Search pattern and occurrences in document (McCreight's Algorithm)

By : Rajat Jain (4446187) , bohrarajat@knights.ucf.edu

***Abstract*—In this paper, search using McCreight's algorithm is implemented. The main aim of McCreight's algorithm is to construct suffix tree in linear time. A suffix tree of a string A, is a compressed trie of all the suffixes of the string A$. A suffix tree is a data structure which is built from a string of linear function of length of the string which is constructed in linear time.**

I. Introduction

A suffix tree for a n character string S is a rooted directed tree. It has exactly n number of leaves from 1 to n. Other than root, each internal node has at least 2 children. Each edge is labeled and two edges cannot have edge labeled starting with same character. $ is concatenated with the given string S to assure that no suffix of the string S matches a prefix of another suffix of the string S and suffix tree is constructed for the the string S$. Suffix tree is built by generating all the suffixes of the given string. All suffixes are considered as individual words and compressed trie is built. Suffix tress are used for pattern searching in a greater set of strings. It is also used for the search to be efficient.

II. McCreight's algorithm

Suffix tree construction starts with an empty tree and repeatedly insert other suffixes. Suffix tree is formed when all the suffixes are inserted. Suffix links are used to increase the speed of the insertion of suffixes. McCreight's algorithm inserts suffixes in the order from 1 to n. To form tree T_i, $suffix_i$ is inserted into tree T_{i-1}. External node is created when the traversal does not end at an internal node. The total run-time of the algorithm is $O(n^2)$. Suffix links are used to achieve linear time complexity. McCreight's algorithm is auxiliary search tree which is used to find specific sub string in the given main string.

McCreight's algorithm is similar to the brute force algorithm with but it creates suffix links which is helpful to achieve linear time complexity. The main problem of the algorithm is to find the extended locus T_{i-1} if $head_i$, which is the longest prefix of $Suffix_i$. To over come this problem, suffix links are used. Suffix links are also useful when we insert $Suffix_i$ in T_{i-1} and v be the internal node, then $path-label(v) = c\alpha$ is a prefix of $Suffix_{i-1}$ and there must be another suffix $Suffix_j$ which also contains $c\alpha$ as its prefix. Since the previous suffix of $Suffix_j$ is already inserted, there is a α path from root in T_{i-1}. To insert suffix quickly, if the end of the label α is found, the comparison of characters in suffix can begin after the prefix α.

III. Algorithm

```
Input: text T[0..n] (T[n] = $)
Output: suffix tree of T: root, child, parent, depth, start, slink
(1)  create new node root; depth(root) ← 0; slink(root) ← root
(2)  u ← root; d ← 0        // (u, d) is the active locus
(3)  for i ← 0 to n do      // insert suffix T_i
(4)      while d = depth(u) and child(u, T[i + d]) ≠ ⊥ do
(5)          u ← child(u, T[i + d]); d ← d + 1
(6)          while d < depth(u) and T[start(u) + d] = T[i + d] do d ← d + 1
(7)      if d < depth(u) then      // (u, d) is in the middle of an edge
(8)          u ← CreateNode(u, d)
(9)      CreateLeaf(i, u)
(10)     if slink(u) = ⊥ then ComputeSlink(u)
(11)     u ← slink(u); d ← d − 1
```

Fig. 1: McCreight's algorithm

```
ComputeSlink(u)
(1)  d ← depth(u)
(2)  v ← slink(parent(u))
(3)  while depth(v) < d − 1 do
(4)      v ← child(v, T[start(u) + depth(v) + 1])
(5)  if depth(v) > d − 1 then      // no node at (v, d − 1)
(6)      v ← CreateNode(v, d − 1)
(7)  slink(u) ← v
```

Fig. 2: Suffix Links

IV. Implementation

My implementation includes :

1. Creation of suffix tree using McCreight's algorithm
2. Implemented search in data using suffix tree
3. Counted occurrences search keyword in given dataset
4. Displaying all the sentences where search keyword has occurred
5. Below are the graph showing performance of my program

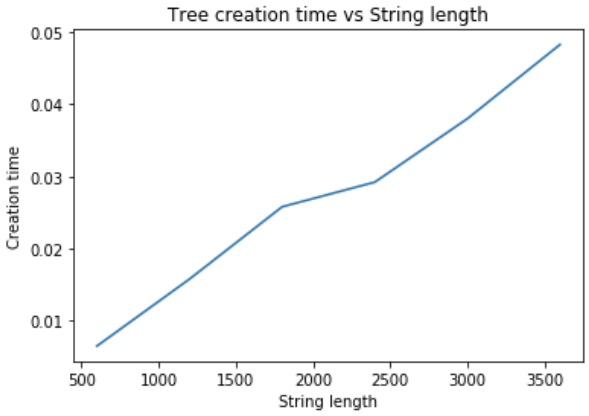

Fig. 3: Suffix Tree creation time

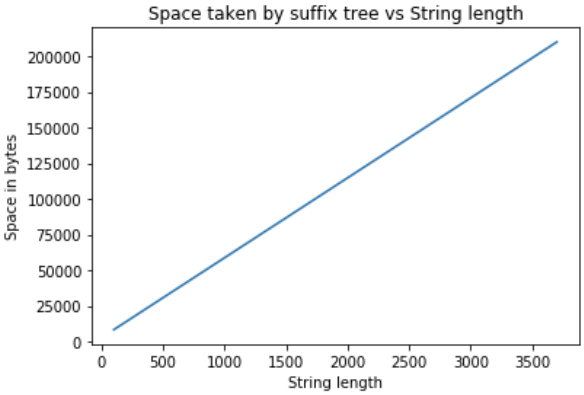

Fig. 4: Space for creation of suffix tree

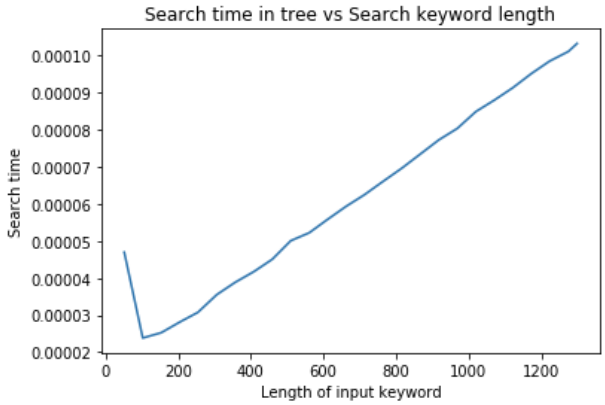

Fig. 5: Search pattern in created tree

V. Conclusion

In general, the suffix tree built using McCreight's algorithm does not differ from other suffix trees built by using Ukkonen's and Weiner's method. They also construct suffix tree in linear time based on the length of the input string. The main advantage of McCreight's algorithm is that it requires less space compared to that of other algorithms. Saving the data space implies saving time.

References

[1] McCreight, Edward M. "A space-economical suffix tree construction algorithm." Journal of the ACM (JACM) 23.2 (1976): 262-272.

[2] Weiner, Peter. "Linear pattern matching algorithms." Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on. IEEE, 1973.

[3] Ukkonen, Esko. "On-line construction of suffix trees." Algorithmica 14.3 (1995): 249-260.

[4] Aluru, Srinivas. Suffix Trees and Suffix Arrays. Handbook of Data Structures and Applications (2004).

[5] Knuth, Donald E., James H. Morris, Jr, and Vaughan R. Pratt. "Fast pattern matching in strings." SIAM journal on computing 6.2 (1977): 323-350.

[6] Ukkonen, Esko. "Approximate string-matching over suffix trees." Annual Symposium on Combinatorial Pattern Matching. Springer, Berlin, Heidelberg, 1993.