Introduction to Bioinformatics Algorithms
Homework 6
Solution

Saad Mneimneh
Computer Science
Hunter College of CUNY

Problem 1: Suffix trees
Describe how you can find in linear time the following, using a suffix tree data structure:

(a) a longest match between x and y

Solution: Let $l(v)$ be the length of the of the string obtained by concatenating the edge labels on the path from the root to v. $l(v)$ can be computed in linear time by a breadth first traversal of the tree from the root and using $l(v) = l(parent(v)) + L(parent, v)$ where $L(e)$ is the length of the label on edge e and $l(root) = 0$. Labeling the nodes by x and/or y as described in class can be done in linear time by a bottom up approach. Then we need to find a node v labeled both x and y with largest $l(v)$. This can be done by a linear time traversal of the tree. The label up to node v is a longest match between x and y. Note that v is left diverse; otherwise, it cannot have the largest $l(v)$. The match itself is obtained by going through the edge labels on the path from the root to v.

(b) a longest **unique** match between x and y if it exists

Solution: This is alomst as above except that we would like the match to be unique, and hence it may not exist. For instance $x = aaabbbaaa$ and $y = aaa$ do not have a unique longest match because any match is present twice in x. But what does it mean for a match to be unique in terms of the suffix tree? It must correspond to a node v labeled both x and y (for it to be a match) such that there is exactly one leaf for x and exactly one leaf for y in v's subtree. This means that v must have two children that are both leaves, one for x and one for y. So among all nodes labeled both x and y we need to find node v with the largest $l(v)$ such that v has two children that are leaves. This can be done in linear time as well since the checking for this extra condition is trivial.

(c) a longuest repeat in x

Solution: Any internal node in the suffix tree for x represents a repeat. Therefore, we need to find an internal node v with the largest $l(v)$.

(d) a longest **non-overlapping** repeat in x

Solution: Any internal node v in the suffix tree for x represents a repeat. Let i and j be two leaves in v's subtree. If $|i-j| \geq l$, then the repeat is non-overlapping since the two identical substrings of length $l(v)$ occur at positions i and j. For every node v, let $min(v)$ and $max(v)$ be the minimum and maximum index of a leaf in v's subtree. $min(v)$ and $max(v)$ for every node v can be computed in linear time by a bottom up approach. Then v represents a non-overlappping repeat iff $max(v) - min(v) \geq l(v)$. Therefore, we need to find the node v with the largest $l(v)$ satisfying $max(v) - min(v) \geq l(v)$.

Problem 2: Parsimony
The following dynamic programming algorithm solves the maximum parsimony problem for a general distance criterion and one character.

$$f_v(a) = 0, v \text{ is a leaf labeled } a$$

$$f_v(a) = \infty, v \text{ is a leaf not labeled } a$$

$$f_v(a) = \sum_{w \in \delta(v)} \min_{b \in A} [f_w(b) + d(a,b)]$$

$$M_w(a) = \{b : f_w(b) + d(a,b) \text{ is minimal}\}$$

We seek $\min_{a \in A} f_{root}(a)$ and M can be used for backtracking.

(a) What is the running time and space requirement for this algorithm? Assume $|A| = r$ (r states), we have n leaves (objects), and m characters.

Solution: As we have seen in class, given a character, for every state and every vertex, we check every state for every child. This is

$$\sum_a \sum_v \sum_{\delta(v)} \sum_a 1$$

which is $O(nr^2)$ time, so we need $O(mnr^2)$ time for all characters.

(b) Adapt this algorithm to the special case when the distance is defined as (obtain better running time):

$$d(a,b) = \begin{cases} 0 & a = b \\ 1 & a \neq b \end{cases}$$

Solution: We can now do this in $O(mnr)$ time. Fix one character as before. For every vertex v, define S_v as the set of all states that v can be labeled with and achieve the minimum distance in its subtree. We can show that the optimal solution can label every vertex v with a state in S_v (though not every optimal solution must do so). Assume a vertex v is labeled by a, and $a \notin S_v$. Then, by definition, the distance in v's subtree can decrease by at least 1. Let $b \in S_v$ and replace a by b. We reduce the total distance in v's subtree by at least 1, and we gain at most a 1 in the distance between v and its parent node. Therefore, our labeling must still be optimal.
We can compute S_v for each v using dynamic programming, and based on the above, obtain the optimal solution by backtracking: Set the label of the root to any state in S_{root}, and for each vertex v, set its label to be the same of its parent if that state is in S_v, otherwise, any state in S_v.
To compute S_v, it must contain all states that occur the most in $\bigcup_{w \in \delta(v)} S_w$. This can be obtained in $O(r|\delta(v)|)$ time. For instance, for each vertex we can keep a vector of size r such that $vector[i] = 1$ iff state $i \in S_v$. The sum of these vectors for v's children will determine which states occur the most. Once this information is obtained, we can compute the vector for v. Summing over all v's we get $O(rn)$. Therefore, we can do this in $O(mnr)$ time for all characters.

Problem 3: Perfect phylogeny (optional)
Consider binary characters and let 1_i be the set of objects that have state 1 for character i. Define 0_i similarly.

(a) Consider the following character state matrix:

$$\begin{array}{ccc} & c_1 & c_2 \\ A & 0 & 1 \\ B & 1 & 1 \\ C & 1 & 0 \end{array}$$

According to the condition stated in class, namely that 1_i and 1_j are either disjoint or one is a subset of the other, a perfect phylogeny does not exist. However, the colored graph produced by the state matrix is acyclic, implying that a perfect phylogeny does exist. Which interpretation is correct?

(b) Consider ordered undirected (the state tree is unrooted) non-binary characters. Show that for each character i, the state tree can be rooted in such a way that for each binary factor j, $|0_j| \geq |1_j|$. *Hint*: root the tree arbitrarily, then reverse edges that violate the condition, and argue that you would still have a tree.

Problem 4: Viterbi (optional)
A loaded die has probability 1/2 for 6. The probability of switching to a loaded die is 0.05. The probability of switching to a fair die is 0.1. We observe the following:

```
651166453132651245366646316366631623264552362666666625151631
```

Which parts of this sequence are generated by the loaded die? Use the Viterbi algorithm to figure this out. Use the log transformation in your implementation.