1748. Sum of Unique Elements

Easy | Array | Hash Table | Counting

Leetcode Link

Problem Description

The problem is centered on being presented with an array `nums` that consists of integer elements. The primary task is to calculate the sum of the unique elements within this array. An element is considered unique if it appears only once in the array. Thus, if an element appears more than once, it should not be included in the sum.

For example, if the `nums` array is `[1, 2, 2, 3, 4]`, the unique elements are `1, 3,` and `4` as they appear exactly once. Adding these up gives us a sum of `1 + 3 + 4 = 8`.

Intuition

The intuition behind the solution is to first identify the unique elements. To do this systematically, we need to go through the array and keep track of the frequency or count of each element. This can be done efficiently using a hash map, which allows constant time insertions and lookups.

In Python, the `Counter` class from the `collections` module does exactly this – it creates a dictionary where keys are the elements of the array and values are the counts of those elements.

Once we have the frequency counts, the next step is to iterate over the count dictionary and collect all keys (the array elements) that have a count of exactly one (1), which indicates that they are unique. The final step is to sum up all these keys to get the desired output.

This approach is efficient because it operates in linear time, dependent on the size of the input array. The space used by the Counter may vary, but in the worst case, if all elements are unique, it will be proportional to the size of the input array.

Solution Approach

The solution provided uses Python's `Counter` from the `collections` module, which is a subclass of `dict`. Essentially, it is a dictionary where elements are stored as dictionary keys and their counts are stored as dictionary values.

Here's the step-by-step approach to the solution:

1. We instantiate a Counter with the `nums` array: `cnt = Counter(nums)`. At this point, `cnt` is a dictionary-like object where each key is a number from the `nums` array and the corresponding value is how many times that number appears in `nums`.
2. Next, we iterate through this `Counter` dictionary with a generator expression: `(x for x, v in cnt.items() if v == 1)`. This examines each pair of (element, count) in the dictionary. The generator expression yields each element `x` if its associated count `v` is exactly 1, which means `x` is a unique element in the original array.
3. The unique elements retrieved from the generator expression above are fed directly into the `sum()` function: `return sum(x for x, v in cnt.items() if v == 1)`. `sum()` takes an iterable as an argument, and here it cumulatively adds up all of the unique elements which were yielded by the generator expression.

By using a `Counter` to get the frequencies and a generator expression to filter and sum the unique elements, the solution sorts the problem efficiently in both time and space complexity—in linear time, since each element in the array is processed exactly once.

Example Walkthrough

Let's consider a small example to illustrate the solution approach. Suppose we have the following `nums` array: `[4, 3, 2, 4, 1]`.

1. We first create a Counter with our `nums` array, resulting in the structure: `cnt = Counter([4, 3, 2, 4, 1])`. Now, `cnt` would look like this: `{4: 2, 3: 1, 2: 1, 1: 1}` where the keys are the numbers from `nums` and their corresponding values indicate the number of times they appear in `nums`.
2. Next, we iterate through the `Counter` dictionary with a generator expression. To visualize this, we tackle each (element, count) pair:
   - `(4, 2)`: Since the count is not 1, `4` is not unique, and we do not consider it for our sum.
   - `(3, 1)`: Since the count is 1, `3` is unique, and we consider it for our sum.
   - `(2, 1)`: `2` has a count of 1, hence it is unique and included in our sum.
   - `(1, 1)`: `1` also has a count of 1, indicating it's unique, so we include it in our sum.
3. Now, we supply the unique elements yielded from the step above to the `sum()` function. Our generator expression would yield values `3`, `2`, and `1` which are the unique numbers.
4. Finally, the `sum()` function adds up these yielded values: `3 + 2 + 1 = 6`.

The final output of our example would thus be `6`, since it is the sum of the unique elements in the provided `nums` array. This walkthrough demonstrates how the provided solution approach would work with an actual array of integers.

Python Solution

```python
from collections import Counter

class Solution:
    def sumOfUnique(self, nums: List[int]) -> int:
        # Create a counter object to tally the frequency of each number in the nums list
        num_counts = Counter(nums)

        # Calculate and return the sum of all the numbers
        # that appear exactly once (unique numbers)
        return sum(number for number, count in num_counts.items() if count == 1)

```

Java Solution

```java
class Solution {
    public int sumOfUnique(int[] nums) {
        // Create an array to keep track of the count of each number
        int[] count = new int[101]; // Assuming that the elements in nums are within [1, 100]

        // Iterate over each element in the input array and increment the corresponding count
        for (int num : nums) {
            count[num]++;
        }

        int sum = 0; // Initialize sum to store the sum of unique elements

        // Iterate over the count array
        for (int i = 0; i < 101; i++) {
            // If the count of a number is exactly 1, it is unique and add it to the sum
            if (count[i] == 1) {
                sum += i;
            }
        }

        // Return the sum of all unique elements
        return sum;
    }
}

```

C++ Solution

```cpp
#include <vector> // Required for the std::vector type

class Solution {
public:
    // Function to calculate the sum of all unique elements in the vector nums.
    int sumOfUnique(vector<int>& nums) {
        // Create an array to store the count of each number.
        int counts[101] = {0}; // Initialize an array to store the count of each number.

        // Increment the count for each number in nums.
        for (int num : nums) {
            ++counts[num];
        }

        int sum = 0; // Variable to hold the sum of unique numbers.

        // Iterate through the counts array.
        for (int i = 0; i < 101; ++i) {
            // If a number occurs exactly once, add it to the sum.
            if (counts[i] == 1) {
                sum += i;
            }
        }

        // Return the total sum of unique numbers.
        return sum;
    }
};

```

Typescript Solution

```typescript
// This function calculates the sum of unique numbers in an array.
// A number is unique if it appears exactly once in the array.
function sumOfUnique(nums: number[]): number {
    // Create an array to keep track of the count of each number (0-100) within the nums array.
    const count = new Array(101).fill(0);

    // Iterate over the nums array to increment the count at each number's index.
    for (const num of nums) {
        ++count[num];
    }

    // Initialize sum to store the sum of unique elements.
    let sum = 0;

    // Iterate over the count array to check if a number appeared exactly once.
    for (let i = 0; i <= 100; ++i) {
        // If the count is exactly 1, add the number to the sum.
        if (count[i] === 1) {
            sum += i;
        }
    }

    // Return the total sum of unique numbers.
    return sum;
}

```

Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the input list `nums`. This is because the `Counter(nums)` operation iterates over the list once to count the occurrences of each element, and the `sum` function iterates over the counted elements once. Both operations are linear in terms of the number of elements in `nums`.

The space complexity of the code is also `O(n)` since it stores a count of each unique element in the list `nums`, creating a dictionary that in the worst case could have as many entries as there are elements in the list, if all elements are unique.