- This problemset has *two* questions.
- To get the credit for questions marked as SPOJ, you must get them accepted on `http://www.spoj.com/AOS`, but you **don't** have to send any explanation!
- For other questions, either send the solutions to `gawry1+aos@gmail.com`, or leave them in the envelope attached to the doors of my office (room 321).

1. Assume that you are given a word w and its suffix array (with the lcp information). Show how to count the number of **different** nonempty subwords of w, i.e., calculate $|\{s[i..j] : 1 \leq i \leq j \leq |w|\}|$ in $\mathcal{O}(n)$ time. For instance, $aaab$ has 7 different subwords.

   Solution: First we have to count the number of all subwords of w (including the repetitions). Since w is of length n we have $\frac{n(n+1)}{2}$ of possibly repeated subwords. From this number we have to subtract the number of repeated subwords. We can get this information from the LCP array. $LCP[i] = j$ means that j prefixes of the i-th word in the suffix array actually occur somewhere before, too. So we have to sum all the numbers from LCP array and to subtract this result from the total number of possibly repeated subwords. Clearly, scanning through the LCP array takes linear time, so the total running time is linear.

2. The goal of this (long) problem is to show that in the RMQ problem we can actually reduce the general case to the special one where $|A[i+1] - A[i]| \leq 1$, which we call *bounded* RMQ. Recall that RMQ is to preprocess an array $A[1..n]$ so that we can find the **position** of the minimum in any $A[i..j]$ efficiently.

   (a) Lowest common ancestor problem (or LCA) is to preprocess a rooted tree on n nodes so that given any two nodes we can find their lowest common ancestor (lowest means closest to the nodes). First show how to preprocess the tree in linear space so that given two nodes you can decide whether one is an ancestor of the other in constant time (think about preorder numbers using in the depth-first search).

   Solution: We can perform DFS and for each node to mark the time when we have entered this node and the time when we have left this node. Then u is an ancestor of w if the interval of entering and leaving time of w is contained in the interval of entering and leaving time of u.

   (b) Design an $\mathcal{O}(n \log n)$ time and space preprocessing algorithm which allows computing the LCA of any two nodes in $\mathcal{O}(\log n)$ time using a binary search-like procedure (use the doubling and powers-of-two trick).

Solution: For each vertex we calculate its 2^j-th ancestow using the following recurrence relation: $P[i][0] = \text{parent}[i]$, and $Pi[][j+1] = P[P[i][j]][j]$. We can calculate and store all $P[i][j]$ in $\mathcal{O}(n \log n)$ time and space.

Now consider a query $\text{LCA}(u, v)$. If neither u is an ancestor of v not b is an ancestor of a, we try to find the highest ancestor of u which is not an ancestor of v (so, parent of that highest ancestor of u is an ancestor of v). In other words, we are looking for a vertex x such that x is an ancestor of u, x is not ancestor of v, and $P[x][0]$ is an ancestor of v. We can find such x using the precomputed values in $\mathcal{O}(\log n)$ time as follows. Start with $j = \log n$ and $x = u$. Repeat as long as $j \geq 0$: if $P[x][j]$ is an ancestor of v decrease j by one, otherwise set $x = P[x][j]$ and decrease j by one, too. Each check is done in constant time using the method described in the previous part.

(c) Consider the following procedure: start at the root and traverse the whole tree in a depth-first fashion, at each step moving either one edge down or one edge up. Prove that the total number of steps is $\mathcal{O}(n)$. Then consider the sequence of numbers $1, -1$ denoting whether you went one edge down or up. What is the relation between the LCA of two nodes and an array of prefix sums of your sequence? Observe that this relation allows you to reduce LCA on a tree of size n to bounded RMQ on an array of length $\mathcal{O}(n)$.

Solution: Each edge is traversed twice. In a tree the number of edges is $n-1$, so the total number of steps is $\mathcal{O}(n)$. Each prefix sum is really the depth of some node. More precisely, the whole prefix sums array corresponds to a traversal of the tree, where for each node we first output its depth, then recurse on the left child, then output the depth again, then recurse on the right child, and output the depth once more. At every step we go one level up, or down, so the difference between 2 adjacent values is $+1$ or -1 To compute the LCA of nodes u and v, find the RMQ between (any of) their occurrences in the prefix sum array.

(d) The *Cartesian tree* of a given array $A[1..n]$ is recursively defined as follows: choose the minimum element $A[i]$ and make it the root. Then recurse on $A[1..i-1]$ and make the resulting tree the left child of the root, and recurse on $A[i+1..n]$ and make the resulting tree the right child of the root. Show how to construct the Cartesian tree in $\mathcal{O}(n)$ time by starting with the empty tree, and then constructing the trees for $A[1..1]$, $A[1..2]$, $A[1..3]$, $\ldots$, $A[1..n]$ (this is probably the most complicated part).

Solution: Assume we have created the Cartesian tree C_i for all elements of the array up to i-th position. Then we want to construct the Cartesian tree C_{i+1}, by inserting the element $A[i+1]$ into our tree C_i. We observe that A_{i+1} must be the rightmost element of the new tree C_{i+1}, so we have to find its place in C_i by scanning the right spine (which is the rightmost path of the tree) of C_i. We can perform this operation in constant amortized time, using a stack. More precisely, if a node belongs to the right spine and we scan trough it, it won't belong to the right spine in the whole future.

(e) Find a relation between RMQ query about $A[i..j]$ and a LCA query on the corresponding Cartesian tree. Observe that this relation allows you to reduce RMQ on an array of length n to LCA on a tree of size n.

Solution: The minimum element in $A[i..j]$ is the LCA of i and j in the corresponding Cartesian tree.

(f) Finally, combine the observations to show that RMQ on an array of length n can be reduced to bounded RMQ on an array of length $\mathcal{O}(n)$.

Solution: We have shown that:

1. RMQ on an array of length n can be reduced to LCA on a tree of size n,
2. We can reduce LCA on a tree of size n to bounded RMQ on array of length $\mathcal{O}(n)$.

So RMQ on an array of length n can be reduced to bounded RMQ on an array of length $\mathcal{O}(n)$.