cannot be extended to the left nor to the right with the same period, namely neither $s[b-1..e]$ nor $s[b..e+1]$ has period c. Since the maximality is essential for runs, in this paper, we refer $[b-1, e+1]$ as the *margined-interval* of a run $\langle b, e, c\rangle$ in s, which is the minimal interval where we can see the maximality of the run. Note that for any run $\langle b, e, c\rangle$ in s, every substring of length $2c$ in $s[b..e]$ is a square. Let $Runs(s)$ denote the set of all runs in s.

A string s is said to be a *palindrome* if $s = s^R$. A string s is said to be a *gapped palindrome* if $s = xux^R$ for some strings $u \in \Sigma^*$ and $x \in \Sigma^+$. Note that u may or may not be a palindrome. The prefix x (resp. suffix x^R) of xux^R is called the *left arm* (resp. *right arm*) of gapped palindrome xux^R. If $|u| = g$, then xux^R is said to be a *g-gapped palindrome*. We denote a *maximal g-gapped palindrome* in a string s by a pair $\langle b, e\rangle_g$ such that $s[b..e]$ is a g-gapped palindrome and $s[b-1..e+1]$ is not. Similarly to the case of runs, $[b-1, e+1]$ is said to be the *margined-interval* of the maximal g-gapped palindrome. Let $gPals(s)$ denote the set of all maximal g-gapped palindromes in s.

Given a text string $s \in \Sigma^+$ and a pattern string $p \in \Sigma^+$, we say that p *occurs* at position i $(1 \leq i \leq |s| - |p| + 1)$ iff $s[i..i + |p| - 1] = p$. Let $Occ(s, p)$ denote the set of positions where p occurs in s.

Lemma 1. *(See [17].) For any strings $s, p \in \Sigma^+$ and any interval $[b, e]$ with $1 \leq b \leq e \leq b + |p|$, $Occ(s, p) \cap [b, e]$ forms a single arithmetic progression if $Occ(s, p) \cap [b, e] \neq \emptyset$.*

We refer to an arithmetic progression representing $Occ(s, p) \cap [b, e]$ with $1 \leq b \leq e \leq b + |p|$ as a *serial-ap* of p.

2.2. Straight-line programs

A *straight-line program* (*SLP*) $\mathcal{S}$ of size n is a set of productions $\mathcal{S} = \{X_i \to expr_i\}_{i=1}^n$, where each X_i is a distinct variable and each $expr_i$ is either $expr_i = X_\ell X_r$ $(1 \leq \ell, r < i)$, or $expr_i = a$ for some $a \in \Sigma$. Note that X_n derives a single string and, therefore, we view the SLP as a compressed representation of the string s that is derived from the variable X_n. Recall that the length N of the string s is $O(2^n)$ and in some instances $N = \Theta(2^n)$. However, it is always the case that $n \geq \log N$. For any variable X_i, let $val(X_i)$ denote the string that is derived from variable X_i. Therefore, $val(X_n) = s$. When it is not confusing, we identify X_i with the string represented by X_i.