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 (α, β, γ) from Q;
    if α = k or β = k or γ = k then
        return True;
    end
    for (α', β', γ') in Generate(α, β, γ) do
        if (α', β', γ') ∉ S then
            Enqueue (α', β', γ') into Q and add (α', β', γ') to S;
        end
    end
end
return False;
```

Algorithm 2: Generate

Input: Current state (α, β, γ), 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 source container, 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. We prove this theorem in several steps:

Invariant Maintenance: We define the following invariants: a) The sum of water in all containers remains constant throughout the algorithm. b) The amount of water in each container never exceeds its capacity. c) Every state in the queue Q and set S is reachable from the initial state. Proof: These invariants hold initially. In each iteration, we only consider states generated by the Generate function, which we proved maintains these properties. Therefore, these invariants are maintained throughout the algorithm. Termination: The algorithm always terminates because: a) The number of possible states is finite, bounded by $A \times B \times C$. b) Each state is enqueued at most once due to the visited set S. c) In each iteration, we dequeue one state and potentially enqueue only unvisited states. Therefore, the queue will eventually become empty, and the algorithm will terminate. Completeness: If a solution exists, the algorithm will find it. This is because: a) The algorithm uses breadth-first search, which explores all states reachable from the initial state in order of their distance from the initial state. b) The Generate function produces all valid next states for each explored state. c) The algorithm only terminates when all reachable states have been explored. Therefore, if there exists a sequence of pouring operations leading to a state with k pints in any container, the algorithm will explore this state before terminating. Soundness: If the algorithm returns true, a solution exists. This is because: a) The algorithm only returns true when it finds a state (α, β, γ) where $\alpha = k$ or $\beta = k$ or $\gamma = k$. b) All states explored by the algorithm are reachable from the initial state (from invariant 1c). Therefore, if the algorithm returns true, there exists a sequence of valid pouring operations leading to a state with k pints in one container. Correctness when returning false: If the algorithm returns false, no solution exists. This is because: a) The algorithm has explored all states reachable from the initial state (from completeness). b) None of these states had k pints in any container (otherwise, the algorithm would have returned true). Therefore, if the algorithm returns false, there is no sequence of pouring operations that can result in k pints in any container.

From steps 1-5, we have proved that the 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. □

4 Runtime Analysis