Brought Down the Grading Server? (balance)

by Lukas Michel

We say that the submissions to a task are balanced if the maximum and minimum number of submissions for this task during the rejudging differ by at most one.

Subtask 1. $S = 2$ and $N, T \leq 20$

In this subtask, we can simply enumerate all possible ordered assignments and output one for which the submissions to all tasks are balanced. This can be implemented in time $O(2^N \cdot (N + T))$.

Subtask 2. $S = 2$

First, consider the case where the total number of submissions to each task is divisible by 2. In fact, in this case we can assume that every task has exactly 2 submissions: we can simply replace a task with s submissions by $s/2$ tasks with 2 submissions each. If the number of submissions to each of the new tasks is balanced, this is also true when we replace them again with the initial task.

Now we have to make sure that the two submissions to each task and the two submissions in the list of each core are assigned to different timeslots. Note that these submissions form cycles of submissions that are pairwise either to the same task or in the list of the same core. Once we assign one of these submissions to a timeslot, this means that the other submission of the corresponding task and the corresponding core have to be assigned to the other timeslot. This, in turn, implies that two other submissions have to be assigned to the same timeslot as the initial submission, and so on.

This means that once we assign a single submission of a cycle to a timeslot, this determines uniquely to which timeslots all other submissions of this cycle have to be assigned to. Moreover, by construction, the two submissions to each task are assigned to different timeslots, and so this ensures that the submissions to all tasks are balanced, as required. This can be implemented in time $O(N + T)$, solving the first group in this subtask.

If there are tasks whose number of submissions is not divisible by 2, we can replace them by multiple tasks with 2 submissions each and one task with 1 submission. Then, in addition to cycles, we will also have paths, but the same construction also applies to them. This can still be implemented in time $O(N + T)$, and solves the entire subtask.

Subtask 3. $N \cdot S \leq 10\,000$

From this subtask on, we frame the problem in the terms of graph theory. More precisely, we construct a bipartite graph with the left vertices being the cores, the right vertices being the tasks, and the submissions being edges between them.

Again, we will first focus on the case where the number of submissions to each task is divisible by S, or equivalently where the degree of every right vertex of the bipartite graph is divisible by S. As before, we can assume that every degree is exactly S by splitting a vertex of degree d into d/S vertices with degree S each.

Now, for each minute, our assignment has to pick exactly one submission from the list of every core while also selecting exactly one submission to each task. In our bipartite graph, such a set of submissions corresponds to a perfect matching. Fortunately, it is known that every regular bipartite graph—that is, a bipartite graph where the degrees of all vertices are the same—has a perfect matching. This can be proven using Hall's Theorem, for example.

From this, we get the following algorithm: first, compute a perfect matching in the bipartite graph, and let these be the submissions evaluated by the cores in the first minute. Then, remove those edges from the graph. The resulting bipartite graph is still regular, so we still know that it contains a perfect matching. Therefore, we can repeatedly compute perfect matchings and remove them from the graph until we have a set of submissions evaluated by the cores for every minute.

Since the bipartite graph has N left vertices and $N \cdot S$ edges, there are simple matching algorithms that run in time $O(N^2 \cdot S)$. This yields an overall runtime of $O(N^2 \cdot S^2)$, solving the first group in this subtask.

Similar to before, if the degrees of some right vertices are not divisible by S, we can replace them by multiple vertices with degree S each and one vertex with degree in $[1, S-1]$. However, in this bipartite graph, simply finding complete left-to-right matchings and removing them repeatedly might not work as we have no guarantee that such matchings will always exist.

Instead, we can transform our graph into a regular bipartite graph by adding new left vertices and connecting them to right vertices with degree lower than S. If we pick perfect matchings in this regular bipartite graph, they reduce to complete left-to-right matchings in the original graph, as required. The regular bipartite graph will have at most $N + T$ left vertices, so the previous algorithm runs in time $O((N+T)^2 \cdot S^2)$, solving the second group of this subtask.

Finally, if $T \geq N$, we can observe that before constructing the regular bipartite graph, we can simply merge any two right vertices if the sum of their degrees is at most S. Once such merges are no longer possible, we will have $T \leq 2N$, and so we can apply the previous algorithm. This solves the entire subtask with a runtime of $O(N^2 \cdot S^2)$.

Subtask 4. No further constraints.

In this subtask, instead of removing matchings one-by-one, we will employ a divide-and-conquer approach: we want to split the edges of the bipartite graph into two sets such that for every vertex (both left and right), half of its incident edges, up to rounding, are in each set. Since S is a power of two, we can apply this recursively, and this will produce the required balanced ordered assignment: this splits the edges incident to left vertices equally, so every core will evaluate exactly one submission per minute, and it also splits the edges incident to right vertices equally, which means that the submissions to any task will be balanced in the end.

To split the edges of the bipartite graph into two such sets, we can use Euler tours. First, if the degree of every vertex is even, we can take any Euler tour. Then, we put every second edge into one set and every other edge into the other set. Since consecutive edges are in different sets, this ensures that exactly half of the edges incident to any vertex end up in the first set and half of the edges end up in the second set, as required. If every degree was divisible by S in the beginning, the degree of every vertex will stay even throughout this process, and so this solves the first group of this subtask. Since Euler tours can be computed in linear time, this gives a runtime of $O(N \cdot S \log S)$.

Finally, whenever some right vertices have odd degree, we can add a new left vertex connected to all of these right vertices with odd degree. Then, every vertex will have an even degree, and we can apply the approach from before to partition the edges into two sets. If we remove the additional left vertex, this still guarantees that for every vertex, up to rounding, half of its incident edges are in each set. This solves the entire task, with a runtime of $O(N \cdot S \log S)$.

Final remarks.

The task could also be solved if S is not a power of two. For this, we can combine the ideas of Subtasks 3 and 4. If S is odd, we can remove a perfect matching from the graph, and if S is even, we can split the edges into two sets as above. Overall, this would give a runtime of $O(N^2 \cdot S \log S)$, or better if the matching algorithm is more efficient. However, this would be more annoying to implement, and it was difficult to prevent efficiently implemented matching solutions to subtask 3 from solving these testcases as well, which is why such a subtask was not included in this problem.