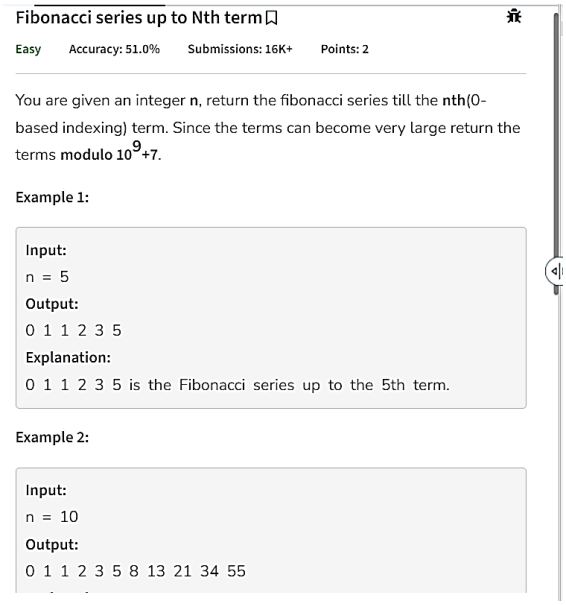
PROBLEM



Fibonacci series up to Nth term

Easy Accuracy: 51.0% Submissions: 16K+ Points: 2

You are given an integer **n**, return the fibonacci series till the **nth**(0-based indexing) term. Since the terms can become very large return the terms **modulo** 10^9+7.

Example 1:

Input:
n = 5
Output:
0 1 1 2 3 5
Explanation:
0 1 1 2 3 5 is the Fibonacci series up to the 5th term.

Example 2:

Input:
n = 10
Output:
0 1 1 2 3 5 8 13 21 34 55

Explanation:
0 1 1 2 3 5 8 13 21 34 55 is the Fibonacci series up to the 10th term.

Your Task:
You don't need to read input or print anything. Your task is to complete the function **Series()** which takes an Integer **n** as input and returns a Fibonacci series up to the **nth** term.

Expected Time Complexity: O(n)
Expected Space Complexity: O(n)

Constraint:
$1 <= n <= 10^5$

EXPLANATION

```
class Solution:
    def series(self, n):
        # Base cases
        if n == 0:
            return [0]
        elif n == 1:
            return [0, 1]

        # Initialize the Fibonacci series list with the first two terms
        fib = [0, 1]

        # Compute Fibonacci series modulo 10^9+7 up to the nth term
        for i in range(2, n + 1):
            fib.append((fib[i - 1] + fib[i - 2]) % (10**9 + 7))

        return fib
```

We want to generate a series of Fibonacci numbers, but we're not just stopping at the nth term; we also need to make sure that each number in the series doesn't get too large. So, we're going to compute each Fibonacci number modulo 10^9+7.

Here's how the code works:

Base Cases: First, we handle the base cases. If the student asks for the Fibonacci series up to 0th term, we return just [0]. If they ask for up to the 1st term, we return [0, 1]. Why? Because those are the first two Fibonacci numbers.

Initial Setup: We start building our Fibonacci series list with [0, 1]. This is because we know the first two Fibonacci numbers already.

Looping for More Terms: Now, we need to fill in the rest of the series. We do this by looping starting from the 3rd term (index 2) up to the nth term (index n). In each iteration of the loop, we calculate the next Fibonacci number by adding the last two Fibonacci numbers. But remember, we're calculating the numbers modulo 10^9+7. This keeps the numbers from getting too large.

Returning the Result: Once we've calculated all the Fibonacci numbers up to the nth term, we return the entire list.

Example: If the student wants the Fibonacci series up to the 5th term, here's what happens:

We start with **[0, 1].**
Then we calculate the **3rd term: 0 + 1 = 1.**
Next, we calculate the **4th term: 1 + 1 = 2.**
After that, the **5th term: 1 + 2 = 3.**
Finally, we add 3 to our list. So, the result is **[0, 1, 1, 2, 3].**
So, this code is a simple and efficient way to generate Fibonacci numbers while ensuring they don't become too large. It's like constructing a ladder of numbers, each step being the sum of the two steps below it, but we're making sure each step is manageable in size.

Let's say we want to generate the Fibonacci series up to the 5th term. Here's how the process would look visually:

Term: 0 1 2 3 4 5
Value: 0 1 ? ? ? ?

Initially, we know the values of the first two terms (0 and 1), so our graph looks like this:

Term: 0 1 2 3 4 5
Value: 0 1 1 ? ? ?

Now, we need to calculate the value for the 3rd term. According to the Fibonacci sequence, it's the sum of the previous two terms: 0 + 1 = 1. So, our graph now looks like this:

Term: 0 1 2 3 4 5
Value: 0 1 1 ? ? ?

Next, we calculate the value for the 4th term. Again, it's the sum of the previous two terms: 1 + 1 = 2.

Term: 0 1 2 3 4 5
Value: 0 1 1 2 ? ?

Then, we calculate the value for the 5th term: 1 + 2 = 3.

Term: 0 1 2 3 4 5
Value: 0 1 1 2 3 ?

Finally, we have the entire Fibonacci series up to the 5th term: [0, 1, 1, 2, 3].