Task 5: Fruits (`towers`)

Authored by: Benson Lin Zhan Li

Prepared by: Benson Lin Zhan Li and Marc Phua

Terminology

We say that a fruit is fixed if it is already placed, and free otherwise. Similarly, a section is fixed if a fruit is assigned to it, and free otherwise.

Subtask 1

Limits: $N \leq 8$

Since N is very small, we can afford to test every single possible permutation of fruits.

For each of the $N!$ possible permutations, we first check that the fixed fruits are in the correct sections. Then we can compute, for each prefix of sections, what the cost incurred is.

Time Complexity: $O(N \cdot N!)$

Subtask 2

Limits: $A_j = -1$ for all $1 \leq j \leq n$

In this subtask, we are free to place the fruits in any permutation that we want. If Benson only takes fruits from the first k sections, we should ensure that he takes the k most expensive fruits.

This is possible by placing fruit $n - k + 1$ at section 1, fruit $n - k + 2$ at section 2 and so on, with fruit n at section k. This gives us a cost of $C_{n-k+1} + C_{n-k+2} + \cdots + C_n$. This can be computed quickly using a suffix sum on C_i.

Time Complexity: $O(N)$

Subtask 3

Limits: $N \leq 200$

Let's take some optimal solution for a certain prefix k and consider what happens when we place the some fruits. The fruits that we have yet to place can be divided into 2 categories, those worse than the current best and those better than the current best (best meaning highest tastiness).

We realise that the exact fruits that are worse than the current best don't matter anymore, since at this point all of them are simply filler fruits. Thus, we can represent the current state using the number of sections we have filled so far and the value of the best fruit.

Thus, we let $dp[x][v]$ be the maximum cost using the first x sections such that the best fruit used is fruit v. Invalid states are set to $-\infty$ and $dp[0][0] = 0$. There are 2 transitions:

- $A_x \neq -1$:

  In this case, the fruit at this section is fixed. Thus, the best fruit up to this point is no worse than A_x and we should only consider $v \geq A_x$.

  If $v = A_x$, then the previous best fruit must be some fruit $w < v$, so $dp[x][v] = C_v + \max(dp[x-1][w])$ across all $w < v$

  If $v > A_x$, then the previous best fruit must be fruit v, so $dp[x][v] = dp[x-1][v]$

- $A_x = -1$:

  In this case, the fruit at this section is not fixed. If the current fruit is the best fruit, then the total cost is $C_v + dp[x-1][w]$ where w is the previous best fruit. Otherwise the best fruit had already been placed, so the number of fruits taken is $dp[x-1][v]$.

  Thus $dp[x][v]$ is the best of $dp[x-1][v]$ and $C_v + dp[x-1][w]$ across all $w < v$.

There are $O(n^2)$ states and $O(N)$ transition, which gives us an $O(N^3)$ solution.

Time Complexity: $O(N^3)$

Subtask 4

Limits: $N \leq 2000$

We can speedup the transition from $O(N)$ to $O(1)$ by precomputing the prefix maximums $\max(dp[x-1][w])$ for all w in $O(N)$ time, and thus the solution runs in $O(N^2)$.

Time Complexity: $O(N^2)$

Subtask 5

Limits: $C_i = 1$ for all $1 \le i \le n$

Take an example test case of $N = 13, A = \{-1, -1, 5, 6, -1, -1, 7, 11, -1, -1, 10, -1, -1\}$. The dp table is as follows ($dp[x][v]$ is on the xth column from the left and the vth row from the bottom)

1	**2**	2	2	**5**	**6**	6	6	**8**	**9**	9	**9**	**9**
1	**2**	2	2	**5**	**6**	6	6	**8**	**8**	8	**8**	-
-	-	-	-	-	-	-	7	**7**	**7**	7	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-
1	**2**	2	2	**5**	**6**	6	-	-	-	-	-	-
1	**2**	2	2	**5**	**5**	5	-	-	-	-	-	-
-	-	-	-	-	-	5	-	-	-	-	-	-
-	-	-	4	**4**	**4**	-	-	-	-	-	-	-
-	-	3	-	-	-	-	-	-	-	-	-	-
1	**2**	-	-	-	-	-	-	-	-	-	-	-
1	**2**	-	-	-	-	-	-	-	-	-	-	-
1	**2**	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	-	-

If we look at the sections which are free (marked in bold), we see that if $dp[x][v]$ and $dp[x][v-1]$ are valid, then

1. $dp[x][v] - dp[x][v-1] \ge 0$
2. $dp[x][v] - dp[x][v-1] \le 1$

In fact, we can rigorously prove these observations. The non-decreasing condition can be shown with some manipulation of the transition.

$$\begin{aligned} dp[x][v] &= \max(dp[x-1][v], 1 + \max_{w<v}(dp[x-1][w])) \\ &\ge \max(1 + dp[x-1][v-1], 1 + \max_{w<v-1}(dp[x-1][w])) \\ &\ge \max(dp[x-1][v-1], 1 + \max_{w<v-1}(dp[x-1][w])) \\ &= dp[x][v-1] \end{aligned}$$

To prove the second condition, we first extend it to all columns (not just the sections that are free) and use induction. In the 0th column (i.e. $dp[0]$), $dp[0][0] = 0$ and $dp[0][v] = -\infty$ for all $v \geq 1$ so the condition is true.

For a fixed section x, if A_x is useless (i.e. never used even if we place the smallest possible fruits in front), then $dp[x][v] = dp[x-1][v]$ for all v. Otherwise, $dp[x][v] = dp[x-1][v]$ for all $v > A_x$ and $dp[x][v] = -\infty$ for all $v < A_x$. The condition is true for all $v > A_x$ based on the previous section, and is true for $v = A_x$ trivially because $dp[x][A_x - 1]$ is not a valid state.

For a free section x, notice that the transitions for $dp[x][v]$ and $dp[x][v-1]$ differ in only 2 areas. The transition for $dp[x][v]$ includes $1 + dp[x-1][v-1]$ instead of $dp[x-1][v-1]$ and includes $dp[x-1][v]$. Since $dp[x-1][v] - dp[x-1][v-1] \leq 1$, we have $\max(dp[x-1][v], 1 + dp[x-1][v-1]) - dp[x-1][v-1] \leq 1$. All other parameters in the $\max$ are the same, so $dp[x][v] - dp[x][v-1] \leq 1$.

An immediate consequence of this observation is that the dp states look like ranges of identical values that increment by 1 as we move up the column. Thus, instead of tracking the exact value of each state, we can track the ranges of states that all have the same value.

Now we need to solve the problem of accurately modelling the transitions between columns using these ranges. Consider the partial dp table below.

c+1	c+2	c+3
c+1	c+2	c+2
c+1	c+1	c+2
c	c+1	c+1
c	c	c
-	-	-

We can make 2 observations.

- The first is that a range of values will increment together by moving up and right by 1 (e.g the range of c in the first column moves to the range of $c+1$ in the second column).
- The second is that a new range may need to be added at the lowest valid tastiness.

Thus our solution is as follows:

- Maintain a double-ended queue containing tuples of (section,value,lower bound), each representing one of the ranges.
- Bottom or lower tuples refer to those representing dp states with lower tastiness values, top or higher tuples refer to those representing dp states with higher tastiness values.

- The answer we want at each index is at the top of the deque.
- To process a fixed section, we pop ranges from the bottom of the deque and take a range max, then add a new range (which may combine with another existing range)
- To process a free section, we check if any ranges at the top need to be popped out and check if any new ranges need to be added at the bottom.

Since each range is added and removed once and there are at most O(N) ranges, adding and removal of ranges is amortized O(N). Checking the answer is an O(1) computation since we access the top of the deque, so this is also O(N) overall.

Time Complexity: $O(N)$

Subtask 6

To simplify the remainder of the editorial, we will do a number of preprocessing steps.

We can do a linear pass to eliminate all fixed fruits that will never be picked even if we use the smallest possible free fruits at each section.

We can now rephrase the problem. Instead of having fixed fruits in the sections, we have n' sections that are all free, and the n' free fruits have tastiness from 1 to n' based on their original order. C_i is now equal to the cost of the free fruit with tastiness i.

The fixed fruits are offered to Benson after he passes a specific section, possibly multiple times in a row. Each fixed fruit has a new tastiness, equal to the number of free fruits which originally were less tasty than this fixed fruit.

Let n' be the number of free fruits, and let us renumerate the free fruits to take on indices from 1 to n'. Also, he now picks a fixed fruit if its tastiness is **at least** the maximum in the basket (free fruits continue to follow the strict inequality).

We let $dp[x][v]$ be the maximum cost using the first x sections such that the tastiest fruit has tastiness v **without taking the fixed fruits offered at** x. Bear in mind that this tastiest fruit might not be a free fruit. For simplicity, we can take $dp[0][0] = 0$ and $dp[0][v] = -\infty$ for all $1 \leq v \leq n'$.

If $x > v$, then the state $dp[x][v]$ does not make sense, as using any combination of x free fruits will contain one with tastiness $> v$, so such states are invalid and will not be considered.

We can generalise the observation in Subtask 5 that adjacent $dp[x][v]$ values can only differ by at most 1 with the following claim: **For all** x, v **where** $dp[x][v]$ **and** $dp[x][v-1]$ **are valid states, we have** $0 \leq dp[x][v] - dp[x][v-1] \leq C_v$**.** This can be shown by modifying the proof in Subtask 5 with the additional fact that $C_v \geq C_{v-1}$.

Now consider the optimal configuration for $dp[x][v]$. The immediate consequence of this claim is that if this configuration does not use any fixed fruit between section $x-1$ and section x, then the optimal transition is simply $dp[x][v] = C_v + dp[x-1][v-1]$. These transitions can be chained together, similar to how the ranges in Subtask 5 move together.

Suppose the last fixed fruit that the configuration used was just before section x' with tastiness t, and after that last fixed fruit the total cost was c. We can compute $dp[x][v]$ as

$$C_v + C_{v-1} + \cdots + C_{\max(t+1, v+x'-x+1)} + c$$

by chaining the optimal transitions together.

Letting $p_v = C_v + C_{v-1} + \cdots + C_1$, we get

$$dp[x][v] = p_v - p_{\max(t, v+x'-x)} + c$$

which not only allows $O(1)$ computation on the fly, it also means that we no longer need to explcitly store the dp states; we simply need to maintain the tuples of (c, x', t) to compute the cost. Let's call these tuples **bases**.

We maintain a double-ended queue consisting of bases that cover at least one dp state at the current section. Initially, this deque only contains the base $(0, 0, 0)$. Free sections can be handled similarly to that in Subtask 5; we truncate the top base if necessary, and add another base at the bottom if necessary.

There are 4 steps to processing a fixed section x.

1. **Compute new base**:

   This requires us to process each dp state with tastiness $\leq A_x$. This can be done by processing each base in the deque from bottom to top. Note that we only need to check the highest possible tastiness dp state for each base since the dp values are non-decreasing.

2. **Delete unusable / non-optimal bases**:

   Unusable bases are bases that cover a range of dp states that all have tastiness $\leq A_x$. Non-optimal bases are bases whose dp states have lower values than those provided by the new base.

   Since dp states are non-decreasing from bottom to top, these states are at the bottom of the deque, and can be removed one by one.

3. **Truncate bottom base**:

   When we add the new base, there may be a base that it does not **entirely** remove, i.e. only a part of the dp states of that base are less than optimal.

   Here, we can perform a binary search to determine where the cutoff point is. This takes $O(\log N)$ time.

4. **Add new base**:

   We push the new base to the bottom of the deque.

The computation of the new bases can be done concurrently with the removal of unusable bases. Since the dp values within a base is non-decreasing, removing non-optimal bases can be done in O(1) each by testing the highest dp state within that base.

All deque operations are $O(N)$ overall, and we perform at most N binary searches. Hence, the overall solution is $O(N \log N)$.

Time Complexity: $O(N \log N)$ with small constant