2894. Divisible and Non-divisible Sums Difference

Easy Math

Leetcode Link

Problem Description

In this problem, we are given two positive integers, `n` and `m`. Our task is to calculate the difference between the sum of all integers from 1 to `n` that are not divisible by `m`, and the sum of all integers from 1 to `n` that are divisible by `m`. To clarify, we define two sums:

- `num1` is the sum of all integers within the range `1` to `n` that cannot be evenly divided by `m`.
- `num2` is the sum of all integers within the same range that can be evenly divided by `m`.

The goal is to find the result of `num1 - num2`, which will be an integer. For a simple example, if `n` is 5 and `m` is 2, then `num1` (sum of numbers not divisible by 2) would be 1 + 3 + 5 = 9, and `num2` (sum of numbers divisible by 2) would be 2 + 4 = 6. Thus, the answer would be 9 - 6 = 3.

Intuition

To solve this problem, we can iterate through all numbers from 1 to `n` and categorize them based on whether they are divisible by `m` or not. If a number `i` is not divisible by `m`, we add it to `num1`. Otherwise, if it is divisible by `m`, we add it to `num2`. To find the final answer, we subtract `num2` from `num1`.

However, instead of maintaining two separate sums and calculating the difference at the end, we can simply keep a running total that adds the value of numbers not divisible by `m` and subtracts the value of numbers that are divisible by `m`. This approach eliminates the need for separate variables and consolidates the operation into a single expression.

We arrive at this solution approach because we recognize that adding and subtracting can be performed during a single pass through the range of numbers. This way, the solution becomes more efficient and we can immediately obtain the result after the pass without additional calculations.

Solution Approach

The solution provided follows a straightforward simulation approach. This approach is about going through each number in the range `[1, n]` one by one and directly applying the logic or operation required by the problem. In programming terms, this is typically done via a `for` loop, which iterates through the given range of numbers.

We don't use any additional data structures like arrays or lists to store intermediate results because we only need to keep track of a running total. This running total is updated on each iteration of the loop.

The algorithm can be described in the following steps:

1. Initialize a variable to store the running total. This can start at 0 because initially, we have no numbers to add or subtract.
2. Use a loop to iterate through all numbers from `1` up to and including `n`. In Python, this is done using the `range` function, i.e., `range(1, n + 1)`.
3. For every number `i` in the range:
   - If `i` is divisible by `m` (i.e., `i % m == 0`), subtract `i` from the running total. In Python, the modulo operator `%` is used to determine if there is any remainder from the division. A remainder of 0 means the number is divisible.
   - Otherwise, if `i` is not divisible by `m`, add `i` to the running total.
4. After the loop completes, return the final running total, which represents `num1 - num2`.

The solution translates this algorithm into a compact Python list comprehension and a generator expression inside the `sum` function. This is a common pattern in Python that allows for efficient iteration and computation of sums without the explicit need for a loop structure in the code.

The Python code `sum(i if i % m else -i for i in range(1, n + 1))` succinctly represents this process. It iterates over the range, and for each `i`, it adds `i` to the sum if `i` is not divisible by `m` and subtracts `i` otherwise, directly evaluating our condition in a one-liner.

This solution approach delivers an elegant and efficient way to calculate the desired difference without resorting to multi-step processes or additional memory usage.

Example Walkthrough

Let's illustrate the solution approach with a small example. Suppose we choose `n = 7` and `m = 3`. Our task is to find the difference between the sum of all integers from `1` to `7` that are not divisible by `3`, and the sum of all integers from `1` to `7` that are divisible by `3`.

To clarify:

- Sum of numbers not divisible by `3` (`num1`): 1 + 2 + 4 + 5 + 7 = 19
- Sum of numbers divisible by `3` (`num2`): 3 + 6 = 9

We want to find `num1 - num2`, which in this case is `19 - 9 = 10`.

Now, according to our solution approach:

1. We start by initializing a variable `total` to store our running total (`total = 0`).
2. We iterate through all numbers from `1` to `7` using a loop.
3. As we loop through each number `i` (from `1` to `7`), we check if it's divisible by `3`. If it is not divisible (`i % 3 != 0`), we add `i` to `total`. If it is divisible (`i % m == 0`), we subtract `i` from `total`.

The calculations would proceed as follows:

- Start with `total = 0`.
- i = 1: Not divisible by 3, so `total = total + 1`.
- i = 2: Not divisible by 3, so `total = total + 2`.
- i = 3: Divisible by 3, so `total = total - 3`.
- i = 4: Not divisible by 3, so `total = total + 4`.
- i = 5: Not divisible by 3, so `total = total + 5`.
- i = 6: Divisible by 3, so `total = total - 6`.
- i = 7: Not divisible by 3, so `total = total + 7`.

Adding it all up, we get:

- `total = 0 + 1 + 2 - 3 + 4 + 5 - 6 + 7 = 10`

4. After the loop completes, `total` gives us the final result. In this case, `total = 10`, which matches our manual calculation (`num1 - num2 = 19 - 9 = 10`).

Our Python code for this would look like this:

```python
total = sum(i if i % 3 else -i for i in range(1, 8))
print(total)  # Output: 10
```

The code uses a generator expression inside the `sum` function to apply the solution approach directly, resulting in an efficient and concise way to compute the answer. The generator expression takes each number in the range, checks if it is divisible by 3, and adds or subtracts it accordingly, performing the entire calculation in a single line.

Python Solution

```python
class Solution:
    def difference_of_sums(self, n: int, m: int) -> int:
        """
        Calculate the difference of sums where the range from 1 to n is considered.
        For each number i in the range, it is added to the sum if it is not divisible by m,
        and subtracted if it is divisible by m.

        :param n: The upper limit of the range to calculate the sum for.
        :param m: The divisor used to determine if a number should be subtracted from the sum.
        :return: The calculated difference of sums.
        """
        # Initialize the total_difference variable to store the cumulative sum
        total_difference = 0

        # Iterate through the range from 1 to n, inclusive
        for i in range(1, n + 1):
            # Check if the current number i is divisible by m
            if i % m == 0:
                # If divisible, subtract it from the total_difference
                total_difference -= i
            else:
                # If not divisible, add it to the total_difference
                total_difference += i

        # Return the calculated total difference of sums
        return total_difference

# Example usage:
# solution = Solution()
# result = solution.difference_of_sums(10, 2)
# print(result) # The result will be the difference of sums for numbers from 1 to 10 with m=2

```

Java Solution

```java
class Solution {
    // Method to calculate the difference between the sum of numbers not divisible by 'm'
    // and the sum of numbers divisible by 'm' within the range 1 to 'n'
    public int differenceOfSums(int n, int m) {
        // Initialize answer to store the final result
        int answer = 0;

        // Loop through numbers from 1 to 'n'
        for (int i = 1; i <= n; ++i) {
            // Check if the current number is divisible by 'm'
            if (i % m == 0) {
                // If it is divisible, subtract it from the answer
                answer -= i;
            } else {
                // If not, add it to the answer
                answer += i;
            }
        }

        // Return the computed difference
        return answer;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to calculate the difference between the sum of numbers
    // from 1 to n that are not multiples of m and the sum of numbers
    // that are multiples of m.
    int differenceOfSums(int n, int m) {
        // Variable to store the final result
        int difference = 0;

        // Iterate over each number from 1 up to and including n
        for (int i = 1; i <= n; ++i) {
            // Check if the current number 'i' is not a multiple of 'm'
            if (i % m != 0) {
                // If not a multiple, add it to 'difference'
                difference += i;
            } else {
                // If it is a multiple, subtract it from 'difference'
                difference -= i;
            }
        }

        // Return the final calculated difference
        return difference;
    }
};

```

Typescript Solution

```typescript
/**
 * Calculate the difference between the sum of numbers that are
 * not multiples of `m` and the sum of numbers that are multiples of `m`,
 * from 1 to `n`.
 *
 * @param {number} n - The upper limit of the range to consider.
 * @param {number} m - The modulus value for determining multiples.
 * @return {number} - The difference of sums.
 */
function differenceOfSums(n: number, m: number): number {
    // Initialize the answer to zero.
    let answer = 0;

    // Iterate over the range from 1 to `n` inclusive.
    for (let i = 1; i <= n; ++i) {
        // Add `i` to the answer if `i` is not a multiple of `m`,
        // otherwise subtract `i` from the answer.
        answer += i % m ? i : -i;
    }

    // Return the computed difference of sums.
    return answer;
}

```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the given integer. This is because the code iterates a single loop from `1` to `n`, performing a constant amount of work for each iteration.

The space complexity of the code is `O(1)`. Only a finite number of variables are used, and there is no data structure whose size scales with `n`, so the amount of memory used does not grow with the input size.