Problem Statement

We have three containers whose sizes are A pints, B pints, and C pints, respectively, where A, B, and C are all positive integers. In the beginning, the A-pint container has a pints of water, the B-pint container has b pints of water, and the C-pint container has c pints of water, where a, b, and c are non-negative integers. (For example, we might have $A = 10$, $B = 7$, $C = 4$, and $a = 0$, $b = 6$, $c = 4$.)

We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that, in the end, leaves exactly k pints of water in any of the three containers. (So the answer we seek is either YES or NO.)

1. Model this problem as a graph problem: give a precise definition of the graph involved, and state the specific question about this graph that needs to be answered.

2. Design an efficient algorithm to solve the above problem.

Main Idea

We can model this problem as a graph, where each node represents a configuration of the containers defined by the tuple (a, b, c), where a, b, and c are the amounts of water in containers A, B, and C, respectively. The edges of the graph represent the valid operations of pouring water between the containers, specifically the following operations: pouring from A to B, pouring from A to C, pouring from B to A, pouring from B to C, pouring from C to A, and pouring from C to B.

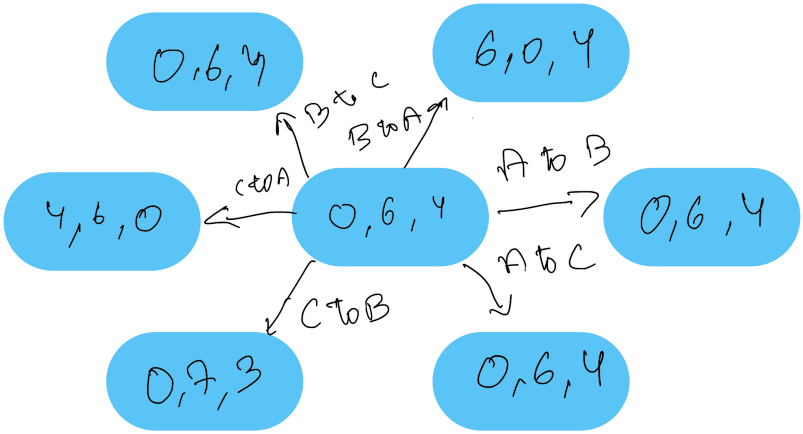

Figure 1: All possible out edges for a Node

The central blue Node in the diagram represents the initial state $(0, 6, 4)$, where container A is empty, container B holds 6 pints, and container C holds 4 pints. From this start state, the algorithm explores all possible pouring operations, which are represented by the arrows leading to other states:

- $B \to A$: Pouring from container B into container A results in the state $(6, 0, 4)$. Here, B is emptied, and A takes in all 6 pints.
- $C \to A$: Pouring from container C into container A results in the state $(4, 6, 0)$. Container A receives 4 pints from C, leaving C empty.
- $C \to B$: Pouring from container C into container B results in the state $(0, 7, 3)$. Container B fills to its maximum capacity, and C is left with 3 pints.

These transitions illustrate the algorithm's process of exploring all possible states from a given configuration by executing valid pouring operations.

Key Considerations

- **Avoiding Duplicates:** The state $(0, 6, 4)$ is shown twice in the diagram. In the actual algorithm, each state would only be explored once to avoid redundancy.
- **Invalid Transitions:** Some arrows in the diagram lead back to the same state without any change. These do not contribute to finding a solution and should be excluded.
- **Completeness:** While this diagram focuses on the immediate transitions from the start state, the algorithm would also consider further transitions from the resulting states.

This diagram serves as a simple example of how the algorithm explores possible states starting from an initial configuration. To accurately represent the algorithm, it's important to ensure that each state is unique, all valid transitions are included, and redundant or invalid transitions are avoided. This approach ensures the algorithm efficiently finds whether the desired state can be reached.

Pseudocode

Algorithm 1 Water Pouring Algorithm

```
1: procedure CanReachK(A, B, C, a, b, c, k)
2:     Create a queue Q and a set visited
3:     Enqueue (a, b, c) into Q
4:     Add (a, b, c) to visited
5:     while Q is not empty do
6:         (x, y, z) ← Dequeue Q
7:         if x = k or y = k or z = k then return YES
8:         end if
9:         for each valid pouring operation do
10:            Generate new states based on pouring operations
11:            (x', y', z') ← new state from pouring
12:            if (x', y', z') not in visited then
13:                Add (x', y', z') to visited
14:                Enqueue (x', y', z') into Q
15:            end if
16:        end for
17:    end while
       return NO
18: end procedure
```

Proof of Correctness

Definitions:

- Let a state be represented as (x, y, z), where x, y, and z are the amounts of water in containers A, B, and C, respectively.
- Valid operations involve pouring water from one container to another until either the source is empty or the destination is full.

Claim: The algorithm correctly determines whether it is possible to reach a state with exactly k pints in any container.

Proof:

1. **Initialization:** The algorithm starts with the state (a, b, c). If $a = k$ or $b = k$ or $c = k$, it returns "YES", covering the case where the desired amount is already present.

2. **BFS Properties:** The algorithm employs BFS, which explores all states at depth d before moving to depth $d+1$. Thus, if there is a valid sequence of pourings to achieve k pints, BFS will reach it.

3. **State Generation:** The algorithm generates new states through valid pouring operations, ensuring that all reachable configurations are explored. Let S be the set of reachable states. If k is reachable, then $k \in S$.

4. **Goal State Check:** For each state dequeued from Q, the algorithm checks if $x = k$, $y = k$, or $z = k$. If true, it returns "YES". If all states are explored without finding k, it returns "NO".

5. **Exploration of States:** Each state is explored only once, as the algorithm maintains a visited set. This prevents revisiting states and ensures that the algorithm does not get stuck in infinite loops.

Since the algorithm exhaustively explores all configurations through valid operations, checks for the goal condition at each state, and avoids revisiting states, it correctly concludes whether achieving k pints is possible. Thus, the algorithm is correct.

Time Complexity

The time complexity of the water pouring algorithm primarily hinges on the number of distinct states that can be generated. Each state represents the amounts of water in the three containers, which we can denote as (x, y, z), where x, y, and z correspond to the amounts in containers A, B, and C, respectively. The maximum number of unique states is approximately $(A+1)(B+1)(C+1)$, since each container can have a value from 0 up to its maximum capacity. A key feature of the algorithm is that it processes each state only once, thanks to the use of a visited set. For every state examined, a fixed number of operations are performed to generate new states through valid pouring actions. As a result, we can conclude that the overall time complexity is $O(A \times B \times C)$.