1. We construct of a directed graph G with N vertices, each of which represents one of the given statements, numbered from 1 to N. A directed edges (x, y) is added to the graph if statement x comments on statement y. If x says y is TRUE, then (x, y) is a "true edge" to which we assign a weight 0; if x says y is FALSE, then (x, y) is "false edge" to which we assign a weight 1.

   Then the group of statements will form a paradox if and only if there exists an odd-weighted cycle in the graph G, i.e., a cycle with an odd number "false edges". The reason is that, starting from any vertex in this cycle, classifying the starting vertex as either TRUE or FALSE, going through this cycle will negate the initial classification of the starting vertex, which is exactly how a contradiction is formed.

   Knowing this properties, we can simply perform DFS on the graph G. During the procedure, whenever we encounter a back edge, i.e., we got a cycle, we backtrack the $\pi[u]$ pointers and compute the total weight of the edges in the cycle (we stop backtracking when we reach the vertex which the back edge is pointing at). If the weight is odd, then we can terminate and output "PARADOX"; if the weight is even, keeping going with DFS. If we finish DFS on G (i.e., all vertices in G are finished) and there is not an odd-weighted cycle, then we can output "NO PARADOX".

   Runtime Analysis: constructing the graph takes $\Theta(N)$, the DFS (including the backtracking) takes $\Theta(|V|+|E|)$ which is also $\Theta(N)$. Therefore the overall worst-case runtime of this algorithm is $\Theta(N)$.

   Another approach for this problem is using vertex colouring, i.e., the group of statements do not form a paradox if and only if you can colour all vertices using two colours (representing the classification of true or false for each statement), where if statement x says y is true then x and y should be of the same colour and if statement x says y is false then x and y should be of different colours. You can then use either BFS or DFS to traverse and colour the graph, and whenever you find that a valid colouring is impossible you return "PARADOX".

2. We construct a directed graph G with $(n+1)(m+1)$ vertices. Each vertex represents a pair (x, y) where x and y are integers that satisfy $0 \leqslant x \leqslant n$ and $0 \leqslant y \leqslant m$. Each vertex has at most 6 outgoing edges that correspond to 6 different possible operations, i.e.,

   - Fill bucket A.
   - Fill bucket B.
   - Empty bucket A.
   - Empty bucket B.
   - Fill A with B until one is full or empty.
   - Fill B with A until one is full or empty.

   We don't add an edge if it is a self-loop, i.e., the amounts of water in the buckets do not change after the operations. Overall, the graph has $(n+1)(m+1)$ vertices and at most $6(n+1)(m+1)$ edges. We use adjacency list to implement the graph.

   Then the problem becomes finding the shortest path from vertex $(0, 0)$ to a vertex (x, y) such that $x = k$ or $y = k$. For this, we do the following steps.

   (a) Perform a BFS on G, starting from vertex $(0, 0)$.

   (b) Go through all the vertices and select a vertex r whose pair contains k and has the smallest distance value $d[v]$ among all vertices whose pairs contain k.

   (c) Tracing back the $\pi[v]$ pointers, the shortest-path found between $(0, 0)$ and r represent the minimum sequence of moves that achieves measuring exactly k litres of water.

(d) If there all vertices whose pairs contain k have distance value ∞, then there is no way to reach the desired measure, and the algorithm returns *NIL*.

Runtime analysis: Graph construction takes $\mathcal{O}(mn)$; BFS takes $\mathcal{O}(|V| + |E|) = \mathcal{O}(mn)$ since $|V|$ and $|E|$ are both in $\mathcal{O}(mn)$; selecting the result takes $\mathcal{O}(|V|) = \mathcal{O}(mn)$. So overall the worst-case runtime of our algorithm is in $\mathcal{O}(mn)$.