Code Jam 2021 - Round 2

Analysis: Hidden Pancakes

Test Set 1

For Test Set 1, the number of pancakes is small enough to do something exponential. One possible way would be to go through every possible order of pancakes. That would be too slow, but we can identify some repetition: any hidden pancakes at any point could be hidden in multiple ways, and that does not affect how we continue the process. Removing the cost of calculating these overlapping cases leads to a dynamic programming solution.

We can formalize the idea above as follows: instead of remembering exactly the current stack of pancakes as the current state, we can store one of 3 states for each pancake: unused, visible, or hidden. The visible pancakes are always stacked in decreasing order of radius, and the position in the stack of hidden pancakes, as argued above, does not affect the process. Thus, we can define a function f recursively that takes a state description and returns the number of valid ways to finish a pancake stack from that state. Applying f to the state in which all pancakes are visible gives the answer of the problem.

Let's define $f(s)$ recursively, where s is a state. If no pancakes are unused in s, then the number of ways to finish it is simply 1. This is our base case. If there are unused pancakes, we can go through all of them to choose which is the next pancake that should be cooked. If s_i is the state that results from cooking the pancake of radius i centimeters when the state was s, then $f(s) = \sum_i f(s_i)$ where the summation is over pancakes that are unused in s.

Since each pancake can independently be in one of 3 states, the domain of f has $3^{\mathbf{N}}$ overall states. The non-recursive cost of computing f is a low-degree polynomial on $\mathbf{N}$. The exact degree depends on the implementation, but it is not too hard to do it in linear time, yielding an overall algorithm that runs in $O(3^{\mathbf{N}} \cdot \mathbf{N})$.

We can also use backtracking to solve the problem. We start building the permutations one element at a time, checking that the $\mathbf{V_i}$s for this partitial permutation match the ones needed for this test case. It can be proven that the complexity of such a solution is bounded by $O(2^{\mathbf{N}}\mathbf{N}^3)$, which is significantly better than $O(\mathbf{N}!)$ of the simplest brute force algorithm, and is enough to solve this test set.

Test Set 2

From the input, we can keep track of the current list of visible pancakes. To make things easier, when a pancake gets covered up in the stack, we will view this as the pancake being "removed" from the list of visible pancakes. We can create a list of inequalities relating the sizes of the pancakes. Let P_i be the radius of the i-th pancake added.

First, it's worth noting that each time we add a pancake to the stack, the size of the list of visible pancakes will either increase by 1, stay the same, or decrease. If the size of the list ever increases by more than one, then this case is impossible so our answer is 0.

Let's consider the possible scenarios when we are adding the i-th When the size of the list increases by 1 ($\mathbf{V_i} = \mathbf{V_{i-1}} + 1$), we know that the new pancake is smaller than all of the pancakes in our list. We can add the inequality $P_x > P_i$ where x is the pancake that was previously at the end of the list.

If the size of the list stays the same or decreases ($\mathbf{V_i} \le \mathbf{V_{i-1}}$), then we know that the new pancake is larger than the last $\mathbf{V_i} - \mathbf{V_{i-1}} + 1$ pancakes on the list and each of these pancakes gets removed from our list. We can add the inequality $P_i > P_x$ where x is the last pancake we removed from the list. Also, if $\mathbf{V_i} > 1$, then the new pancake is strictly smaller than the rest of the pancakes in the list. In this case, we can add the inequality $P_y > P_i$ where y is the first pancake we did not "remove".

Note that in the latter case, we would already have added an inequality saying that $P_y > P_x$. But, this inequality is redundant now that we have the extra inequalities $P_y > P_i$ and $P_i > P_x$. Therefore, we can remove the inequality $P_y > P_x$ to prevent us from having any redundant inequalities that can cause us issues later.

After removing the redundant edges, each pancake is on the right side of at most one inequality in the form of $A > B$. Because of this, the inequalities can be modeled as a tree where we have an edge from A to B if $A > B$. We can then solve for the number of valid cooking orders of $\mathbf{N}$ pancakes recursively starting at the root of the tree which is the largest pancake. Note: the largest pancake is whichever pancake was at the beginning of our list at the end.

Let $s(i)$ be the number of pancakes in the subtree rooted at i. Also, let $f(i)$ be equal to the number of valid permutations of the pancake sizes (ranging from 1 to $s(i)$) in the subtree starting at i.

To solve for $f(i)$ we need to count the number of ways we can assign pancake sizes to the subtrees belonging to our direct children. Then, each child subtree can permute itself in $f(j)$ ways (where j is a direct child of i). So, $f(i)$ is the product of all $f(j)$'s and the number of ways we can assign pancake sizes to our subtrees.

Since i is the largest pancake in our subtree, it must be given the largest size. The other $s(i) - 1$ pancake sizes for our subtree can be assigned to any subtree. The number of ways to do this can be counted using Multinomial Coefficients.

If we precompute the factorials and inverse factorials (to allow for division under mod), we can count the number of valid cooking orders in $O(\mathbf{N})$. Building the tree of inequalities also can be done in linear time. This gives us a final time complexity of $O(\mathbf{N})$ (assuming we precompute factorials and inverse factorials in linear time).

An alternate solution

There is a different solution to the problem that requires two observations: (1) the largest pancake (the one with a radius of $\mathbf{N}$ cm) can only be placed at position k, where k is the largest integer such that $\mathbf{V_k} = 1$ and (2) since the pancake with a radius of $\mathbf{N}$ covers all other pancakes, the order of the pancakes before the largest pancake does not impact the $\mathbf{V_i}$s for the pancakes after the largest pancake. This allows us to split the problem into two independent parts. We solve the left part and the right part separately. For the right part, the largest pancake will always be visible, so we subtract 1 from all $\mathbf{V_i}$ in the right part to account for it (note we do not actually do the subtraction, because that would be too slow, but we just implicitly do it). We must also choose which pancakes were in the left and right parts (there are $\binom{\mathbf{N}-1}{k-1}$ ways of doing this where the largest pancake was at index k). As base cases: If the range is empty, then there is 1 valid ordering, and if there is no $\mathbf{V_i} = 1$, then there are 0 valid orderings. Otherwise, the product of the left part's answer, the right part's answer, and the binomial coefficient is the total number of orderings.