

# On the Size of Generalised Suffix Trees Extended with String ID Lists

*Norwegian University of Science and Technology  
Technical Report IDI-TR-2007-01*

ISSN: 1503-416X

Nils Grimsmo\*  
and  
Truls Amundsen Bjørklund  
`{nilsgri, trulsamu}@idi.ntnu.no`

Department of Computer and Information Science  
Sem Sælands vei 7-9  
NO-7491 Trondheim  
NORWAY

2007-02-02

---

\*Supported by the Research Council of Norway under the grant NFR 162349.

## Abstract

The document listing problem can be solved with linear preprocessing and optimal search time by using a generalised suffix tree, additional data structures and constant time range minimum queries. A simpler solution is to use a generalised suffix tree in which internal nodes are extended with a list of all string IDs seen in the subtree below the respective node. This report makes some remarks on the size of such a structure. For the case of a set of equal length strings, a bound of  $\Theta(n\sqrt{n})$  for the worst case space usage of such lists is given, where  $n$  is the total length of the strings.

## 1 Document listing problem

The *occurrence listing problem* is given a string  $T \in \Sigma^*$  of length  $n$ , which can be pre-processed, and a pattern  $P$  of length  $m$ , find all  $z$  occurrences of  $P$  in  $T$ . This classical problem can be solved with  $\Theta(n)$  preprocessing and optimal  $O(m + z)$  time queries with a suffix tree [McC76] if  $|\Sigma|$  is constant. The *document listing problem* is, given a set of strings  $T = \{T_1, \dots, T_t\}$  of total length  $n$ , which can be preprocessed, find all  $y$  strings in  $T$  which contain the pattern  $P$ . This can also be solved with  $\Theta(n)$  preprocessing and  $O(m + y)$  time queries, by augmenting a generalised suffix tree with additional data, and using constant time minimum range queries [Mut02].

This report considers the complexity of a simpler solution, which does not have linear time preprocessing or linear space usage, but has been used to solve a problem in information retrieval [ZKNY07], without considering the complexity.

## 2 Suffix tree definition

A suffix tree [McC76] for a string  $T$  of length  $n$  from the alphabet  $\Sigma$  is a compacted trie containing the first  $n$  suffixes of  $T\$$ , where  $\$ \notin \Sigma$  is a unique terminator. Compaction here means that all internal non-branching nodes in the trie have been removed, joining adjacent edges. As  $\$$  is not seen in  $T$ , no suffix is a prefix of another, and all  $n$  suffixes are represented by a unique leaf node. Since all internal nodes are branching, their number is strictly upper bounded by  $n$ . If the edges in the tree are represented as pointers into  $T$ , the suffix tree can be represented in  $\Theta(n)$  space. It can be constructed in  $\Theta(n)$  time if the alphabet is constant [Wei73, McC76, Ukk95] or integer [Far97].

Given a pattern  $P$  of length  $m$ , it can be decided whether it occurs in  $T$  by trying to follow its path downwards from the root in the suffix tree. The time complexity is  $O(m)$  if child lookup is  $\Theta(1)$ , which is trivially achieved if  $|\Sigma|$  is constant. The parent child relationship is typically implemented with linked sibling lists. An expected lookup time of  $O(m)$  can be achieved with hashing [McC76]. Perfect hashing [FKS84] can be used to get  $O(m)$  worst case lookup,

at the cost of a longer construction time. After the position representing  $P$  in the tree has been located, all  $z$  hits can be extracted in  $\Theta(z)$  time, as all leaf nodes in the subtree correspond to unique hits and all internal nodes in the subtree are branching.

### 3 Generalised suffix tree

A generalised suffix tree for a set of strings  $T = \{T_1, \dots, T_s\}$  of total length  $n = n_1 + \dots + n_s$  is a compacted trie containing for each  $T_i$ , all the first  $n_i$  suffixes of  $T_i\$, where  $\$$  is a unique terminator string. The tree will have  $n$  leaf nodes, and can be constructed in  $\Theta(n)$  time and space [Gus97].$

### 4 String ID list extension

A generalised suffix tree is an asymptotically optimal substring index for a set of strings from an constant alphabet. The cost of a query is linear in the size of the input and output. But in many cases, you do not want to extract the set of hit positions, but the set of strings in which the hits occur. In cases where substrings are repeated in the strings indexed, a string ID can be seen many times in a subtree, and the set of unique string IDs will be smaller than the set of hit positions. To avoid this overhead, each node can store a list of the string IDs seen in its subtree. This can speed up the search for strings with matches. A drawback is a non-linear space usage in the worst case, which is shown below.

### 5 A lower bound for worst case space usage

Here follows a proof for a lower bound of  $\Omega(n\sqrt{n})$  for the worst case space usage of the suffix trees extended with string ID lists. This is shown through the  $\Theta(n\sqrt{n})$  space usage for a family of sets of strings.

Assume we have the set of strings  $T = \{T_1, \dots, T_t\}$ , where the strings are the rotations of  $(1, \dots, t)$ , such that

$$T_i = ((0 + i) \bmod (t + 1), \dots, (t - 1 + i) \bmod (t + 1)) \quad (1)$$

The total length of the strings is  $n = t^2$ . Figure 1 shows a generalised suffix tree for this set of strings for  $t = 4$ .

Let  $U = (1 \bmod (t + 1), \dots, (2t - 1) \bmod (t + 1))$ . All strings in  $T$  are substrings of  $U$ . Figure 2 shows this for  $t = 4$ . Consider the substring  $U[p \dots q]$ , where

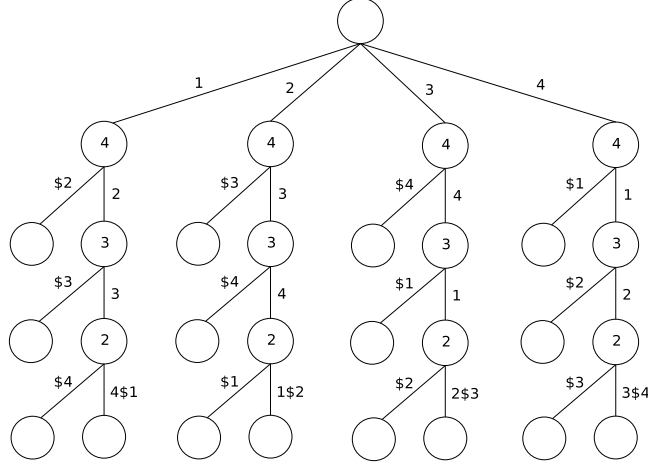


Figure 1: Generalised suffix tree for the padded strings  $(1, 2, 3, 4, \$1)$ ,  $(2, 3, 4, 1, \$2)$ ,  $(3, 4, 1, 2, \$3)$  and  $(4, 1, 2, 3, \$4)$ . Number of unique string IDs in subtree shown in internal nodes.

1	2	3	4	1	2	3	
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	\$1			
	2	3	4	1	\$2		
		3	4	<b>1</b>	<b>2</b>	\$3	
			4	<b>1</b>	<b>2</b>	3	\$4

Figure 2: Showing how the strings in  $T$  are substrings of  $U$  for  $t = 4$ . For each  $T_i \in T$ , each substring of length less than  $t$  is seen two places in  $U$ .

$1 \leq p \leq t$  and  $p \leq q < p + t$ , which is equal to  $U[p + t \dots q + t]$  if  $q < t$ . It is seen in  $T_i$ , where  $i \leq p$  or  $q < i$ , which gives a total of  $p + (t - q)$  locations.

If  $p + (t - q) > 1$ , there will be an internal node representing the substring  $U[p \dots q]$ , as it is followed by  $U[q + 1]$  in some string, and by an end of string symbol in the padded string  $T_{q-t+1}\$_{q-t+1}$ . Hence the total number of string IDs stored in the lists will be

$$|L| = \sum_{p=1}^t \sum_{q=p}^{p+t-2} p + (t - q) = \sum_{p=1}^t \sum_{q=0}^{t-2} t - q = t \sum_{k=2}^t k = \frac{t^3 + t^2 - 2t}{2} \quad (2)$$

Inserting  $t = \sqrt{n}$ , we get

$$|L| = \frac{n\sqrt{n} + n - 2\sqrt{n}}{2} \in \Theta(n\sqrt{n}) \quad (3)$$

As the space usage for a normal generalised suffix tree is  $\Theta(n)$ , we get a total space usage of  $\Theta(n\sqrt{n})$  for a tree extended with string ID lists for this family

of sets of strings, and a bound of  $\Omega(n\sqrt{n})$  for the worst case space usage of this data structure in general.

## 6 An upper bound for strings of equal length

Assume we have  $T = \{T_1, \dots, T_t\}$ , and equal string lengths with  $|T_i| = r$ . The total length of the strings is  $n = tr$ . Since there are less than  $n$  internal nodes in the tree, and each list contains at most  $t$  IDs, the total length of the lists is bounded as

$$|L| < tn = t(tr) = t^2r \in O(t^2r) \quad (4)$$

Seen differently, the  $j$ th suffix of each string has length  $r - j + 1$ , so there can be at most  $r - j$  internal nodes above the leaf node representing the suffix, to which it can contribute to the string ID list. This gives a bound

$$|L| \leq t \sum_{j=1}^r r - j = t \sum_{k=0}^{r-1} k = t \frac{(r-1)r}{2} \in O(tr^2) \quad (5)$$

Together, these two bounds give

$$|L| \in O(\min(t^2r, tr^2)) \quad (6)$$

which is maximal for  $r = t$ . This gives  $n = t^2 = r^2$ , and the bound

$$|L| \in O(n\sqrt{n}) \quad (7)$$

Combining with Equation 3, we get a bound of  $\Theta(n\sqrt{n})$  for the worst case space usage for equal length strings.

## References

- [Far97] M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. FOCS*, pages 137–143, 1997.
- [FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a Sparse Table with 0 (1) Worst Case Access Time. *J. ACM*, 31(3):538–544, 1984.
- [Gus97] D. Gusfield. *Algorithms on strings, trees, and sequences: Computer science and computational biology*. 1997.

- [McC76] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.
- [Mut02] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. SODA*, pages 657–666, 2002.
- [Ukk95] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(5):249–260, 1995.
- [Wei73] P. Weiner. Linear pattern matching algorithms. In *IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.
- [ZKNY07] Liang Zuopeng, Hu Kongfa, Ye Ning, and Dong Yisheng. An Efficient Index Structure for XML Based on Generalized Suffix Tree. *Information Systems*, 32:283–294, 2007.