EGOI 2024 Editorial - Bike Parking

Problem author: Nils Gustafsson.

The problem

There are N tiers of parking slots, numbered from 0 (best tier) to $N-1$ (worst tier). There are x_i parking slots in the i-th tier, and y_i users that target the i-th tier. There are enough slots for all users overall ($\sum_i y_i \leq \sum_i x_i$), but not necessarily within the target tier. We need to assign exactly one slot for each user. Our score is defined by U-D, where U is the total number of users assigned to a lower tier than their target and D is the total number of users assigned to a higher tier than their target. In other words we get $+1$ for assigning to a lower tier than target, 0 for assigning to the target tier, and -1 for assigning to a higher tier than target. What is the maximum possible score?

Test group 1: two tiers

In this test group, $N = 2$ and $x_i, y_i \leq 100$.

A user with target tier 0 gives a score of 0 when assigned to tier 0, and a score of -1 when assigned to tier 1, while a user with target tier 1 gives a score of $+1$ when assigned to tier 0 and a score of 0 when assigned to tier 1. So tier 0 always gives 1 point more than tier 1. Therefore all users are essentially equivalent, and we should just make sure to fill tier 0 spots before filling tier 1 spots. As the number of tiers is fixed, the time complexity is constant ($O(1)$).

Test group 2: everything of the same size

In this test group there exists a constant c such that $x_i = y_i = c$ for all i.

We can get a score of 0 by just assigning everybody to their target tier. However, when $N \geq 3$ we can do better: let us assign everybody to their target tier minus one, but assign users with target tier 0 to tier $N-1$.

This way the score will be $(N-2) \cdot c$, as we will get $+1$ from $N-1$ tiers and -1 from only one tier. We can implement this while just traversing through the tiers and the users, so, the time complexity of this approach is $O(N)$.

Test group 5: general case

In the last test group $N \leq 3 \cdot 10^5$, $x_i, y_i \leq 10^9$.

The solution is greedy, but of course there are many greedys that work and many that do not.

The key idea is: suppose we have already finalized some partial assignment of users to slots in an optimal solution. Let us look at the remaining yet unassigned users and slots. Consider a user with target tier a and slot in tier b such that $a > b$ and there are neither unassigned users

nor unassigned slots in all tiers c such that $a > c > b$. Then we can assign user a to slot b and the solution can still be completed to an optimal solution.

In order to prove this, let us use the *exchange argument*: suppose there is a solution with a higher score where user a is assigned to another slot d, and another user c is assigned to slot b (in case slot b is free in an optimal solution, the same construction works too).

Let us take that solution but reassign a to b and c to d. What happens to the score? By the choice of a and b, c and d are not between a and b, and moreover $d \neq b$ and $c \neq a$ (otherwise we do have a user with target tier a already assigned to a slot from tier b), so we have the following cases remaining:

- $d < b < a < c$: in this case the score for these two users changes from $+2$ to $+2$, so the new solution is also optimal.
- $b < a < d$: in this case the score for a changes from -1 to $+1$, so no matter what happens to c the new solution is optimal.
- $c < b < a$: in this case the new score for a is $+1$, and the old score for c is -1, so the situation could have only improved and the new solution is optimal.
- $b < a = d$: in this case the score for a changes from 0 to $+1$, and the score for c decreases by at most 1 since c cannot be between b and d, so the new solution is optimal.
- $d <= c = b < a$: in this case the new score for a is $+1$ and the new score for c is at least as good as the old score for c, so the new solution is optimal.

In all cases we find an optimal solution where a is assigned to b.

Therefore, we can greedily assign users to slots lower than their target as long as there are no unassigned users and slots in between. There are many ways to do it, here is one:

In the first pass, let us go through the users in the increasing order of tier and try to assign slots in a lower tier (for a score of $+1$) to everybody. If we have multiple options, we choose the slot with a higher number (as close as possible to the target tier, but still lower than it). If we cannot do it, we skip the user for now.

Having done those assignments we can no longer get any additional $+1$ scores, so we should focus on getting on as many 0 scores as possible. In order to do this, we run a second pass over all tiers and assign as many users to their target tier as possible.

The proof above is quite tedious, but of course you did not to be as rigorous during the contest. As soon as you have a part of that proof or the corresponding intuition, it is completely reasonable to implement and submit, since the code is really short.

In order for this solution to run in time, we need to process the users with the same circumstances in batches. So if the greedy solution above assigns a user from tier a to a slot in tier b, the total amount of unassigned users in tier a is u, and the total amount of unassigned slots in tier b is v, we need to do the same assignment $\min(u, v)$ times in one go. When we do this, either the number of users or the number of slots in a tier goes to 0 every time, so the running time of this approach is $O(N)$.

If you found some but not all ideas needed to solve the general case, you might already be able to solve test groups 3 or 4. For example, in test group 3 you can afford to go through users one by one instead of in batches, and in test group 4 you can afford to be quadratic in the number of users, or cubic or higher in the number of tiers.