Suffix Tree

aka: Type of Compressed Trie / Position Tree / Suffix Trie

Overview

Suffix Tree is compressed trie of all suffixes in given text T.

Representation

Suffix tree is constructed following below steps:

1. Generate all suffixes of given text.
2. Consider all suffixes as individual words and build a compressed trie.

Let us consider an example text `banana$` where `$` is string termination character not there in the input string. This ensures that no suffix is a prefix of another, and that there will be n leaf nodes. Following are all suffixes of `banana$`

```
banana$
anana$
nana$
ana$
na$
a$
$
```

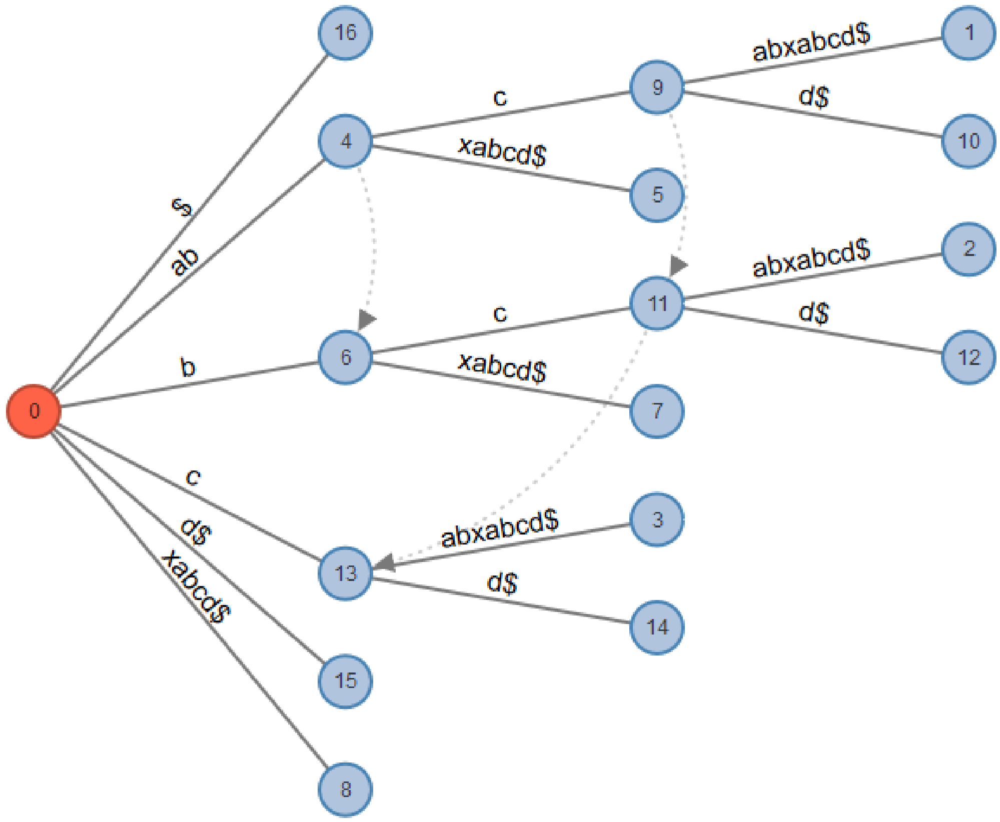
Next *Ukkonen's algorithm* is explained which is a linear-time, *online algorithm* for constructing suffix trees.

Ukkonen's Algorithm

Reference Document

Ukkonen's algorithm is divided into m phases (one phase for each character in the string with length m). In phase $i+1$, tree $T_i + 1$ is built from tree T_i. Each phase $i+1$ is further divided into $i+1$ extensions, one for each of the $i+1$ suffixes of $S[1 \,..\, i+1]$.

Input `abcabxabcd$`



Time and Space Complexity

The naive implementation for generating a suffix tree requires $O(n^3)$ time complexity, where n is the length of the string. ***Ukkonen's algorithm*** reduced this to $O(n)$ time, for constant-size alphabets, and $O(n * log(n))$ in general. If each node and edge can be represented in $\Theta(1)$ space, the entire tree can be represented in $\Theta(n)$ space. The total length of all the strings on all of the edges in the tree is $O(n^2)$, but each edge can be stored as the position and length of a substring of S, giving a total space usage of $\Theta(n)$.

Note: Comparing to other popular pattern search algorithms (KMP, Rabin-Karp, DFA, Boyer-Moore) after the pre-processing step the worst time complexity for searching a pattern in text is $O(n)$ where n is the length of the text. While with suffix tree after pre-processing step it take $O(m)$ time where m is the length of the pattern.

Applications

Suffix tree can be used for a wide range of string problems, following are some of the primary applications.

- String search, in $O(m)$ complexity, where m is the length of the sub-string (Initial $O(n)$ time required to build the suffix tree from text of length n).
- Finding the *longest repeated substring*.
- Finding the *longest common substring*.
- Finding the *longest palindrome* in a string.
- Suffix trees are often used in bio-informatics applications, searching for patterns in DNA or protein sequences.
- Variants of the LZW compression schemes use suffix trees.

References

- Data Structures and Algorithms in Java Book
- Suffix Tree Wikipedia
- Animation explaining Ukkonen's Algorithm
- StackOverFlow thread