

2283. Check if Number Has Equal Digit Count and Digit Value

EasyHash TableStringCounting

Leetcode Link

Problem Description

You are given a string `num`, which is indexed starting from 0 and has a length `n`. This string is made up of numeric characters (digits from 0 to 9). The problem poses a condition that must be evaluated for every character in the string `num`: the digit at each index `i` (with `i` ranging from 0 to `n - 1`) should appear exactly `num[i]` times anywhere within the string `num`. If this condition holds true for every index in the string, the function should return `true`; otherwise, it should return `false`.

To put it in simpler terms, if the first character of the string (index 0) is '2', there should be exactly two '0's present in the string. If the second character (index 1) is '1', there should be exactly one '1' in the string, and so on for each character in the string.

Intuition

Given the problem's constraints, we understand that we need to count the occurrences of each digit within the string `num`. We will then use these counts to verify the stated condition: does the digit `i` actually occur `num[i]` times in the string `num`.

One intuitive way to solve this problem is by using a counting method. Particularly in Python, we can utilize the `Counter` class from the `collections` module to count the occurrences of each character (digit) in `num`.

After we have the counts of each digit, we can iterate over each index `i` of the string `num`. For each index `i`, we check if the count of the digit as a string (since indices in the string `num` are represented as characters) is equal to `num[i]` converted to an integer. This is because `Counter` returns counts indexed by characters, and `num[i]` is also a character; we need to interpret it as an integer for the comparison.

The `all()` function is used to verify that the condition holds for every index `i`. If, at any point, the condition is not met, `all()` will immediately return `false`. If we successfully iterate over all indices without this happening, it will return `true`, which is the expected result.

We can thus summarize our solution approach as follows:

- Count the occurrences of each digit using a `Counter`.
- Iterate over each index and character of the string `num`.
- Check if the count matches the expected number of occurrences as stated by the character at that index.
- Use `all()` to determine if the condition holds for every index in the string.

Solution Approach

The solution uses the `Counter` class from Python's `collections` module to count the occurrences of each digit within the string `num`. Here's the step-by-step explanation of the implementation:

- `Counter(num)` is used to create a counter object, which is essentially a dictionary where each key is a digit from `num` represented as a string, and the corresponding value is the number of times that digit appears in `num`. For example, if `num='1210'`, `Counter(num)` would yield `Counter({'1': 2, '2': 1, '0': 1})`.
- We then use list comprehension combined with the `all()` function to check our condition across all indices of `num`. The `all()` function takes an iterable and returns `True` if all of the iterable's elements are truthy (i.e., evaluate to `True`). Otherwise, it returns `False`.
- Inside the list comprehension, we iterate over the string `num` using `enumerate(num)`. By using `enumerate`, we get both the index (`i`) and the value (`v`) at that index for each iteration.
- The list comprehension contains the expression `cnt[str(i)] == int(v)`. For each index-value pair (`i`, `v`), this checks if the count for the digit `i` in string form (`str(i)`) equals the numeric representation of the value at that index (`int(v)`). If the digit doesn't appear in `num`, `cnt[str(i)]` would return `0`, as `Counter` objects return a count of zero for missing items.
- Finally, the `all()` function aggregates these individual boolean checks. If all checks pass, it will return `True`, otherwise, it will return `False`.

By using a `Counter` and the `all()` function, the code efficiently checks the condition across all indexes with a concise and readable expression, without needing additional loops or conditional statements.

Example Walkthrough

Let's illustrate the solution approach by walking through a small example:

Suppose we are given the string `num = "1210"`. The expected result is for the function to return `true` because each digit appears exactly as many times as its index states it should.

Here are the steps following the solution approach described earlier:

- We use `Counter(num)` to get the counts of each digit in `num`. This gives us `Counter({'1': 2, '2': 1, '0': 1})` which means:
 - The digit '1' appears 2 times.
 - The digit '2' appears 1 time.
 - The digit '0' appears 1 time.
- Using the `all()` function combined with list comprehension, we verify if every condition holds true where `num[i]` should equal the count of the digit `i`.
 - List comprehension goes through each character `num[i]` with its respective index `i`.
- We now check each index and character:
 - At index 0, we have the character '1'. The count for '0's should be 1, according to our counter, it's true (`Counter({'1': 2, '2': 1, '0': 1}) - '0'` appears 1 time).
 - At index 1, we have the character '2'. The count for '1's should be 2, according to our counter, it's true (`Counter({'1': 2, '2': 1, '0': 1}) - '1'` appears 2 times).
 - At index 2, we have the character '1'. The count for '2's should be 1, which matches our counter (`Counter({'1': 2, '2': 1, '0': 1}) - '2'` appears 1 time).
 - At index 3, we have the character '0'. The count for '3's should be 0, and since '3' does not appear in `num`, the counter implicitly gives it a count of 0, which is correct.
- Because the list comprehension would evaluate to `[True, True, True, True]`, the `all()` function would return `True`.

Therefore, the function should return `true` for the input '1210' as all conditions are satisfied according to our solution approach.

Python Solution

```
1 from collections import Counter
2
3 class Solution:
4     def digitCount(self, num: str) -> bool:
5         # Count the occurrences of each digit in the string
6         digit_count = Counter(num)
7
8         # Iterate over each index and its corresponding value in the string
9         for index, value in enumerate(num):
10            # Convert the index to a string to use it as a key for the digit_count
11            # Convert the value at the current index to an integer
12            # Check if the count of the digit (which should be represented by the index) matches the value
13            # If any of them do not match, return False
14            if digit_count[str(index)] != int(value):
15                return False
16
17        # If all counts match, return True
18        return True
19
```

Java Solution

```
1 class Solution {
2     public boolean digitCount(String num) {
3         // Array to hold the count of digits from 0 to 9
4         int[] digitCount = new int[10];
5
6         // The length of the input string
7         int length = num.length();
8
9         // First loop: count how many times each digit appears in the string
10        for (int i = 0; i < length; ++i) {
11            // Increment the count for the current digit
12            digitCount[num.charAt(i) - '0']++;
13        }
14
15        // Second loop: check if the digit count matches the expected values
16        for (int i = 0; i < length; ++i) {
17            // If the actual count of digit 'i' is not equal to the value at
18            // index 'i' in the string, return false
19            if (digitCount[i] != num.charAt(i) - '0') {
20                return false;
21            }
22        }
23
24        // If all counts match, return true
25        return true;
26    }
27 }
28
```

C++ Solution

```
1 class Solution {
2 public:
3     // This method checks if the digit count matches the index of the digit in the string.
4     bool digitCount(string num) {
5         int count[10] = {0}; // Initialize a count array for digits 0-9.
6
7         // Increment the count for each digit found in the string.
8         for (char& c : num) {
9             ++count[c - '0'];
10        }
11
12        // Check if every digit in the string matches the count for its index.
13        for (int i = 0; i < num.size(); ++i) {
14            if (count[i] != num[i] - '0') {
15                // If the count doesn't match the digit at the index 'i', return false.
16                return false;
17            }
18        }
19
20        // If all digits have the correct count as their index value, return true.
21        return true;
22    }
23 };
24
```

Typescript Solution

```
1 function digitCount(num: string): boolean {
2     // Get the length of the input number string
3     const length = num.length;
4
5     // Initialize an array to count the occurrences of each digit (0 to 9)
6     const counts = new Array(10).fill(0);
7
8     // Fill the 'counts' array with the frequency of each digit in 'num'
9     for (let i = 0; i < length; i++) {
10        const digit = Number(num[i]);
11        if (digit < counts.length) { // Ensure the digit is within the array bounds to avoid overwriting other indices
12            counts[digit]++;
13        }
14    }
15
16    // Iterate over the number string and decrement the count for the digit at each index
17    for (let i = 0; i < length; i++) {
18        const countIndex = Number(num[i]);
19        if (countIndex < counts.length) { // Check if index is within bounds to avoid negative indexing
20            counts[countIndex]--;
21        }
22    }
23
24    // Check if all counts have returned to zero
25    return counts.every((count) => count === 0);
26 }
27
```

Time and Space Complexity

Time Complexity

The time complexity of the code is determined by two main operations: constructing the `Counter` object for the string `num` and the loop that checks if each digit's count matches its expected frequency according to its position.

- Constructing the `Counter` object takes $O(n)$ time, where `n` is the length of the input string `num`. This is because it involves iterating through each character in `num` to count the occurrences.
- The loop using `all` iterates through each character in the string `num` and compares the count from `Counter` object. It runs in $O(n)$ time, since it must make `n` comparisons.

Since these operations are performed sequentially, the overall time complexity is the sum of their individual complexities. Therefore, the total time complexity is $O(n) + O(n)$, which simplifies to $O(n)$.

Space Complexity

The space complexity of the code is determined by the additional space used by the `Counter` object and the space required for the `all` function to iterate.

- The `Counter` object stores an integer for each unique character in `num`. Because `num` is a string of digits, there are at most 10 unique characters (digits 0 to 9), so the space used by `Counter` is constant, $O(1)$.
- The `all` function does not require additional space proportional to the input size since it evaluates the generator expression lazily.

The overall space complexity of the code is therefore the maximum of the individual space complexities, which is $O(1)$ since both the `Counter` object and the `all` function use constant space in the context of this problem.