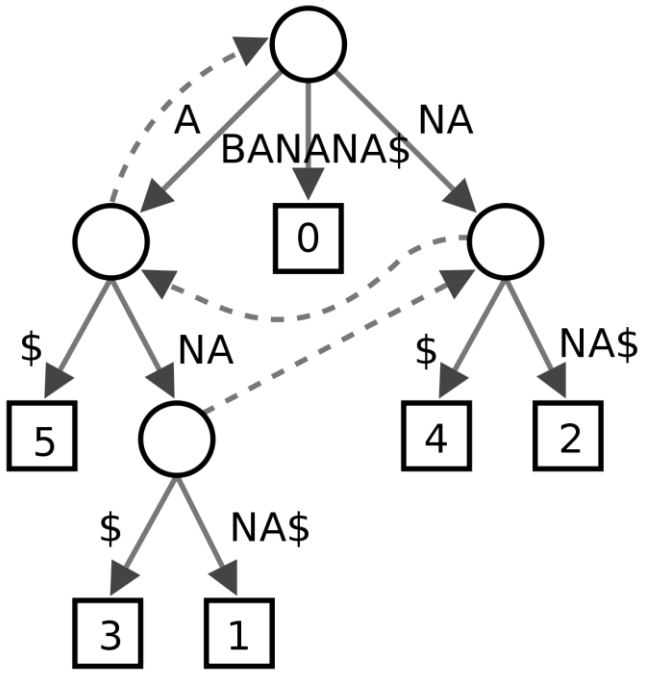
Suffix Trees

Abstract
In computer science, a suffix tree is a compressed trie containing all the suffixes of the given text as their keys and positions in the text as their values.Substring searching can be done in Linear time .It can be used to list all the occurrences of a query string in the document,give the first occurrence of a given query-string,rank the list of documents in order of relevance.



Preprocessing

For taking a whole content of a tale as a string, we remove all the newline characters from each content . To distinguish between the content and next story, I am putting a '$' character .This '$' character tells that the end of the a tale is reached and a new tale is starting.
For removing the newline characters, I am going through each of the character in input File 'AesopTales.txt' and putting it in the 'out.txt' unless it is '\n' and checking the number of newline characters.If there are three ,then we adding the '$' to the 'out.txt' in the next line.

Preprocessing the input file file takes O(n) time where 'n' is the length of the all the tales int the AesopTales.txt.

Suffix Tree Construction

Algorithm Used: Ukkonen's Suffix Tree Construction, from geeksforgeeks as reference.

Approach For First Sub-Problem

After building a suffix tree ,we will traverse the edge character by character .When the substring is matched ,we will count the no.of leaf nodes. The no. Of leaf nodes will give the no. of substrings matched and we can get the index of it as well.

For printing the line in which we have the matched substring.I have checked for previous fifty characters and post 50 characters from the index where the substring is found.And check for a secial terminating characters ie. ' . ', ' ? ', ' ! ' , ' , ' etc. And print the line between that.

Time Complexity is O(m+z) where m is the length of the substring & z is for the dfs .

Approach For Second Sub-Problem

For every suffix of the query string we will check the no. of characters of the prefixes that matched for that suffix. We keep a record of that .
We search for each suffix the no_of_matches. The max_matches will give us the max no. of characters in substring that is matched.
If the no_of _matches is equals to the max_matches. That is the longest substring of the query string in that document.Then we print the first occurence of it.

Time Complexity is O(m^2) for each document ,where m is the length of the substring.
It took a total of 9.2 msec.

Approach For Third Sub-Problem

We will first find the words in the query string(tokenizing the query string).
Then we keep an array to store the count of how many words of query string found in the document.And also we have an array to keep the index of the stories. So, that stories can be mapped to the found word.
Then we will sort the no_of_words_found and the stories respectively in descending order.
The document at the first index is the most relevant document.
Then we print it.

Time Complexity is O(p^2)*O(n) for each document ,where p is the length of the token,n is the no of tokens in the query string.