

1342. Number of Steps to Reduce a Number to Zero

Easy

Bit Manipulation

Math

[Leetcode Link](#)

Problem Description

Given an integer `num`, the task is to find out how many steps it would take to reduce this number to zero. The rule for each step of the reduction process is simple:

- If `num` is even, divide it by 2.
- If `num` is odd, subtract 1 from it.

The process is repeated until `num` becomes zero and we need to count and return the total number of steps taken.

Intuition

The intuition behind the solution is to use a loop that keeps running until the number becomes zero. Inside the loop, we check if the current number is odd or even. The bitwise AND operation `num & 1` is a quick way to check if a number is odd (1) or even (0). If the number is odd (last bit is 1), we subtract 1 from it. If it is even (last bit is 0), we shift the bits to the right by one position using `num >>= 1`, which effectively divides the number by 2. After each operation, we increment a counter `ans` by 1 to keep track of the number of steps taken. Once the loop ends (when `num` is zero), we return the counter as our result.

So, the intuition leverages simple bitwise operations for efficiency, and a while loop to iteratively reduce the number to zero, counting the steps along the way.

Solution Approach

The solution is implemented using a straightforward approach that doesn't rely on any complex algorithms or advanced data structures. The primary tools used here are bitwise operations and a loop. Here's how the implementation works:

1. We define a function `numberOfSteps` that takes an integer `num` as an input and initializes a counter `ans` to 0. This counter will hold the total number of steps required to reduce `num` to zero.
2. The solution employs a `while` loop that runs as long as `num` is not zero. Within the loop, we perform one of the two operations in each iteration:
 - If `num` is odd (which we check using the bitwise AND operation `num & 1`), we subtract 1 from it. This step ensures that an odd number becomes even, and thus can be halved in the next step.
 - If `num` is even, we use the right shift bitwise operation `num >>= 1` to divide `num` by 2. Bit shifting to the right by one position is synonymous with integer division by two for non-negative integers.
3. After each operation inside the loop, we increment the counter `ans` by 1. This accounts for the step we've just performed.
4. Once `num` is zero, the loop exits, and the function returns the `ans` counter, which by then has accumulated the total number of steps taken to reduce `num` to zero.

The solution uses no auxiliary data structures; it only manipulates the given `num` and maintains a simple integer counter. The approach is efficient both in terms of time and space complexity, as it operates in linear time with respect to the number of bits in `num` and only uses a constant amount of additional space for the counter.

Example Walkthrough

Let's walk through a small example to clearly illustrate the solution approach using the integer 14 as our `num`.

1. We call the function `numberOfSteps` with `num = 14`.
2. Since the while loop condition (`num != 0`) holds true, we enter the while loop.
3. We check if 14 is odd or even using `num & 1`. Since `14 & 1` equals 0, 14 is even.
4. We divide 14 by 2 using `num >>= 1`, which results in 7. Now, `num = 7`. We increment `ans` to 1.
5. Since 7 is odd (`7 & 1` equals 1), we subtract 1 from it resulting in 6. Now, `num = 6`. We increment `ans` to 2.
6. 6 is even (`6 & 1` equals 0), so we divide it by 2 using `num >>= 1`, resulting in 3. Now, `num = 3`. We increment `ans` to 3.
7. 3 is odd (`3 & 1` equals 1), so we subtract 1 from it, resulting in 2. Now, `num = 2`. We increment `ans` to 4.
8. 2 is even (`2 & 1` equals 0), so we divide it by 2 using `num >>= 1`, yielding 1. Now, `num = 1`. We increment `ans` to 5.
9. 1 is odd (`1 & 1` equals 1), so we subtract 1 from it, resulting in 0. Now, `num = 0`. We increment `ans` to 6.
10. The while loop condition `num != 0` no longer holds true, so we exit the loop.
11. Finally, `numberOfSteps` returns `ans` which is 6. Therefore, it takes 6 steps to reduce the number 14 to zero.

Here we took six steps to reduce the number from 14 to 0. The approach involved alternating bitwise operations and arithmetic subtractions until reaching zero, tracking each operation as a step.

Python Solution

```
1 class Solution:
2     def numberOfSteps(self, number: int) -> int:
3         # Initialize the step counter
4         step_count = 0
5
6         # Keep iterating until the number is reduced to zero
7         while number:
8             # Check if the current number is odd
9             if number & 1:
10                # If odd, subtract 1 from it
11                number -= 1
12            else:
13                # If even, right-shift the number (equivalent to dividing by 2)
14                number >>= 1
15            # Increment the step counter for each operation (subtract or shift)
16            step_count += 1
17
18        # Return the total number of steps taken to reduce the number to zero
19        return step_count
20
```

Java Solution

```
1 class Solution {
2     // Method to calculate the number of steps to reduce a number to zero
3     public int numberOfSteps(int num) {
4         int steps = 0; // Counter for the number of steps taken
5
6         // Loop until the number is reduced to zero
7         while (num != 0) {
8             // If the number is odd, subtract 1, else right shift (divide by 2)
9             num = (num & 1) == 1 ? num - 1 : num >> 1;
10            // Increment the step counter
11            ++steps;
12        }
13
14        // Return the total number of steps taken
15        return steps;
16    }
17 }
18
```

C++ Solution

```
1 class Solution {
2 public:
3     // Function to compute the number of steps to reduce the number 'num' to zero.
4     int numberOfSteps(int num) {
5         int steps = 0; // Variable to count the number of steps taken.
6         while (num) { // Continue the process until 'num' becomes zero.
7             // If 'num' is odd, subtract 1 from it; otherwise, right shift by 1 (divide by 2).
8             num = (num & 1) ? num - 1 : num >> 1;
9             steps++; // Increment the steps counter after each operation.
10        }
11        // Return the total number of steps taken.
12        return steps;
13    }
14 };
15
```

Typescript Solution

```
1 /**
2  * This function returns the number of steps to reduce the number to zero.
3  * In each step, if the current number is even, you have to halve it,
4  * otherwise, you have to subtract 1 from it.
5  * @param num The number to be reduced to zero.
6  * @returns The number of steps to reduce the number to zero.
7  */
8 function numberOfSteps(num: number): number {
9     let steps = 0; // Initialize the step counter to zero.
10
11     // Loop until the number is reduced to zero.
12     while (num) {
13         // If the number is odd, subtract 1 from it; if it's even, divide it by 2.
14         // This is done using bitwise operations: '&' to check if odd and '>>= 1' to right shift (divide by 2).
15         num = num & 1 ? num - 1 : num >>= 1;
16         steps++; // Increment the step counter.
17     }
18
19     return steps; // Return the total number of steps.
20 }
21
```

Time and Space Complexity

The time complexity of the given code is $O(\log n)$. This is because each operation either decreases the number by one if it is odd or divides the number by two if it is even. In the worst-case scenario, for a number with all bits set to 1, it will take at most $2 * \log_2(n)$ steps to reduce the number to 0, since each bit position would need a subtraction and a right shift to clear it.

The space complexity of the code is $O(1)$ since the algorithm uses a fixed amount of space. The extra space used by the algorithm does not grow with the input size `n`. Only a constant number of variables (`num` and `ans`) are used regardless of the size of the input.