Suffix Trees

Hitesh Kumar
PES1201701511
PES University
Bangalore , India
hiteshkumarhk.info99@gmail.com

***Abstract*—Problem Statement : In computer science, a suffix tree (also called PAT tree or, in an earlier form, position tree) is a compressed trie containing all the suffixes of the given text as their keys and positions in the text as their values. Suffix trees allow particularly fast implementations of many important string operations. The problem statement is to implement a generalized suffix tree for any number of given documents. For a given pattern the algorithm is supposed to return the index of the first occurence of the pattern in each document. If the pattern is not found in any document, and a substring of the document is found, we return the index of the longest occuring substring.**

I. Introduction

Suffix Tree is very useful in numerous string processing and computational biology problems. Many books and e-resources talk about it theoretically and in few places, code implementation is discussed. But still, I felt something is missing and it's not easy to implement code to construct suffix tree and it's usage in many applications. This is an attempt to bridge the gap between theory and complete working code implementation. Here we will discuss Ukkonen's Suffix Tree Construction Algorithm. We will discuss it in step by step detailed way and in multiple parts from theory to implementation. Ukkonen's algorithm constructs an implicit suffix tree Ti for each prefix S[l ..i] of S (of length m). It first builds T1 using 1st character, then T2 using 2nd character, then T3 using 3rd character, . . . , Tm using mth character. Implicit suffix tree Ti+1 is built on top of implicit suffix tree Ti. The true suffix tree for S is built from Tm by adding $. At any time, Ukkonen's algorithm builds the suffix tree for the characters seen so far and so it has on-line property that may be useful in some situations. Time taken is O(m).

II. Implementation

A. Approach

In Suffix Tree Construction of string S of length m, there are m phases and for a phase j (1 ¡= j ¡= m), we add jth character in tree built so far and this is done through j extensions.

To do jth extension of phase i+1 (adding character S[i+1]), we first need to find end of the path from the root labelled S[j..i] in the current tree. One way is start from root and traverse the edges matching S[j..i] string. This will take O(m3) time to build the suffix tree. Using few observations and implementation tricks, it can be done in O(m) which we will see now.

We will have SuffixTreeNode structure to represent each node in tree. SuffixTreeNode structure will have following members:

children – This will be an array of alphabet size. This will store all the children nodes of current node on different edges starting with different characters. suffixLink – This will point to other node where current node should point via suffix link. start, end – These two will store the edge label details from parent node to current node. (start, end) interval specifies the edge, by which the node is connected to its parent node. Each edge will connect two nodes, one parent and one child, and (start, end) interval of a given edge will be stored in the child node. Lets say there are two nods A (parent) and B (Child) connected by an edge with indices (5, 8) then this indices (5, 8) will be stored in node B. suffixIndex – This will be non-negative for leaves and will give index of suffix for the path from root to this leaf. For non-leaf node, it will be -1 .