1 Catfish Farm

Written by: **Lim Rui Yuan** (Singapore), NUS High School
Prepared by: Mushthofa, Prabowo Djonatan
Solutions, review, and other problem preparations by: Jonathan Irvin Gunawan, Muhammad Ayaz Dzulfikar, Wiwit Rifa'i
Analysis author: Muhammad Ayaz Dzulfikar

Recall that a pier of length k covers cells from row 0 to row $(k-1)$.

1.1 Subtask 1

In this subtask, all catfish are located in even-indexed columns.

We can build piers of length N in all odd-indexed columns to catch all the catfish. Thus, the answer for this subtask is the total weight of all catfish.

Time complexity: $O(M)$

1.2 Subtask 2

In this subtask, all catfish are located at either column 0 or column 1.

When $N = 2$, the optimal solution is to either build a pier of length N in column 0, or a pier of length N in column 1. This way, we either get all the catfish in column 1, or all the catfish in column 0.

When $N > 2$, apart from above solution, another possible optimal solution is by building a pier of length k $(k < N)$ in column 1 and a pier of length N in column 2. Thus, we can catch the catfish of some prefix in column 0 and catfish of some suffix in column 1.

Therefore, the solution for this subtask is the maximum among all those possible optimal solutions.

Time complexity: $O(N + M)$

1.3 Subtask 3

In this subtask, all catfish are located at row 0.

For each column, we either do not build a pier, or build a pier of length 1. Notice that whether a catfish in column i is caught or not only depends on our decision in column $i-1$, i, and $i+1$. Thus, we can formulate a dynamic programming (DP) with the following states: the index of the current column and the decision in the last two columns.

Time complexity: $O(N + M)$

1.4 Subtask 4

In this subtask, $N \leq 300$ and $Y[i] \leq 8$ for all catfish.

The idea is an extension of subtask 3. Denote Y_{max} as the maximum value from array Y. Notice that there exists an optimal solution in which the piers' length is at most $Y_{max} + 1$. Thus, we can change our DP formulation to instead track the index of the current column and the piers' length in the last two columns. To speed up the calculation of the total catfish weight, we can build a prefix sum for them.

Time complexity: $O(N \times {Y_{max}}^3 + M)$.

1.5 Subtask 7

In this subtask, there are at most 2 catfish in each column.

The idea is to extend the solution from subtask 4 using the following observation:

Observation 1.1. *Let $R(i)$ be the set of rows containing catfish in column i. Then, there exists an optimal solution where for each $0 \leq i < N$ we either do not build a pier in column i, or we build a pier in column i of length r, where $(r-1) \in (R(i-1) \cup R(i+1))$.*

The intuition is that it is better to have the end of the pier to be directly connected to the left or right of a catfish. If it is not, then we can just shorten it until it is.

Thus, for each column, there are at most 4 rows that we must consider to be the end of the pier. Then, we can plug in the solution from subtask 4 to solve this subtask.

Time complexity: $O(N + M)$

1.6 Subtask 5

In this subtask, we have $N \leq 300$.

The crux of the previous solutions is that we need to keep track of the piers' length from the previous two columns. To tackle this, first, we start with some definitions.

Definition 1.1. *A valley is an index i, such that:*

- *We build piers in column $(i-1)$, i, and $(i+1)$, and*
- *The length of the pier in column i is shorter than the pier in column $(i-1)$ and column $(i+1)$.*

Definition 1.2. *A bitonic sequence of piers is a continuous segment of piers in which there is no valley in it.*

Finally, we use the following observation:

Observation 1.2. *There exists an optimal solution with no valley in it. In other words, there exists an optimal solution where the constructed piers form several disjoint bitonic sequences.*

Proof. Observe that valleys will not contribute at all to the total weight; Thus, it is better to not build a pier in column i, as there might be catfish that can be caught if we do not build that pier at all. □

Thus, we can reformulate our DP states to be the index of the column, the length of the previous columns' pier, and whether right now the length is increasing or decreasing. As one of the dimensions changes from $O(N)$ to $O(1)$, the time complexity becomes $O(N^3 + M)$.

Time complexity: $O(N^3 + M)$

1.7 Subtask 6

In this subtask, we have $N \leq 3000$.

It is possible to speed up the DP from Subtask 5. Rather than trying all possible lengths for the current column, we can find the optimal length by storing the prefix maximum and suffix maximum of the DP from the next column. This will remove another $O(N)$, hence the time complexity becomes $O(N^2 + M)$.

Time complexity: $O(N^2 + M)$

1.8 Subtask 8

To fully solve this problem, we need to combine Observation 1.1 and Observation 1.2. Observation 1.1 implies there are only $O(N + M)$ possible DP states. Meanwhile, Observation 1.2 (and optimization from Subtask 6) implies all of the states can be computed in $O(1)$. This yields a solution with time complexity $O(N + M \log M)$ (due to sorting), which is sufficient to solve this problem.

Time complexity: $O(N + M \log M)$