Code Jam to I/O for Women 2022 - Code Jam to I/O for Women 2022 Ingredient Optimization

PROBLEM ANALYSIS

Analysis

The problem is talking about two types of events:

- Delivery: when an amount of leaves is added to the stock with a specific expiry time.
- Order: when an amount $\mathbf{U}$ of leaves is removed from the stock if we have enough.

In order to keep track of the amount of stock we have at some point of time, we can loop through the events in a sorted order by delivery/order time. This approach helps us avoid using invalid delivery for a certain order (For example: a delivery at time 3 is not valid to be used in an order at time 2).

For example: If we have a delivery at time 1, 3, 5 and order at time 2, 3, 6, we will process them in the following order:

[Delivery1, Order2, Delivery3, Order3, Delivery5, Order6]

Note that if an order and delivery have the same time stamp, we need to process the delivery first.

This can be done by using two pointers (one for the deliveries and one for the orders) and pick the smaller delivery/order time first. Another alternative is adding all of them into one array and sort it.

Test Set 1

For Test Set 1, no Thai basil leaf spoils before an order comes. So, we don't really care about the expiry time. In this case, we can represent the stock by a single number and loop through the delivery/order events. Whenever we have a:

- Delivery: the stock is increased by the delivered amount.
- Order: the stock is decreased by $\mathbf{U}$ amount only if we can fulfill the order.

```
if currentEvent is Delivery:
    stockAmount += currentEvent.deliveredAmount
else if currentEvent is Order:
    if stockAmount >= U:
        stockAmount -= U
        fulfilledOrders += 1
```

Time Complexity

Since the deliveries and orders are given in ascending order, we only need to go through each event. Therefore the time complexity equals $O(\mathbf{N} + \mathbf{D})$.

Test Set 2

For Test Set 2, we need to take the expiry time for each delivery into consideration. The key observation is when we have multiple leaves to choose from, we must always choose the $\mathbf{U}$ leaves that spoil first. This way we increase the chances to fulfill more future orders. Therefore, having a sorted list of deliveries by expiry time will help optimize using the leaves.

We need to store a sorted list which contains detailed information about deliveries. For each delivery, we need to store the delivery time, expiry time, and amount of Thai leaves. Whenever we have a:

- Delivery: the delivery information is added to the deliveries list in the correct position so that all the deliveries that spoil before it are placed before it on the list. Also, the amount of the stock is increased by the deliveried amount.

- Order: we check our stock amount. If it's greater than $\mathbf{U}$, then we can fulfil this order. In this case, we must use the $\mathbf{U}$ non-spoiled leaves that will spoil first in the future and remove all the used leaves from the list of deliveries.

Note that we continuously need to remove the spoiled leaves before processing the delivery/order event.

The approach explained above is using the insertion sorting algorithm to add the delivery in the correct order which gives a time complexity $O(\mathbf{D}^2)$. A better approach is to use a Min-Heap, such that the earliest leaves to spoil are always found on the top of the heap. In this case, adding a delivery to it's right place in the heap will be $O(\log \mathbf{D})$. While, removing the used leaves from the heap should be $O(\log \mathbf{D})$ as well.

Time Complexity

In case of using insertion sort, the total complexity of inserting the deliveries in their right place will be the complexity of insertion sort, $O(\mathbf{D}^2)$. And for fulfilling the orders, we iterate through $O(\mathbf{N})$ orders and visit and remove at most $O(\mathbf{D})$ deliveries from the list. So the total complexity is $O(\mathbf{D}^2 + \mathbf{N})$.

In case of using a Min-Heap, we insert to the heap for each delivery, and remove a delivery from the top of the heap when it is used up or spoiled, so there are totally $O(\mathbf{D})$ insertions and removals on the heap. The top of the heap is checked for each order and whenever a previous top is removed, so there are totally $O(\mathbf{D} + \mathbf{N})$ checks. Since the operation on a heap is $O(\log \mathbf{D})$, the total complexity $O((\mathbf{D} + \mathbf{N}) \log \mathbf{D})$.

Test Data

We recommend that you practice debugging solutions without looking at the test data.