PBL 4: Shortest Superstrings

This problem is designed to have the teams explore some issues of algorithm design and complexity. The context is the quest to assemble DNA fragments into longer strings of bases.

To discuss the complexity of an algorithm, one needs to decide how to measure the size of input data and the effort needed by the algorithm to process these data. Part of the solution to this problem will depend on your investigation of these measures.

Throughout this problem let $\mathcal{A}$ be an alphabet (a set of characters). The measure of $\mathcal{A}$ is the number of elements it contains. We write

$$\text{size of } \mathcal{A} = |\mathcal{A}| = K$$

1. Let X be a finite set of strings on $\mathcal{A}$.
   (a) Propose measures for X.
2. The *trie* for X, $\mathcal{T}(X)$, is the rooted tree with the following properties
   (a) Its edges are labeled using elements of $\mathcal{A}$
   (b) Sibling edges (edges with a common node) are labeled with distinct elements of $\mathcal{A}$
   (c) Paths from root to leaf form the strings in X in the sense that, for each of these paths, the sequence of labels of edges in the path forms one of these strings.

   Consider such a trie, $\mathcal{T}(X)$.
   (a) Propose measures for $\mathcal{T}(X)$.
   (b) Determine how the measures for the trie are related to the measures for the set of strings.
   (c) Give an example of a set of strings X and the corresponding trie $\mathcal{T}(X)$ and compute the measures described above. [*N.B.* Remember the context of the problem.]
3. Let y be a single string in $\mathcal{A}^*$. Let the length of $y = |y| = n$. Then y gives rise naturally to a set of strings, namely the set of *suffixes* of y. Specifically, let the string $y = y_1 y_2 \dots y_n$. Denote the i^{th} suffix of y by $y(i, n) = y_i y_{i+1} \dots y_n$ where $i = 1, 2, \dots, n$.
   (a) Specialize the measures for a trie that you developed above to a suffix trie.
   (b) Give an example of a string y and its associated suffix trie. Compute the trie measures for this example.
4. To perform the computations necessary to find a shortest superstring, we need to convert the suffix trie to a *suffix tree*. This is done by collapsing paths that go through nodes of degree 1 and concatenating the edge labels into a substring of the original string y. For subsequent computations, we must assume that y_n is distinct from every other character in the given string. [*N.B.* Figure out where this assumption is actually used.] Denote the suffix tree of y by $\mathcal{S}(y)$.

   Under this assumption, compute, discover, or otherwise ascertain:

(a) The number of leaves of $\mathcal{S}(y)$.

(b) The substrings of y represented by the paths from the root to the leaves.

(c) A reasonable labeling scheme for the leaf nodes.

(d) The degree of the root node of $\mathcal{S}(y)$.

(e) The degree of an internal node of $\mathcal{S}(y)$.

(f) The substring of y represented by an internal node and the relationship of this substring to the child nodes of the internal node.

(g) The number of nodes of $\mathcal{S}(y)$.

5. The complexity of an algorithm depends on both the amount of time (or number of operations) that must be performed and on the amount of space that must be used for the data. Let's consider an efficient way of storing the suffix tree. The efficiency comes from a scheme of referring to the given string and its substrings by using the location indices of characters in the string. In this way the given string is stored only once.

   Let $y(j,k)$, where $j \leq k$, denote the substring of y with indices j through k, namely $y_j y_{j+1} \ldots y_k$. Note that $y(j,j) = y_j$ and $y(j,k)$ is undefined if $k < j$. We'll store a node in the suffix tree by using a cell that holds three values:

   - the pair of indices that defines the substring labeling the edge leading into the node. *N.B.* the pair of indices for the root will defined to be (0,0).
   - the pointer to the cell of the left-most child of the node.
   - the pointer to the cell of the next sibling of the node. *N.B.* If the target cell of a pointer does not exist, the pointer is set to λ.

   (a) For the example suffix tree you constructed above, draw the storage diagram using this scheme.

   (b) Describe measures for this storage scheme and calculate them for your example.

6. For the string TCACTGACTGTGAC construct the suffix tree representing it by the storage scheme described above. Use the *Brute Force* algorithm for the construction, which proceeds as follows:

   (a) The tree for the empty suffix is denoted by Y_0 and is stored as

(0,0)	λ	λ

   (b) Construct the tree iteratively (**for** $i = 1$ **to** n **do**) by *inserting* the path corresponding to the suffix $y(i,n)$ into the partial suffix tree Y_{i-1} to create the partial suffix tree Y_i. The trick for this step is to find a way to insert the new path that uses as many of the edges of the partial tree as possible. Intuitively this means that we should split $y(i,n)$ into two pieces: a beginning piece that duplicates existing edges and an ending piece that is the new part of the tree leading to the leaf that represents $y(i,n)$. We will use the notation $y(i,n) = head_i tail_i$ to show the division. Then use one of the following constructions.

- If $head_i = \lambda$, then connect a new node to the root and label appropriately.
- If $head_i \neq \lambda$, then insert a new node on the edge whose label starts with $head_i$ and attach to the new node a leaf node that is labeled with $tail_i$.

7. To solve the shortest superstring problem we need to construct a suffix tree that represents all the strings in the given set. This can be done as described above attaching each suffix of each string to a common root.

   (a) Construct a suffix tree for the set of strings

   $$X = \{alf, ate, half, lethal, alpha, alfalfa\}$$

   (b) Using the measures you've developed above to check the size of the suffix tree you've constructed.

8. *(Constructing a superstring)* Let's first define some operations on a set of strings X. For any strings x, y in X,

   - let $ov(x, y)$ be the longest suffix of x which is a prefix of y. Give examples.
   - let $lov(x, y) = |ov(x, y)|$.
   - let $blend(x, y)$ be the shortest string that has x as a prefix and y as a suffix. Give examples.
   - let $pref(x, y)$ be the prefix of x which when concatenated with $ov(x, y)$ gives x, that is, $x = pref(x, y)ov(x, y)$. Give examples.

   Write the greedy algorithm for constructing a superstring using the suffix tree data structure you constructed above.

9. Give an estimate for the measures of the typical superstring construction problem.