In this paper are proposed a couple of algorithms for computing the LPrF tables, which are used (as per conclusion) in several contexts such as string algorithms, text compression or RNA secondary structure prediction. Anyway, because of that, I believe that the title is too generic and should emphasize more that they actually discuss efficient LPrF computations.
* Good overall structure, with well-explained motivation and work that has already been done, but with some unclear explanations such as:
* Comments on 2.2 SA-IS definition: 1) “X[i..n] is j the suffix of x in ascending order” - this way of speaking doesn’t describe clearly that X[i...n] is equal to j if X[i...n] is on the j-th position after sorting all of x’s suffixes; 2) In the same definition, it is firstly mentioned string y, but afterward they talk about string x; 3) Initially SA is indexed from 1 to n, but in the example, it is indexed from 0 to (n-1); The string from the table’s caption is different from the one in the table since it has an extra ‘A’ at the end 4) It is mentioned the $ symbol, but it is never used in their example
* Several typos such as: In 2.2. 1) “The suffix array construction algorithm call SA-IS algorithm is” -- “call” should have been “called”; 2) “substrings by developed on top of the following classification“ - “developed” should have been “developing”; In 3.1. 1) “suffix target for each states in the position heap” - “states” should have been “state”; In section 5, the conclusion, there’s a missing “p” from palindrome.
* In section 3.1 y reversed is different from y since it also has the ending symbol $, so it lacks consistency
* Throughout sections 2 and 3 it is mentioned about the so-called “suffix target for each states”, but it’s not clear what is a suffix target or a state.
* Great plot at the end comparing the memory needed by the two presented algorithms, but they could have also provided a comparison with some other existing methods for computing the LPrF table.