Name: Aviral Joshi
SRN: 01FB15ECS062

Assignment - 2 on Suffix Trees - UE15CS311 (AdvAlgo) (October 2017)

Aviral Joshi
Student, PES University
joaviral@gmail.com

Abstract

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. We will start with brute force way and try to understand different concepts, tricks involved in Ukkonen's algorithm and in the last part, code implementation will be discussed.

1. Introduction

The AesopTales Dataset is a dataset of short tales which will be used as the primary source of reference for this article. However the algorithm provided can be used over any document.

Problem Statement:

Given a set of documents, build a suffix tree to perform the following operations.

1. List all the occurrences of a query-string in the set of documents.
2. List only the first occurrence of a given query-string in every document. If the query-string is not present, find the first occurrence of the longest substring of the query-string.
3. For a given query-string (query of words), list the documents ranked by the relevance.

2. Implementation

Suffix Trees were built using the Ukkonen Algorithms.
Class Node, SuffixTree and Documents were created.
The Node class represents a single node in the suffix tree
The SuffixTree class is an abstract representation of the suffix tree.
The Document class uses the SuffixTree and Node class internally to solve problems previously mentioned.

3. Approach

The data-set contains 312 tales of not more than 150 words.
The Data is loaded. For each Tale in the data-set a new suffix tree is built.

A) List all the occurrences of a query-string in the set of documents.
The query to be searched for in the document is inputted and searched in all the 312 suffix trees.
The Document classes method all_in_all implicitly calls the find method of the SuffixTree class to retrieve all the matched indexes. The index is used to identify the position of the word in the

Name: Aviral Joshi
SRN: 01FB15ECS062

respective tale and the circumscribing sentence is extracted.
All the sentence containing the query word along with the name of the tale are returned and displayed.

B) List only the first occurrence of a given query-string in every document. If the query-string is not present, find the first occurrence of the longest substring of the query-string.

A similar approach to problem 1) is adopted here but when no matching strings are found in the suffix tree of a tale the longest common substring is searched for.
The Documents class call the lcs function (aka longest common substring) to determine the substrings that match the query-string.
The result from the lcs function is appended to the result from the find function and is returned to the user.

C) For a given query-string (query of words), list the documents ranked by the relevance.

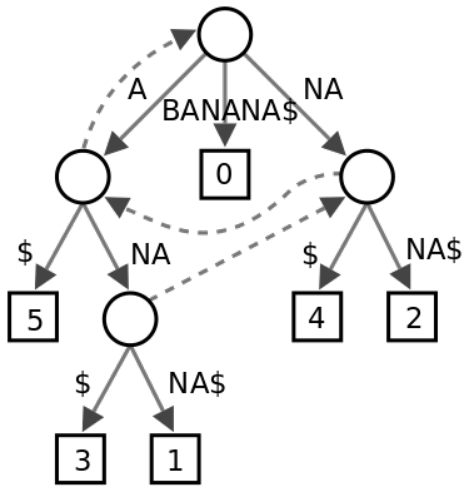
Relevance can be thought of as the closeness or similarity between the document and the query-string. To determine this the find method and the lcs method are used in combination and a score is assigned based on the measure of similarity.

Score of 10 is given if an exact match is found.
Score of 0.1* length(substring) is given if a matching substring is found.

Total Score = Score for exact match + Score for substring match

4. Representation and Diagrams

Suffix Tree:



1) Fully Connected/ Strong Lines:
Direct Transition from parent to leaf/internal node

2) $:
Global End i.e. end of the suffix representing the end of the suffix of the input string

3) Dotted Lines:
Suffix Links

5. Conclusion

Run-Time Analysis

Building Suffix Trees:
3.1 s ± 15.7 ms

List all the occurrences of a query-string in the set of documents.

1.83 ms ± 5.8 µs

List only the first occurrence of a given query-string in every document. If the query-string is not present, find the first occurrence of the longest substring of the query-string.
9.63 ms ± 23.7 µs

For a given query-string (query of words), list the documents ranked by the relevance.

Name: Aviral Joshi
SRN: 01FB15ECS062

9.74 ms ± 10.1 µs

The Above Results show the versatility of Suffix Trees in searching for words in a large piece of text.Suffix Trees can be used to efficiently search through large text of data to find matching patterns ,common substrings and much more.

6. References

1)
http://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-1/

2)
https://www.youtube.com/watch?v=aPRqocoBsFQ&t=3461s

3)
http://brenden.github.io/ukkonen-animation/