



[Coding Problems](#) · ⌚ 7 minute read

Water Jug Problem

October 25, 2021



Water Jug Problem



Table Of Contents



Problem Statement

Breadth-First Search (BFS) Approach

 C++ Implementation

 Java Implementation

 Python Implementation

Mathematical Approach

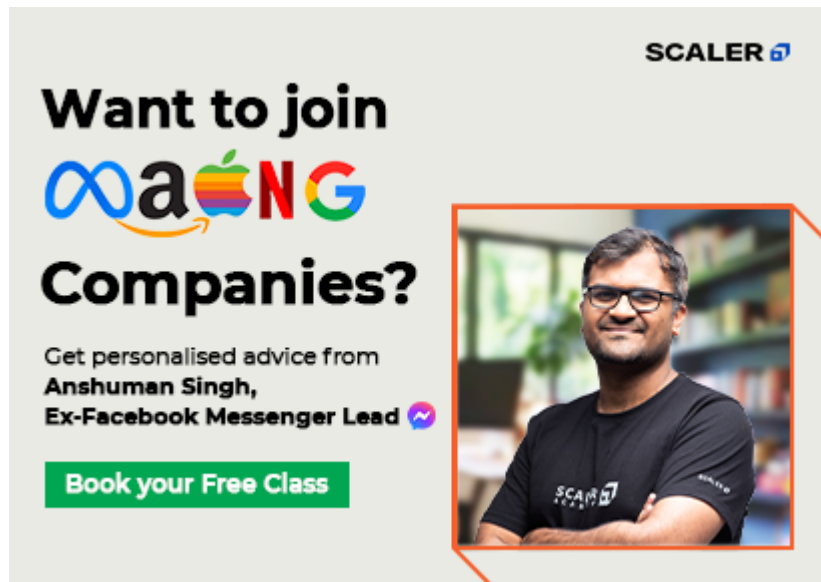
 C++ Code For Mathematical Approach

 Java Code For Mathematical Approach

Problem Statement

Given two water jugs with capacities **X** and **Y** litres. Initially, both the jugs are empty. Also given that there is an infinite amount of water available. The jugs do not have markings to measure smaller quantities.

One can perform the following operations on the jug:



The advertisement features the SCALER logo in the top right corner. The main text reads 'Want to join' followed by a row of logos for Amazon, Apple, and Google, and then 'Companies?'. Below this, it says 'Get personalised advice from Anshuman Singh, Ex-Facebook Messenger Lead' with a Messenger icon. A green button at the bottom left says 'Book your Free Class'. On the right, there is a portrait of Anshuman Singh, a man with glasses wearing a black SCALER t-shirt, with his arms crossed.

- Fill any of the jugs completely with water.
- Pour water from one jug to the other until one of the jugs is either empty or full, $(X, Y) \rightarrow (X - d, Y + d)$
- Empty any of the jugs

The task is to determine whether it is possible to measure **Z** litres of water using both the jugs. And if true, print any of the possible ways.

Examples:

Input: $X = 4, Y = 3, Z = 2$

Output: $\{(0, 0), (0, 3), (3, 0), (3, 3), (4, 2), (0, 2)\}$

Explanation:

- Fill the 4 litre jug completely with water.
- Empty water from 4-litre jug into 3-litre (leaving 1L water in 4L jug and 3L completely full).
- Empty water from 3L.
- Pour water from 4L jug into 3L jug (4L being completely empty and 1L water in 3L litre jug)
- Fill the 4L jug with water completely again.
- Transfer water from 4L jug to 3L jug, resulting in 2L water in 4L jug.

Input: X = 3, Y = 5, Z = 4

Output: 6

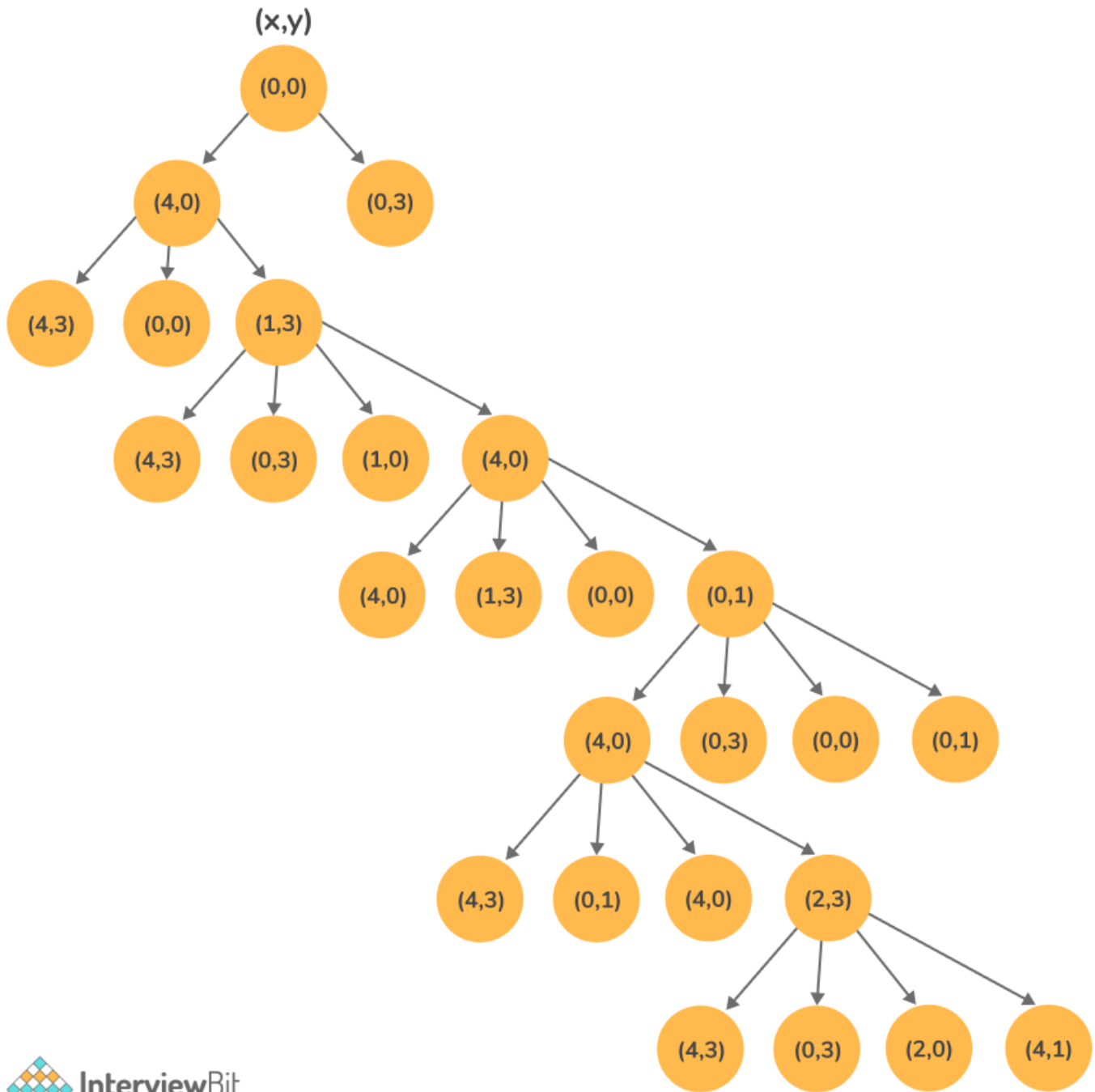
Explanation:

- Fill 5-litres jug to its maximum capacity.
- Transfer 3-litres from 5-litres jug to 3-litres jugs.
- Empty the 3-litres jug.
- Transfer 2-litres from 5-litres jug to 3-litres jug.
- Fill 5-litres jug to its maximum capacity.
- Pour water to 3L jug from 5L jug until it's full.

Breadth-First Search (BFS) Approach

The idea is to run a Breadth-First Search(BFS). The BFS approach keeps a track of the states of the total water in both the jugs at a given time, The key idea is to visit all the possible states and also keep track of the visited states using a visiting array or a hashmap. In case, the total amount of water in any of the jugs or summation of the total amount of water in both the jugs is equal to **Z**, return True and print the resulting states.

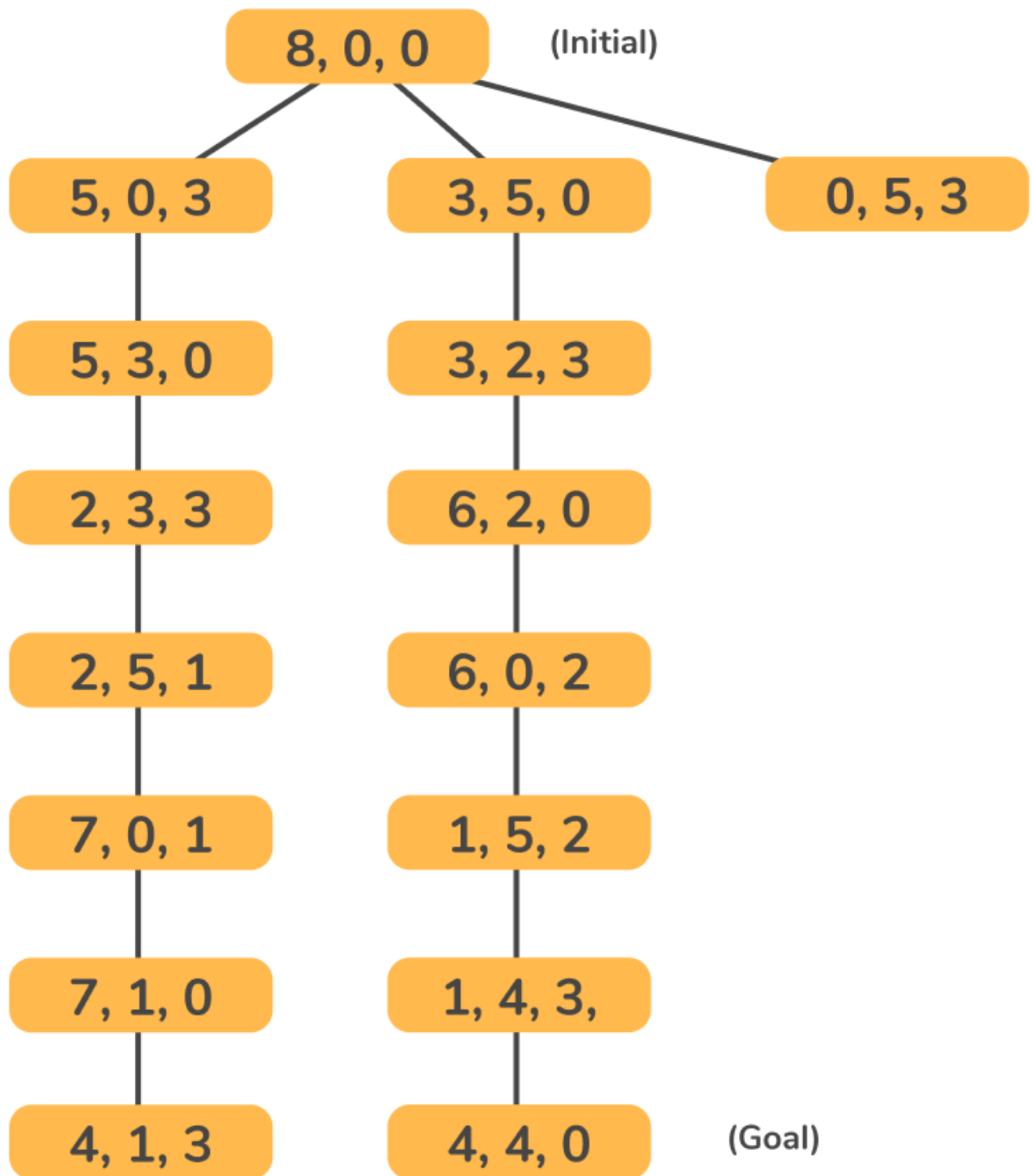
Recursion Tree:



Algorithm

- Initialise a queue to implement **BFS**.
- Since, initially, both the jugs are empty, insert the state $\{0, 0\}$ into the queue.
- Perform the following state, till the queue becomes empty:
 - Pop out the first element of the queue.
 - If the value of popped element is equal to **Z**, return True.
 - Let **X_left** and **Y_left** be the amount of water left in the jugs respectively.
 - Now perform the **fill** operation:
 - If the value of $X_left < X$, insert $\{X_left, Y\}$ into the hashmap, since this state hasn't been visited and some water can still be poured in the jug.
 - If the value of $Y_left < Y$, insert $\{Y_left, X\}$ into the hashmap, since this state hasn't been visited and some water can still be poured in the jug.

- Perform the **empty** operation:
 - If the state $(\{0, Y_left\})$ isn't visited, insert it into the hashmap, since we can empty any of the jugs.
 - Similarly, if the state $(\{X_left, 0\})$ isn't visited, insert it into the hashmap, since we can empty any of the jugs.
- Perform the **transfer of water** operation:
 - $\min(\{X-X_left, Y\})$ can be poured from second jug to first jug. Therefore, in case – $\{X + \min(\{X-X_left, Y\}), Y - \min(\{X-X_left, Y\})\}$ isn't visited, put it into hashmap.
 - $\min(\{X_left, Y-Y_left\})$ can be poured from first jug to second jug. Therefore, in case – $\{X_left - \min(\{X_left, Y - X_left\}), Y + \min(\{X_left, Y - Y_left\})\}$ isn't visited, put it into hashmap.
- Return False, since, it is not possible to measure **Z** litres.



```

typedef pair < int, int > pii;
void printpath(map < pii, pii > mp, pii u) {
    if (u.first == 0 && u.second == 0) {
        cout << 0 << " " << 0 << endl;
        return;
    }
    printpath(mp, mp[u]);
    cout << u.first << " " << u.second << endl;
}
void BFS(int a, int b, int target) {
    map < pii, int > m;
    bool isSolvable = false;
    vector < tuple < int, int, int >> path;
    map < pii, pii > mp;

    queue < pii > q;

    q.push(make_pair(0, 0));
    while (!q.empty()) {

        auto u = q.front();
        q.pop();
        if (m[u] == 1)
            continue;

        if ((u.first > a || u.second > b || u.first < 0 || u.second < 0))
            continue;

        m[{
            u.first,
            u.second
        }] = 1;

        if (u.first == target || u.second == target) {
            isSolvable = true;

            printpath(mp, u);
            if (u.first == target) {
                if (u.second != 0)
                    cout << u.first << " " << 0 << endl;
            } else {
                if (u.first != 0)
                    cout << 0 << " " << u.second << endl;
            }
            return;
        }
        if (m[{
            u.first,
            b
        }] != 1) {
            q.push({
                u.first,

```

```

        b
    });
    mp[{
        u.first,
        b
    }] = u;
}

if (m[{
    a,
    u.second
}] != 1) {
    q.push({
        a,
        u.second
    });
    mp[{
        a,
        u.second
    }] = u;
}

int d = b - u.second;
if (u.first >= d) {
    int c = u.first - d;
    if (m[{
        c,
        b
    }] != 1) {
        q.push({
            c,
            b
        });
        mp[{
            c,
            b
        }] = u;
    }
} else {
    int c = u.first + u.second;
    if (m[{
        0,
        c
    }] != 1) {
        q.push({
            0,
            c
        });
        mp[{
            0,
            c
        }] = u;
    }
}

```



```

}
d = a - u.first;
if (u.second >= d) {
    int c = u.second - d;
    if (m[{
        a,
        c
    }] != 1) {
        q.push({
            a,
            c
        });
        mp[{
            a,
            c
        }] = u;
    }
} else {
    int c = u.first + u.second;
    if (m[{
        c,
        0
    }] != 1) {
        q.push({
            c,
            0
        });
        mp[{
            c,
            0
        }] = u;
    }
}

if (m[{
    u.first,
    0
}] != 1) {
    q.push({
        u.first,
        0
    });
    mp[{
        u.first,
        0
    }] = u;
}

if (m[{
    0,
    u.second
}] != 1) {
    q.push({

```

```

        0,
        u.second
    });
    mp[{
        0,
        u.second
    }] = u;
}

}
if (!isSolvable)
    cout << "No solution";
}

```

Java Implementation

```

class Pair {
    int first, second;

    Pair(int f, int s) {
        first = f;
        second = s;
    }
}

void BFS(int a, int b, int target) {
    int m[][] = new int[1000][1000];
    for (int[] i: m) {
        Arrays.fill(i, -1);
    }

    boolean isSolvable = false;
    Vector < Pair > path = new Vector < Pair > ();

    Queue < Pair > q = new LinkedList < Pair > ();
    q.add(new Pair(0, 0));

    while (!q.isEmpty()) {
        Pair u = q.peek();

        q.poll();

        if ((u.first > a || u.second > b ||
            u.first < 0 || u.second < 0))
            continue;
    }
}

```

```

    if (m[u.first][u.second] > -1)
        continue;

    path.add(new Pair(u.first, u.second));

    m[u.first][u.second] = 1;

    if (u.first == target || u.second == target) {
        isSolvable = true;
        if (u.first == target) {
            if (u.second != 0)

                path.add(new Pair(u.first, 0));
        } else {
            if (u.first != 0)

                path.add(new Pair(0, u.second));
        }

        int sz = path.size();
        for (int i = 0; i < sz; i++)
            System.out.println("(" + path.get(i).first +
                ", " + path.get(i).second + ")");
        break;
    }

    q.add(new Pair(u.first, b));
    q.add(new Pair(a, u.second));

    for (int ap = 0; ap <= Math.max(a, b); ap++) {

        int c = u.first + ap;
        int d = u.second - ap;

        if (c == a || (d == 0 && d >= 0))
            q.add(new Pair(c, d));

        c = u.first - ap;
        d = u.second + ap;

        if ((c == 0 && c >= 0) || d == b)
            q.add(new Pair(c, d));
    }

    q.add(new Pair(a, 0));
    q.add(new Pair(0, b));
}

if (!isSolvable)
    System.out.print("No solution");
}

```

Python Implementation

```
def BFS(a, b, target):  
    m = {}  
    isSolvable = False  
    path = []  
  
    q = deque()  
    q.append((0, 0))  
  
    while len(q) > 0:  
        u = q.popleft()  
  
        if (u[0], u[1]) in m:  
            continue  
  
        if u[0] > a or u[1] > b or u[0] < 0 or u[1] < 0:  
            continue  
  
        path.append([u[0], u[1]])  
  
        m[(u[0], u[1])] = 1  
  
        if u[0] == target or u[1] == target:  
            isSolvable = True  
  
            if u[0] == target:  
                if u[1] != 0:  
                    path.append([u[0], 0])  
            else:  
                if u[0] != 0:  
                    path.append([0, u[1]])  
  
            sz = len(path)  
            for i in range(sz):  
                print("(", path[i][0], ",", path[i][1], ")")  
            break  
  
        q.append([u[0], b]) # Fill Jug2  
        q.append([a, u[1]]) # Fill Jug1  
  
        for ap in range(max(a, b) + 1):  
            c = u[0] + ap  
            d = u[1] - ap
```

```

        if c == a or (d == 0 and d >= 0):
            q.append([c, d])

        c = u[0] - ap
        d = u[1] + ap

        if (c == 0 and c >= 0) or d == b:
            q.append([c, d])

    q.append([a, 0])

    q.append([0, b])

if not isSolvable:
    print("No solution")

```

Mathematical Approach

Since we need to find if **Z** can be measured from the given jugs of **X** and **Y** litres. This can be written in a single equation as follows:

$$A * X + B * Y = Z$$

where A and B are integers. Now, this is a linear diophantine equation and is easily solvable iff. **GCD(X, Y)** divides **Z**. So, the conditions to solve this problem is:

- $Z \% \text{GCD}(X, Y) = 0$
- $X + Y > Z$

C++ Code For Mathematical Approach

```

int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }

    return gcd(b, a % b);
}

```

```

bool waterJug(int x, int y, int z) {
    if (x + y < z) {
        return false;
    }

    if (x == 0 and y == 0) {
        if (z == 0) {
            return true;
        } else {
            return false;
        }
    }

    if (z % gcd(x, y) == 0) {
        return true;
    } else {
        return false;
    }
}

```

Java Code For Mathematical Approach

```

public class Solution {
    public static int gcd(int a, int b) {
        if (b == 0) {
            return a;
        }

        return gcd(b, a % b);
    }

    public static boolean waterJug(int x, int y, int z) {
        if (x + y < z) {
            return false;
        }

        if (x == 0 && y == 0) {
            if (z == 0) {
                return true;
            } else {
                return false;
            }
        }

        if (z % gcd(x, y) == 0) {
            return true;
        }
    }
}

```

```
    } else {  
        return false;  
    }  
}  
}
```

Python Code For Mathematical Approach

```
def gcd(a, b):  
    if(b == 0):  
        return a  
  
    return gcd(b, a % b)  
  
def waterJug(x, y, z):  
    if(x + y < z):  
        return False  
  
    if(x == 0 and y == 0):  
        if(z == 0):  
            return True  
  
        else:  
            return False  
  
    if(z % gcd(x, y) == 0):  
        return True  
  
    else:  
        return False
```

Practice Question

[N Queens Problem](#)

FAQs

What other problems can be solved using the same approach?

The N queen problem, 8 queen problem uses the same approach.

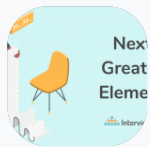
What is the worst case and best case space complexity in the queue approach?

In the worst case, the queue would contain $X * Y$ distinct states, hence, the space complexity would be $O(X * Y)$. The best space complexity is $O(\min(X, Y))$.

Water Jug Problem



Previous Post



[Coding Problems](#)

Next Greater Element

October 25, 2021

Next Post

[Coding Problems](#)

8 Queens Problem

October 25, 2021

