1394. Find Lucky Integer in an Array

Easy Array Hash Table Counting Leetcode Link

Problem Description

In this problem, we are given an array of integers called `arr`. We need to find an integer that is considered "lucky". A *lucky integer* is defined as an integer that occurs in the array with a frequency equal to its value. In other words, if we pick a number from the array, that number is lucky if it appears on the list exactly that many times. Our task is to return the *largest* such lucky integer found in the array. If there is no integer that meets the criteria for being lucky, we should return `-1`.

To solve this, one can inspect every unique number in the array and check if its value equals the number of times it appears in the array. Then, the largest of these numbers that satisfy the condition will be our answer. If we do not find any number meeting the condition, our answer would default to `-1`.

Intuition

The intuition behind the solution is to use a hash map (dictionary in Python) to keep track of the frequencies of each unique integer in the given array. We do this using `Counter`, which is a specialized dictionary from Python's `collections` module, designed for counting hashable objects.

Once we have the frequency of each integer, we iterate through the items of this dictionary. Here we make two checks for each element `x` with frequency `v`:

1. If the value `x` equals its corresponding frequency `v` in the array (i.e., `x` is a lucky number).
2. If this lucky number is larger than any we have seen earlier.

If both conditions are true, we update our answer `ans` to the current number `x`. If, after checking all elements, we don't find any lucky number, `ans` remains `-1`, which is also our default answer.

This solution ensures that we consider the largest lucky integer because we keep updating `ans` every time we find a larger lucky integer. The final answer is the largest one found or `-1` if no lucky integers are present.

Solution Approach

The implementation of the solution provided uses a `Counter` from Python's `collections` module which is a subclass of dictionary that is designed to efficiently count the frequency of items.

Here is the step-by-step explanation of the implementation:

1. We first create a `Counter` for the input array `arr`. The `Counter` collects the frequency of each integer value into a dictionary-like object, where keys are the integer values from the array, and their corresponding values are their respective counts (frequencies).

   ```python
   cnt = Counter(arr)
   ```

2. We initialize a variable `ans` with a value of `-1`. This will hold our largest lucky integer value or remain `-1` if no lucky integers are found.

   ```python
   ans = -1
   ```

3. The next step is to iterate over the items in the `Counter` object. We use a for-loop that gets each item as a tuple `x, v` where `x` is the integer and `v` is its frequency in `arr`.

   ```python
   for x, v in cnt.items():
   ```

4. Inside the loop, we apply our logic to check if an integer is lucky. For an integer to be lucky, its value must be equal to its frequency (`x == v`). If this condition is true, we are also interested if it's larger than our current answer `[ans < x]`.

   If an integer satisfies both conditions, it becomes our new answer as it's a lucky number and it is greater than any lucky number found so far. This ensures that by the end of the iteration, `ans` will hold the highest lucky integer.

   ```python
   if x == v and ans < x:
       ans = x
   ```

5. After the iteration completes, we return the value stored in `ans`. This will either be the largest lucky integer in the array or `-1` if no lucky integers were found.

   ```python
   return ans
   ```

The algorithm used in this solution has a time complexity of `O(n)`, where `n` is the number of elements in the array. This is due to the fact that we pass through the array once to create the frequency counter, and we pass through the dictionary of unique values, which also requires `O(n)` in the worst case when all elements are unique.

One key point to underline here is the use of the `Counter` data structure, which is optimized for counting and can be far more efficient than manually implementing a frequency count, especially for large datasets. The use of this data structure greatly simplifies the code and reduces the opportunity for errors.

Example Walkthrough

Let's consider an example where the given array `arr` is `[2, 2, 3, 4, 4, 4, 4, 5, 5]`.

1. First, create a `Counter` object for the array which will count the frequencies of each integer. The `Counter` will look as follows:

   ```python
   cnt = Counter([2, 2, 3, 4, 4, 4, 4, 5, 5])
   # Counter({4: 4, 2: 2, 5: 2, 3: 1})
   ```

2. Now initialize a variable `ans` with a value of `-1` to hold the largest lucky integer.

   ```python
   ans = -1
   ```

3. Iterate over the items in the `Counter`. Each item returned by `cnt.items()` will be a key-value pair of the integer and its frequency. For example, the first pair would be `(4, 4)`.

   ```python
   for x, v in cnt.items():
   # loop iterations:
   # (4, 4)
   # (2, 2)
   # (5, 2)
   # (3, 1)
   ```

4. Inside the loop, check if an integer is lucky (i.e., its value is equal to its frequency). For example, when `x` is `4` and `v` is `4`, check `x == v`. Since 4 is equal to its frequency, check if it is larger than `ans`. As `ans` is `-1`, update `ans = 4`.

   Continue the loop and find that `2` is also lucky since its value matches its frequency. However, `2` is not larger than the current `ans` which is `4`, so no update is made. Similarly, `5` and `3` are not lucky integers because their values do not match their frequencies.

5. Upon completion of the loop, return the value of `ans`. Since `4` was the largest lucky integer found, `ans = 4` will be returned.

Therefore, in this example, the largest lucky integer is `4`, and that is the value that our algorithm would accurately return for the given array.

Python Solution

```python
from collections import Counter

class Solution:
    def findLucky(self, arr: List[int]) -> int:
        # Create a counter to count the occurrences of each number in the array
        number_counter = Counter(arr)

        # Initialize the answer to -1, as -1 would be the default return value if no lucky integer is found
        lucky_integer = -1

        # Iterate through the items in the counter
        for number, count in number_counter.items():
            # A lucky integer is defined as an integer whose value is equal to the number of times it appears in the array
            if number == count and lucky_integer < number:
                # Update the lucky integer to the current number if it's greater than current lucky_integer
                lucky_integer = number

        # Return the largest lucky integer found, or -1 if none found
        return lucky_integer

```

Java Solution

```java
class Solution {
    // This method finds the lucky number in an array. A lucky number is defined as
    // a number that has a frequency in the array equal to its value.
    public int findLucky(int[] arr) {
        // Create an array to store the frequency of each number. Assuming the maximum
        // number value is 500, thus the length is 510 to include zero-indexing padding.
        int[] frequency = new int[510];

        // Iterate through the input array and increment the frequency count for each number.
        for (int num : arr) {
            // Increment the frequency for the number found in the array.
            ++frequency[num]; // Correct the loop to iterate over the 'arr' elements.
        }

        // Initialize the lucky number to -1, as a default value if there's no lucky number.
        int luckyNumber = -1;

        // Iterate through the frequency array to find a lucky number starting from 1
        // since the problem statement implies lucky numbers are positive.
        for (int i = 1; i < frequency.length; ++i) {
            // If the frequency of a number is the same as its value, update the luckyNumber.
            if (frequency[i] == i) {
                luckyNumber = i; // A bigger number will always override previous one, if found.
            }
        }

        // Return the lucky number, which will be -1 if none was found.
        return luckyNumber;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <algorithm> // for fill_n

class Solution {
public:
    /**
     * Find the largest lucky number in the array.
     * A lucky number is defined as a number which has a frequency in the array equal to its value.
     * @param arr the input vector of integers
     * @return the largest lucky number or -1 if no such number exists
     */
    int findLucky(vector<int>& arr) {
        // Create an array to store counts of each number, initial size is enough for constraints
        int count[510];
        // Initialize all elements of count array to 0
        fill_n(count, 510, 0);

        // Iterate through the input array and increment the corresponding count index
        for (int num : arr) {
            ++count[num];
        }

        // Initialize the answer as -1, which means no lucky number found
        int ans = -1;
        // Loop through the count array to find the largest lucky number
        for (int i = 1; i < 510; ++i) {
            if (count[i] == i) {
                ans = i; // Update the answer if a new lucky number is found
            }
        }

        // Return the largest lucky number or -1 if none were found
        return ans;
    }
};

```

Typescript Solution

```typescript
function findLucky(arr: number[]): number {
    // Create an array to count the frequencies of each number, large enough to hold all potential values.
    const frequencyCounter = new Array(510).fill(0);

    // Iterate through the array and increment the count of each number in the frequencyCounter.
    for (const number of arr) {
        frequencyCounter[number]++;
    }

    // Initialize the lucky number to -1, indicating that no lucky number has been found yet.
    let luckyNumber = -1;

    // Iterate through the frequencyCounter, starting from 1 since 0 can't be a lucky number.
    for (let i = 1; i < frequencyCounter.length; ++i) {
        // Check if the current number's count is the same as the number itself.
        if (frequencyCounter[i] === i) {
            // Update the lucky number since a new lucky number is found.
            // As we iterate from low to high, this will ensure the largest lucky number will be recorded last.
            luckyNumber = i;
        }
    }

    // Return the lucky number, which is -1 if none was found or the largest lucky number from the array.
    return luckyNumber;
}

```

Time and Space Complexity

Time Complexity

The time complexity of the provided code can be analyzed as follows:

1. Counting the frequencies of elements in `arr` using `Counter(arr)` – This operation runs in `O(n)` time where `n` is the length of `arr`, as it requires a single pass over all the elements.

2. Iterating over the items in the `cnt` dictionary to find the lucky integer – The iteration over the dictionary items will take at most `O(n)` time since there can't be more unique elements than the number of elements in `arr`.

As a result, the overall time complexity is `O(n)` + `O(n)` = `O(2n)` which simplifies to `O(n)`.

Space Complexity

The space complexity of the provided code can be considered in the following parts:

1. The space used by the `Counter` to store the frequency of each element – This takes up `O(u)` space where `u` is the number of unique elements in `arr`, which in the worst case can be up to `n`.

2. The space used for the variable `ans` – This is constant space, `O(1)`.

Thus, the overall space complexity is `O(u)` + `O(1)` which simplifies to `O(u)`. When considering the worst case where all elements are unique, this becomes `O(n)`.