2057. Smallest Index With Equal Value

Easy Array Leetcode Link

Problem Description

The problem provides an array `nums`, which has non-negative integers, and it is indexed from `0`, meaning it's a 0-indexed array. The task is to find the index `i` in this array which satisfies the condition `i mod 10 == nums[i]`. Here, `i mod 10` means the remainder when the index `i` is divided by `10`. If such an index exists, the function should return the smallest one; if no such index exists, the function should return `-1`.

Simply put, you need to check each element in the list and see if the index of that element has the same remainder when divided by 10 as the value of the element itself. The question asks for the smallest index where this condition is true, meaning if the condition is true for multiple indices, you should return the first one (smallest index) you find that satisfies the condition.

Intuition

To solve this problem, we can take a straightforward approach by iterating over each element in the array and checking the condition for each index.

- We start from the first element (at index 0), then proceed to the second element (index 1), and so on.
- At each step, we perform the modulo operation (`i % 10`) and check if it equals the value of the element at that index (`nums[i]`). The modulo operation finds the remainder when `i` is divided by `10`.
- The moment we find a match, we know we have found the smallest index that satisfies the condition, because we are moving from the start of the array towards the end. Hence, we can immediately return the current index.
- If we go through the entire array without finding any index that satisfies the condition, we return `-1`, as directed by the problem statement.

The algorithm has a linear time complexity `O(n)` because, in the worst case, we might have to check every element until we find the correct index or determine that no such index exists.

Solution Approach

The solution uses a simple `for` loop to iterate over the array. This approach does not require any additional data structures or complex algorithms. The pattern used is direct iteration, which is often the go-to strategy for problems that require examining each element of a collection one by one.

Here are the steps followed in the solution:

1. Utilize Python's built-in `enumerate()` function to get both the index `i` and the value `v` at that index while iterating through `nums`. This is a common Python pattern for when you need to access both the elements and their indices in a list.

2. Use the modulo operator `%` to calculate `i % 10` at each step. In Python, `x % y` returns the remainder of `x` divided by `y`.

3. Compare the result of `i % 10` with the current value `v`. If they are equal, immediately return the current index `i` because that's the smallest index that satisfies the condition. This step is the implementation of the condition provided in the problem description.

4. If the loop terminates without finding any index meeting the condition, return `-1`. This is achieved using a single return statement at the end of the function, ensuring that the function exits with the correct value if no valid index is found during the iteration.

The overall structure of the solution is straightforward, with simplicity being a key virtue here. There is no optimization or shortcut because the problem requires checking each index explicitly and the problem description does not provide any constraints that would allow for skipping certain elements.

By following these steps, the function implemented in Python meets the problem's requirements with an efficient and easily understandable solution.

Example Walkthrough

Let's consider a small array `nums` to demonstrate the solution approach:

```
nums = [0, 2, 9, 4, 7, 1]
```

We will walk through the list and apply the steps provided in the solution approach:

1. Start with the first element:
   - Index `i = 0`, Value `v = nums[i] = 0`
   - Calculate `i % 10 = 0 % 10 = 0`
   - Compare `i % 10` with `nums[i]`. Since both are `0`, the condition is satisfied.
   - Return the index `0` as it satisfies `i % 10 == nums[i]`.

The function would thus conclude at the very first step since the first index (which is the smallest possible index) contains a value equal to `i % 10`. There is no need to continue checking the following elements, as we were tasked with finding the smallest index that satisfies the condition. The function would return `0` in this case.

However, for the sake of illustration, let's assume we continue to check other elements (although, in practice, the function would have stopped at the first matching index):

2. Look at the second element (although this step would be skipped in an actual implementation once a match is found):
   - Index `i = 1`, Value `v = nums[i] = 2`
   - Calculate `i % 10 = 1 % 10 = 1`
   - Compare `i % 10` with `nums[i]`. Since `1 != 2`, the condition is not satisfied.
3. Look at the third element:
   - Index `i = 2`, Value `v = nums[i] = 9`
   - Calculate `i % 10 = 2 % 10 = 2`
   - Compare `i % 10` with `nums[i]`. Since `2 != 9`, the condition is not satisfied.
4. Continue this process for the remaining elements:
   - For `i = 3`, `nums[i] = 4`, `i % 10 = 3`. But `3 != 4`, condition not satisfied.
   - For `i = 4`, `nums[i] = 7`, `i % 10 = 4`. But `4 != 7`, condition not satisfied.
   - For `i = 5`, `nums[i] = 1`, `i % 10 = 5`. But `5 != 1`, condition not satisfied.

Since we found a match at the very first index (index `0`), the function would have returned `0`. If the first element did not satisfy `i % 10 == nums[i]`, the function would continue checking subsequent indices in the same way until either a matching index is found or the end of the array is reached, in which case the function would return `-1`.

Python Solution

```python
class Solution:
    def smallestEqual(self, nums: List[int]) -> int:
        # Iterate over the list of numbers with their indices.
        for index, value in enumerate(nums):
            # Check if the current index modulo 10 equals the value.
            if index % 10 == value:
                # If condition matches, return the current index.
                return index
        # If no index-value match found, return -1.
        return -1

```

Java Solution

```java
class Solution {
    public int smallestEqual(int[] nums) {
        // Loop over each element in the given array
        for (int index = 0; index < nums.length; ++index) {
            // Check if the current index mod 10 equals the value at this index in the array
            if (index % 10 == nums[index]) {
                // If condition is true, return the current index as the answer
                return index;
            }
        }
        // If no index satisfies the condition, return -1
        return -1;
    }
}

```

C++ Solution

```cpp
#include <vector>

/**
 * Finds the smallest index such that the index divided by 10 gives the same remainder as the value at that index.
 * @param nums A vector of integers to be searched.
 * @return The smallest index fulfilling the condition or -1 if no such index exists.
 */
int smallestEqual(std::vector<int>& nums) {
    // Iterate through the vector of numbers
    for (int index = 0; index < nums.size(); ++index) {
        // Check if the current index modulo 10 is equal to the value at that index
        if (index % 10 == nums[index]) {
            // If the condition is satisfied, return the current index
            return index;
        }
    }
    // If no index satisfies the condition, return -1
    return -1;
}

```

Typescript Solution

```typescript
/**
 * Finds the smallest index such that the index divided by 10 gives the same remainder as the value at that index.
 * @param {number[]} nums - An array of numbers to be searched.
 * @returns {number} The smallest index fulfilling the condition or -1 if no such index exists.
 */
function smallestEqual(nums: number[]): number {
    // Iterate through the array of numbers
    for (let index = 0; index < nums.length; index++) {
        // Check if the current index modulo 10 is equal to the value at that index
        if (index % 10 === nums[index]) {
            // If condition is satisfied, return the current index
            return index;
        }
    }
    // If no index satisfies the condition, return -1
    return -1;
}

```

Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the `nums` list. This is because the function involves a single loop that iterates through each element of the list exactly once.

The space complexity of the code is `O(1)`, which means it uses a constant amount of extra space. The space used does not depend on the input size, as it only utilizes a few variables for its operations, regardless of the length of the input list.