2543. Check if Point Is Reachable

Hard Math Number Theory

Leetcode Link

Problem Description

The problem presents an infinite grid where you start at the coordinates `(1, 1)` and your goal is to reach a target point `(targetX, targetY)` through a series of steps. At each step, you are allowed to move to a new point by either doubling the x or y coordinate, or subtracting the value of y from x, or the value of x from y. The task is to determine if it is possible to reach the target using a finite number of these operations. You need to return `true` if the target can be reached, or `false` if it is impossible.

Intuition

The key to solving this problem lies in recognizing that the operations can be performed in a reversible manner, implying that starting from the target point `(targetX, targetY)` and working our way back to `(1, 1)` is equivalent to moving from `(1, 1)` to `(targetX, targetY)`. In particular, because the subtract operations could potentially make coordinates negative (which is not allowed in the problem), we realize that the only way to reach `(targetX, targetY)` without ever being in the negatives is if both targetX and targetY have a common factor that is a power of 2.

The solution, therefore, is to calculate the greatest common divisor (GCD) of `targetX` and `targetY`. If their GCD is a power of 2, it would mean that we can scale down both coordinates by dividing by the GCD and eventually reach `(1, 1)` without encountering any issues with negative numbers.

To check if the GCD is indeed a power of 2, we use the bit manipulation approach: a number `n` is a power of 2 if and only if `n & (n - 1) == 0` where `&` is the bitwise AND operator. This condition checks whether `n` has exactly one non-zero bit, which is the characteristic property of powers of 2.

The insight that the path can be traced backwards efficiently is crucial, as trying to simulate all potential forward paths would be impractical given the size of the grid and the number of steps potentially involved.

Solution Approach

The solution leverages a mathematical approach rather than a step-by-step simulation, which would be computationally intensive. This approach uses both number theory and bit manipulation to solve the problem. Here's a step-by-step breakdown of the algorithm implemented in the provided solution:

1. The solution begins by invoking the `gcd` function from Python's `math` module to calculate the Greatest Common Divisor (GCD) of `targetX` and `targetY`. The function `gcd` takes two integers and returns their largest common factor. In mathematical terms, `x = gcd(targetX, targetY)`.
2. After calculating the GCD, the solution checks if `x` is a power of two. This is accomplished using a bit manipulation trick: `x & (x - 1) == 0`. In this expression, the bitwise AND operator `&` is used between the number `x` and the number `x - 1`.
   - If `x` is a power of two, it means it has a binary representation with only a single `1` bit set, and all other bits set to `0`.
   - The expression `x - 1` converts the rightmost `1` bit of `x` to `0` and flips all the bits to the right of it. For example, if `x` is `8` (binary `1000`), then `x - 1` is `7` (binary `0111`).
   - When you perform the bitwise AND of `x` with `x - 1`, if `x` is a power of two, the result will be `0` because there are no overlapping `1` bits in the two numbers.
3. The final step is to return whether the GCD is a power of two using the result of the bit manipulation check. If the check passes, the function returns `True`, indicating the target point `(targetX, targetY)` is reachable. Otherwise, it returns `False`.

By explicating the approach in terms of number theory (GCD) and optimizing with bit manipulation, the solution circumvents the need for complex data structures or algorithms. It elegantly exploits the properties of numbers and operations allowed to deduce the reachability of the target coordinates.

Example Walkthrough

Let's illustrate the solution approach using a small example where the target coordinate is `(12, 8)`. Our task is to determine if it is possible to reach this target coordinate `(12, 8)` from the starting coordinate `(1, 1)` by either doubling the x or y coordinate or by subtracting the value of y from x or the value of x from y.

1. **Calculating the GCD**: First, we find the GCD of the target coordinates (targetX = 12) and (targetY = 8). Using Python's `math.gcd()` function, we find that the GCD of 12 and 8 is 4.
2. **Checking if the GCD is a power of 2**: Our next step is to check if the GCD (which is 4 in this case) is a power of 2. To perform this check, we use the expression `gcd & (gcd - 1) == 0`.
   - For our GCD, 4, we calculate 4 & (4 - 1) which is 4 & 3.
   - In binary form, 4 is `100` and 3 is `011`. The bitwise AND of `100` & `011` results in `000`, which is 0.
3. **Returning the result**: Since the calculation from the previous step gives us 0, it confirms that our GCD, 4, is indeed a power of 2. Thus, according to our algorithm, this implies that the target coordinate `(12, 8)` can be reached from `(1, 1)`.

Following these steps, the final output for our example is `True`, which means the movement from `(1, 1)` to `(12, 8)` is possible with a finite number of the described operations.

Python Solution

```python
from math import gcd  # Import the gcd function from the math module

class Solution:
    def isReachable(self, target_x: int, target_y: int) -> bool:
        # Calculate the greatest common divisor (GCD) of target_x and target_y
        greatest_common_divisor = gcd(target_x, target_y)

        # Check if the GCD is a power of two
        # A number is a power of two if it's bitwise 'and' with itself minus one is zero
        return greatest_common_divisor & (greatest_common_divisor - 1) == 0

```

Java Solution

```java
class Solution {
    // Method to check if the point (targetX, targetY) is reachable
    public boolean isReachable(int targetX, int targetY) {
        // Calculate the greatest common divisor (GCD) of targetX and targetY
        int gcdValue = gcd(targetX, targetY);

        // Check if gcdValue is a power of 2 by using bitwise AND with gcdValue - 1
        // If the result is 0, then gcdValue is indeed a power of 2
        return (gcdValue & (gcdValue - 1)) == 0;
    }

    // Helper method to calculate the greatest common divisor (GCD) of two numbers using recursion
    private int gcd(int number1, int number2) {
        // If number2 is 0, return number1 as the GCD
        if (number2 == 0) {
            return number1;
        }
        // Recur with number2 and the remainder of number1 divided by number2
        return gcd(number2, number1 % number2);
    }
}

```

C++ Solution

```cpp
#include <algorithm> // Include the algorithm header for the 'std::gcd' function

class Solution {
public:
    // Function to check if a point (targetX, targetY) is reachable
    // The point is reachable if the greatest common divisor of X and Y is a power of 2
    bool isReachable(int targetX, int targetY) {
        // Finding the greatest common divisor (GCD) of targetX and targetY
        int gcdValue = std::gcd(targetX, targetY);

        // Check if gcdValue is a power of 2 by using the property:
        // A number n is a power of 2 if and only if n & (n - 1) is 0,
        // where & is the bitwise AND operator.
        // This works because a power of 2 has only one bit set in its binary representation,
        // and (n - 1) would have all the bits set before that particular bit.
        return (gcdValue & (gcdValue - 1)) == 0;
    }
};

```

Typescript Solution

```typescript
// Calculate the greatest common divisor (GCD) of two numbers
function gcd(a: number, b: number): number {
    // Base case: if b is 0, a is the gcd
    if (b === 0) {
        return a;
    }
    // Recursive case: call gcd with b and the remainder of a divided by b
    return gcd(b, a % b);
}

// Determine if the location (targetX, targetY) is reachable
function isReachable(targetX: number, targetY: number): boolean {
    // Find the gcd of the target coordinates
    const greatestCommonDivisor = gcd(targetX, targetY);

    // Check if the gcd is a power of two (i.e., has only one '1' bit in binary)
    // This is done by performing the binary AND operation between x and (x - 1)
    // If the result is zero, then x is a power of two
    // This step is crucial because being able to reach a position (x, y) on a
    // number line using moves that double the number at each step is only possible
    // when the gcd of (x, y) is a power of two
    return (greatestCommonDivisor & (greatestCommonDivisor - 1)) === 0;
}

```

Time and Space Complexity

The time complexity of the code is `O(log(min(targetX, targetY)))`, driven by the `gcd` function which performs a series of modulus operations until it finds the greatest common divisor of `targetX` and `targetY`. This process takes a time proportional to the logarithm of the smaller of the two inputs.

The space complexity of the code is `O(1)` since it uses only a constant amount of additional memory. The gcd function stores intermediate values, but these do not scale with the input size and are instead limited to a fixed number of simple variables necessary for the computation.