COMP 9601 Theory of Computation and Algorithms Design

Assignment 3 Due on mid-night Nov 16, 2015

Warm-up. (1) Suppose you are given a polynomial-time algorithm that can determine whether a given Boolean formula has a satisfying assignment (i.e., the output is either yes or no). Show how you could make use of this algorithm to obtain a polynomial-time algorithm to find a satisfying assignment for any given Boolean formula (if exists).
(2) Prove or disprove the following: Let T be a suffix tree for some string. Let the string α be the label of an edge in T and let β be a proper prefix of α. It is not possible to have an internal node x in T such that the path from the root to x is labeled β.
(3) Give a linear time algorithm to convert an implicit suffix tree to a suffix tree. Make sure your algorithm works if the given suffix tree is derived from a string with all "a"s.

You are expected to answer all five questions, each carries 20 points. Below $T[1..n]$ denotes a string of length n, with the last character being a special character '\$' which is lexicographically smaller than any other character).

Problem 1. Let X be a collection of strings with a total length of n. The number of strings in X can be $\Omega(n)$. Give an algorithm to compute the longest common substring of all the strings in X. What is the complexity of your algorithm?

Problem 2. Suppose that the suffixes of T have already been sorted in ascending order with respect to their first $k \geq 1$ characters and the ordering has been recorded in an array $A[1..n]$. (Note that such an ordering is not unique as we only consider the first k characters; also some suffixes have length less than k). Furthermore, it is given another array $Start[1..n]$ such that $Start[i] = 1$ if and only if the $[A[i-1]..n]$ is lexicographically smaller than the the suffix $T[A[i]..n]$ with respect to the first k characters. Based on the arrays A and $Start$, show how to construct A' and $Start'$ which have the same definition as A and $Start$, respectively, except that the ordering is with respect to $2k$ characters. Your algorithm should run in $O(n)$ time and cannot make reference to T.

Problem 3. Define $SA[i]$ to be the starting position of the i-smallest suffix of T. And define the suffix range of a pattern P to be $[\ell, r]$ such that ℓ and r are respectively the rank of the smallest and largest suffix of T containing P as a prefix. Suppose that you are given the arrays $SA[1..n]$ and $SA^{-1}[1..n]$. Show an algorithm which, given the suffix ranges of any two patterns P and P', can compute the suffix range of PP' (i.e., the concatenation of P and P') in $O(\log n)$ time using SA and SA^{-1}.

Problem 4. With respect to T, define $SA[i]$ to be the starting position of the i-smallest suffix of T. Furthermore, define $\Psi[i] = SA^{-1}[SA[i] + 1]]$, and for each character a in A, we define $count[a]$ to be the number of characters in T that are lexicographically smaller than a.

1. Show how you would use Ψ to compute $SA^{-1}[1]$, $SA^{-1}[2]$, $SA^{-1}[3]$, $SA^{-1}[4]$, $\cdots$.

2. Computing $SA[i]$ from Ψ is slightly more complicated. We make use of additional $O(n)$ bits to store $SA[i]$ for all $SA[i] = 1$, $\log n$, $2\log n, \cdots$ and a bit vector $B[1..n]$ such that $B[i] = 1$ if and only if $SA[i]$ is stored. Give an algorithm to compute $SA[i]$ for any i using $O(\log n)$ time. (NB. Without B, it would take $O(\log^2 n)$ time.)

Problem 5 Define $BWT[i] = T[SA[i]-1]$ if $SA[i] > 1$. For the special case when $SA[i] = 1$, define $BWT[i] = T[n] = \$$. For each distinct character x, define $count[x]$ to be the number of characters in T that are lexicographically smaller than x.

1. Show how to compute T from BWT and $count$.
2. Show how to compute the $\Psi[1..n]$ from BWT and $count$ directly (i.e., without computing the suffix array SA). What is the time complexity of your algorithm?