Journal of Discrete Algorithms 28 (2014) 9–22

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Reverse engineering of compact suffix trees and links: A novel algorithm

Bastien Cazaux, Eric Rivals *

L.I.R.M.M., University of Montpellier II, CNRS U.M.R. 5506, 161 rue Ada, F-34392 Montpellier Cedex 5, France

ARTICLE INFO

Article history:
Available online 22 July 2014

Keywords:
Text indexing
Suffix tree
Characterisation
Recognition
Graph
Eulerian tour
Permutation
Connectivity

ABSTRACT

Invented in the 1970s, the Suffix Tree (ST) is a data structure that indexes all substrings of a text in linear space. Although more space demanding than other indexes, the ST remains likely an inspiring index because it represents substrings in a hierarchical tree structure. Along time, STs have acquired a central position in text algorithmics with myriad of algorithms and applications to for instance motif discovery, biological sequence comparison, or text compression. It is well known that different words can lead to the same suffix tree structure with different labels. Moreover, the properties of STs prevent all tree structures from being STs. Even the suffix links, which play a key role in efficient construction algorithms and many applications, are not sufficient to discriminate the suffix trees of distinct words. The question of recognising which trees can be STs has been raised and termed Reverse Engineering on STs. For the case where a tree is given with potential suffix links, a seminal work provides a linear time solution only for binary alphabets. Here, we also investigate the Reverse Engineering problem on ST with links and exhibit a novel approach and algorithm. Hopefully, this new suffix tree characterisation makes up a valuable step towards a better understanding of suffix tree combinatorics.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Forty years after its invention, the Suffix Tree (ST) remains a ubiquitous data structure for indexing all substrings of a text [2] and still inspires many researchers. Given a word w, the ST of w can be built in linear time and space using well-known construction algorithms [8,5,7,1,2]. To achieve linear time construction, all internal nodes of an ST are equipped with an edge of a different type called a *suffix link*. Once built, the ST is kept in memory and serves for matching exactly any given pattern in a time linear in the pattern length rather than in the text length [2]. However, approximate pattern matching or overlap matching can also be achieved with STs, as well as many other applications, for instance in bioinformatics [2].

To any word corresponds a suffix tree. As illustrated in Fig. 1, two distinct words can give rise to the same suffix tree (in terms of structure, not in terms of labels). Moreover, not all tree structures can be a suffix tree. Hence, the reverse engineering problem (REST), which, given a tree asks for a word whose suffix tree is isomorphic to the input tree, is a natural, but non-trivial question. How does one characterise a tree that is a suffix tree? Which strings do generate the same ST? How many distinct STs exist for strings of a given length? All these questions remain open, not mentioning

* Corresponding author.
E-mail addresses: Bastien.Cazaux@lirmm.fr (B. Cazaux), rivals@lirmm.fr (E. Rivals).

http://dx.doi.org/10.1016/j.jda.2014.07.002
1570-8667/© 2014 Elsevier B.V. All rights reserved.

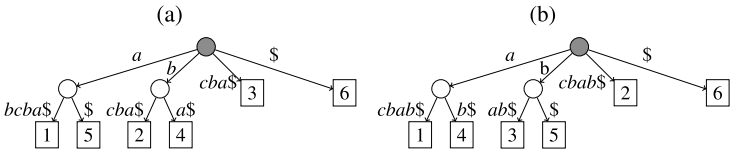

Fig. 1. Two different strings with an identical suffix tree structure: in (a) the string *abcba*\$ and in (b) the string *acbab*\$. Note that in both trees the suffix links (not represented) of internal nodes are identical (they go to the root). Almost all labels from internal nodes to leaves are distinct between the two suffix trees, even their lengths differ. No permutation of the alphabet symbols can transform *abcba*\$ into *acbab*\$.

enumeration or random sampling. To our knowledge, the problem REST has been studied in a master thesis [6],[1] and in a special case in one article, namely when the tree and its potential suffix links[2] are given as inputs [3,4]. We call this version the *SLI-REST* (for Suffix Links on Internal nodes – REST). Both reverse engineering problems are fundamental for enumerating, counting, and even uniformly sampling suffix trees at random, in other words for understanding their combinatorics. The reverse engineering problem has been raised and addressed for other data structures like border arrays, suffix arrays, etc. (see references in [4]).

The SLI-REST problem is defined as follows: given a tree T and a tree of additional links F (on the same nodes as T), find a string whose ST and Suffix Links (SL) are isomorphic to (T, F). In [3,4], the authors first define SLI-REST, but then investigate a restricted version of it where T is an ordered tree and is equipped with a *labelling function* giving for each edge the first symbol of its label. They propose an algorithm that builds the Suffix Tour Graph on the nodes of T, and checks whether it contains an Eulerian cycle including the root and all leaves of T. On a binary alphabet, the authors show with combinatorial arguments that the number of labelling functions is bounded by a constant, and hence their algorithm runs in linear time. Since it explores all labelling functions, the same algorithm can also output all strings solving SLI-REST. However, the combinatorial explosion of possible labelling functions for a larger alphabet makes it inappropriate. Here, we propose a different approach for the general version of SLI-REST (where T is neither ordered, nor labelled) and organise the paper as follows. Notation, definitions, and known properties are written below for strings, suffix trees, and other concepts. Section 2 explains how to compute the labels for all internal edges, Section 3 details the case where all nodes are equipped with potential suffix links, while Section 4 explains how to find the potential suffix links of leaves and proposes an algorithm for solving SLI-REST. Finally, Section 5 summarises the differences and improvements between [4] approach and ours, and Section 6 concludes.

1.1. Notation on strings, trees and graphs

An *alphabet* Σ is a finite set of *letters*. A *word* or *string* over Σ is a finite sequence of elements of Σ. The set of all words over Σ is denoted by $\Sigma^\star$, and ε denotes the empty word. For a word x, $|x|$ denotes the *length* of x. Given two words x and y, we denote by xy the *concatenation* of x and y. For every $1 \le i \le j \le |x|$, $x[i]$ denotes the i-th letter of x, and $x[i; j]$ denotes the *substring* $x[i]x[i+1]\dots x[j]$. The suffix of x starting at position i is denoted $\mathrm{suff}_i(w)$. We denote by $\#(\Lambda)$ the cardinality of any set Λ.

A directed graph $G := (V_G, E_G)$ consists of a set of vertices V_G and a set of directed edges E_G that is a subset of $V_G \times V_G$. An *Eulerian cycle* in a graph G is a simple path that visits each edge exactly once (with the same start and end vertices). It can be seen as a cyclic permutation of E_G. An *Eulerian route* is a traversal of an Eulerian cycle starting at a given vertex. The $\#(E_G)$ possible routes of an Eulerian cycle are all cyclic shifts one of another. A tree is a graph in which any two vertices can be connected by a unique simple path. In a rooted tree T, the *root* is denoted by $\perp_T$, the vertices of V_T are partitioned into *internal* nodes and *leaves*; so, $V_T := V_T^{in} \cup V_T^{leaf}$. For any node $v \in V_T$, $f_T(v)$ denotes its unique *father*, while $Children_T(v)$ is its set of *children*. A leaf has no children and the root has no father. Moreover, a child of v that is a leaf is termed a *chleaf*; thus, children of v are partitioned into chleaves and non-leaf children (also called *chin*), which we denote by $Chleaves_T(v) := Children_T(v) \cap V_T^{leaf}$ and $Children_T(v) := Chleaves_T(v) \cup Chin_T(v)$. In addition, $V_T(v)$ denotes the set of nodes of the subtree of T rooted in v.

An *alphabetisation* of a tree T on an alphabet Σ is an injective mapping from the set of edges going out $\perp_T$ to Σ. A *labelling* l of a tree T on an alphabet Σ is a mapping from E_T onto Σ^*: it maps an edge e to a word $l(e)$ of Σ^*. Let $u, v \in V_T$ be such that $v \in V_T(u)$, $\tilde{l}(u, v)$ denotes the word formed by the concatenation of edge labels on the (unique) path joining u to v in T, and $l(v) := l(f_T(v), v)$. The (unique) *word represented by node* v with labelling l is $\tilde{l}(v) := \tilde{l}(\perp_T, v)$.

Note that the notion of labelling function of [4] assigns the first letter to the labels of all edges; so it does not coincide with our notion of labelling.

[1] A study that did not come up with a characterisation.
[2] We qualify those links of potential to distinguish them from true suffix links.

Download English Version:

https://daneshyari.com/en/article/431277

Download Persian Version:

https://daneshyari.com/article/431277

Daneshyari.com