EGOI 2025 Editorial - Giftboxes

Task Author: Anja Dožić

July 16, 2025

The problem

The organizers are giving gift boxes to competitors and want to gift one box to each team. There are T teams and N competitors waiting in line for gifts. The problem is that girls in a line are mixed up such that the girls from the same team might not be standing next to each other, and there may be multiple girls per team in the line. The organizers have to pause the gifting process once and let some girls pass without receiving the gift boxes. The goal is to maximize the number of teams receiving gift boxes, but to avoid giving more than one gift box to the same team (we can give a gift box to at most one competitor from one team, while others must be passed).

Subtask 1: $N = T + 1$

In this subtask, only one team has two members in the line, while the remaining all have exactly one.

Here, it is possible to gift every team by passing one of two participants from a team that appears twice. This is true because when we remove that one participant from an array, we will have an array with all distinct elements (because all other participants are from different teams).

To find the team that appears twice, go through the given array and mark every element once it appears. This can be done by maintaining a separate array b of boolean values, where the value of $b[i]$ is changed to *true* once the number i appears. If $b[i]$ is already true, that means that i has already appeared in the array once and we found the second appearance, so we can print index i twice to remove that participant.

Subtask 2: $N = 2 \cdot T$ and every team will appear once in each half

In this subtask, it is also be possible to gift every team. This can be done by choosing a half and passing it completely. This strategy is correct because it is guaranteed that there is exactly one appearance of each team in each half, so if we remove one half, the other will contain all elements from 0 to $T - 1$ exactly once. Therefore, printing 0 and $N/2$ or $N/2 + 1$ and N solves this subtask.

Subtask 3: $T, N \leq 500$

This subtask can be solved by iterating through all possible pause intervals and choosing the best one. In detail, this works as follows:

Iterate through all possible start and end points of the pause interval. For a specific interval, we have to check whether the array consists of distinct numbers after removing that interval. This can be done using a similar strategy as in the first subtask. In the end, we output the shortest interval that fulfilled this condition.

Note that the time complexity of this solution is $O(N^3)$, which is fast enough this subtask, but is too slow for the remaining subtasks.

Subtask 4: $N = 2 \cdot T$ and every team will appear twice

In this subtask, the problem boils down to finding the shortest subarray that contains all elements at least once. This is sufficient because if all elements appear twice, it is necessary to remove at least one of the appearances for each of them.

This subarray can be found using the two-pointer method. At the beginning, the left pointer is on the first element, while the right pointer is on the first appearance of the element that appears last. The subarray between those two pointers contains all elements at least once (since it contains the first appearance of each element).

Now, we need to move the pointers in a way such that the array between them always satisfies this condition. This can be done in the following way:

In each round, first move the left pointer right by one. Then, we removed the first appearance of an element from the subarray. Because we need the subarray to contain all elements, we need to make sure that it will contain the second appearance of the removed element. This is done by moving the right pointer right until the subarray contains the second appearance of the removed element.
In the end of each round, we calculate the length of the constructed subarray. This process ends once the left pointer reaches the second appearance of some element. Finally, the shortest valid subarray is chosen.

Subtask 5: $M, N \leq 5000$

In this subtask, we are aiming to reduce our time complexity of Subtask 3 from $O(N^3)$ to $O(N^2)$. To do that, we want to accelerate the check whether a subarray satisfies the conditions.

First, count for each element how often it appears. Then, we again iterate through all subarrays. First, fix the start of the subarray and iterate through all possible endings. While doing this, we can maintain an array b, such that $b[i]$ is the number of appearances of the i-th element in the subarray. Also, maintain a counter c of the number of elements that appear more than once outside of the subarray. For an element i, this is the case precisely if the difference between $b[i]$ and the total number of appearances is 1 or more. A subarray satisfies the condition if and only if $c = 0$. Both b and c can be updated in constant time when the ending of the subarray is moved right by one. In the end, the shortest subarray that satisfies the condition is chosen.

Subtask 6: No additional constraints

For the full solution, the following observation is needed: if an element appears more than twice, every appearance between the first and the last will be in the subarray we intend to remove.

Firstly, why is this true? Let us assume that we have one element which appears more than twice, and there is an appearance that is not first nor last and that is not in the chosen subarray. The subarray must be either left or right from the appearance of the element. If it is left, then we know that the last appearance of the element, as well as the one we are looking at, will be in the array that is left when we remove the chosen subarray. That means that the conditions of the task are not satisfied. If it is right, we get an analogous contradiction.

We will call those appearances (those which are neither first nor last) *middle appearances*. Using the observation, we can compress the given array in the following way: we will remove the subarray bounded by the first and last middle appearances among all elements because we know those numbers will definitely be in the chosen subarray.

Now, we simplified the full task to a subtask 4 with two additional constraints:

1. Elements appear at most twice, so there might be some that appear once.
2. We must include the removed elements in the chosen subarray.

Since the elements that appear at most once can be ignored when choosing the subarray, we can use a similar algorithm as in subtask 4, and then unite the found interval with the interval that we already removed.