CS 224 HW 3 Due October 27.

1. Recall that the label of a node v in a suffix tree is the string created by concatenating the substrings on the path from the root to node v. Suppose u is labeled with the string $x\alpha$, where x is a single character, and α is a string (possibly empty).

Give a justification that there must also be a node v' in the suffix tree labeled with the string α. Recall that a *suffix-link* in a suffix tree is a pointer from a node labeled $x\alpha$ to the unique node labeled α. Several algorithms that use suffix trees depend on the suffix-links.

The algorithm we developed in class (and homework 2) to construct a suffix tree from a suffix array and its LCP array, does not find the suffix links in the tree, unlike all prior linear-time algorithms to build suffix trees.

Suppose you have a suffix tree without suffix links, and you want to explicitly put in the suffix links, of course only using linear time. That is possible by using a linear-time preprocessing, constant-time lookup Lowest Common Ancestor algorithm. Show how that works.

Another idea for finding the suffix links is to use two simultaneous depth-first searches in the suffix tree, one that lags the other by one character. Then, whenever the first search is at a node v labeled $x\alpha$, the lagging search should be at the node v' labeled α. Now if you actually moved character by character through the tree in the two searches, the time bound for the searches would not be linear, because the number of characters represented on the edges of the suffix tree is quadratic in the length of the string. Instead you want to do each dfs by moving from node to node in the tree, since the number of nodes is linear in the length of the string.

Complete the details of how to do the two simultaneous searches, paying attention to the complication that the path spelling out $x\alpha$ may have a different number of edges than the path spelling out α.

Prove that the total time for this approach is linear in the length of the string (not linear in the number of characters labeling the edges of the suffix tree).

2. Read the posted notes on the Unique Decypherability problem and prove that the graph-based algorithm is correct and runs in polynomial time. Optional: think about how to implement the building of the graph in $O(nm)$ time, using a suffix tree or array.

3. In the LCA method discussed in the video, the LCA problem on a tree

T of n nodes and E = n-1 edges was reduced to a problem on two arrays of size 2E+1 = 2n-1 by doing a dfs of T, recording each node visited during the dfs in an aray N, and also recording its depth in a second array D. Array D has the property that consecutive values differ by plus or minus one.

When T is binary, there is a different reduction of the LCA problem to two arrays, N' and D', of sizes n by doing an in-order traversal (traverse-left, visit-node, traverse-right) adding the visited node v and its depth at the ends of N' and D' only when the traversal backs up to v. As in the first reduction, given two nodes u and v, we can find the LCA(u,v) by finding the positions of the entries for u and v in N', and then finding the minimum depth entry between those positions in D'.

Give a complete explanation for why this is correct. Does D' have the property that consecutive values differ by plus or minus one? If not, does that cause a serious problem for using this smaller N' and D' in place of the original arrays. That is, is the plus-minus-one property of the original D really critical to the original method?

4. The LCA algorithm reduces the LCA problem to the problem of finding the minimum entry in an interval of an array D, in constant time, when the consecutive values in D differ by plus or minus one, after linear-time preprocessing. Surprisingly, the requirement that the consecutive entries in D differ by plus or minus one, can be removed, allowed D to be arbitrary. This is done with the use of a Mystery Tree:

> A Mystery Tree of an array A[l, r] is a binary tree M(A) whose root is a minimum element of A, labeled with the position i of this minimum. The left child of the root is the Mystery Tree of A[l, i-1] if $i > l$, otherwise it has no left child. The right child is defined similarly for A[i + 1, r].

Problem: Assume that the Mystery Tree can be built in time linear in the size of A (you can work this out if you want - it is not very hard, but it is not the topic of this course). Show how to solve the problem of finding the minimum entry an interval of an array A, when the values of A are arbitrary, using only constant time to find the minimum, after linear-time preprocessing.