Problem 2

I would like to approach this question from a DP perspective. So I want to think about the state.

- I need to choose exactly `m` dishes from a total of `n`, and the **order** in which I eat them matters because of the potential bonus scores from eating dish `x` before dish `y`.

So at any state in the DP, I want to know:

- Which dishes I've already picked
- What the last dish I picked was (so I can apply the correct bonus if I choose a next one)

My first instinct was to loop through combinations, but I quickly realised that a **bitmask DP** would be ideal here since `n ≤ 18`, allowing `2^n` states. I created a `dp[mask][last]` table that stores the **maximum satisfaction** I can achieve by picking the subset `mask` of dishes, ending with dish `last`.

Initially, I tried implementing this using normal arrays and looping over subsets. I got a solution working on some sample inputs, but failed on larger hidden test cases.

The issue was with the way I was representing the state: I hadn't properly considered the **multiple paths** that could lead to the same subset ending in the same dish. In some cases, I was only considering one path (essentially greedy), instead of keeping track of the **maximum** among all possible valid transitions.

So I refined the approach:

- For each subset `mask`, and for each `last` dish in that mask, I try all `next` dishes not yet picked, and update `dp[new_mask][next]` accordingly.
- I also had to fix a bug with input parsing — the bonus rules used 1-based indexing, but my arrays were 0-based, which led to an out-of-bounds crash.

After maintaining **both the full subset state and the last picked dish**, and ensuring all transitions were correctly tracked with `max(...)`, the solution passed all test cases.

I could have caught this earlier with a better set of custom test cases that ensured bonus rules were non-trivial and required ordering awareness.