Kick Start 2022 - Round F

Analysis: Story of Seasons

For each type of seeds i, let us use $deadline_i$ to represent the latest day it should be planted to earn profits, where $deadline_i$ is calculated as

$$deadline_i = \mathbf{D} - \mathbf{L_i}.$$

Which means if one seed of type i is planted on or before $deadline_i$, it can mature and let us earn $\mathbf{V_i}$ eventually. Otherwise it cannot. In other words, for each type i, if the current day is already later than $deadline_i$, then all seeds of type i are not worthy to plant anymore.

Test Set 1

Since each vegetable has one seed and we can plant only one seed per day, we should plant one seed everyday from the day one, until there is no seed available or worth to plant. Since $\mathbf{N}$ is very small in this case, we can use brute force search with dynamic programming to implement it.

We can maintain a set S that contains the types of seeds that are not planted yet. And on each day d, we will try to plant every type i from S. If $deadline_i \leq d$, then we can earn $\mathbf{V_i}$ from this day's work, otherwise we earn 0. And then we go on to the next day d with remaining types $S - \{i\}$ to see what maximum profit we can earn. Formally, we have the recursive function as below:

$$Profit(d, S) = MAX_{i \in S}(Earn(d, i) + Profit(d + 1, S - \{i\})),$$

where $Earn(d, i) = \mathbf{V_i}$ if $d \leq deadline_i$ otherwise $Earn(d, i) = 0$. Note that if d is larger than $\mathbf{D}$ or if S is empty, then $Profit(d, S)$ should directly return 0. Then $Profit(1, \{1, \ldots, N\})$ is the answer of the problem.

We can use dynamic programming for the search. That is, prepare a table to store the answer of every state (d, S). Note that the value of d is actually implied by the associated S that $d = \mathbf{N} - |S| + 1$. Therefore the total number of states equals to the number of combinations of S, which is $O(2^{\mathbf{N}})$. And in each searching state, we iterate through all the available types in S, which is an $O(\mathbf{N})$ operation. Therefore, the time complexity of this solution is $O(\mathbf{N} \times 2^{\mathbf{N}})$.

Test Set 2

Instead of searching for all possible combinations, we can use greedy strategy on this problem. For simplicity, let us still use the limits that $\mathbf{X} = 1$ and $\mathbf{Q_i} = 1$ for all i.

We might not know which type to plant on day 1 in order to achieve the optimal solution, but we can find out which type to plant on day $\mathbf{D} - 1$. We just need to gather all types that have $deadline_i \geq \mathbf{D} - 1$ and choose the one with the maximum $\mathbf{V_i}$ to plant on day $\mathbf{D} - 1$.

Let us prove that this is always the best choice. Say type g is the one we select for day $\mathbf{D} - 1$ in our greedy strategy, we need to prove that for any other schedule that does not plant type g on day $\mathbf{D} - 1$, we can always achieve better or the same profit by modifying it to plant g on day $\mathbf{D} - 1$:

- If a schedule does nothing on day $\mathbf{D} - 1$, and the seed g does not exist in this schedule: We can put g in day $\mathbf{D} - 1$ of this schedule to increase the total profit by $\mathbf{V_g}$.

- If a schedule does nothing on day $\mathbf{D} - 1$, but seed g is scheduled to be planted on some other day d: We can move g from day d to day $\mathbf{D} - 1$ of this schedule and the total profit will stay the same.
- If a schedule plant other seed o on day $\mathbf{D} - 1$, and the seed g does not exist in this schedule: We can replace seed o with seed g on day $\mathbf{D} - 1$ to increase the total profit by $\mathbf{V_g} - \mathbf{V_o}$. Since g is the one with maximum value among the non-expired seeds in day $\mathbf{D} - 1$, we have that $\mathbf{V_g} - \mathbf{V_o} \geq 0$.
- If a schedule plant other seed o on day $\mathbf{D} - 1$, but the seed g is scheduled to be planted on some other day d: We can exchange seed o and seed g so that o is planted on day d and g is planted on day $\mathbf{D} - 1$, and the total profit will stay the same.

Therefore, we are sure that planting the type with maximum $\mathbf{V}$ among all non-expired types of day $\mathbf{D} - 1$ is always the best. And for day $\mathbf{D} - 2$, $\mathbf{D} - 3$, $\ldots$, to day 1, we do the same for each of them because they can be seen as a smaller problem of the original one.

To implement this, we can sort the types by their deadline, and maintain a max-heap to store the types with their values. Then we iterate from day $\mathbf{D} - 1$ to day 1, put all the types whose deadline is equal to the current day into the max-heap, and take the one with the maximum value from the heap to assign for the current day.

This strategy can also be used on the case that $\mathbf{X} > 1$ and $\mathbf{Q_i} > 1$. For each day with $\mathbf{X}$ slots, we first insert all types that has deadline equal to the current day, then repeatedly take one type from the heap to assign for the current day until the heap is empty or the $\mathbf{X}$ slots are filled.

In this implementation, we insert each type to the heap on the days equal to their deadline, so we totally insert the heap $O(\mathbf{N})$ times. We pop out each type whenever they are used up, so we totally pop the heap $O(\mathbf{N})$ times. We access the top from the heap on each new day and whenever a previous type is used up, so the number of times we access the top of the heap is $O(\mathbf{D} + \mathbf{N})$. Therefore the total time complexity we spend on operating the heap is $O((\mathbf{D} + \mathbf{N}) \times \log \mathbf{N})$. And we spent $O(\mathbf{N} \times \log \mathbf{N})$ time on sorting the types by their deadline. Therefore the total time complexity of this implementation is $O((\mathbf{D} + \mathbf{N}) \times \log \mathbf{N})$.

Test Set 3

In the implementation of Test Set 2, we iterate through all $\mathbf{D}$ days to fill the slots. In this case with $\mathbf{D}$ up to 10^{12}, we are not able to iterate through all of them. However, only the days that new types are added to the heap need our attention, otherwise every day is just some slots to fill.

Therefore, we can divide these $\mathbf{D}$ days into several segments, where the end of each segment is the deadline of some types. Then the number of slots of each segment is the length of the segment times $\mathbf{X}$. Then we iterate through the segments from the latest one, put all types with associated deadline into the heap, and take the most expensive ones from the heap to fill the slots of the current segment.

In this implementation, the number of segments is $O(\mathbf{N})$. The number of times we insert and pop the heap is the same as the implementation of Test Set 2, which is $O(\mathbf{N})$. And we access the top of the heap when we process each new segment and whenever a previous type is used up, so the number of time we access the top is $O(\mathbf{N})$. Therefore the total time complexity we spend on operating the heap is $O(\mathbf{N} \times \log \mathbf{N})$. With the $O(\mathbf{N} \times \log \mathbf{N})$ time we spent on sorting the deadlines and $O(\mathbf{N})$ time we spent on building the segments, the total time complexity of this solution is $O(\mathbf{N} \times \log \mathbf{N})$.