CS 224 Fall 2011, Final Exam. It may be long. Do the best you can.

0.1 Suffix Trees

1. Let T be a suffix tree for string S over a finite alphabet. Each node v in T is labeled with the string w, where the walk from the root to v spells out a string w. That is, w is the concatenation of the strings on the edges of the walk to v.

1a. Prove that for any node v in T, if v is labeled with string $x\alpha$, where x is a single character and α is a string, then there is a unique node v' in T labeled with string α. Remember that an edge in T can be labeled with a string of length more than one.

For each such ordered pair of nodes (v, v') in T, where v is labeled $x\alpha$ and v' is labeled α, a link from v' to v is called a *reversed suffix-link*; the link is labeled with character x (i.e, the first character of the label on v). We didn't discuss this in class, but all the reversed suffix-links in a suffix tree can be found in linear time.

1b. Suppose suffix tree T for string S has been constructed, and all of the reversed suffix-links have been added to T. Develop an algorithm that takes in a pattern P, and finds the longest suffix of P that occurs in S, in time linear in the length of P. The algorithm is not allowed to build any new data structures - it just uses T and the reversed suffix links. Moreover, it can traverse at most one edge of T, and otherwise only traverses reversed suffix links. Explain the correctness of your algorithm and its linear time bound.

0.2 Suffix Arrays

Recall the following definitions given when we discussed the LCP array:

Given a string S, define $Suff_k$ as the suffix of string S starting at position k. (We start indexing at 1, not 0.) Define $lcp(S_1, S_2)$ as the length of the longest common prefix of strings S_1 and S_2. If POS is the suffix array of a string S, and k is an entry at a position, say i, of POS, then define Pred(k) as the entry in position $i - 1$ of POS. That is, Pred(k) is the entry in POS just to the left of where k is in array POS. $lcp(Suff_k, Suff_{Pred(k)})$ is defined to be the *length* of the longest common prefix of $Suff_k$ and $Suff_{Pred(k)}$.

Recall that LCP is an array of length $|S|$ (where the indexing starts at 1), and for each i from 2 to $|S|$, the value of LCP[i] is $lcp(Suff_k, Suff_{Pred(k)})$ where k is the value of POS at position i.

For an arbitrary pair of distinct positions k and k' in S, we want to determine $lcp(Suff_k, Suff_{k'})$. Suppose that k is the value at position i of POS, and k' is the value at position i' of POS. (It helps to draw a picture.)

I claim that $lcp(Suff_k, Suff_{k'})$ is equal to the minimum value in the array LCP between positions $i+1$ and i', inclusive.

2. Is this claim (essentially) correct? By essentially, I mean that if it is only wrong due to an index being off by one, or some other simple issue, just fix it. If it is essentially correct (modulo any little needed fix), give a convincing explanation for it. I think a good way to proceed is to visualize what this is saying on a suffix tree and how a suffix tree relates to a suffix array.

0.3 BWT

3a. State a three-line (at most) description of a way to create the BWT string (not the inverse) of a string S in linear time. You do not need to explain or justify it.

3b. Briefly explain as best as you understand it, why the BWT tends to bring identical characters together.

0.4 Phylogenetic Networks

4. Give a definition of a Buneman graph for a binary matrix M.

0.5 Self-Derivability and MinARGs

Definition Given a set of sequences M, we say that M is *self-derivable* (SD) if M can be generated on an ARG $\mathcal{N}$ where *every* node in $\mathcal{N}$ (including the root) is labeled with a sequence in M. In that case, we also say that $\mathcal{N}$ self-derives M.

Definition We say that M is *unary-self-derivable* if it is self-derivable on an ARG $\mathcal{N}$ where no edge is labeled with more than one site. In that case, we also say that $\mathcal{N}$ unary-self-derives $M(S)$.

Recall that $H(M)$ denotes the haplotype bound, which is the number of distinct rows of M, minus the number of distinct columns of M, minus one.

Problem 5: Prove the following:

Lemma Suppose that M has no duplicate columns. If M is unary-self-derivable on an ARG $\mathcal{N}$, then $\mathcal{N}$ contains exactly $H(M)$ recombination nodes, i.e., the haplotype bound is tight.

0.5.1 Building a MinARG

Definition For a set of sequences K, let $SD(K)$ be a Boolean variable that is set to **true** if the sequences in K are self-derivable, and **false** otherwise.

Definition If the set of sequences K is self-derivable, let $r(K)$ denote the *minimum* number of recombination nodes in any ARG that self-derives K. Note that $r(K)$ is undefined if $SD(K)$ is **false**.

We use a dynamic programming approach to determine if a set of sequences S is self-derivable, and if so, to compute $r(S)$. To start the dynamic program to test if S is self-derivable, set $SD(K)$ to **true**, and $r(K)$ to zero, for every singleton subset K in S. Then in order of increasing size, consider each subset K of two or more sequences in S. When a subset K is considered, set $SD(K)$ to **false** and $r(K)$ to ∞, and then execute the following two steps for each sequence $s \in K$ such that $SD(K - \{s\})$ is **true**:

1) Enumerate each pair of sequence s', s'' in $K - \{s\}$, and set $SD(K)$ to **true** if s can be created by a recombination of s' and s''. If s can be created by a recombination of s' and s'', set $r(K)$ to the minimum of the current $r(K)$ and $r(K - \{s\}) + 1$.

2) Enumerate each sequence s' in $K - \{s\}$, and set $SD(K)$ to **true** if s differs from s' precisely at a set of sites C, where for each site $c \in C$, all of the sequences in $K - \{s\}$ have the same state at site c. {In this case, s can be derived from s' by a mutation at each of the sites in C.}

If s can be derived from s' by mutations, set $r(K)$ to the minimum of the current $r(K)$ and $r(K - \{s\})$.

Claim After the iteration where $K = S$, $SD(S)$ will have been computed correctly, and if $SD(S)$ is **true**, then $r(S)$ will be less than infinity and will have been computed correctly.

Problem 6a: Is the Claim correct? Justify.

Finally a MinARG Now we can fully detail how to find $Rmin(M)$ and a MinARG for M. See Algorithm MinARG in Figure 1.

Problem 6b: Did I get the algorithm right this time?

```
ALGORITHM MINARG (M)
    Let P(m) be the set of all binary sequences of length m
    Set variable $r^*$ equal to $nm$, and variable MA to the empty DAG.
    **for** each subset $\mathcal{S}$ of P(m) that includes $M$ and has size at most $nm + m + 1$ **do**
        Test if $\mathcal{S}$ is self-derivable, and if it is, compute $r(\mathcal{S})$
        If $\mathcal{S}$ is self-derivable, and $r(\mathcal{S}) < r^*$, set $r^*$ to $r(\mathcal{S})$ and
        set MA to the ARG created in the self-derivability test.
    **endfor**
    Output the value of MA and $r^*$.
    {MA is a MinARG for $M$ using $r^*$ recombination nodes, so $Rmin(M) = r^*$.}
```

Figure 1: *Algorithm MinARG* is guaranteed to find $Rmin(M)$ and MinARG for M.