Advanced Algorithms Suffix trees, LCA, RMQ Teaching Block 1

TAs : Kheeran K. Naidu, Alex Carpenter

Problem sheet 3 (Solutions)

Please feel free to discuss these problems on the unit discussion board or directly with your colleagues. If you would like to have your answers marked, please either hand them in in person or email them to me with the email subject "Problem sheet 3". Submitted work will be marked as quickly as possible, ideally within one week of being handed in. (I will add more questions as needed to this sheet.)

1. (From Gusfield) Given a set S of k strings, we want to find every string in S that is a substring of some other string in S. That is, we want to find the smallest subset of strings $S' \subset S$ such that no string in $S - S'$ is a substring of any other string in that set. Assuming that the total length of all the strings is n, give an $O(n)$-time algorithm to solve this problem.

Solution.

```
alg(S) :
    1. Create a generalised suffix tree on all k strings
    2. S' ← ∅
    3. For each s_i ∈ S :
    4.         walk down the tree from its root following s_i
    5.         if the walk stops at an internal node
    6.                 S' ← S' ∪ {s_i}
    7. return S'
```

Time complexity. Let n_i be the length of each string $s_i \in S$. Constructing a suffix tree on the k strings in line 1 takes $O(n_1 + ... + n_k) = O(n)$ time since $\sum_{i=1}^{k} n_i = n$. Walking down the tree for some string s_i takes n_i steps and $O(n_i)$ time. Determining if it is an internal node takes $O(1)$ time. We do this for each $s_i \in S$, so the total time required for the loop is $O(n_1 + ... + n_k + k) = O(n)$ where we have reasonably assumed that the number of substrings k is $O(n)$. Hence, the overall time taken is $O(n)$.

Correctness. Assume for a contradiction that $\exists s_i \in S'$ such that s_i is only a substring of itself. When processing string s_i in line 4, the walk would end at a leaf node since s_i is a suffix in the generalised suffix tree. Hence line 6 would not have executed and s_i would not have been added to S', a contradiction. A similar argument holds for proving that $\nexists s_j \in (S - S')$ such that s_j is a substring of a string other than itself.

✓

2. (From Gusfield) A suffix tree for a string S can be viewed as a keyword tree, where the strings in the keyword tree are the suffixes of S. In this way, a suffix tree is useful in efficiently building a keyword tree when the strings for the tree are only *implicitly* specified. Now consider the following implicitly specified set of strings : Given two strings S_1 and S_2, let D be the set of all substrings of S_1 that are not contained in S_2. Assuming the two strings are of length n, show how to construct a keyword tree for set D in $O(n)$ time.

Solution.

(a) Construct a generalised suffix tree on S_1 and S_2

(b) Colour all nodes (including leaves) as follows :
- — **green** if all leaves in the subtree from the node are from S_1
- — **red** if all leaves in the subtree from the node are from S_2

(c) Remove all red nodes and the the edges leading to them

The keywords in this tree are the strings whose walks from the root end on an edge leading up to a green node.

Time complexity. Constructing the (compacted) suffix tree in step (a) takes $O(n + n) = O(n)$ time. We can perform step (b) by traversing up the tree from every leaf to root and tracking the unique terminating symbols seen at each node along the way, appropriately colouring the nodes green or red if only one unique symbol has been seen so far. Since there are $O(n)$ edges [1] in a suffix tree, this takes $O(n)$ time. Finally, there are $O(n)$ nodes so deleting only the red ones takes $O(n)$ time using a Breadth First Search. Overall, this takes $O(n)$ time as required.

Correctness. Observe that for any generalised atomic suffix tree [2], paths from root to any node or leaf represent substrings since prefixes of suffixes are substrings. We will prove that in the above colouring of the suffix tree, only green nodes are keywords for the set D and no others. Consider a path from root to a green node x and its corresponding substring s_x. We have that s_x is not a substring of S_2. Assume for a contradiction that s_x is a substring of S_2. This means that s_x is a prefix for some suffix of S_2 which implies that there is a path from x to a leaf of S_2, a contradiction since the subtree from the green node x has a leaf in S_2. By similar arguments we can show that paths from root to red nodes

1. This is because there are $O(n)$ nodes (including leaves), and exactly one edge leads to each of them (excluding the root).

2. WLOG we consider atomic suffix trees instead of the space efficient compacted ones, which doesn't affect the correctness of the algorithm.

represent substrings of only S_2, and paths from root to uncoloured nodes represent substrings of both S_1 and S_2.

✓

3. Assume you are given two strings S_1 and S_2 of length n. Describe how you can find the length of the longest substring in S_2 that is also a substring of S_1 in $O(n)$ time. You should describe the different parts of your algorithmic solution in detail and also give the total running time.

Solution.

(a) Construct a generalised suffix tree on S_1 and S_2

(b) Colour all nodes (including leaves) as follows :
- — **green** if all leaves in the subtree from the node are from S_1
- — **red** if all leaves in the subtree from the node are from S_2

(c) Find the uncoloured node with the largest root to node length, and output the corresponding word at that node

Time Complexity. Step (a) and (b) take $O(n)$ time as shown in Q2. We can perform step (c) by running a Breadth First Search (BFS) and summing the lengths of the edges along the way, storing the cumulative sum at each node and adding pointers to the uncoloured ones. A BFS visits each node only once and at each visit adding the length of the edge to the cumulative length so far takes $O(1)$ time. Since there are $O(n)$ edges in the suffix tree, this takes $O(n)$ time. Then, we can simply check every uncoloured node (using the pointers) for the largest root to node length in $O(n)$ time. This takes $O(n)$ time overall.

Correctness. The correctness of Q2 proves that only the paths from root to uncoloured nodes represent the words which are substrings in both S_1 and S_2. Hence, the longest such path is the longest common substring.

✓

4. This question is about the LCA problem. For the example tree in the lectures with 11 nodes, write out the complete reduction to the ± 1 RMQ problem. What is the total size of the data created by the LCA preprocessing ?

Solution. For simplicity, we will store all information as data entries in respective arrays.

(a) Euler tour of T to construct arrays N and D, simultaneously. Both N and D have $2n-1$ entries each, totalling $4n-2$ entries.

(b) Storing pointers for each of the n nodes to an index in N can be done with an array of length n.

Hence, the total size of all arrays this problem stores during preprocessing is given by

$$4n - 2 + n = 5n - 2.$$

✓

5. If you are given an independent and efficient solution to the full RMQ problem, how can you reduce the space needed for the arrays created in the reduction from LCA to RMQ ? Show your new reduction for the example tree in the lectures with 11 nodes.

Solution. First observe that the values at $D[i]$ and $N[i]$ correspond to the same node, since the two arrays are constructed during the Euler tour. Further, the value of $RMQ(i, j)$ in both N and D will always refer to the same node, since the (unique) node with lowest depth in a given range is also guaranteed to have the smallest ID in that range, as node IDs strictly increase with depth.

Now that we can answer any RMQ query efficiently, rather than just the ± 1 cases, we can perform range minimum queries on N instead of D. So the total space needed for the arrays is just the size of N, since D is no longer needed. So the total space used reduces from $5n - 2$ to $3n - 1$.

✓