Suffix trees

TDT4187 – Exercise 3 October 11, 2017

Suffix Tree Construction & Features

1) Create the suffix tree for the string ω = GCACGCA using Ukkonen's algorithm. Note down the implicit suffix tree after execution of each phase.

   (a) Create keyword tree for the same string, and pinpoint the differences. Explain why suffix tree is more efficient both in time and space than a keyword tree.

   (b) Consider case where the suffix tree was created using high-level Ukkonen algorithm (slide 4/26; all extensions explicit). Which extension rules were used for each suffix? Pinpoint steps where the efficient version of Ukkonen algorithm performs explicit extension (as opposed to implicit).

   (c) In which steps of the efficient Ukkonen algorithm is the implicit suffix tree also a genuine suffix tree?

2) Consider the following set of strings $\{S_i = a^i b^i a^{i-1} b^{i-1}\}_{i \in \mathbb{N}}$. Consider the suffix tree for arbitrary word from this set. Determine (asymptotically) the total length of edge labels in the suffix tree (w.r.t. the length of the word). That is, how many letters would be necessary to write down all the edge labels explicitly? If your result is in $\omega(n)$, what does it tell about the efficient Ukkonen algorithm?

   hint: The difficulty lies in some branches of the tree being shared by a number of suffixes of the word. Try to focus on branches where you know the number of suffixes sharing them.

antonink@idi.ntnu.no

NTNU
Det skapende universitet

Algorithms

3) You are given string ω_1 and string ω_2, both of size n, over fixed-size alphabet Σ. For $\mathcal{O}(n)$ preprocessing time, you then want to be able to answer in $\mathcal{O}(|P|)$ time for any given pattern P over the same alphabet, whether $P \in \mathcal{D}$, where $\mathcal{D}$ is set of substrings of ω_1 not contained in ω_2. Design an algorithm that acheives this.

*4) Consider the `All pairs prefix-suffix matching problem` (slide 22 - Suffix tree applications). Design an algorithm for this problem which doesn't use Suffix trees. The algorithm shall have a complexity at most $\mathcal{O}(k \cdot m)$, where k is the number of patterns and m is the total length of the patterns.

hint: Linear time pattern search algorithm, such as KMP algorithm, could be useful.

Some more algorithms

*5) A string $\mathbf{p} = p_1...p_n$ is a palindrome if it spells the same string when read backward; that is, $p_i = p_{n+1-i}$. Design an efficient algorithm for finding all palindromes (of all lengths) in a text.

hint 1: Lowest Common Ancestor (LCA) retrieval for suffix trees can be queried for two vertices v_1, v_2 at constant ($O(1)$) time, at the cost of $O(n)$ preprocessing on the suffix tree.

hint 2: Longest Common Extension of two suffixes S_i, S_j can be queried in O(1), using the LCA preprocessing.

Jones & Pevzer: p. 337, Problem 9.7

6) Design an efficient algorithm for finding the longest exact repeat within a text.

Jones & Pevzer: p. 337, Problem 9.8

7) Design an efficient algorithm for finding the longest exact tandem repeat within a text.

Jones & Pevzer: p. 337, Problem 9.9