

BCB 567/CprE 548
Fall 2007
Homework 3
Solutions

- 1.
2. To solve this problem, we will show that the number of unique substrings of S corresponds to the summation of the lengths of all edge labels in the suffix tree of S , excluding the character \$. For all of the reasoning below, ignore the \$ symbol.

Let α be the path label from the root to some internal node u and β be the edge label from u to v , where v is a child of u . Denote each distinct prefix of β as β_i . The edge label from u to v corresponds to $|\beta|$ strings, $\alpha\beta_1, \alpha\beta_2, \dots, \alpha\beta_{|\beta|}$. Each of these strings corresponds to a distinct substring of S , starting at some position j , where $suff_j$ is in the subtree rooted at v . We notice that each $\alpha\beta_i$ corresponds to a unique path from the root. Moreover, given some substring γ of S starting at position t , there exists some nodes u and v , such that $\gamma = \alpha\beta_i$ for some i . This is because the path labeled γ must exist in the graph, as this is a property of suffix trees.

Therefore, our algorithm for counting the number of distinct suffixes of S must simply add together the string lengths of all edges in the suffix array, discounting the \$. This can be done using a tree traversal, which takes $O(E)$ time, where E is the number of edges. In the case of the suffix tree, $O(E) = O(n)$.

3. (5 points)
 - (a) Because we can find the child of interest in constant time, the total time needed to search for the pattern P is $O(|P|)$ time. However, since we require $O(|\Sigma|)$ space per node, the total space requirement is $O(|\Sigma|n)$.
 - (b) Because we can find the child of interest in $O(|\Sigma|)$ time by traversing the linked list, the total time needed to find pattern P is $O(|P||\Sigma|)$ in the worst case. This occurs when the number of nodes visited during search is $O(|P|)$. For each child, there is exactly one node in the linked list of the parent. Therefore, there is a constant amount of space required per node to store the linked lists. Therefore, the tree requires $O(n)$ space.
 - (c) Because we can find the child of interest in $O(\log_2|\Sigma|)$ time, the total time needed to find pattern P is $O(|P|\log_2|\Sigma|)$. For each child, there is exactly one node in the balanced tree. Therefore, there is a constant amount of space required per node to store the balanced trees. Therefore, the tree requires $O(n)$ space.

4. No, a prefix tree does not work for pattern matching. The only patterns that would be found in a prefix tree are those patterns that start at the beginning of the string. Any pattern starting in the middle of the string would not match characters in the prefix tree starting at the root (unless the pattern happened to be a repeat that also was found in the beginning of the string). Depending on your assumptions, a prefix tree would either be a single edge (this would be called an implicit prefix tree) or would be a tree of height n , with the empty string labeling n edges. A rather useless data structure.

Note: The prefix tree as defined in the problem is not the same as the suffix tree of the reverse of the string. The suffix tree of the reverse of a string could be thought of as the reverse prefix tree (a compacted trie that contains the reverse of all prefixes of a string.) But really this is just a convoluted suffix tree. The point of this question was to demonstrate that a true prefix tree is worthless.

5. (5 points) The statement is false. A counterexample is easy to construct one we realize what such a statement means. Internal nodes correspond to distinct right maximal repeats. However, if you reverse the string, then internal nodes correspond to what used to be distinct left maximal repeats in the original string (right maximal in the reverse). Therefore, to find a counterexample, all that we need to do is to construct a string that has a different number of left maximal and right maximal repeats. Counterexample:

ABABCA

By inspection, we see that this string has three right maximal repeats: $\{a, ab, b\}$. The string has two left maximal repeats $\{a, ba\}$. Therefore, without even constructing the suffix trees, we know that the suffix tree corresponding to the forward string will have three internal nodes while the suffix tree for the reverse string will have two internal nodes.

6. (a) We convert the problem to the suffix tree formulation: The longest repeated prefix of S starts at both index 1 and index $|\beta|$ in the string. Therefore, the longest repeated prefix of S is a common prefix of length $|\beta|$ between suff_1 and $\text{suff}_{|\beta|}$. This means that the longest repeated prefix of a string correspond to the longest path in the suffix tree of S that starts at the root and is labeled β (possibly stopping in the middle of an edge) such that leaves labeled 1 and $|\beta|$ are found in the subtree of S rooted at the internal node v corresponding to endpoint of the edge on which we stop.
- Let the variable $depth$ represent the string depth of the current node under consideration.
 - Let the variable len_β represent the length of the longest repeated prefix found thus far in the algorithm.
 - To initialize the algorithm, find leaf 1 in $O(n)$ time using any sort of tree traversal. Initialize $depth = n$. Initialize $len_\beta = 0$.
 - Starting from leaf 1, we will walk up the tree toward the root, one edge per iteration. In each iteration, we will update $depth$ and len_β .

- Consider the current iteration in which we have walked from node v to node u (u is the parent of v .) Let (i, j) be the edge label of the edge connecting u to v .
 - Set $depth = depth - (j - i + 1)$.
 - If $len_\beta \geq depth$ then we quit. This is because we cannot find a new repeated prefix that is longer than the longest repeated prefix already found. Report $S[1..len_\beta]$ as the answer. (*Note that if the longest repeated prefix is of length zero, the algorithm will terminate when we reach the root with $len_\beta = 0$. In this case we should report the empty string as the answer.*)
 - Otherwise, traverse the tree rooted at v (not including the tree rooted at u from whence we came). For each leaf encountered labeled j , if $j \leq depth$ and $len_\beta < j$ then set $len_\beta = j$.

Once the tree traversal completes, continue with the next iteration by walking up to the parent of u .

Because each edge in the tree is traversed exactly once, this algorithm runs in $O(n)$ time.

- (b) Finding the repeated suffix is now very easy. First reverse the string. Then run the algorithm to find the repeated prefix of a string.