SUFFIX TREES

Vibha Satyanarayana, 01FB15ECS346

Abstract—Description of suffix trees and their possible applications. Implementation of a suffix tree using naive construction algorithm. Application of the suffix tree for efficient pattern matching in the given document.

Index Terms—, Pattern matching ,substring,suffix trees, suffix tries

1 INTRODUCTION:

SUFFIX TREE: A suffix tree is a compressed trie containing all possible suffixes of a given text as its nodes.The position of the suffix in the text is the value of its node. This makes the tree space linear. Suffix trees allow a multitude of sophisticated operations to be performed efficiently.

APPLICATIONS: Given a query q to a suffixtree constructed using text T, we could check if q is a substring of T, find the suffix of q in T, find the number of occurrences of q in T,find the longest repeat in T, find the lexicographically first suffix etc

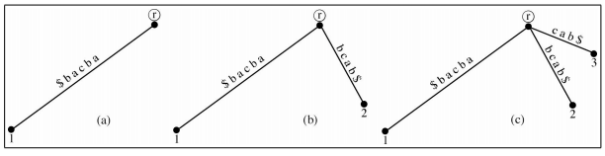
GENERALIZED SUFFIX TREE: A generalized suffix tree is a suffix tree for a set of strings, each marked by a delimiter. If there are n strings S1,S2,S3...Sn of total length m, the generalized suffix tree is created using all m suffixes. Each string has a unique delimiter to identify it.

2 DEFINITION:

Given a string S of length m, its suffix tree is defined such that:

- it has exactly m leaves numbered from 1 to m
- every internal node has at least two children, except the root.
- each edge is labeled with a substring of S
- no two edges starting from a node can begin with the same character
- the string obtained by concatenating all the string labels found on the path from the root to leaf i is the suffix S[i..m] where 1<i<m

3 CONSTRUCTION ALGORITHM - NAIVE:



Constructing first 3 suffixes of abcab$

Let T (S) be the suffix tree of string S. The algorithm works in an incremental way, by processing the m suffixes S[i..m] of S, from i = 1 to m, one by one. Let Ti be the tree obtained at step i. We need to construct Ti+1 from Ti. Tree Tm will be the final tree T (S). At the first step, S[1..m]$ is inserted in an empty tree and suffix tree T1 is composed of a unique node. At step i+ 1, suffix S[i+ 1..m]$ is inserted in Ti as follows: traverse the tree starting at the root r and find the longest prefix that matches a path of Ti. If such prefix is not found, this is because none of the previous suffixes S[j..m]$, for j = 1 to j = i, starts by character S[i + 1]. In this case, a new leaf numbered i + 1 with edge-label S[i + 1..m]$ is created and joined to the root. Thus, assume there is a path such that S[i + 1..m]$ is a prefix of maximal length of that path. Because of the presence of the termination character $, the prefix cannot be a substring of a suffix entered into the tree early on. Therefore, there is a character at position k such that S[i + 1..k]$ is the prefix; let S[k..m]$ be the non-matching substring. Let (u, v) be the edge of the path where character S[k] is found. The algorithm creates a new node w and a leaf numbered i + 1. Node w splits edge (u, v) into edges (u, w) and (w, v), and edge joining w and leaf i + 1 receives edge-label S[i + 1..k]$. The algorithm finishes when the termination character is inserted.

4 IMPLEMENTATION:

A generalized suffix tree was constructed for all documents. Each document was delimited by a '$' symbol. The tree was implemented using a class where each node is its object. Each edge is stored using the start index and end index of the substring it represents. Each node contains information about its incoming edge. This information includes the following:

- serial number of the documnet the node belongs to
- start and end index of the edge (stored in a structure)
- a variable to store the number of matches made per edge during construction
- other variables used for debugging (eg: queue used for level order traversal of the tree)

CONSTRUCTION:

Every suffix of each document is sent to the construct function. The construct function sends every pattern sent to it, to the findNode function along with the current active node. The function findNode checks every child of the current node and returns the matching node. There are 3 possibilities:
1. findNode finds no matching child - In this case, findNode returns NULL and construct has to create a new node and make it the child of the current node.

2. findNode finds a perfect match (highly unlikely because the tree is built documents wise and there cannot be two documents with the exact same characters) - In this case, findNode returns the matched child. Construct creates a new leaf node with the new document details and makes the node returned by findNode its parent.

3. findNode finds a partial match - In this case, findNode returns the node containing the partially matched edge along with the number of matched characters. Construct then creates a new node with the characters matched so far. This becomes the current node. The old node is split into 2 parts - one child with the remaining unmatched part of the old node and one child with the remaining unmatched part of the newly added word.

PREPROCESSING:

The following preprocessing was done on the input file before the construction of the tree:

- All titles were stored in a seperate text file
- All newline characters were removed
- Each document was delimited by '$'
- All titles were delimited by '.' and were made a part of the rest of the document
- A new file was created for every document for easier access

SEARCH:

1. Search finds the first occurrence of the query or its subquery (if the query doesnt exist) and returns its document number. This is used by findTitle to find the the title of the document it belongs to. Then the document is scanned for the query and the matched text along with the surrounding text is returned.

2. SearchForAll finds all occurrences of the query in all documents. It finds the node upto which the query matches and performs a depth first search to find all documents to which the query (or subquery) belongs. These numbers are stored in a list and is then used to display all the titles and matched text (as done previously).

RELEVANCE OF DOCUMENTS:

- Documents with a higher document number are listed first (i.e. most recently constructed).
- Documents with more number of matches are listed first.
- Query match is listed before subquery match

5 ANALYSIS:

Let m be the size of the entire file and let n be the average size of every document.

1. Preprocessing: Text clean up and document creation takes O(m) time (assuming creating a document takes constant time).
2. Storing all characters in an array for index reference- O(m)
3. Tree construction by sending every suffix in the document to the construct function: $O(n^2)$ per document
4. Search - O(n + z) where z is the number of leaf nodes
5. Finding surrounding matched text: $O(n^2)$

6 RESULT:

Query string: "much ado about nothing"

Function	Time (in milli seconds)
Construction	725111
First Occurrence	171
All documents containing the query string	8
All occurrences	63