Problem 365: Water and Jug Problem

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given two jugs with capacities

x

liters and

y

liters. You have an infinite water supply. Return whether the total amount of water in both jugs may reach

target

using the following operations:

Fill either jug completely with water.

Completely empty either jug.

Pour water from one jug into another until the receiving jug is full, or the transferring jug is empty.

Example 1:

Input:

x = 3, y = 5, target = 4

Output:

true

Explanation:

Follow these steps to reach a total of 4 liters:

Fill the 5-liter jug (0, 5).

Pour from the 5-liter jug into the 3-liter jug, leaving 2 liters (3, 2).

Empty the 3-liter jug (0, 2).

Transfer the 2 liters from the 5-liter jug to the 3-liter jug (2, 0).

Fill the 5-liter jug again (2, 5).

Pour from the 5-liter jug into the 3-liter jug until the 3-liter jug is full. This leaves 4 liters in the 5-liter jug (3, 4).

Empty the 3-liter jug. Now, you have exactly 4 liters in the 5-liter jug (0, 4).

Reference: The

Die Hard

example.

Example 2:

Input:

x = 2, y = 6, target = 5

Output:

false

Example 3:

Input:

x = 1, y = 2, target = 3

Output:

true

Explanation:

Fill both jugs. The total amount of water in both jugs is equal to 3 now.

Constraints:

1 <= x, y, target <= 10

3

Code Snippets

C++:

```
class Solution {
public:
bool canMeasureWater(int x, int y, int target) {

}
};
```

Java:

```
class Solution {
public boolean canMeasureWater(int x, int y, int target) {

}
}
```

Python3:

```
class Solution:
def canMeasureWater(self, x: int, y: int, target: int) -> bool:
```

Python:

```
class Solution(object):
def canMeasureWater(self, x, y, target):
"""
:type x: int
:type y: int
:type target: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} y
* @param {number} target
* @return {boolean}
*/
var canMeasureWater = function(x, y, target) {

};
```

TypeScript:

```
function canMeasureWater(x: number, y: number, target: number): boolean {

};
```

C#:

```
public class Solution {
public bool CanMeasureWater(int x, int y, int target) {

}

}
```

C:

```
bool canMeasureWater(int x, int y, int target) {

}
```

Go:

```
func canMeasureWater(x int, y int, target int) bool {

}
```

Kotlin:

```
class Solution {
fun canMeasureWater(x: Int, y: Int, target: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func canMeasureWater(_ x: Int, _ y: Int, _ target: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_measure_water(x: i32, y: i32, target: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer} target
# @return {Boolean}
def can_measure_water(x, y, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @param Integer $target
* @return Boolean
*/
function canMeasureWater($x, $y, $target) {

}
}
```

Dart:

```
class Solution {
bool canMeasureWater(int x, int y, int target) {

}
}
```

Scala:

```
object Solution {
def canMeasureWater(x: Int, y: Int, target: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_measure_water(x :: integer, y :: integer, target :: integer) ::
boolean
```

```
def can_measure_water(x, y, target) do

end
end
```

Erlang:

```
-spec can_measure_water(X :: integer(), Y :: integer(), Target :: integer())
-> boolean().
can_measure_water(X, Y, Target) ->
.
```

Racket:

```
(define/contract (can-measure-water x y target)
(-> exact-integer? exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Water and Jug Problem
* Difficulty: Medium
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool canMeasureWater(int x, int y, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Water and Jug Problem
* Difficulty: Medium
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean canMeasureWater(int x, int y, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Water and Jug Problem
Difficulty: Medium
Tags: math, search

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def canMeasureWater(self, x: int, y: int, target: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canMeasureWater(self, x, y, target):
"""
:type x: int
:type y: int
:type target: int
:rtype: bool
```

```

"""
```

JavaScript Solution:

```
/**
* Problem: Water and Jug Problem
* Difficulty: Medium
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} x
* @param {number} y
* @param {number} target
* @return {boolean}
*/
var canMeasureWater = function(x, y, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Water and Jug Problem
* Difficulty: Medium
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function canMeasureWater(x: number, y: number, target: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Water and Jug Problem
* Difficulty: Medium
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CanMeasureWater(int x, int y, int target) {

}
}
```

C Solution:

```
/*
* Problem: Water and Jug Problem
* Difficulty: Medium
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

bool canMeasureWater(int x, int y, int target) {

}
```

Go Solution:

```
// Problem: Water and Jug Problem
// Difficulty: Medium
// Tags: math, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach
```

```
func canMeasureWater(x int, y int, target int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canMeasureWater(x: Int, y: Int, target: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canMeasureWater(_ x: Int, _ y: Int, _ target: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Water and Jug Problem
// Difficulty: Medium
// Tags: math, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn can_measure_water(x: i32, y: i32, target: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer} target
# @return {Boolean}
def can_measure_water(x, y, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @param Integer $target
* @return Boolean
*/
function canMeasureWater($x, $y, $target) {

}
}
```

Dart Solution:

```
class Solution {
bool canMeasureWater(int x, int y, int target) {

}
}
```

Scala Solution:

```
object Solution {
def canMeasureWater(x: Int, y: Int, target: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_measure_water(x :: integer, y :: integer, target :: integer) ::
boolean
```

```
def can_measure_water(x, y, target) do

end
end
```

Erlang Solution:

```
-spec can_measure_water(X :: integer(), Y :: integer(), Target :: integer())
-> boolean().
can_measure_water(X, Y, Target) ->
.
```

Racket Solution:

```
(define/contract (can-measure-water x y target)
(-> exact-integer? exact-integer? exact-integer? boolean?)
)
```