3005. Count Elements With Maximum Frequency

Easy Array Hash Table Counting

Problem Description

In this problem, we are given an array of positive integers, and we need to find the "total frequencies" of the elements having the "maximum frequency". The frequency of an element is defined by how many times that element appears within the array. In simpler terms, we must:

1. Count how many times each unique element appears in the given array.
2. Identify the highest frequency - that is, the maximum number of times any element is repeated in the array.
3. Add up the frequencies of all the elements that are repeated as many times as the maximum frequency.
4. Return this sum as our answer.

This is a problem of calculating occurrences and then working with those counts to determine which numbers are the most common and to what extent.

Intuition

The intuition behind the solution is based on two steps: counting and comparison. The approach goes as follows:

1. Count each element's occurrences: We can use a data structure, like a hash table, to keep track of how many times each element appears in the array. In Python, this can be efficiently done using the `Counter` class from the `collections` module.
2. Identify and sum the maximum frequencies: Once we have the counts, we look through them to find the maximum frequency value. With that value in hand, we can then sum up the occurrences of all elements that have this maximum frequency.

Solution Approach

The solution makes use of a counting approach, which is both intuitive and efficient for this type of problem. It leverages the following algorithms, data structures, and patterns:

1. **Hash Table (`Counter` from Python's `collections` module)**: This data structure is ideal for counting instances of elements because it provides constant-time lookups and insertions ($O(1)$). We use `Counter` to create a hash table where keys are the elements of `nums` and values are their corresponding frequencies.

   Implementation snippet:

   ```
   cnt = Counter(nums)
   ```

2. **Finding Maximum Value**: After counting, we need to find the maximum frequency. We can achieve this by utilizing the built-in `max` function to traverse the values in the counter and find the highest count.

   Implementation snippet:

   ```
   mx = max(cnt.values())
   ```

3. **Summing Specific Frequencies**: Lastly, we need to return the sum of the frequencies of the elements that are exactly equal to the maximum frequency found. We achieve this by using a generator expression that iterates over the values of our counter and sums up those that equal `mx`.

   Implementation snippet:

   ```
   return sum(x for x in cnt.values() if x == mx)
   ```

By combining these steps, the solution is able to find the "total frequency" of the maximum frequency elements in `nums`. The `Counter` simplifies the counting process, `max` helps us quickly find the highest frequency, and the generator expression is a Pythonic way to compute the conditional sum we're interested in.

This is a direct approach working well for the problem due to its simplicity and the efficiency of counting and hash table operations in Python.

Example Walkthrough

Let's go through an example to illustrate the solution approach described above. Suppose we have the following array of integers:

```
nums = [2, 3, 3, 2, 5]
```

First, we shall count each element's occurrences using the `Counter` class.

```
cnt = Counter(nums)  # cnt will be Counter({2: 2, 3: 2, 5: 1})
```

In this step, the `Counter` object `cnt` gives us `{2: 2, 3: 2, 5: 1}` which shows that '2' appears 2 times, '3' appears 2 times, and '5' appears 1 time.

Next, we need to identify the maximum frequency from the `Counter` object. In this case, both 2 and 3 have the same highest frequency, which is 2.

```
mx = max(cnt.values())  # mx will be 2
```

Here, `mx` is the variable holding the maximum frequency, which equals 2.

Finally, we sum the frequencies of all elements that have this maximum frequency. Since the maximum frequency is 2, and elements 2 and 3 both occur twice, we sum these frequencies:

```
return sum(x for x in cnt.values() if x == mx)  # returns 2 + 2 = 4
```

The generator expression iterates over the values `2, 2, 1` in the `cnt`, but only sums those equal to `mx` (which is 2), resulting in `2 + 2`, and thus the total frequency sum we return is 4.

Our final result for the `nums` array using the steps described in the solution approach is 4. This demonstrates the correctness and efficiency of the described approach in finding the sum of the frequencies of the elements with the maximum frequency in the array.

Solution Implementation

Python

```python
from collections import Counter
from typing import List

class Solution:
    def maxFrequencyElements(self, nums: List[int]) -> int:
        # Count the frequency of each element in the nums list using Counter
        frequency_counter = Counter(nums)

        # Find the maximum frequency among all elements
        max_frequency = max(frequency_counter.values())

        # Calculate the sum of elements that have the maximum frequency
        # This is the count of elements that appear the most frequently in the list
        max_frequency_elements_count = sum(frequency for frequency in frequency_counter.values() if frequency == max_frequency)

        return max_frequency_elements_count

# Example usage:
# solution = Solution()
# result = solution.maxFrequencyElements([1, 2, 2, 3, 3, 3])
# print(result)  # Output: 3, since '3' appears 3 times, which is the max frequency
```

Java

```java
class Solution {
    public int maxFrequencyElements(int[] nums) {
        int[] frequency = new int[101]; // Array to store the frequency of elements, assuming the values range between 0 and 100
        // Count the frequency of each number in the given array
        for (int num : nums) {
            ++frequency[num];
        }

        int totalMaxFrequency = 0; // This will hold the sum of the maximum frequencies
        int maxFrequency = -1; // Store the current maximum frequency found

        // Iterate over the frequency array to find the highest frequency count(s)
        for (int freq : frequency) {
            if (maxFrequency < freq) { // Found a new maximum frequency
                maxFrequency = freq; // Update the maximum frequency
                totalMaxFrequency = freq; // Reset the total to the current max frequency
            } else if (maxFrequency == freq) { // Found another frequency count that matches the maximum
                totalMaxFrequency += freq; // Add to the total the frequency of this value
            }
        }
        return totalMaxFrequency; // Return the sum of the highest frequencies
    }
}
```

C++

```cpp
#include <vector>

class Solution {
public:
    // Function to find the maximum frequency of elements in the given vector
    int maxFrequencyElements(vector<int>& nums) {
        int counts[101] = {0}; // Initialize array to store frequencies of elements, assuming elements range from 0 to 100
        // Count frequency of each element in nums
        for (int num : nums) {
            ++counts[num];
        }
        int maxFrequency = 0; // Variable to hold the maximum frequency
        int maxOccurringElementCount = -1; // Helps in tracking the largest count found so far

        // Iterate over the frequency array to find the maximum frequency
        for (int freq : counts) {
            if (maxOccurringElementCount < freq) {
                // New maximum found, update max variables
                maxOccurringElementCount = freq;
                maxFrequency = freq;
            } else if (maxOccurringElementCount == freq) {
                // If current frequency matches max, accumulate the result
                maxFrequency += freq;
            }
        }

        // Return the maximum frequency of any element in nums
        return maxFrequency;
    }
};
```

TypeScript

```typescript
// Function to find the maximum frequency of elements in an array.
// The input is limited to integers between 0 and 100 inclusive.
function maxFrequencyElements(nums: number[]): number {
    // Initialize an array to hold counts for each possible number from 0 to 100.
    const counts: number[] = new Array(101).fill(0);

    // Populate the counts array with the frequency of each number.
    for (const num of nums) {
        ++counts[num];
    }

    // Initialize variables to store the maximum frequency found and
    // the sum of frequencies that are equal to the maximum.
    let maxFrequencySum = 0;
    let currentMaxFrequency = -1;

    // Iterate through the counts array to find the maximum frequency.
    for (const frequency of counts) {
        if (currentMaxFrequency < frequency) {
            // If the current frequency is higher than the previous maximum,
            // update the maximum frequency and reset the sum.
            currentMaxFrequency = frequency;
            maxFrequencySum = frequency;
        } else if (currentMaxFrequency === frequency) {
            // If the current frequency matches the maximum frequency,
            // add it to the sum.
            maxFrequencySum += frequency;
        }
    }

    // Return the sum of frequencies that are equal to the maximum frequency found.
    return maxFrequencySum;
}
```

```python
from collections import Counter
from typing import List

class Solution:
    def maxFrequencyElements(self, nums: List[int]) -> int:
        # Count the frequency of each element in the nums list using Counter
        frequency_counter = Counter(nums)

        # Find the maximum frequency among all elements
        max_frequency = max(frequency_counter.values())

        # Calculate the sum of elements that have the maximum frequency
        # This is the count of elements that appear the most frequently in the list
        max_frequency_elements_count = sum(frequency for frequency in frequency_counter.values() if frequency == max_frequency)

        return max_frequency_elements_count

# Example usage:
# solution = Solution()
# result = solution.maxFrequencyElements([1, 2, 2, 3, 3, 3])
# print(result)  # Output: 3, since '3' appears 3 times, which is the max frequency
```

Time and Space Complexity

Time Complexity

The time complexity of the given code can primarily be broken down into two segments:

1. Constructing the counter `cnt` from the list `nums` via `Counter(nums)`, which requires iterating through all the elements in `nums`. The time complexity for building the counter is $O(n)$, where n is the length of the list `nums`.
2. Finding the maximum frequency with `max(cnt.values())` and then summing up instances where the frequency is equal to this maximum frequency `sum(x for x in cnt.values() if x == mx)`. The time to find `max(cnt.values())` is $O(k)$, where k is the number of unique elements in `nums`. The summing part involves going through these values at most once, which is again $O(k)$.

Since $k <= n$, both steps are bounded by $O(n)$ complexity. Therefore, when considering the combined time complexity of these operations, the total time complexity is $O(n)$.

Space Complexity

The space complexity is determined by:

1. The space required to store the `cnt` counter, which can contain at most n unique elements if all elements of `nums` are unique. Thus, the space complexity due to the counter is $O(n)$.
2. The space required for variable `mx` is constant, $O(1)$.

Given that $O(n) + O(1)$ simplifies to $O(n)$, the overall space complexity of the code is $O(n)$.