412. Fizz Buzz

Easy Math String Simulation Leetcode Link

Problem Description

The problem "FizzBuzz" is a classic example often used in coding interviews to assess basic programming abilities. Given a positive integer `n`, the task is to construct an array of strings that follow a specific pattern based on whether the index of the array (starting from 1) is divisible by 3, 5, or both:

- For each index 'i' that is divisible by both 3 and 5 (for example, 15, 30, 45), the array should contain the string "FizzBuzz" at that index.
- If the index 'i' is only divisible by 3 (like 3, 6, 9), the array should have "Fizz" at that index.
- If the index 'i' is only divisible by 5 (such as 5, 10, 20), the string "Buzz" should appear at that index.
- For all other indexes that don't meet the above divisibility conditions, the array should store the index 'i' itself as a string.

The function should return the array, which is described as `answer`, and it should be 1-indexed, meaning the first element corresponds to `i=1`.

Intuition

The solution is straightforward — it systematically checks each number from 1 to `n` against the divisibility rules provided in the problem statement. The process can be summarized as follows:

- Iterate over the range from 1 to `n` (inclusive), representing the index 'i'.
- For each 'i', examine the remainder when 'i' is divided by 3, 5, and the least common multiple of 3 and 5 (which is 15), to determine divisibility.
- If 'i' is divisible by 15, it means it's divisible by both 3 and 5, so append the string "FizzBuzz" to the answer list.
- Otherwise, if 'i' is divisible by just 3, append "Fizz".
- Otherwise, if 'i' is divisible by just 5, append "Buzz".
- If none of these conditions match, convert 'i' to a string and append it to the answer list.

This logic utilizes the properties of divisibility and the observation that any number divisible by both 3 and 5 is also divisible by 15, allowing for a clean and organized check without further complication or redundant conditions.

Solution Approach

The implementation of the solution for the FizzBuzz problem uses a simple for-loop and basic control statements (if-elif-else). Here's how each part of the code contributes to the solution:

- **Data Structure**: A list `ans` is used to collect the results. Lists in Python are dynamic arrays that can grow as needed, which is perfect for this use case since we initially don't know how many "Fizz", "Buzz", or "FizzBuzz" entries we will have.
- **Algorithm**:
  - A for-loop iterates over the range of numbers from 1 to `n` (inclusive), which directly translates to the 1-indexed array requirement of the problem.
  - Inside the loop, the first condition checked is `if i % 15 == 0`:
    - The use of 15 is a result of recognizing that 15 is the least common multiple (LCM) of 3 and 5, so any number divisible by both 3 and 5 is divisible by 15.
    - If this condition is true, "FizzBuzz" is appended to `ans`.
  - The `elif` conditions `i % 3 == 0` and `i % 5 == 0` follow:
    - These conditions use the modulus operator `%` to check for divisibility by 3 or 5, respectively.
    - If `i` meets either condition, "Fizz" or "Buzz" is appended to `ans`.
  - The `else` case handles numbers not divisible by 3 or 5 by converting the number `i` to a string using `str(i)` and appending it to `ans`.
- **Code Complexity**:
  - Time complexity: O(n), where `n` is the input number. Each number up to `n` is checked exactly once.
  - Space complexity: O(n), since a list of `n` strings is being constructed to store the results.
- **Patterns Used**:
  - The implementation demonstrates the use of modulo arithmetic to check for divisibility, a typical pattern when handling such conditions.
  - Following a sequential pattern, checking the most stringent condition first (divisibility by both 3 and 5) simplifies the logic and prevents redundant checks.

By adhering to the outlines provided in the problem statement, the solution methodically constructs the required FizzBuzz sequence in the form of a list of strings.

Example Walkthrough

Let's go through each step of the FizzBuzz problem solution using the example where `n` is 16. This will demonstrate how the algorithm works by iterating through each number from 1 to 16 and deciding what to append to the answer list.

1. **Initialization**: Create an empty list `ans` to store the results.
2. **Iteration**: Use a for-loop to iterate through numbers from 1 to 16 (inclusive). These numbers represent the index 'i'.
   - When `i = 1`, none of the special conditions are met, so "1" is appended to `ans`.
   - When `i = 2`, similarly to `i = 1`, it does not meet any of the special conditions; "2" is appended.
3. **Divisibility Check**:
   - For `i = 3`, `i` is divisible by 3. Since `3 % 3 == 0` and `3 % 15 != 0`, "Fizz" is appended to `ans`.
   - For `i = 4`, it is neither divisible by 3 nor 5; thus, "4" is appended.
   - For `i = 5`, `i` is divisible by 5. Since `5 % 5 == 0` and `5 % 15 != 0`, "Buzz" is appended.
   - This process continues until `i = 15`, where special case for "FizzBuzz" is encountered because `15 % 15 == 0`.
   - Lastly, `i = 16` is another number that does not meet any special conditions, so "16" is appended.
4. **Constructing Final Output**: As the loop finishes, the `ans` list looks like this: `["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz", "11", "Fizz", "13", "14", "FizzBuzz", "16"]`.

Every other number from 1 to 16 checks each if-elif-else condition in the algorithm to determine the correct string to append. By the end of the loop, we get the complete FizzBuzz pattern up to the number 16.

5. **Return Result**: The function will return the `ans` list containing the final sequence adhering to the FizzBuzz rules for `n = 16`.

By closely adhering to the proposed solution approach, the FizzBuzz sequence is created with efficient checks for divisibility and by appending the correct responses to a dynamically growing list that is structured as per the problem requirements.

Python Solution

```python
from typing import List

class Solution:
    def fizzBuzz(self, n: int) -> List[str]:
        # Initialize an empty list to hold the results
        results = []

        # Loop through numbers from 1 to n
        for number in range(1, n + 1):
            # If number is divisible by both 3 and 5 (or 15)
            if number % 15 == 0:
                results.append('FizzBuzz')
            # If number is only divisible by 3
            elif number % 3 == 0:
                results.append('Fizz')
            # If number is only divisible by 5
            elif number % 5 == 0:
                results.append('Buzz')
            # If number is not divisible by either 3 or 5
            else:
                results.append(str(number))

        # Return the list containing the FizzBuzz results
        return results

```

Java Solution

```java
import java.util.List;
import java.util.ArrayList;

class Solution {
    /**
     * Generates a list of strings representing the FizzBuzz sequence up to n.
     *
     * @param n The number up to which the FizzBuzz sequence should be generated.
     * @return A list of strings corresponding to the FizzBuzz sequence.
     */
    public List<String> fizzBuzz(int n) {
        // Initialize an ArrayList to hold the FizzBuzz results
        List<String> answer = new ArrayList<>();

        // Loop through numbers from 1 to n
        for (int num = 1; num <= n; ++num) {
            // Initialize an empty string to hold the current answer
            String current = "";

            // Check if the number is divisible by 3 and append "Fizz" if it is
            if (num % 3 == 0) {
                current += "Fizz";
            }

            // Check if the number is divisible by 5 and append "Buzz" if it is
            if (num % 5 == 0) {
                current += "Buzz";
            }

            // If the string is still empty, the number is neither divisible by 3 nor 5
            // Convert the number to string and use it as the current answer
            if (current.isEmpty()) {
                current += Integer.toString(num);
            }

            // Add the current answer to the list
            answer.add(current);
        }

        // Return the complete list of FizzBuzz results
        return answer;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <string>

class Solution {
public:
    // Function to solve the FizzBuzz problem
    std::vector<std::string> fizzBuzz(int n) {
        // Initialize an empty vector of strings to store the result
        std::vector<std::string> result;

        // Loop from 1 to n
        for (int i = 1; i <= n; ++i) {
            // Initialize an empty string for the current element
            std::string element;

            // If i is divisible by 3, append "Fizz" to the element string
            if (i % 3 == 0) {
                element += "Fizz";
            }

            // If i is divisible by 5, append "Buzz" to the element string
            if (i % 5 == 0) {
                element += "Buzz";
            }

            // If the element string is still empty, it's neither divisible by 3 nor 5
            // Convert the number to string and use that as the element
            if (element.empty()) {
                element = std::to_string(i);
            }

            // Add the element string to the result vector
            result.push_back(element);
        }

        // Return the final result vector
        return result;
    }
};

```

Typescript Solution

```typescript
// Type annotation for the function parameter
const fizzBuzz = function (n: number): string[] {
    // Explicitly define type for the array that will hold the FizzBuzz string sequence
    let sequence: string[] = [];

    // Iterate from 1 to n
    for (let i = 1; i <= n; i++) {
        // If the current number i is divisible by 15, add "FizzBuzz"
        if (i % 15 === 0) sequence.push('FizzBuzz');
        // If the current number i is divisible by 3, add "Fizz"
        else if (i % 3 === 0) sequence.push('Fizz');
        // If the current number i is divisible by 5, add "Buzz"
        else if (i % 5 === 0) sequence.push('Buzz');
        // Otherwise, add the number as a string
        else sequence.push(i.toString());
    }

    // Return the array containing the FizzBuzz sequence
    return sequence;
};

```

Time and Space Complexity

Time Complexity

The given code loops through the numbers from 1 to `n`, performing a constant amount of work for each number. This results in a time complexity of `O(n)`, where `n` is the input number to the `fizzBuzz` function.

Space Complexity

The space complexity of the function is primarily dictated by the output list `ans`. Since the list `ans` will contain exactly `n` strings, the space complexity of the code is `O(n)`, reflecting the space required to store the output.