Theoretical Computer Science 544 (2014) 60–73

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Efficient counting of square substrings in a tree ☆

Tomasz Kociumaka [a,1], Jakub Pachocki [b,2], Jakub Radoszewski [a,3,*], Wojciech Rytter [a,c], Tomasz Waleń [a]

[a] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
[b] Carnegie Mellon University, United States
[c] Faculty of Mathematics and Computer Science, Copernicus University, Toruń, Poland

ARTICLE INFO

Article history:
Received 15 May 2013
Received in revised form 19 February 2014
Accepted 12 April 2014

Keywords:
Tree
Square in string
Pattern matching

ABSTRACT

We give an algorithm which in $O(n \log^2 n)$ time counts all distinct squares in a labeled tree. There are two main obstacles to overcome. The first one is that the number of distinct squares in a tree is $\Omega(n^{4/3})$ (see Crochemore et al., 2012 [7]), which differs substantially from the case of classical strings for which there are only linearly many distinct squares. We overcome this obstacle by using a compact representation of all squares (based on maximal cyclic shifts) which requires only $O(n \log n)$ space. The second obstacle is lack of adequate algorithmic tools for labeled trees, consequently we design several novel tools, this is the most complex part of the paper. In particular we extend to trees Imre Simon's compact representations of the failure table in pattern matching machines.

1. Introduction

Repetitions play an important role in combinatorics on words with particular applications in pattern matching, text compression, computational biology etc. For a survey on known results related to repetitions in words and their applications see [2]. The basic type of a repetition are squares: strings of the form ww. Here we consider square substrings corresponding to simple paths in labeled unrooted trees. Squares in trees and graphs have already been considered e.g. in [3,4]. There have also been results on squares in partial words [5] and squares in the context of games [6].

Recently it has been shown that a tree with n nodes can contain $\Theta(n^{4/3})$ distinct squares, see [7], while the number of distinct squares in a string of length n does not exceed $2n - \Theta(\log n)$, as shown in [8–10]. This paper can be viewed as an algorithmic continuation of [7].

Enumerating squares in ordinary strings is already a difficult problem, despite the linear upper bound on their number. Complex $O(n)$ time solutions to this problem using suffix trees [11] and runs [12] are known. Two notions that we introduce in this paper (semiruns and packages of cyclically equivalent squares) are in a sense an extension of the techniques used in [12].

☆ A preliminary version of this paper appeared at ISAAC 2012 [1].

* Corresponding author.
E-mail addresses: kociumaka@mimuw.edu.pl (T. Kociumaka), pachocki@cs.cmu.edu (J. Pachocki), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), walen@mimuw.edu.pl (T. Waleń).

[1] Supported by Polish budget funds for science in 2013–2017 as a research project under the 'Diamond Grant' program (Ministry of Science and Higher Education, Republic of Poland, grant number DI2012 01794).
[2] This work was done while at University of Warsaw.
[3] The author receives financial support of Foundation For Polish Science (START programme number 94.2013).

http://dx.doi.org/10.1016/j.tcs.2014.04.015
0304-3975/

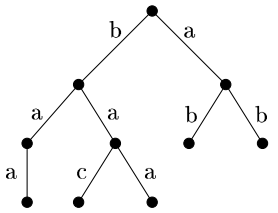

Fig. 1. This tree contains the following squares: *aa*, *aaaa*, *abab*, *baba*, *bb*. We have here $|\mathrm{sq}(T)| = 5$. Note that the squares *abab* and *baba* correspond to the same path, one is read from left to right and the other is read from right to left.

Assume we have a tree T with n nodes whose edges are labeled with symbols from an integer alphabet Σ. We assume that Σ is polynomially bounded in terms of n, i.e. $\Sigma \subseteq \{0, \ldots, n^C\}$ for some positive integer constant C. If u and v are two nodes of T, then let $val(u, v)$ denote the sequence of labels of edges on the path from u to v. We call $val(u, v)$ a *substring* of T. (Note that a substring is a string, not a path.) Denote by $\mathrm{sq}(T)$ the set of different square substrings in T. The main problem we consider is as follows:

Input: A labeled tree T.

Output: $|\mathrm{sq}(T)|$, the number of distinct square substrings in T.

Example: For the tree in Fig. 1 we have $|\mathrm{sq}(T)| = 5$.

Our result: We compute $|\mathrm{sq}(T)|$ in $O(n \log^2 n)$ time.

In the same time complexity we provide a compact representation of the set of all distinct squares in T. The representation consists of $O(n \log n)$ packages, each package stores a pair of nodes x, y that represents the maximum cyclic rotation $u = val(x, y)$ of a square half and a cyclic interval I such that for all $i \in I$ the cyclic rotation of u by i letters is a square half (see Fig. 4 for an example).

The structure of the algorithm:

1. We reduce the problem to finding a compact representation of all squares *anchored* at a given node r of the tree. Afterwards we consider trees rooted at specific node r.
2. For a given root r we compute a set of paths, called *semiruns*, which contain all squares anchored at r.
3. We reorganize the data related to semiruns to get an unambiguous representation of squares in terms of cyclic rotations. This unambiguity allows efficient counting of squares.

The hardest and the most interesting part of the paper is efficient computation of several basic tables. We structure the paper in such a way that this part is moved after the presentation of the main algorithm (which is described in Section 4). Before that we present combinatorial tools related to strings and labeled trees which are the base of the algorithm design.

In the last section we present a linear time algorithm for counting squares in a special family of trees called combs. The trees from this family turn out to maximize the asymptotic number of square substrings [7].

2. Combinatorial tools for squares in trees

Centroid decomposition The centroid decomposition enables to consider paths going through the root in rooted trees instead of arbitrary paths in an unrooted tree. Let T be an unrooted tree of n nodes. Let $T_1, T_2, \ldots, T_k$ be the connected components obtained after removing a node r from T. The node r is called a *centroid* of T if $|T_i| \leq n/2$ for all T_i. The *centroid decomposition* of T, $CDecomp(T)$, is defined recursively:

$$CDecomp(T) = \{(T, r)\} \cup \bigcup_{i=1}^{k} CDecomp(T_i).$$

Note that for every path p in T there exists an element $(T', r') \in CDecomp(T)$ such that p is a path in T' that passes through r'. This can be proved by a simple induction on $|T|$: either p passes through r in T, or we use the inductive hypothesis for the subtree T_i that contains p.

Every tree has a centroid, see [13], and a centroid of a tree can be computed in $O(n)$ time. The recursive definition of $CDecomp(T)$ implies a bound on its total size.

Fact 1. *For a tree T with n nodes, the total size of all subtrees in $CDecomp(T)$ is $O(n \log n)$. The decomposition $CDecomp(T)$ can be computed in $O(n \log n)$ time.*