Algorithmica (1995) 14: 249–260

Algorithmica

On-Line Construction of Suffix Trees[1]

E. Ukkonen[2]

Abstract. An on-line algorithm is presented for constructing the suffix tree for a given string in time linear in the length of the string. The new algorithm has the desirable property of processing the string symbol by symbol from left to right. It always has the suffix tree for the scanned part of the string ready. The method is developed as a linear-time version of a very simple algorithm for (quadratic size) suffix *tries*. Regardless of its quadratic worst case this latter algorithm can be a good practical method when the string is not too long. Another variation of this method is shown to give, in a natural way, the well-known algorithms for constructing suffix automata (DAWGs).

Key Words. Linear-time algorithm, Suffix tree, Suffix trie, Suffix automaton, DAWG.

1. Introduction. A *suffix tree* is a trie-like data structure representing all suffixes of a string. Such trees have a central role in many algorithms on strings, see, e.g., [3], [7], and [2]. It is quite commonly felt, however, that the linear-time suffix tree algorithms presented in the literature are rather difficult to grasp.

The main purpose of this paper is an attempt to develop an understandable suffix tree construction based on a natural idea that seems to complete our picture of suffix trees in an essential way. The new algorithm has the important property of being on-line. It processes the string symbol by symbol from left to right, and always has the suffix tree for the scanned part of the string ready. The algorithm is based on the simple observation that the suffixes of a string $T^i = t_1 \cdots t_i$ can be obtained from the suffixes of string $T^{i-1} = t_1 \cdots t_{i-1}$ by catenating symbol t_i at the end of each suffix of T^{i-1} and by adding the empty suffix. The suffixes of the whole string $T = T^n = t_1 t_2 \cdots t_n$ can be obtained by first expanding the suffixes of T^0 into the suffixes of T^1 and so on, until the suffixes of T are obtained from the suffixes of T^{n-1}.

This is in contrast with the method by Weiner [13] that proceeds right to left and adds the suffixes to the tree in increasing order of their length, starting from the shortest suffix, and with the method by McCreight [9] that adds the suffixes to the tree in decreasing order of their length. It should be noted, however, that despite the clear difference in the intuitive view on the problem, our algorithm and McCreight's algorithm are in their final form functionally rather closely related.

[1] This research was supported by the Academy of Finland and by the Alexander von Humboldt Foundation (Germany).

[2] Department of Computer Science, University of Helsinki, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland. ukkonen@cs.Helsinki.FI.

Received March 4, 1993; revised December 8, 1993. Communicated by K. Mehlhorn.

Our algorithm is best understood as a linear-time version of another algorithm from [12] for (quadratic-size) suffix *tries*. The latter very elementary algorithm, which resembles the position tree algorithm in [8], is given in Section 2. Unfortunately, it does not run in linear time—it takes time proportional to the size of the suffix trie which can be quadratic. However, a rather transparent modification, which we describe in Section 4, gives our on-line, linear-time method for suffix trees. This also offers a natural perspective which makes the linear-time suffix tree construction understandable.

We also point out in Section 5 that the suffix trie augmented with the suffix links gives an elementary characterization of the suffix automata (also known as directed acyclic word graphs or DAWGs). This immediately leads to an algorithm for constructing such automata. Fortunately, the resulting method is essentially the same as already given in [4]–[6]. Again it is felt that our new perspective is very natural and helps in understanding the suffix automata constructions.

2. Constructing Suffix Tries. Let $T = t_1 t_2 \cdots t_n$ be a string over an alphabet Σ. Each string x such that $T = uxv$ for some (possibly empty) strings u and v is a *substring* of T, and each string $T_i = t_i \cdots t_n$ where $1 \leq i \leq n+1$ is a *suffix* of T; in particular, $T_{n+1} = \varepsilon$ is the *empty* suffix. The set of all suffixes of T is denoted $\sigma(T)$. The *suffix trie* of T is a trie representing $\sigma(T)$.

More formally, we denote the suffix trie of T as $STrie(T) = (Q \cup \{\perp\}, root, F, g, f)$ and define such a trie as an augmented DFA (deterministic finite-state automaton) which has a tree-shaped transition graph representing the trie for $\sigma(T)$ and which is augmented with the so-called suffix function f and auxiliary state $\perp$. The set Q of the states of $STrie(T)$ can be put in a one-to-one correspondence with the substrings of T. We denote by $\bar{x}$ the state that corresponds to a substring x.

The initial state *root* corresponds to the empty string ε, and the set F of the final states corresponds to $\sigma(T)$. The transition function g as defined as $g(\bar{x}, a) = \bar{y}$ for all $\bar{x}$, $\bar{y}$ in Q such that $y = xa$, where $a \in \Sigma$.

The *suffix function* f is defined for each state $\bar{x} \in Q$ as follows. Let $\bar{x} \neq root$. Then $x = ay$ for some $a \in \Sigma$, and we set $f(\bar{x}) = \bar{y}$. Moreover, $f(root) = \perp$.

Auxiliary state $\perp$ allows us to write the algorithms in what follows such that an explicit distinction between the empty and the nonempty suffixes (or, between *root* and the other states) can be avoided. State $\perp$ is connected to the trie by $g(\perp, a) = root$ for every $a \in \Sigma$. We leave $f(\perp)$ undefined. (Note that the transitions from $\perp$ to *root* are defined consistently with the other transitions: State $\perp$ corresponds to the inverse a^{-1} of all symbols $a \in \Sigma$. Because $a^{-1}a = \varepsilon$, we can set $g(\perp, a) = root$ as *root* corresponds to ε.)

Following [9] we call $f(r)$ the *suffix link* of state r. The suffix links are utilized during the construction of a suffix tree; they also have many uses in the applications (e.g., [11] and [12]).

Automaton $STrie(T)$ is identical to the Aho–Corasick string matching automaton [1] for the key-word set $\{T_i \mid 1 \leq i \leq n+1\}$ (the suffix links are called *failure transitions* in [1]).