- This problemset has *three* 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. Given two strings s and t, we are interested in computing the length of their longest common substring, which is a string occurring in both s and t. Show how to solve this problem in linear time using the suffix array. You can assume that the lcp array is available, too.

Solution: Construct the suffix array for the concatenation $w = s\#t$ of both strings. Then it's enough to locate, for each suffix $t[i..|t|]$ of t, the suffix $s[j..|t|]$ of s maximizing the longest common prefix. Each $t[i..|t|]$ is also suffix of w, so it occurs in the suffix array. For each i we would like to find the suffix $w[j..|w|]$ with $j \le |s|$ maximizing the longest common prefix. We find two candidates for this j, one being before and one being after $t[i..|t|]$ in the suffix array. Situation is symmetric so let's focus on finding the former.

Among all suffixes $w[j..|w|]$ with $j \le |s|$ occurring before $t[i..|t|]$ in the suffix array we are interested in the one which is as close to $t[i..|t|]$ as possible. So, we can sweep through the suffix array from left to right. Whenever we encounter a suffix $w[j..|w|]$ with $j \le |s|$ we update the best suffix seen so far. Whenever we encounter a suffix $t[i..|t|]$ we take the best suffix seen so far as the suffix maximizing the longest common prefix among all suffixes $w[j..|w|]$ with $j \le |s|$ occurring before $t[i..|t|]$ in the suffix array.

By repeating this twice, we get two candidates for each i. To check which of them is better, we can use the constant time longest common prefix machinery (although simpler solution is also possible: maintain not only the best suffix but also its longest common prefix with the current suffix, this can be updated using the lcp array). Then choose i maximizing the answer.

2. Given a permutation on $\{1, 2, \ldots, n\}$, we want to find a word $w \in \Sigma^n$ such that its suffix array SA_w is exactly the given permutation.

   (a) Show a linear time algorithm solving the problem for $\Sigma = \{a, b\}$.

Solution: We want to construct $w[1..n]$ such that its suffix array is a given permutation $SA[1..n]$. In the suffix array you first have the suffixes starting with a, then starting with b. If you know where the boundary between these two groups is, you know the whole word. Think what happens when $w[1..n]$ ends with ba^k. Then the suffix array begins with n, $n-1$, $n-2$, ..., $n-k+1$, and $n-k$ is somewhere further to the right (so, the suffix array looks like $[n, n-1, n-2, ..., n-k+1, ..., n-k, ...]$

with something between $n-k+1$ and $n-k$; if there is nothing between them, the text looks like $bb...bbaa...aaa$, but then $aa...aa$ gives the same suffix array). Then the boundary must be between $n-k$ and its predecessor. This is because in other case we would have a suffix starting with b that appears before ba^k in the suffix array. So, such suffix is $bx < ba^k$, but this is only possible when the length of x is smaller than k, and all such suffixes start with a.

This gives us a characterization: find largest k such that the suffix array begins with $n, n-1, n-2, ..., n-k+1$, and $n-k$ is somewhere further. The position of $n-k$ gives us the boundary, and we can reconstruct w (and construct its suffix array, and check if it's the same permutation as the given one).

(b) For extra credit: show a linear time algorithm solving the problem for $\Sigma = \{a, b, \dots, z\}$.

Solution: See Theorem 6 in `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.7800`.

(SPOJ) 3. Given two strings of length n, $p = p_1 \dots p_n$ and $q = q_1 \dots q_n$, define $M(i, j, k)$ as the number of mismatches between $p_i \dots p_{i+k-1}$ and $q_j \dots q_{j+k-1}$ (basically it is a substring Hamming distance). Given an integer K, find the maximum length L such that there exists pair of indices (i, j) for which we have $M(i, j, L) \le K$.

Solution: Guess $\delta = i - j$. Then image that you align p and q with a shift of δ. Then find the longest fragment $p[i..i+k-1]$ such that the part of q below, so $q[\delta+i..\delta+i+k-1]$ is exactly the same. This can be done by (conceptually) creating a new string t, where $t[i] = 1$ or 0 depending on whether $p[i] = q[\delta+i]$. Then if $K = 0$ we are simply looking for the longest run of ones in t. If $K > 0$ the situation is slightly more complex: we want to find the longest substring containing at most K zeroes. So, sweep t from left to right. For each j maintain the smallest i such that $t[i..j]$ contains at most K zeroes. Then when we increase j by one, we should either keep i unchanged, or increase it by one, too. The complexity, for a fixed δ, is $\mathcal{O}(n)$, so the total time is quadratic.