2190. Most Frequent Number Following Key In an Array

Easy Array Hash Table Counting

Leetcode Link

Problem Description

The problem presents us with an array of integers named `nums` and an integer `key`, which is guaranteed to be found within `nums`. Our objective is to identify the integer in `nums` that follows `key` the most frequently. To clarify, we are looking for the value `target` that appears exactly one position after `key` (i.e., `nums[i + 1] == target` when `nums[i] == key`) the greatest number of times throughout the array.

We should iterate through the array, checking pairs of consecutive numbers (`nums[i]` and `nums[i + 1]`). We need to keep track of how many times each `target` comes after `key`, and then return the `target` number that has the highest frequency of appearing immediately after `key`. If multiple `target` numbers have the same frequency, we only return the one with the maximum count, which per the problem description, is unique.

Intuition

To solve the problem, one efficient approach is to traverse through `nums` and use a data structure to keep a tally of the frequencies of each `target` that follows `key`. A common data structure that can be used for this purpose is a Counter (available in Python's `collections` module), which will hold `target` integers as keys and their counts as values.

We start by initializing an empty Counter object. As we loop through the array, we examine each pair of adjacent elements (`nums[i]` and `nums[i + 1]`). When we find an occurrence of `key`, we increment the count for the following number (`nums[i + 1]`). While doing this, we keep track of both the number with the maximum count encountered so far, and the current maximum count. If at any point we find a `target` with a higher count than the previous maximum, we update both the `ans` (answer) with the new `target` and `mx` (maximum count) with the new count.

Finally, once we've passed through the array, the variable `ans` will hold the `target` with the highest frequency of occurrence immediately following `key`, and that is what we return.

Solution Approach

The implementation of the solution uses a `Counter` from Python's `collections` module to track the frequency of each integer appearing immediately after the `key`. Also, it leverages the `pairwise` iterator from Python's `itertools` module to efficiently iterate over the array in adjacent pairs. Here is a step-by-step explanation of the code:

1. Initialize a `Counter` object named `cnt` to hold the frequencies, and two variables `ans` and `mx` to keep track of the answer (the most frequented target number) and the maximum frequency encountered so far, respectively.
2. Use `pairwise(nums)` to create an iterator that returns consecutive pairs of elements in `nums`. This will look like `(nums[0], nums[1]), (nums[1], nums[2]), ..., (nums[n-2], nums[n-1])`.
3. Iterate over these pairs of numbers `a` (current key) and `b` (potential target):
   - If `a` (the current number) is equal to `key`, it means `b` is immediately following `key`. Then increment the counter for `b` by 1: `cnt[b] += 1`.
   - After incrementing, check if the count for `b` exceeds the current maximum (`mx`). If it does, update `mx` to the new count for `b`, and set `ans` to `b` because `b` is now the new target with the maximum count following `key`.
4. Once the loop concludes, `ans` will hold the value of the most frequent target after `key`, and we return `ans` as the solution.

The use of a `Counter` object is crucial in this approach for efficient frequency tracking, which allows for the update and query operations to happen in constant time (O(1)). The `pairwise` utility simplifies the process of inspecting adjacent elements without having to manage index values manually. Together, these strategies offer a straightforward and efficient solution for the given problem description.

Example Walkthrough

Let's illustrate the solution approach with a small example:

```
nums = [1, 2, 3, 2, 2, 3, 4, 2, 5]
key = 2
```

Here, our task is to find the number that most frequently appears immediately after the `key`, which in this case is the integer `2`.

1. We initialize the `Counter` object `cnt` as empty, `ans` as `None`, and `mx` (maximum frequency) as `0`.
2. We create `pairwise` pairs from `nums`, which would look like this:

   ```
   (1, 2), (2, 3), (3, 2), (2, 2), (2, 3), (3, 4), (4, 2), (2, 5)
   ```

   Notice that the number `2`, our key, appears in the first slot of several pairs.
3. We iterate over each pair `(a, b)`:
   - The first interesting pair is `(2, 3)`. Since `a` is `2` (our `key`), we increment the counter for `b`, which is `3`: `cnt[3]` becomes `1`. `mx` was `0`, so `mx` is updated to `1`, and `ans` is set to `3`.
   - The next occurrence of `2` is in `(2, 2)`. We increment the counter for `2`: `cnt[2]` becomes `1`. Since `mx` is `1` and `cnt[2]` equals `mx`, nothing changes with `mx` and `ans`.
   - We encounter `(2, 3)` again. Incrementing the counter for `3`: `cnt[3]` becomes `2`. `mx` is then updated to `2`, and `ans` becomes `3`.
   - Finally, we see `(2, 5)`. We increment the counter for `5`: `cnt[5]` becomes `1`. Since `mx` is still higher (2), we do nothing.
4. After completion, the `Counter` object `cnt` contains:

   ```
   cnt = {3: 2, 2: 1, 5: 1}
   ```

`ans` holds `3`, and `mx` holds `2`. Thus, the most frequent number following `2` is `3`.

Therefore, the solution would return `3` as the target number that most frequently appears immediately after the `key` in the array `nums`.

Python Solution

```python
from collections import Counter
from itertools import pairwise

class Solution:
    def mostFrequent(self, nums, key):
        # Initialize a counter to keep track of the occurrences after the key
        count = Counter()

        # Variables to keep track of the element with the highest frequency
        most_frequent_element = 0
        max_frequency = 0

        # Iterate over the pairwise elements of nums to identify pairs where the first element is the key
        for current, following in pairwise(nums):
            # Check if the current element is equal to the key
            if current == key:
                # Increment the counter for the following element
                count[following] += 1
                # Check if the count of the following element is greater than the max frequency seen so far
                if max_frequency < count[following]:
                    # Update the max frequency and the most frequent element
                    max_frequency = count[following]
                    most_frequent_element = following

        # Return the element that appears most frequently immediately after the key
        return most_frequent_element
```

Note that the `pairwise` function is used here which requires Python 3.10 or newer. If an older version of Python is used, one could create pairs using a loop or by using the `zip` function like so:

```python
# Example for pairwise utility using zip when itertools.pairwise isn't available
def custom_pairwise(iterable):
    a, b = tee(iterable)
    next(b, None)
    return zip(a, b)
```

Java Solution

```java
class Solution {

    // Finds the number that appears most frequently immediately after the key in an array
    public int mostFrequent(int[] nums, int key) {
        // Array to store the counts of numbers
        int[] count = new int[1001]; // Assuming the input numbers will not exceed 1000
        int answer = 0; // Variable to store the most frequent number following the key
        int maxCount = 0; // Variable to store the max frequency

        // Loop through the array, but not including the last element
        // because it cannot be followed by any other number
        for (int i = 0; i < nums.length - 1; ++i) {
            // Check if the current element is the key
            if (nums[i] == key) {
                // Increment the count of the number that follows the key
                count[nums[i + 1]]++;

                // If the new count is greater than the current maxCount, update maxCount and answer
                if (maxCount < count[nums[i + 1]]) {
                    maxCount = count[nums[i + 1]];
                    answer = nums[i + 1];
                }
            }
        }

        // Return the number that is most frequently observed after the key
        return answer;
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    // Function to find the most frequent element in the array following the 'key' element
    int mostFrequent(vector<int>& nums, int key) {
        int count[1001] = {1}; // Initialize an array to store frequency of elements with all values set to 0
        int answer = 0;        // Variable to store the most frequent element
        int maxFrequency = 0; // Variable to store the maximum frequency

        // Iterate through the array, except the last element
        for (int i = 0; i < nums.size() - 1; ++i) {
            // Check if the current element is equal to the 'key'
            if (nums[i] == key) {
                // Increment the frequency of the element following the 'key'
                int nextElement = nums[i + 1];
                count[nextElement]++;

                // Update the answer if the next element's updated frequency is greater than the maxFrequency
                if (maxFrequency < count[nextElement]) {
                    maxFrequency = count[nextElement];
                    answer = nextElement;
                }
            }
        }

        // Return the most frequent element that follows the 'key'
        return answer;
    }
};
```

Typescript Solution

```typescript
function mostFrequent(nums: number[], key: number): number {
    // Initialize an array to count frequencies of elements following the key
    const frequencyCounter: number[] = new Array(1001).fill(0);

    // Variables for tracking the most frequent element and its frequency
    let mostFrequentElement = 0;
    let maxFrequency = 0;

    // Iterate through the array, but stop one element before the end
    for (let i = 0; i < nums.length - 1; ++i) {
        // Check if the current element is the key
        if (nums[i] === key) {
            // Increment the frequency count of the element after the key
            const target = nums[i + 1];
            frequencyCounter[target]++;

            // If the new frequency count is greater than the max frequency,
            // update the most frequent element and the max frequency
            if (maxFrequency < frequencyCounter[target]) {
                maxFrequency = frequencyCounter[target];
                mostFrequentElement = target;
            }
        }
    }

    // Return the element that appeared most frequently after the key
    return mostFrequentElement;
}
```

Time and Space Complexity

The time complexity of the provided code is `O(n)` where `n` is the length of the input list `nums`. This is because we are iterating through the list exactly once with `pairwise(nums)`, and the operations within the loop are performed in constant time.

The space complexity is `O(u)` where `u` is the number of unique elements that come immediately after `key` in the list `nums`. The worst case for space complexity occurs when every element following `key` is unique, in which case the `Counter` will store each unique element. However, on average, the space used by the `Counter` would probably be less than `n` because not all elements in `nums` would be unique or follow the `key`.