*** I was granted a 2-day extension from Professor Jiang because my town lost internet service ***

Problem Statement

We have three containers whose sizes are A pints, B pints and C pints, respectively, where A, B, 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.)

Your task:

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

The nodes in this graph are represent different combinations of possible states (the amount of liquid currently in each container) of the 3 containers (A, B, C). There will be a node with the initial volumes (a, b, c) and nodes containing unique combinations of (volume A, volume B, volume C). A node can have maximum volumes of A, B, C. However, the minimum volumes can only get as low as zero for each container. Volume A can range from 0 to A, Volume B can range from 0 to B, and Volume C can range from 0 to C. This means that the node for container A can have a possible Volume A ranging from $0 \leq$ Volume A $\leq A$; similar rules apply for containers B and C. The maximum number of nodes is based on the total number of possible combinations. For container A, it can have a volume of 0, 1, 2, ... A; this means container A has (A + 1) possible volumes. This is similar for the other two containers, meaning that the maximum number of nodes is equal to (A + 1) * (B + 1) * (C + 1). The edges in this graph are representative of pouring between two containers/nodes. Since you can pour from one container to another, and vice versa, this means that the graph is undirected; water can be poured from one container to another, as long as the receiving container is not at its maximum volume or the pouring container isn't empty.

The specific question about this graph that needs to be answered: Is there a path from the initial volume node (a, b, c) to another node where Volume A, Volume B, or Volume C (any available container) that has a volume of exactly k?

To solve this problem, we need an algorithm that searches all possible states while trying to find an ideal path. To do this, use Breadth-First Search. The starting node would be the initial volumes (a, b, c). Then, iteratively look at the surrounding neighbors, and try to find a node with volume k. If it is not found in the initial neighbors, you keep searching the next layer of neighboring nodes and the next layer, until you either find a node with volume k or have searched all nodes.

Pseudocode

Algorithm 1 Find Container with k Volume

1: **function** CONTAINERGRAPH(void)

2: Initialize global variables:

- Initial volumes of each container:
  - initalVolA, initialVolB, initialVolC
- Maximum volumes of each container:
  - maxA, maxB, maxC
- TargetVol = k
- graphQueue = states/nodes to explore
- visitedNodes = keep track of all states/nodes that have already been explored.

3: Iterate using while loop, until the queue is empty:

- currentNode = current state being explored from the queue
- currA, currB, currC = volumes of the current node
- if (currA == k or currB == k or currC == k):
  - return("Yes") - ***search is over because there exists a path to a node containing a volume of exactly k***
- else: - ***generate neighboring nodes to search***
  - Generate all possible configurations of volumeA, volumeB, volumeC without exceeding maximum volumes (maxA, etc.) of each container and keeping minimum volume of 0.
  - If these generated nodes/states are not in visitedNodes (don't want to conduct redundant searches):
    - Add nodes to graphQueue
    - Add to visitedNodes - ***node with these volumes have not already been checked***

4: If you reach outside of while loop without finding k:

5: return("No") - ***if you reach the end of the while loop, this means there is not a path to a node containing a volume of exactly k***

6: **end function**

Proof of Correctness

The base case for this algorithm is when the containers are at their initial volumes (a, b, c). If a = k, b= k, or c = k, then this means that one of the containers starts out with exactly a volume of k; the algorithm can immediately return "yes" without needing to search any of the neighboring nodes. This means the algorithm took (n = 0) pours to be correct.

However, if the base case is not true, then the more intensive Breadth-First Search can begin. Assuming there is a state/node with a volume k after (n) pours, then there should be a path to a node where one of the containers reaches a volume of k. For (n + 1) pours, the algorithm will generate all possible pours from where the containers left off after (n) pours. This additional pour will ensure that all possible states of container volumes have been checked. Thus, by induction, this algorithm should be able to find the path to a node containing at least one volume of exactly k (if it exists) or search all possible iterations before returning "no" if this node does not exist.

Time Complexity

The time complexity of this algorithm is based on the most time-consuming part of this algorithm, which is the while loop when conducting the Breadth-First Search. Since the worst-case scenario would involve needing to check every possible combination of values ranging from (0 - A, 0 - B, 0 - C), the time complexity of this algorithm would be equal to O(A * B* C).