ZCO Mock 1 Solutions

windreaper, saarang

1 Error Bottles

1.1 Subtask 1 [3 points] : $K = 1$

Since Paras can make only 1 move, Paras only has 3 choices, which are to either fill bottle A or fill bottle B or leave them both empty.

1.2 Subtask 2 [8 points] : $A = B$

The two bottles have the same capacities. As was the case with subtask 1, there are only 4 reachable configurations : Both bottles empty, Exactly one bottle filled and both bottles filled.

1.3 Subtask 3 [14 points] : $1 \leq K \leq 8$

The small limit on K hints towards a brute force approach to this subtask. At every step, we have 3 allowed moves which can be performed on either bottle, giving us 6 distinct choices at every step. It is thus possible to explicitly calculate all reachable configurations by calculating the end result of every set of commands.

Time Complexity : $O(6^K)$

1.4 Subtask 4 [21 points] : $1 \leq A, B, K \leq 100$

It's hard to answer the question: Can we end up with exactly M units of milk in these two buckets after at most K operations?

It's easier to answer the question: Can we end up with A units of milk in the size X bucket and B units of milk in the size Y bucket after at most K operations?

Imagine that we have A units of milk in the size X bucket and B units of milk in the size Y bucket after at most L operations. With this information, there are several possible states that are attainable after at most L+1 operations. For example, just by emptying or filling buckets, we can get the following six states:

- X units of milk in the size X bucket and B units of milk in the size Y bucket.

- 0 units of milk in the size X bucket and B units of milk in the size Y bucket.
- A units of milk in the size X bucket and Y units of milk in the size Y bucket.
- A units of milk in the size X bucket and 0 units of milk in the size Y bucket.
- Pour $\min(X - A, B)$ units of milk in the size X bucket from the size Y bucket.
- Pour $\min(A, Y - B)$ units of milk from the size X bucket to the size Y bucket.

The above observations point toward a DP solution. Define $dp[i][j][k]$ to be $true$ if it is possible for the first bucket contains i units of milk and the second bucket contains j units of milk after performing k moves and $false$ otherwise.
Time Complexity : $O(ABK)$
Space Complexity : $O(ABK)$ or $O(AB)$ depending on implementation
Official Solution from USACO: http://www.usaco.org/current/data/sol_pails_silver_feb16.html

1.5 Subtask 5[26 points]: $1 \leq A, B, K \leq 1000$

For this subtask, we look at something called implicit graphs. Let us define a graph G with AB nodes where each node represents a possible configuration of water in the bottles. The edges (each of weight 1) are formed by the 6 possible moves. We now have a graph with AB nodes and $6AB$ edges.

Observe that the minimum number of moves to transform any node into any other node is equal to the shortest path between the two nodes.

Our initial state is $(0, 0)$ and we need to find a node (a, b) such that $|L-(a+b)|$ is minimum. An easy way to do this would be to find all nodes which can be attained in at most K moves and take the minimum of this value over them all. To find all such nodes, we need to consider their shortest distances from the initial node $(0, 0)$. In this case, a BFS would suffice since all edge weights are 1. **Time Complexity:** $O(AB)$

1.6 Subtask 6[28 points]: No additional constraints

Observation: At every step, at least one of the bottles is either empty or full. Thus, the total number of reachable nodes is $O(A + B)$.

Proof: We start from $(0, 0)$. Every move stops only when either of the bottles is full or empty. Moves 1 and 2 fill and empty the target bottle until no more can be filled or removed respectively. Move 3 stops when either the bottle being decanted is empty, or the bottle being filled is full.

We now have $O(A + B)$ nodes and edges. The solution which will give you full points on this problem can either use a map and the same solution from subtask 5, or you can make a dp : dp[0/1/2/3][max of (a, b)] and fill this dp table in a BFS.

Time Complexity: $O(A + B)$ or $O((A + B) \log A + B)$

2 Problem Setting Challenge

2.1 Subtask 1[6 points] : All difficulty values are in the range $[1, 10^9]$.

Any problem exchange cycle must end in a 0 for it to be a happy outcome. Since the difficulty ratings are only in the range $[1, 10^9]$, there are no $0s$ in the array implying the answer for all N problems of Saarang are -1. Printing N -1s will AC this subtask

2.2 Subtask 3[8 points] : $K = 0$ and all $4N$ difficulty values are distinct

Since $K = 0$, the problemsetter has to always return a problem of the same difficulty as the one he received. Since all difficulty values are distinct, he will never be able to do that. Hence, the answer will always be -1. However, you have to take care of the edge case when one of Shiven's difficulty values is 0 in which case, the answer for that problem will be 1.

2.3 Subtask 7[8 points] : All of Shiven's difficulties are 0

All cycles end once the setter has received a problem of difficulty 0. Since all of Shiven's ratings are 0, the cycle will end after 1 problem no matter what. Printing N 1s will AC this subtask.

All other subtasks are based on this observation:

Let $dp_{i,0}$ = min length of cycle if Saarang gives problem i first.
Let $dp_{i,1}$ = min length of cycle if Shiven gives problem i first.
$dp_{i,0} = 1$ if Shiven's difficulty for i is 0.
$dp_{i,1} = 1$ if Saarang's difficulty for i is 0.

Now instead of starting with problem i and continuing the cycle till we reach a problem of difficulty 0, we will go backwards. We will start at 0 and find the **minimum "distance"** from 0s to every problem i.

Why backwards?

When you start from a problem i, there will be multiple choices for the returned problem and figuring out which of them is most optimal will be inefficient. However, if you start from a 0 and go backwards, we are processing problems from those of minimum distance to those further away (this is exactly the same as what happens in a Breadth First Search (process using a queue). For a 0, the possible problems it could have "come from", would be in the difficulty range of K (in the other setter's view). Hence, we know the problems which are "adjacent" to the 0 will have answer as the answer for the 0 plus 1.

2.4 Subtasks 2, 4, 5, 6, 8

Subtasks $2, 4, 5, 6, 8$ are based on the above solution. The only difference would be in the efficiency with which we traverse the K adjacent problems. Traversing them in $O(n)$ should allow you to pass all subtasks which allow $O(n^2)$. The better way of traversing these would be using the `multiset` datastructure. We insert all problems into 2 `multisets` - one for Shiven and one for Saarang. As and when we want to find the K adjacent problems, we can binary search for them using `C++`'s in-built `lower_bound` and `upper_bound`. After we find the minimum answer for a particular index, we erase an occurrence of them from the sets. Thus, the number of problems we end up visiting is bounded by $4N$. The only difference from the classic BFS which has exactly 1 source is that here there may be multiple 0s. Therefore, we push all the 0s into the queue at the beginning instead of just a single 1 (a multi-source BFS). Performing a BFS for each 0 will let you AC Subtask 6 where there is exactly one 0.

It is possible that computing the above *dp* recursively without the BFS might pass some subtasks (I've personally not verified this).

Other helpful links:

- Official solution from USACO - http://www.usaco.org/current/data/sol_piepie_gold_dec17.html
- https://codeforces.com/blog/entry/93652 - Another blog on this sort of BFS.
- A similar problem - https://codeforces.com/contest/1272/problem/E

Notes: This problem is not a direct application of BFS rather it is quite a *uncommon* trick. If you are unfamiliar with BFS or find this explanation confusing, I'd suggest practicing various BFS problems before coming back to it.