EGOI 2026 Editorial - Ovenmasters

Task Author: Tomohisa Hachiya, Yuki Tanaka

Let us call the pizza currently lying on a table the *top* pizza of that table. After the first M arrivals all tables are occupied. From then on, a baker with rank x chooses the table whose top pizza has the largest rank smaller than x. If no top pizza has rank smaller than x, the baker leaves and does not appear in any tray stack.

The first useful observation is about a single table. Whenever a baker replaces a pizza on a table, her rank must be larger than the rank of the pizza she ate. Therefore, in every stack, the ranks have to be increasing from bottom to top. If this is not true, the answer is immediately `NO`.

The first M arrivals are also fixed: they are the bottom elements of the M stacks, in table order. After this, the actual table labels no longer matter. The only relevant information is the sorted order of the current top pizzas.

Subtask: $M = 1$

With one table, the process is very simple. The stack must be increasing. Any baker not in the stack has to leave, which is possible only after the current top pizza has become larger than that baker's rank.

Thus we can start with the first stack value and then repeatedly take the smallest rank that is currently possible:

- either the next value in the stack, which replaces the current top;
- or a missing value smaller than the current top, which leaves.

If at the end some missing value is larger than the final top pizza, it could never have left, so the instance is impossible.

Subtask: $M = 2$, $N \leq 200$, and no baker leaves

In this subtask every baker appears in one of the two stacks. After the first two arrivals, the next arrival must therefore be the next unused value of one of the two stacks.

Let table 0 be the table with the smaller bottom value, and table 1 the table with the larger bottom value. Their current top values are always $c_0 < c_1$.

Now consider a possible next arrival with rank x.

- If $c_0 < x < c_1$, then table 0 contains a pizza better than x, while table 1 does not. So x must go to table 0.

- If $x > c_1$, then both tables contain a pizza better than x, and the worse of those two pizzas is c_1. So x must go to table 1.
- If $x < c_0$, then there is no better pizza for x. This case cannot happen in this subtask, because no baker leaves.

Therefore, to decide whether the next unused value of a stack can be taken now, we just check whether it falls into the interval for that stack.

At each step there are at most two candidates: the next unused value of the first stack and the next unused value of the second stack. We test which of them are currently possible and append the smaller possible one to the answer. If neither candidate is possible, the answer is `NO`.

This greedy choice gives the lexicographically smallest order, because every valid order with the current prefix must choose one of these possible candidates next. The smaller one is therefore always best.

The implementation can be completely direct and runs in $O(N)$ time.

Keeping the tables sorted

Suppose the current top ranks of the tables, sorted increasingly, are

$$c_0 < c_1 < \cdots < c_{M-1}.$$

A baker with rank x will choose table i exactly when

$$c_i < x < c_{i+1},$$

where for the last table there is no upper bound. This is because c_i is then the worst available pizza that is still better than x.

In particular, if table i is chosen, its top changes from c_i to some value $x < c_{i+1}$. Thus the sorted order of the tables does not change. This lets us sort the tables once by their initial top pizza and keep them in that order for the whole solution.

It is also convenient to add one fake table for the bakers who leave. Its current value is -1, and it is placed before all real tables. A baker who does not appear in any stack can be handled as belonging to this fake table: such a baker may arrive exactly when

$$-1 < x < c_0,$$

which means that no real table has a better pizza for her.

Subtask: $N \leq 200$ and no baker leaves

For more than two tables, the same idea still works. Every baker appears in some stack, so the next arrival must be the first unused value of one of the stacks.

We keep the current top value of every table. Repeatedly look at the next unused value of each stack, check whether that value would currently choose this table, and append the smallest value that passes the check. If no stack has a possible next value, the answer is `NO`.

The lexicographic argument is the same as above: after a fixed prefix, we choose the smallest value that can legally be the next arrival. A straightforward implementation is fast enough for $N \leq 200$, for example in $O(N^2M)$ time.

Subtask: $M \leq 10$

For larger N but small M, we should avoid scanning all unused ranks after each arrival. Instead, we will scan the tables.

First sort the real tables by their first stack value. We also add the fake table for bakers who leave. After sorting, think of these as rows $0, 1, \ldots, M$, where row 0 is the fake table and the remaining rows are the real tables.

Each row i stores two things:

- a current value c_i, initially its first value (-1 for the fake row);
- a list of remaining values that still have to be processed in increasing order.

For a real row, the remaining values are the rest of its stack. For the fake row, they are all ranks that do not appear in any stack, sorted increasingly.

The values c_i are always sorted:

$$c_0 < c_1 < \cdots < c_M.$$

For row i, let u_i be the first value in its remaining list. This is the only value from row i that can be processed next, because values in one stack have to appear in their stack order.

When can we process u_i? The answer is exactly when it would currently choose row i. This means

$$c_i < u_i < c_{i+1},$$

with no upper bound when $i = M$. For the fake row, this says $c_0 < u_0 < c_1$, which is exactly the condition that u_0 has no better pizza available and leaves.

Now scan the rows from left to right and find the first row whose u_i is currently possible. This gives the smallest possible next arrival. Indeed, if row i is possible,

then $u_i < c_{i+1}$. But every value in a later row is larger than its current value, which is at least c_{i+1}. So no later row can give a smaller next rank.

After we find this row, append u_i to the answer, set $c_i = u_i$, and remove u_i from the row. Then start another scan from row 0. If no row is possible but some values are still unprocessed, the answer is `NO`.

This takes $O(NM)$ time after sorting, which is easily fast enough for $M \leq 10$.

Full solution

The full solution is the same row-based algorithm. The only extra care needed is to avoid starting the scan again from row 0 after every processed value.

Build $M + 1$ rows:

- one fake row containing -1 followed by all ranks that do not appear in the input;
- one row for each table stack.

Before sorting the rows, save the first M output values: they are the bottom elements of the original table stacks, in the original table order.

Then sort all rows by their first value. The fake row, whose first value is -1, will become the first row. For each row, remove its first value and store it as c_i. The remaining values in the row are processed from smallest to largest.

Now maintain an index r, initially 0.

- If the current row has no remaining values, move to the next row.
- Otherwise let x be the first remaining value in this row.
- If $x < c_r$, the row is not increasing, so output `NO`.
- If this is the last row, or $x < c_{r+1}$, then x is the smallest possible next arrival. Append it to the answer, set $c_r = x$, remove it from the row, and move one row left if possible.
- Otherwise, x is still too large: it would currently choose a later table. Move one row right.

If the scan finishes and the answer contains all N ranks, output `YES` and the answer. Otherwise output `NO`.

Correctness proof

We prove that the algorithm outputs `NO` exactly for impossible inputs, and otherwise outputs the lexicographically smallest valid order.

First, each real stack has to be increasing. Every time a baker appears in a stack, she replaced a better pizza, hence her rank is larger than the previous top of that table. Therefore a non-increasing stack is impossible, and the algorithm correctly rejects it.

Second, once the real tables are sorted by their current top ranks, their order never changes. If a baker with rank x chooses row i, then $c_i < x < c_{i+1}$, or row i is the last row. After the replacement, the new value of row i is still before row $i+1$ in sorted order.

Third, the fake row exactly represents bakers who leave. A missing rank x can leave precisely when x is smaller than the smallest real current top. This is the same condition as being processable in the fake row.

Now consider any moment during the algorithm. In row i, the first unprocessed value is the only value from that row that can possibly be processed next, because stack order must be preserved. It is possible exactly when it lies between c_i and c_{i+1}, with no upper bound for the last row.

Among all possible next values, the smallest one is in the leftmost possible row: every possible value in row i is smaller than c_{i+1}, while every value in a later row is larger than its own current value, and therefore at least c_{i+1}.

Thus, whenever the algorithm processes a value, it appends the smallest rank that can appear next in any valid continuation. This is exactly the greedy choice needed for lexicographic minimality. If the algorithm cannot process any remaining value, then no valid next arrival exists, so no valid order exists.

Therefore, if the algorithm outputs `YES`, the produced order is valid and lexicographically smallest; if it outputs `NO`, no valid order exists.

Complexity

Sorting the rows and the missing ranks takes $O(N \log N)$ time. The scan over the rows is linear in the number of processed values plus the number of row moves, so it is $O(N + M)$ after sorting.

The memory usage is $O(N)$.