CSE 5311-001 Lab Assignment 3

Due May 4, 2021, 5:00 p.m.

Goals:

1. Understanding of suffix arrays, along with the related notions of suffix rank and longest common prefix (lcp).
2. Review of elementary state machine concepts.
3. Application of the above concepts to the ***longest common substring*** problem.

Requirements:

1. Write (and test) a C program to determine a longest common substring of ***three*** input strings with no more than 500 symbols each. Each input string will be on a line by itself. These strings will be readable using `scanf`, since they will include only upper/lowercase alphabetic symbols or digits. Your code should run in time $O(n)$, not including the time to compute the suffix array, where n is the total number of input symbols.

   Your output will be strings common to all three input strings with monotonically increasing (e.g. non-decreasing) lengths. That is, whenever you find a common substring you will check to see if its length is at least as long as the previous longest common substring. It is important to find ***substrings*** (i.e. contiguous symbols) rather than ***subsequences***.

   Your C program must compile and execute on omega.uta.edu.

2. Submit your C code on Canvas before 5:00 p.m. on Tuesday, May 4. Be sure to include comments regarding how to compile and execute your code.

Getting Started:

1. You may use suffix array code from the course webpage or elsewhere. Code to find a longest common substring, along with common substrings of monotonically increasing length, for ***two*** input strings is available in this lab's web directory.

2. Symbol-to-symbol comparisons should occur only during the preprocessing before finding the longest common substring.

3. When scanning the tables for the results, you will need to examine various intervals that include at least one suffix for each of the three input strings. Since the longest common substring for the interval will have the minimum lcp value in the interval as its length, you only need to process minimal intervals such that leaving out the first or last suffix will leave an interval with only two of the three input strings represented. This notion may be summarized using regular expressions: if x y z is a permutation of {0, 1, 2}, where these correspond to the three input sequences, then a minimal interval will include suffixes according to the regular expression $x\ y^+\ z$. Thus, the scanning can be done by a simple automaton that tracks the current values (if any) for x, y, and z, along with maintaining the indices and updating the longest known common substring.

   Notes that column `t` in the sample outputs indicates the input string corresponding to the beginning of a suffix.

4. You will find ***14.2. Text-Based Preprocessing*** under Canvas and pp. 9-14 of Notes 14 to be useful.

5. The printing of the tables should be disabled in the code you submit.