REPORT-ASSIGNMENT 2

(S.Swetha-01FB15ECS250)

Suffix tree construction:(function \_\_init\_\_ in class SuffixTree)

Class SuffixTree has a Class Node whose members are the starting and ending index of the label of the node,a dictionary of all outgoing edges,and a list of all occurances of the string from the root to the leaf. This list is empty for internal nodes.

For all the three cases, a suffix tree is created for each document(story i.e only the content, titles are stored in a separate list).

If ‘l’ is the length of a story, then the complexity of the tree creation would be O(l^2).

In the best case,(i.e all the characters of story is unique) complexity would be

O(l).

I have referred to the following link for suffixtree creation-http://www.cs.jhu.edu/~langmea/resources/lecture\_notes/suffix\_trees.pdf

followpath:(function followpath)

Given a query string ‘s’ ,the complexity to find the node corresponding to the given query string is O(l1\*dif) where ‘l1’ is the length of the query string ‘s’ and ‘dif’ is the length of the label of a node.

DFS:(function dfs and dfs1)

If the given query string is not a suffix of the story then the node corresponding to the query string is an internal node. In that case, in order to find all the occurances of the query string, a dfs is done to find all it’s leaves.

If ‘k’ is the total number of leaves below the internal node, then the complexity to do a dfs would be O(k).

Printing result:(function printres)

If story index is ‘i’, index of query string is ‘j’, length of query string is’l1’ and length of story is ‘l’, we need to find the sentence in which the query string is present( sentences if it is spanned across multipe sentences. e.g ‘present.What’ which would be spanned across two sentences).

If ‘p’ and ‘q’ is the index of the immediate fullstop before ‘j’ and after ‘j+l1-1’ respectively, then the complexity to find ‘p’ and ‘q’ would be

O((j-p)+(q-(j+l1-1)))=O(q+1-(l1+p))

Complexity to print string spanned from ‘p’ to ‘q’ would be O(q-p).

Hence overall complexity is approximately O(q-p).

Finding all occurances of query string ‘s’:(function alloccur)

Let ‘l’ be length of story,’l1’ be length of query string and ‘dif’ be length of label at a node.

For every story, function alloccur(s,c) is called where ‘c’ is the story index.

In the function, function followpath is called whose complexity is O(l1\*dif).

If the returned node is not a leaf node, then a dfs has to be done to find all the leaf nodes.

Funtion dfs is called whose complexity is O(k\*(q-p)) where ‘k’ is the total number of leaves below the internal node , and q and p are the indices of fullstop before and after query string.

Hence overall complexity of function ‘alloccur’ is O(l1\*dif +k\*(q-p)) for every story.

Finding first occurance of a query string ‘s’:(function firstoccur)

Find 1st occurance of a string ‘s’,if it is not present find first occurance of longest substring of ‘s’ in the story.

In the best case, if query string ‘s’ is suffix in the story, the complexity would be O(l1\*dif+q-p) approx. O(l1\*dif)- l1\*dif for followpath and q-p for printing result.

If query string ‘s’ is a substring but not a suffix, the complexity would be O(l1\*dif+k+q-p) approx O(l1\*dif)-l1\*dif for followpath, k for dfs and q-p for printing result.

If ‘s’ not in story, we need to first generate all substrings and order them based on length-this has a complexity of O(l1^2).

If ‘n’ is the number of substrings, then complexity would be O(l1^2+n\*l1\*dif) approximately for every story.

Rank documents based on relevance:(function relevance)

**Heuristics:**Stories that have exact query string have highest rank.Stories that have all words of querystring have higher relevance than those which have a subset of the words.

e.g : Let query string ‘s’ have n words. Story 1 has higher relevance than Story 2 if it matches n-1 words of ‘s’ while Story 2 matches n-2 words of ‘s’.

Let querystring s=’hello world’.

If story s1 has the querystring s it is ranked 1.

If story s2 has the words ‘hello’ and ‘world’ not in any particular order,then it is ranked 2.

If story s3 has the word ‘hello’ alone(could be present any no. of times),it is ranked 3.

Stories that match same number of words of the query string will have same relevance. Hence many stories can have the same rank.

First, all the words of query string are generated-complexity associated is O(l1) where l1 is length of query string.

For every story,function ‘relevance’ is called.

In function relevance,function followpath is called whose complexity is O(l1\*dif). If it is not present in the story then function followpath is called for every word of the query string.If the word is present in the story a variable no\_occur is incremented. This variable is returned by the function. The returned value is stored in a dictionary rel.

Hence complexity of function relevance is O(l1\*dif+n\*l2\*dif) approximately O(n\*l2\*dif)-where n is the number of words and l2 is length of word.

For k stories, complexity is O(k\*n\*l2\*dif).

The dictionary rel is sorted based on it’s values and they are ranked accordingly.