1 Main Idea

In this problem, we are given three containers with sizes A, B, and C pints where $A, B, C \in \mathbb{Z}^+$. The A pint container initially has a pints of water, the B pint container initially has b pints of water, and the C pint container initially has c pints of water for $a, b, c \in \mathbb{Z}^+ \cup \{0\}$. We want to determine if it is possible to leave exacty k pints of water in one of the containers after performing a sequence of pouring operations, where we can only stop pouring when the source container is empty or the destination container is full.

We can model this problem as a graph $G = (V, E)$ where V is the set of all possible states of the three containers and E is the set of all possible transitions between states. Each state is a triple (α, β, γ) where α is the number of pints in the A pint container, β is the number of pints in the B pint container, and γ is the number of pints in the C pint container. It should hold that, $\alpha + \beta + \gamma$ remains constant throughout the entire sequence of pouring operations, being equal to the initial sum $a + b + c$. An edge $(u, v) \in E$ if and only if it is possible to pour water from state u to state v, following the rules mentioned before. Then the question becomes whether there is a path from the initial state (a, b, c) to a state (α, β, γ) such that any of α, β, or γ is equal to k.

The upshot of the algorithm that solves this efficiently is that we can use breadth-first search to traverse the graph G. First, we initialize a queue Q and a set S to store visited states. We start by adding the initial state (a, b, c) to Q and S. Then, until Q is empty, we dequeue a state (α, β, γ) from Q and check if any of α, β, or γ is equal to k. If so, we return true. Otherwise, we generate all possible states that can be reached from (α, β, γ). For each of these states, we check if it has been visited before. If not, we add it to Q and S. Finally, if we have exhausted all possible states without finding a solution, we return false.

2 Pseudocode

Algorithm 1: Pouring Water

```
Input: Container sizes $A$, $B$, $C$; initial water levels $a$, $b$, $c$; target water level $k$
Output: True if it is possible to leave exactly $k$ pints of water in one of the containers,
        false otherwise
Initialize an empty queue $Q$ and an empty set $S$;
Enqueue $(a, b, c)$ into $Q$ and add $(a, b, c)$ to $S$;
while $Q$ is not empty do
    Dequeue $(\alpha, \beta, \gamma)$ from $Q$;
    if $\alpha = k$ or $\beta = k$ or $\gamma = k$ then
        return True;
    end
    for $(\alpha', \beta', \gamma')$ in Generate$(\alpha, \beta, \gamma)$ do
        if $(\alpha', \beta', \gamma') \notin S$ then
            Enqueue $(\alpha', \beta', \gamma')$ into $Q$ and add $(\alpha', \beta', \gamma')$ to $S$;
        end
    end
end
return False;
```

Algorithm 2: Generate

```
Input: Current state $(\alpha, \beta, \gamma)$, Container sizes $A$, $B$, $C$
Output: List of all possible next states
Initialize an empty list NextStates;
$x = \min(\alpha, B - \beta)$;
if $x > 0$ then
    Add $(\alpha - x, \beta + x, \gamma)$ to NextStates;
end
$x = \min(\alpha, C - \gamma)$;
if $x > 0$ then
    Add $(\alpha - x, \beta, \gamma + x)$ to NextStates;
end
$x = \min(\beta, A - \alpha)$;
if $x > 0$ then
    Add $(\alpha + x, \beta - x, \gamma)$ to NextStates;
end
$x = \min(\beta, C - \gamma)$;
if $x > 0$ then
    Add $(\alpha, \beta - x, \gamma + x)$ to NextStates;
end
$x = \min(\gamma, A - \alpha)$;
if $x > 0$ then
    Add $(\alpha + x, \beta, \gamma - x)$ to NextStates;
end
$x = \min(\gamma, B - \beta)$;
if $x > 0$ then
    Add $(\alpha, \beta + x, \gamma - x)$ to NextStates;
end
return NextStates;
```

3 Proof of Correctness

To prove the correctness of our algorithm, we will first prove the correctness of the Generate function, and then use this to prove the correctness of the main Pouring Water algorithm.

3.1 Correctness of the Generate Function

Lemma 1. *The Generate function produces all and only the valid next states according to the problem rules.*

Proof. We prove this lemma in two parts. First, we want to show that all states produced are valid. For each possible pouring pairing (e.g., from A to B), the function calculates the theoretical amount of water to pour $x = \min(\alpha, B - \beta)$, where α is the current amount in A and $B - \beta$ is the available space in B. Taking the minimum of α and $B - \beta$ ensures the following:

1. We don't pour more water than is available in the source container. In the case that $\alpha < B - \beta$,

the amount of water available in A is less than the space available in B. Thus, the pour is valid.

2. We don't exceed the capacity of the destination container. In the case that $\alpha > B - \beta$, the amount of water available in A exceeds the available space in B. Thus, we only pour the amount that fits in B, so the pour is valid.

Additionally, since the water poured x is always subtracted from the source container and added to the destination container, the sum of water in all containers remains constant, satisfying the problem rules. Therefore, the states produced by the Generate function are valid according to the problem rules. Next, we want to show that the function is exhaustive in producing all possible valid next states. The function considers all six possible pouring operations: A to B, A to C, B to A, B to C, C to A, and C to B. There are only two possible destination containers for each of the three possible source containers, so there are six possible operations. This exhausts all possible single-step transitions in the problem. Therefore, the Generate function produces all and only the valid next states. □

3.2 Correctness of the Pouring Water Algorithm

We will prove the correctness of the main algorithm using invariant maintenance, termination proof, completeness, and soundness.

Theorem 2. *The Pouring Water algorithm correctly determines whether it is possible to leave exactly k pints of water in one of the containers after a sequence of pouring operations.*

Proof. First, consider the loop invariant Q containing reachable unexplored states. Upon initialization, Q will contain the initial state (a, b, c). For each iteration, explored states are dequeued from Q and their neighbors are enqueued, which maintains the invariant. Then, the loop with only terminate when either Q is empty or a state with k pints of water is found. The algorithm is guaranteed to find a solution if one exists, since BFS explores all possible states reachable from the initial state, and the generate function produces all valid next states by Lemma 1. Therefore, the algorithm correctly solves the problem. □

4 Runtime Analysis

Since the number of pints of water each container can hold must be an integer, the total number of states possible for three containers of sizes A, B, and C is exactly $(A + 1)(B + 1)(C + 1)$ (because contains might be empty). In the worst case, i.e. there is no solution, the algorithm will explore all possible states. Therefore, the runtime complexity of the algorithm is $O(ABC)$ where A, B, and C are the sizes of the three containers.