ELSEVIER

Information Processing Letters 83 (2002) 159–161

Information Processing Letters

www.elsevier.com/locate/ipl

A speed-up for the commute between subword trees and DAWGs ☆

Alberto Apostolico [a,b,*], Stefano Lonardi [a,b]

[a] *DEI, University of Padova, Via Gradenigo 6/A, I-35131 Padova, Italy*
[b] *Computer Sciences Department, Purdue University, West Lafayette, IN 47907, USA*

Received 3 March 2001
Communicated by F. Dehne

Abstract

A popular way to describe and build the DAWG or Directed Acyclic Word Graph of a string is by transformation of the corresponding subword tree. This transformation, which is not difficult to reverse, is easy to grasp and almost trivial to implement except for the assumed implication of a standard tree isomorphism algorithm. Here we point out a simple property of subword trees that makes checking tree isomorphism in this context a straightforward process, thereby simplifying the transformation significantly. Subword trees and DAWGs arise rather ubiquitously in applications of string processing, where they often play complementary roles. Efficient conversions are thus especially desirable.

Keywords: Design and analysis of algorithms; Subword tree; DAWG; Tree isomorphism test

1. Introduction

We recall, in informal terms, that a subword tree is the compact tree storing all suffixes of a given string, while the DAWG, or Directed Acyclic Word Graph, for a string is a special finite automaton recognizing all subwords of that string. We assume some familiarity of the reader with the salient properties and constructions of these structures, for which we refer to the bibliography [1–8].

It is customary (see, e.g., [2–5]) to introduce a DAWG as the result of a two-step transformation of a subword tree. The first step involves the identification and juxtaposition of all roots of isomorphic subtrees, beginning with the leaves. This produces an intermediate or "compact" DAWG consisting of a directed acyclic graph with one source and one sink, and requiring linear space for nodes and edges except for the edge labels, that can charge quadratic space in the worst case.

The second step takes care of reducing the overall space to linear. This exploits the fact that the edges reaching a same node are labeled by some consecutive suffixes of a same, longest word. This property supports a more succinct representation for the bundle of edges reaching each node. Achieving such a representation requires the following two simple ac-

☆ Work supported in part by NATO Grant PST.CLG.977017, by Purdue Research Foundation Grant 690-1398-3145, by the Italian Ministry of University and Research under the National Project "Bioinformatics and Genomic Research", and by the Research Program of the University of Padova.

* Corresponding author.
E-mail address: axa@cs.purdue.edu (A. Apostolico).

0020-0190/01/$ – see front matter
PII: S0020-0190(01)00327-1

tions: first, the edge with the longest label is broken down into a chain of unit edges, and then all other edges are re-directed to an appropriate node in the chain.

Of the two main steps above, the second one is trivial to implement. The first one is done by invoking some variant of the classical algorithm for testing tree isomorphism devised by Hopcroft and Tarjan [6] in connection with their linear planarity test (see, e.g., [1, Chapter 3, pp. 84–86]). Although this algorithm is linear and not prohibitively involved, we show here that there is a faster and more natural way to test isomorphism on subword trees. Combined with the simplicity of step 2, this makes it faster to commute a subword tree into its corresponding DAWG.

2. Speeding up the isomorphism test

We use T_x to denote the subword tree of string $x\$$, where \$ is a symbol not in x. The following two easy facts are well known to hold for T_x.

Fact 1. *Let a be a symbol of the alphabet and v a possibly empty string. If the path labeled $w = av$ from the root of T_x ends precisely at a node, then so does the path labeled v.*

The property in Fact 1 finds crucial use in the efficient constructions of T_x. To exploit the property, *suffix* links are maintained in the tree that lead from the terminal node of each string av to the terminal node of its suffix v. Such links are thus a byproduct of the construction.

Fact 2. *Let μ be the node reachable from the root of T_x on the shortest path labeled wv, where v is possibly empty. Then, the starting positions of all the occurrences of w in x are precisely the leaves in the subtree of T_x rooted at μ.*

Fact 2 can be re-phrased by saying that the number of occurrences of w in x equals the number of leaves in the subtree of T_x rooted at μ. A trivial bottom-up computation on T_x will weight each node with the number of leaves in the subtree rooted at that node. Facts 1 and 2 support our main property below. The property shows that, having weighted the tree as stated, the juxtaposition of isomorphic trees in step 1 can be accomplished just by collapsing to a single node each chain of suffix links that connects nodes with the same weight.

Fact 3 (Main fact). *Any two subtrees T^1 and T^2 of T_x are isomorphic if and only if they have the same number of leaves and their roots are connected by a chain of suffix links.*

Proof. Let μ_1 and μ_2 be the nodes that are roots of T^1 and T^2, respectively, and let strings w_1 and w_2, where we take without loss of generality $|w_1| > |w_2|$, be the respective labels of the paths from the root of T_x to those nodes. We also use $endpos(w)$ to denote the set of the end positions of all occurrences of string w in x.

If T^1 and T^2 are isomorphic then the collection of the labels from their respective roots to their leaves must describe a same set of suffixes of x. Thus the trees must have, in particular, the same number of leaves. Let k be the starting position in x of a suffix contained in both T^1 and T^2. Then, by the structure of T_x, leaves $k - |w_1|$ and $k - |w_2|$ must be found in T^1 and T^2, respectively. This is equivalent to saying that $k - 1$ is in $endpos(w_1)$ as well as in $endpos(w_2)$. But then w_2 is a suffix of w_1, and there must be a chain of suffix links from μ_1 to μ_2.

Let us assume now that there is a chain composed by $l > 0$ suffix links from the root of T^1 to the root of T^2 and that T^1 and T^2 have the same number of leaves. From the first one of these assumptions, it must be possible to write $w_1 = uw$, and $w_2 = w$, where $|u| = l$. Now, in general, we have $endpos(uw) \subseteq endpos(w)$. Since we know from the second assumption that $|endpos(uw)| = |endpos(w)|$, then the only possibility is that $endpos(uw) = endpos(w)$, hence T^1 and T^2 are isomorphic. □

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[2] A. Apostolico, Z. Galil (Eds.), Pattern Matching Algorithms, Oxford University Press, New York, 1997.

[3] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, R. McConnell, Complete inverted files for efficient text retrieval and analysis, J. ACM 34 (3) (1987) 578–595.
[4] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen, J. Seiferas, The smallest automaton recognizing the subwords of a text, Theoret. Comput. Sci. 40 (1985) 31–55.
[5] M. Crochemore, W. Rytter, Text Algorithms, Oxford University Press, New York, 1994.
[6] J.E. Hopcroft, R.E. Tarjan, Isomorphism of planar graphs, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 131–150.
[7] E.M. McCreight, A space economical suffix tree construction algorithm, J. ACM 23 (1976) 262–272.
[8] E. Ukkonen, On-line constructin of suffix trees, Algorithmica 14 (1995) 249–260.