Journal of Discrete Algorithms 32 (2015) 14–23

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A suffix tree or not a suffix tree? ☆

Tatiana Starikovskaya [a,*,1], Hjalte Wedel Vildhøj [b]

[a] Higher School of Economics, Russia
[b] Technical University of Denmark, DTU Compute, Denmark

ARTICLE INFO

Article history:
Available online 19 January 2015

Keywords:
Reverse engineering
Suffix trees
Suffix links
Suffix tour graphs

ABSTRACT

In this paper we study the structure of suffix trees. Given an unlabeled tree τ on n nodes and suffix links of its internal nodes, we ask the question "Is τ a suffix tree?", i.e., is there a string S whose suffix tree has the same topological structure as τ? We place no restrictions on S, in particular we do not require that S ends with a unique symbol. This corresponds to considering the more general definition of *implicit* or *extended* suffix trees. Such general suffix trees have many applications and are for example needed to allow efficient updates when suffix trees are built online. Deciding if τ is a suffix tree is not an easy task, because, with no restrictions on the final symbol, we cannot guess the length of a string that realizes τ from the number of leaves. And without an upper bound on the length of such a string, it is not even clear how to solve the problem by an exhaustive search. In this paper, we prove that τ is a suffix tree if and only if it is realized by a string S of length $n-1$, and we give a linear-time algorithm for inferring S when the first letter on each edge is known. This generalizes the work of I et al. (2014) [15].

1. Introduction

The suffix tree was introduced by Peter Weiner in 1973 [19] and remains one of the most popular and widely used text indexing data structures (see [1] and references therein). In static applications it is commonly assumed that suffix trees are built only for strings with a unique end symbol (often denoted \$), thus ensuring the useful one-to-one correspondence between leaves and suffixes. In this paper we view such suffix trees as a special case and refer to them as *\$-suffix trees*. Our focus is on suffix trees of *arbitrary strings*, which we simply call *suffix trees* to emphasize that they are more general than \$-suffix trees.[2] Contrary to \$-suffix trees, the suffixes in a suffix tree can end in internal non-branching locations of the tree, called *implicit suffix nodes*.

Suffix trees for arbitrary strings are not only a nice generalization, but are required in many applications. For example in online algorithms that construct the suffix tree of a left-to-right streaming text (e.g., Ukkonen's algorithm [18]), it is necessary to maintain the implicit suffix nodes to allow efficient updates. Despite their essential role, the structure of suffix trees is still not well understood. For instance, it was only recently proved that each internal edge in a suffix tree can contain at most one implicit suffix node [4].

☆ A preliminary version of this work has been presented at the 25th International Workshop on Combinatorial Algorithms in October 2014.

* Corresponding author.
E-mail addresses: tat.starikovskaya@gmail.com (T. Starikovskaya), hwv@hwv.dk (H.W. Vildhøj).

[1] Partly supported by Dynasty Foundation.

[2] In the literature the standard terminology is *suffix trees* for \$-suffix trees and *extended/implicit suffix trees* [3,11] for suffix trees of strings not ending with \$.

http://dx.doi.org/10.1016/j.jda.2015.01.005
1570-8667/

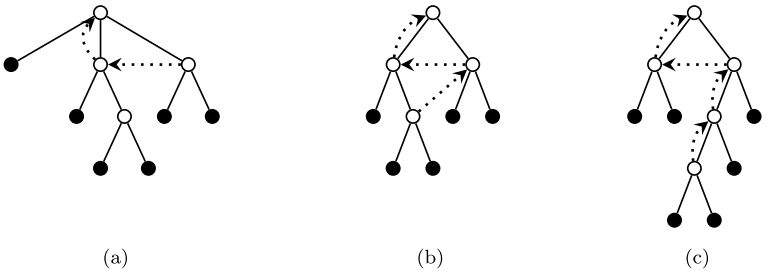

Fig. 1. Three potential suffix trees. Internal nodes are white, and leaves are black. (a) Is a $-suffix tree, e.g. for `ababa$`. (b) Is not a $-suffix tree, but it is a suffix tree, e.g. for `abaabab`. (c) Is not a suffix tree.

In this paper we prove some new properties of suffix trees and show how to decide whether suffix trees can have a particular structure. Structural properties of suffix trees are not only of theoretical interest, but are essential for analyzing the complexity and correctness of algorithms using suffix trees.

Given an unlabeled ordered rooted tree τ and suffix links of its internal nodes, the *suffix tree decision problem* is to decide if there exists a string S such that the suffix tree of S is isomorphic to τ. If such a string exists, we say that τ *is a suffix tree* and that S *realizes* τ. If τ can be realized by a string S having a unique end symbol \$, we additionally say that τ *is a \$-suffix tree*. See Fig. 1 for an example of a \$-suffix tree, a suffix tree, and a tree, which is not a suffix tree.

I et al. [15] recently considered the suffix tree decision problem and showed how to decide if τ is a \$-suffix tree in $O(n)$ time, assuming that the first letter on each edge of τ is also known. Deciding if τ is a suffix tree is much more involved, mainly because we can no longer infer the length of a string that realizes τ from the number of leaves. Without an upper bound on the length of such a string, it is not even clear how to solve the problem by an exhaustive search. In this paper, we give such an upper bound, show that it is tight, and give a linear time algorithm for deciding if τ is a suffix tree.

Our results In Section 2, we start by settling the question of the sufficient length of a string that realizes τ.

Theorem 1. *An unlabeled tree τ on n nodes is a suffix tree if and only if it is realized by a string of length $n-1$.*

As far as we are aware, there were no previous upper bounds on the length of a shortest string realizing τ. The bound implies an exhaustive search algorithm for solving the suffix tree decision problem, even when the suffix links are not provided. In terms of n, this upper bound is tight, since e.g. stars on n nodes are realized only by strings of length at least $n-1$.

The main part of the paper is devoted to the suffix tree decision problem. We generalize the work of I et al. [15] and show in Section 4 how to decide if τ is a suffix tree.

Theorem 2. *Let τ be a tree with n nodes, annotated with suffix links of internal nodes and the first letter on each edge. There is an $O(n)$ time algorithm for deciding if τ is a suffix tree.*

In case τ is a suffix tree, the algorithm also outputs a string S that realizes τ. To obtain the result, we show several new properties of suffix trees, which may be of independent interest.

Related work The problem of revealing structural properties and exploiting them to recover a string realizing a data structure has received a lot of attention in the literature. Besides \$-suffix trees, the problem has been considered for border arrays [17,7], parameterized border arrays [12–14], suffix arrays [2,9,16], KMP failure tables [8,10], prefix tables [5], cover arrays [6], directed acyclic word graphs [2], and directed acyclic subsequence graphs [2].

2. Suffix trees

In this section we prove Theorem 1 and some new properties of suffix trees, which we will need to prove Theorem 2. We start by briefly recapitulating the most important definitions.

The *suffix tree* of a string S is a compacted trie on suffixes of S [11]. Branching nodes and leaves of the tree are called *explicit nodes*, and positions on edges are called *implicit nodes*. The *label* of a node v is the string labeling the path from the root to v, and the length of this label is called the *string depth* of v. The *suffix link* of an internal explicit node v labeled by $a_1a_2\ldots a_m$ is a pointer to the node u labeled by $a_2a_3\ldots a_m$.

We use the notation $v \cdots\!\!\rightarrow u$ and extend the definition of suffix links to leaves and implicit nodes as well. We will refer to nodes that are labeled by suffixes of S as *suffix nodes*. All leaves of the suffix tree are suffix nodes, and unless S ends with a unique symbol \$, some implicit nodes and internal explicit nodes can be suffix nodes as well. Suffix links for suffix nodes form a path starting at the leaf labeled by S and ending at the root. Following [4], we call this path the *suffix chain*.