Homework 4

CS 4364/5364
Spring 2022

Due: 7 March 2022

1. **(20 points) Describe how to *use* the suffix tree data structure to find the longest *palindrome* substring in a string S. Remember a palindrome is a string that is the same forward as in reverse (i.e. `tacocat`). The longest palindrome substring in the example string `bananas` is `anana`: it is both a substring (from the 2nd to 6th characters) and is a palindrome. Include in the submission: (1) an algorithm, (2) an explanation of its correctness, and (3) an analysis of it's running time.**

Algorithm

Adapted from Wing-Kin Sung, Section 3.3.5 (page 63).

(a) Construct $S^R = s_n s_{n-1} s_{n-2} ... s_1$, the reverse of S.

(b) Build a generalized suffix tree on S and S^R.

(c) find the longest common substring between S and S^R using the suffix tree, call its length k and call the two suffix start positions i and j from S and S^R respectively

(d) if $i = n - j - k$, return the string found

(e) otherwise, find the next longest and at step 1c

Explanation

We know from the course discussion that by building a generalized suffix tree we can find an LCS of two strings. Since the two strings are the string itself and its reverse, the longest common substring is going to be a string that is also reversed in the input. This is only a palindrome when it is the *same substring*, meaning the end position in the reverse string corresponds to the start position in the original string. Since we know that index i' in S corresponds to index $n - i'$ in S^R, we can use that along with the common string length to calculate the end position of the substring match in one direction or the other.

Runtime Analysis

Steps 1a and 1b will take $O(n)$, the string needs to be printed, and building is done using Ukkonen's algorithm. Step 1c is worst case $O(n)$ each time it is run as discussed in class. Step 1d is $O(1)$. Step 1e is also $O(1)$ itself, but may cause you to run 1c repeatedly, since the LCS cannot increase, and can only be lowered n times, we know 1c can only be run that many times. Therefore the total running time for all instances of 1c is $O(n^2)$ which is the dominating factor and thus the total running time of the algorithm.