FIRST EUROPEAN JUNIOR OLYMPIAD IN INFORMATICS
SOFIA • BULGARIA • 2017

Analysis

This task looks like a direct dynamic optimization problem but this is not correct. There have to be made some important observations and after them the task becomes more straightforward.

The first subtask is for 10 points. No observations are needed to be made here – considering the constraints it is enough for every substring to be viewed and to be counted how many times every type of letter is met. Next the counts for every type of letter are checked and if they are all equal then the current substring is counted in the answer. The expected complexity is $O(n^3 + kn^2)$.

The second subtask is for 20 points. Here the idea is to optimize what we did before. We will remove the inner cycle – we will calculate the count for every type of letter for a current substring in constant complexity. There are different ways to do that. The author precomputes the prefix counts for every type of letter and with this the count for every type of letter for a current substring can be calculated as a subtraction of prefix counts. So the complexity here is $O(kn^2)$.

The third subtask is for 30 points. We have to make the first observation in the task which is for an easier variant – when we only have two types of letters. Let we do the following change of the letters in the string. We replace the first type of letters with the number $+1$ and the second type – with the number -1 (negative 1). In this way our string becomes an array of numbers. The needed substrings are with equal number of letters from the first and the second type i.e. the sum of the numbers in the array in these positions would be 0 which means we are looking for subintervals with 0 sum in the array. This task is solved easier than the original. Let we calculate the prefix sums in the array $pref$ (we have a fictive prefix sum in the beginning which is 0). The sum of a subinterval of the array from x to y is $pref[y] - pref[x-1]$. We want this to be 0 i.e. for a given y the subintervals ending there with zero sum are with x-es for which $pref[x-1] = pref[y]$. To solve the current subtask we can save how many prefix sums are there for every possible value of a sum in the array cnt and for a current index ind the number of subintervals with the needed property ending there are $cnt[ind]$. We add this to the value of the answer and then we need to make the update: $cnt[ind] + +$. The complexity of the described procedure is linear – $O(n)$ but the solution for the whole task isn't so here it can be made a solution with complexity $O(n \log_2 n)$.

The fourth subtask is for 40 points. The previous subtask was useful because the idea can be adapted for the whole task. Let we fix two types of letters – a and b (these are only fictive and the letters can be different). Let we save the number values in the array $nums_1$. Again in the same way we replace the letters of type a with $+1$, we replace the letters of

type b with -1 but for every letter which is from a different type we replace it with the number 0. Let we take the second type of letters – b and a new type of letters – c and we will save the numbers in a new array $nums_2$. After we make the replacements and save the numbers using the described scheme we take c and another different type of letters (for example d) and we again fill in an array $nums_3$ and we do this while we still have more unused types of letters. A substring that fulfills the requirement in the statement is such that the sums of the numbers in the corresponding subintervals in the arrays $nums_1, nums_2, \dots, nums_{k-1}$ are all zeroes. Now we can approach analogously as in the previous subtask. Here we have a problem – the array cnt which we used now has to have several numbers as a parameter. We can use hashes which isn't expected to be known by the competitors so the author's solution doesn't use this despite providing complexity of $O(kn)$. We can use the data structure map from STL but we have to predefine operator and there will be a big constant which will make the solution slower. The author approaches in the following way. Let for every array $nums_i$ we calculate the corresponding prefix array $pref_i$. For every position i we can make this type of sets S_i $\{pref_1, pref_2, \dots, pref_n\}$. In reality we search for pairs of sets S_x and S_y (of course $x < y$) for which $S_x \equiv S_y$. Let we sort the sets S_i. Now the equivalent sets are adjacent. Let we view all groups of equivalent sets and their number is p and every group's length is num_i. The number of all substrings which fulfill the requirement from the statement is $\sum_{i=1}^{p-1} num_i\,(num_i - 1)$. This is easy to be calculated and solves the task for 100 points. The complexity of the described procedure is $O(kn \log_2 n)$ – this is the complexity of the sorting of the sets S_i which is the dominating in the solution.

The thing which shouldn't be forgotten is that we have to make the calculations by the modulo in the statement. In reality it turns out that the answer can't be too big (the more types of letters the small it gets) – using a convenient sample the maximal number for a given n and k is $\left\lfloor \frac{n}{k} \right\rfloor (n+1) - \frac{k\left\lfloor \frac{n}{k} \right\rfloor \left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right)}{2}$ which for the maximal constraints isn't much bigger than the modulo. The reason there is a modulo is for deception that the answer will be very big and to prevent the competitors from doing cheat solutions. The task is interesting because of the approach that the letters are replaced with numbers which is the big step towards finding a solution with smaller complexity.