Table of Contents

[What is Suffix Tree? 2](#_Toc3056834)

[History 2](#_Toc3056835)

[Suffix Tree Construction 2](#_Toc3056836)

[**A naive algorithm to build a suffix tree:** 2](#_Toc3056837)

[Applications of Suffix Tree 4](#_Toc3056838)

[An Example for Understanding 4](#_Toc3056839)

[Conclusion 5](#_Toc3056840)

[Bibliography 5](#_Toc3056841)

Suffix Trees

# What is Suffix Tree?

Suffix trees constitute a well understood, extremely elegant, but poorly appreciated, data structure with potentially many applications in language processing. A suffix tree is a data structure constructed from a text whose size is a linear function of the length of the text and which can also be constructed in linear time. That the size is linear in the length of the text some 25 to 30 bytes per character of text is fairly obvious from a simple characterization of the structure. That is can be built in linear time is more surprising.

# History

Suffix trees were first introduced in[[1]](#endnote-1) a paper which Donald Knuth characterized as “Algorithm of the Year 1973". A greatly simplified version of algorithm was proposed in[[2]](#endnote-2) and[[3]](#endnote-3). Ukkonen provided the first linear-time online construction of suffix trees, now known as “Ukkonen's algorithm". This latter is generally regarded as the easiest to understand, and it may have marginal advantages in efficiency. I begin my exploration of suffix trees by considering a related but somewhat simpler structure called a *suffix trie*. A trie[[4]](#endnote-4) is a very well-known data structure often used for storing words so that they can be looked up easily. It can be alternatively named by *Digital Search Tree (DST)* to avoid hesitation about how it should be pronounced. A DST is a data structure that represents a set of strings.

# Suffix Tree Construction

### **A naive algorithm to build a suffix tree:**

Given a string S of length m, enter a single edge for suffix S[l ..m]$ (the entire string) into the tree, then successively enter suffix S[i..m]$ into the growing tree, for i increasing from 2 to m. Let Ni denote the intermediate tree that encodes all the suffixes from 1 to i.  
So Ni+1 is constructed from Ni as follows:

* Start at the root of Ni
* Find the longest path from the root which matches a prefix of S[i+1..m]$
* Match ends either at the node (say w) or in the middle of an edge [say (u, v)].
* If it is in the middle of an edge (u, v), break the edge (u, v) into two edges by inserting a new node w just after the last character on the edge that matched a character in S[i+l..m] and just before the first character on the edge that mismatched. The new edge (u, w) is labelled with the part of the (u, v) label that matched with S[i+1..m], and the new edge (w, v) is labelled with the remaining part of the (u, v) label.
* Create a new edge (w, i+1) from w to a new leaf labelled i+1 and it labels the new edge with the unmatched part of suffix S[i+1..m]

This takes O(m2) to build the suffix tree for the string S of length m.  
Following are few steps to build suffix tree based for string “xabxa$” based on above algorithm:

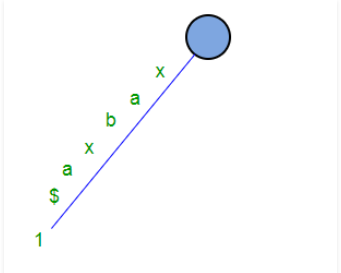
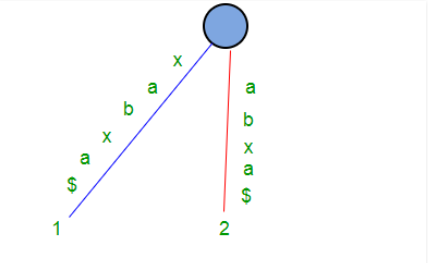
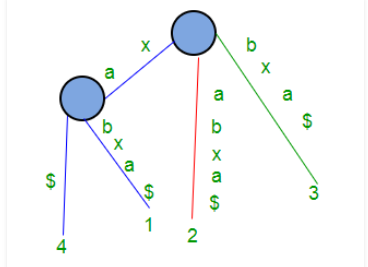


Figure 2: Tree with suffix N1, S[1..6] and N2, S[2...6]

Figure 1: Tree with suffix N1, S[1..6]



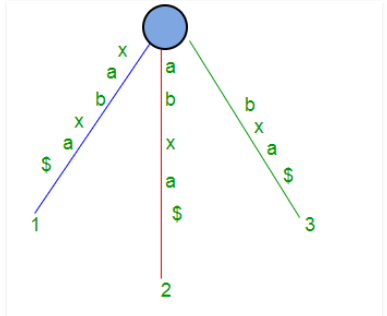


Figure 4:Tree with suffix N1, N2, N3, and N4

Figure 3: Tree with suffixes N1, N2 and N3

# Applications of Suffix Tree

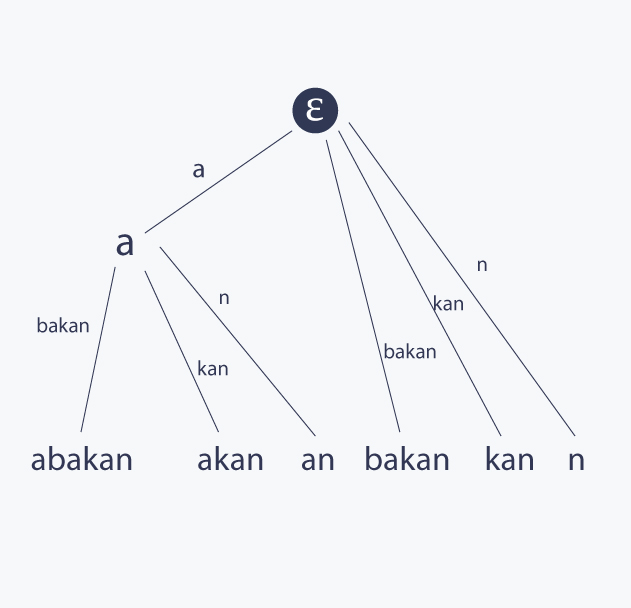
Suffix trees can be used to solve a large number of string problems that occur in text-editing, free-text search, computational biology and other application areas. Primary applications include:

* String Search, in *O(m)* complexity, where *m* is the length of the sub-string (but with initial *O(n)* time required to build the suffix tree for the string)
* Finding the longest repeated substring
* Finding the longest common substring
* Finding the longest palindrome in a string

Suffix trees are often used in bioinformatics applications, searching for patterns in DNA or protein sequences (which can be viewed as long strings of characters). The ability to search efficiently with mismatches might be considered their greatest strength. Suffix trees are also used in data compression; they can be used to find repeated data, and can be used for the sorting stage of the Burrows–Wheeler transform. Variants of the LZW compression schemes use suffix trees (LZSS). A suffix tree is also used in suffix tree clustering, a data clustering algorithm used in some search engines.

# An Example for Understanding

For better understanding, let's consider the suffix tree **T** for a string **s = abakan**. A word abakan has 6 suffixes **{abakan , bakan, akan, kan, an, n}** and its suffix tree looks like this:



There is a famous algorithm by Ukkonen for building suffix tree for s in linear time in terms of the length of s. Suffix trees can solve many complicated problems, because it contain so many information about the string itself. For example, in order to know how many times a pattern P occurs in s, it is sufficient to find P in T and return the size of a sub-tree corresponding to its node. Another well-known application is finding the number of distinct substrings of s, and it can be solved easily with suffix tree, while the problem looks very complicated at first sight.

# Conclusion

Suffix Tree is a natural way to store a string -- search, count occurrences, and many other queries answerable easily. But they are not space efficient: O(n2) space. Suffix trees are space optimal: O(n), but require a little more subtle algorithm to construct. Suffix trees can be constructed in O(n) time using Ukkonen’s algorithm. Similar ideas can be used to store sets of strings.

# Bibliography

1. P. Weiner. Linear pattern matching algorithm.14th Annual IEEE Sym-posium on Switching and Automata Theory, pages 1{11, 1973 [↑](#endnote-ref-1)
2. Edward M. McCreight. A space-economical suffix tree construction algorithm. J0ournal of the ACM, 23(2):262{272, 1976 [↑](#endnote-ref-2)
3. E. Ukkonen. On-line construction of suffix trees. Algorithmica,14(3):249{260, 1995. [↑](#endnote-ref-3)
4. Edward Fredkin. Trie memory. Communications of the ACM, 3(9):490{499, September 1960.

   1. Earth, H. (n.d.). *Trie, Suffix Array, Suffix Tree*. Retrieved from Hacker Earth: https://www.hackerearth.com/practice/notes/trie-suffix-tree-suffix-array/
   2. Geeks, G. f. (n.d.). *Ukkonen Suffix Tree Construction - Part 1 - GeeksforGeeks*. Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-1/
   3. Shamir, P. R. (n.d.). *Cupid - Suffix trees.* Retrieved from Tel Aviv University: https://www.cs.tau.ac.il/~rshamir/algmb/presentations/Suffix-Trees.pdf
   4. Stanford. (n.d.). *Suffix Tree.* Retrieved from Stanford University: https://web.stanford.edu/~mjkay/suffix\_tree.pdf
   5. University, C. M. (n.d.). *Suffix Trees.* Retrieved from Carnegie Mellon University: https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/suffixtrees.pdf
   6. Wikipedia. (n.d.). *Suffix tree*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Suffix\_tree

   [↑](#endnote-ref-4)