Space Efficient Suffix Trees

Ian Munro1, Venkatesh Raman2, and S. Srinivasa Rao2

1 Department of Computer Science, University of Waterloo, Canada N2L 3G1
2 The Institute of Mathematical Sciences, Chennai, India 600 113

Abstract. We first give a representation of a suffix tree that uses \(n \lg n + O(n)\) bits of space and supports searching for a pattern in the given text (from a fixed size alphabet) in \(O(m)\) time, where \(n\) is the size of the text and \(m\) is the size of the pattern. The structure is quite simple and answers a question raised by Muthukrishnan in [17]. Previous compact representations of suffix trees had a higher lower order term in space and had some expectation assumption [3], or required more time for searching [5]. Then, surprisingly, we show that we can even do better, by developing a structure that uses a suffix array (and so \(n \lceil \lg n \rceil\) bits) and an additional \(o(n)\) bits. String searching can be done in this structure also in \(O(m)\) time. Besides supporting string searching, we can also report the number of occurrences of the pattern in the same time using no additional space. In this case the space occupied by the structures is much less compared to many of the previously known structures to do this. When the size of the alphabet \(k\) is not a constant, our structures can be easily extended, using standard tricks, to those that use the same space but take \(O(m \lg k)\) time for string searching or to those that use an additional \(O(m \lg k)\) bits but take the same \(O(m)\) time for searching.

1 Introduction and Motivation

Given a text string, indexing is the problem of preprocessing the text so that search for a pattern can be answered efficiently. Two popular data structures for indexing are suffix trees and suffix arrays. A suffix tree [12] is a trie representing all the suffixes of the given text. Standard representations of suffix trees for texts of length \(n\) take about \(4n\) words or pointers (each word taking \(\log n\) bits of storage) and a search for a pattern of size \(m\) can be performed in \(O(m)\) time (see Section 3 for details). A suffix array [11] keeps an array of pointers to the suffixes of the given text in lexicographic order, and another array of longest common prefixes of some of those suffixes to aid the search. Standard representation of suffix array takes about \(2n\) words [5] and a search in a suffix array can be performed in \(O(m + \log n)\) time [11]. So in [17], Muthukrishnan asked whether there exists a data structure that uses only \(n + o(n)\) words and answers indexing questions in \(O(m)\) time. In this paper, we propose two such structures. The first one uses \(n + O(n/ \lg n)\) words, or equivalently \(n \lg n + O(n)\) bits and supports string searching in optimal \(O(m)\) time. The second structure takes \(n \lceil \lg n \rceil + o(n)\) bits of space and supports searching in \(O(m)\) time. It is
interesting to note that $o(n)$ additional bits are enough besides the $n$ pointers for indexing.

Previously, Colussi and De Col [5] reported a data structure that uses $n \log n + O(n)$ bits, but a search in the structure takes $O(m + \log \log n)$ time. In [3], Clark and Munro gave a version of suffix tree that uses $n$ plus an expected $O(n \log \log n / \log n)$ words under the assumption that the given binary (encoded) text is generated by a uniform symmetric random process and that the bit strings in the suffixes are independent. So our structure is not only more space efficient (in the lower order term), it doesn’t need any assumption about the distribution of characters in the input string.

The main idea in our first representation is the recent $2n + o(n)$ bits encoding of a static binary tree on $n$ nodes [16]. First we observe that a few more operations, than those given in [16] are needed to support our suffix tree algorithms. The next section reviews the binary tree representation and describes algorithms to support the additional operations. Section 3 reviews the suffix tree data structure and explains how our binary tree representation can be used to obtain a space efficient suffix tree. In Section 4 we give a structure that uses a suffix array and a couple of sparse suffix trees taking $n \lceil \log n \rceil + o(n)$ bits of space which also supports searching in optimal time bounds. Section 5 gives concluding remarks and lists open problems.

Our model of computation is the standard unit cost RAM model where we assume that standard arithmetic and boolean operations on $\log n$ bit words and reading and writing $\log n$ bit strings can be performed in constant time. Also $\log$ denotes the logarithm to the base 2 throughout the paper. And by a suffix array, we mean an array of pointers to the suffixes of the given text in lexicographic order (without any extra information).

## 2 Succinct Representation of Trees

A general rooted ordered tree on $n$ vertices can be represented by a nested balanced string of $2n$ parenthesis as follows. Perform a preorder traversal of the tree starting from the root, and write an open parenthesis when a node is first encountered, going down the tree, and then a closing parenthesis while going up after traversing the subtree. However, if we represent a binary tree by such a sequence, it is not possible to distinguish a node with a left child but no right child and one with a right child and no left child.

So Munro and Raman[16] first use the well known isomorphism between the class of binary trees and the class of rooted ordered trees to convert the given binary tree into a general rooted ordered tree and then represent the rooted ordered tree using the above parenthesis representation. In the ordered tree there is a root which does not correspond to any node in the binary tree. Beyond this, the left child of a node in the binary tree corresponds to the leftmost child of the corresponding node in the ordered tree, and the right child in the binary tree corresponds to the next sibling to the right in ordered tree. (See the figure for an example). A node is represented, by convention, by its corresponding left