EFFICIENT AND ELEGANT SUBWORD-TREE CONSTRUCTION†

M. T. Chen
Department of Computer Science
University of Nanjing
People's Republic of China

Joel Seiferas
Department of Computer Science
University of Rochester
Rochester, New York, U. S. A. 14627

Abstract. A clean version of Weiner's linear-time compact-subword-tree construction simultaneously also constructs the smallest deterministic finite automaton recognizing the reverse subwords.

Introduction

Any finite set $S$ of words that is prefix-closed (i.e., $xy \in S \Rightarrow x \in S$) has a prefix tree with node set $S$, ancestor relation "is a prefix of", and father relation "is obtained by dropping the last letter of". The set $S$ of all subwords of a text string is a prefix-closed set whose prefix tree, the text's subword tree, is particularly useful. For example, it lets us test arbitrary words for membership in $S$ in time proportional to their own lengths, regardless of how long the entire text is. Even more useful is the subword tree for $\$w\$, where $\$ and $\$ are delimiting symbols not occurring in $w$. This, for example, lets us test easily whether a word is a prefix or suffix of $w$. In one appropriate walk through the tree, we can easily augment each node with such information as the count of its leaf descendents. Then it becomes convenient to tell how many times a word appears, where a word first or last appears, what is the longest repeated subword, and more. As an example, the subword tree for $aabbab$ is shown in Figure 1.

The number of distinct subwords of a text string of large length $n$ can be very large (proportional to $n^2$ for $a^{n/2}b^{n/2}$, for example), so subword trees can have prohibitively many nodes. Fortunately, however, there are compact but functionally equivalent data structures that can even be built in time proportional to just $n$. Weiner [13], McCreight [5], Pratt [6, 7, 8], and Slisenko [12, Section 2] have described such data structures and algorithms. (See also [1, Section 9.5], [4].) Each of their algorithms is complicated by the maintenance of additional auxiliary structure along with the developing compact subword tree. (In Slisenko's case, more ambitious applications account for an extra measure of additional structure [9-12].) In this report, we describe a version with auxiliary structure that is unusually clean and clearly desirable in its own right.

† Part of this work was done while the first author visited the University of Rochester. The work of the second author was supported in part by the National Science Foundation under grant MCS-8110430.
The most obvious way to represent a subword tree compactly is to omit interior nodes of degree 1, replacing them with through edges. The string corresponding to each remaining node can be represented by a (not necessarily unique) pair of pointers into the text string; or, alternatively, the incremental substring corresponding to each edge can be represented by such a pair. Resulting representations for \$aabab\$ are shown in Figure 2. Either way, no information is lost, and the degree of each remaining node continues to be bounded by the alphabet size. The size of this representation is thus proportional to the number of nodes. And the number of nodes is proportional to the length of the text, because the number of leaves is so bounded (one for each suffix of the text) and because, in a tree without interior nodes of degree 1, the number of interior nodes is bounded by the number of leaves.

We can obtain a quite different compact representation by identifying (edge-)isomorphic subtrees. (The edge labels to be preserved by each such isomorphism are the incremental letters, not their indices in the text.) The edge-labeled version of the subword tree for \$aabab\$ (shown in Figure 3), for example, has isomorphic subtrees below the subwords $b$ and $ab$. The result of making all such identifications is shown in Figure 4. (For expositional clarity in our figures, we revert to explicit edge labels. For later reference, in addition, parenthesized capital letters have been arbitrarily assigned as names for the nodes in Figure 4.) Except for omission of the one nonaccepting state, from which all transitions are self-loops, this directed acyclic graph is just the smallest deterministic finite automaton recognizing the set of subwords of the text. We will see that the size of this representation is again only proportional to the length of the text,