Advanced Algorithms
Assignment 2

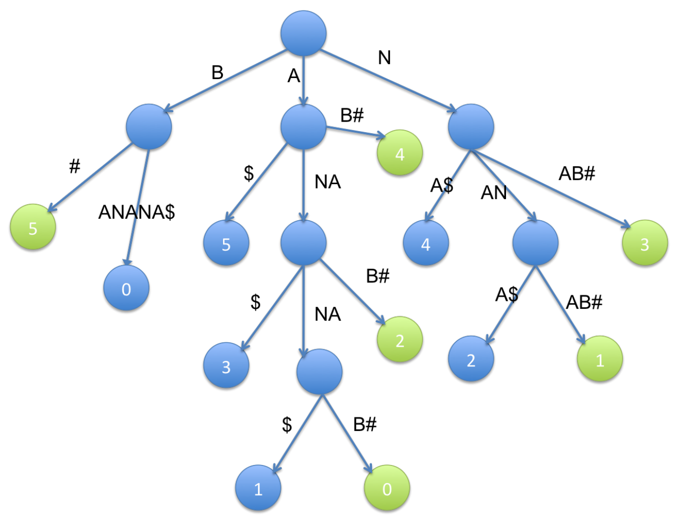
# Suffix trees - Construction and Pattern Matching

Sumanth S Rao

USN - 01FB15ECS314

14th October, 2017

# Introduction



A Suffix Tree for a given text is a compressed trie for all suffixes of the given text.Here we build a Suffix tree for the given dataset which is a document of tales.

## The Dataset

The Data set consists of a set of tales from Aesop's fables,each tale separated by double line.

## Dataset Input

I have taken the Input as StringBuilder in java , where each of the tale is separated by a '$' for easier processing.

I have also taken a HashMap of the Titles of all the Titles of the tale where each tale is indexed by its position in the StringBuilder of all tales.

## Tree Construction

**Method - Naive-Construction**

**Time Complexity - construction - O(n^2)**

**Input - Tales as strings terminated by a '$'**

### My Approach

- I am considering each tale as a string for insertion in the Generalised suffix tree terminated by a '$'.
- For each suffix, start at the root node and match the longest prefix of the suffix, moving down the tree and chop off that part of the suffix.
- if we reach the leaf node:
    - Add that suffix to the leaf
- Else
    - Split the node that is reached and add a new edge for that new suffix.

## Array of suffix trees

In case of questions 2 and 3 asking for results talewise, I am creating an array of suffix trees with one suffix tree for each tale using read_talewise().

## Time Complexity

We have to consider all the possible suffixes of the given string of length ‘n’.

Hence the outer loop runs ‘n’ times.

Considering each suffix Si, We match it with the k characters of of the nodes already added in the suffix tree.(Longest Common prefix whose length is k)

i,e k comparisons.(k<i)

The remaining ‘j’ characters that is Sk+1 to S0 has to be read and added in the new leaf node. AT each level the complexity is O(i).(i=k+j)

The total complexity therefore is

= (n+(n-1)+(n-2)...0          n-length of string

= O(n^2)

## Pattern Matching

My idea :     Each edge in the tree holds three parts.

1)Beginning index     2).Ending index    3).The leaf Nodes it has.

Here the beginning and ending index refers to the position of the substring the edge holds in the stringbuilder used to construct the tree.I have a locate function that matches the pattern against the prefixes in the Tree.The collect node function returns the indices of occurance of the pattern in the string by doing a **DFS** of all the leaf nodes in the matched nodes.

- **PrefixOK** flag can be set to TRUE that will return the leaf nodes even if a prefix of the pattern matches in the tree.Hence the collect function behaves **POLYMORPHICALLY**.

## The Heuristics used for Relevance.

- Match each word in each document,if the match is not found then match the longest prefix and return the same.
- The document with highest exact word matches will have a higher relevance than the one with partial matching(A prefix of the word matching)
- The rank is based on the total number of characters matched in the whole query string in each document.Considering the total words or prefix of words matched in the query.
- **RELEVANCE PERCENTAGE -Relevance score is calculated as**

**total characters that matched ( q )*100 / total length of the query string(n).**

- **THE DOCUMENT MATCHING THE WHOLE QUERY IN THE SAME ORDER HAS TO HAVE A RELEVANCE OF 100%** .If the whole query is not found then split the query into words and look for the number of words matching or prefixes matching.
- In the case of multiple words matching then if the ordering of the words in the original query is same as the matched then ,It will have a higher relevance .(number of chars matched + 1 for the ordering)

## Attachment guide

FILES ATTACHED

1).SuffixTree.java  - Main class consisting of methods

main(), read(), read_talewise(), test() .

2).MySuffixTree.java  - Subclass consisting of methds for tree construction -insert(), locate(),FindAll(),LongestPrefix(),collect() .

3).OUTPUT 1  ,  OUTPUT 2  ,  OUTPUT 3 -The outputs of the questions run on test cases for questions 1,2,3.

## Acknowledgement

I’d like to thank my professor N S Kumar for the thoughtful and knowledge enriching assignment which helped me in evolving my programming skills.A lot of hard work went to the construction of suffix tree function and also the matching function.I would say building the insert function and defining the node structure was the toughest part of this assignment.