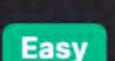
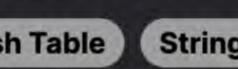
2283. Check if Number Has Equal Digit Count and Digit Value





Problem Description

String Hash Table Counting Leetcode Link

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

One intuitive way to solve this problem is by using a counting method. Particularly in Python, we can utilize the Counter class from

then use these counts to verify the stated condition: does the digit i actually occur num[i] times in the string num.

Given the problem's constraints, we understand that we need to count the occurrences of each digit within the string num. We will

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

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

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

result. We can thus summarize our solution approach as follows:

immediately return false. If we successfully iterate over all indices without this happening, it will return true, which is the expected

2. Iterate over each index and character of the string num.

3. Check if the count matches the expected number of occurrences as stated by the character at that index. 4. Use all() to determine if the condition holds for every index in the string.

1. Count the occurrences of each digit using a Counter.

- Solution Approach

Counter(num) would yield Counter({'1': 2, '2': 1, '0': 1}).

Here's the step-by-step explanation of the implementation:

1. 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',

The solution uses the Counter class from Python's collections module to count the occurrences of each digit within the string num.

- 2. 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.
- and the value (v) at that index for each iteration. 4. 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.

3. Inside the list comprehension, we iterate over the string num using enumerate(num). By using enumerate, we get both the index (i)

return False. By using a Counter and the all() function, the code efficiently checks the condition across all indexes with a concise and readable

5. Finally, the all() function aggregates these individual boolean checks. If all checks pass, it will return True, otherwise, it will

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

Here are the steps following the solution approach described earlier:

1. 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:

expression, without needing additional loops or conditional statements.

 The digit '1' appears 2 times. The digit '2' appears 1 time.

2. Using the all() function combined with list comprehension, we verify if every condition holds true where num[i] should equal

List comprehension goes through each character num[i] with its respective index i.

The digit '0' appears 1 time.

exactly as many times as its index states it should.

3. We now check each index and character:

the count of the digit i.

from collections import Counter

return True

15

16

17

18

19

19

20

21

22

23

24

25

26

28

27 }

'2': 1, '0': 1}) - '1' appears 2 times).

for index, value in enumerate(num):

return False

If all counts match, return True

if (digitCount[i] != num.charAt(i) - '0') {

return false;

// If all counts match, return true

- 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,
 - At index 2, we have the character '1'. The count for '2's should be 1, which matches our counter (Counter ((1'1': 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.

Iterate over each index and its corresponding value in the string

Convert the index to a string to use it as a key for the digit_count

- Therefore, the function should return true for the input '1210' as all conditions are satisfied according to our solution approach. Python Solution
 - class Solution: def digitCount(self, num: str) -> bool: # Count the occurrences of each digit in the string digit_count = Counter(num)

4. Because the list comprehension would evaluate to [True, True, True, True], the all() function would return True.

```
# Convert the value at the current index to an integer
11
               # Check if the count of the digit (which should be represented by the index) matches the value
12
               # If any of them do not match, return False
13
               if digit_count[str(index)] != int(value):
14
```

```
Java Solution
   class Solution {
       public boolean digitCount(String num) {
           // Array to hold the count of digits from 0 to 9
           int[] digitCount = new int[10];
           // The length of the input string
           int length = num.length();
           // First loop: count how many times each digit appears in the string
9
           for (int i = 0; i < length; ++i) {
10
               // Increment the count for the current digit
               digitCount[num.charAt(i) - '0']++;
12
13
14
15
           // Second loop: check if the digit count matches the expected values
           for (int i = 0; i < length; ++i) {</pre>
16
               // If the actual count of digit 'i' is not equal to the value at
17
               // index 'i' in the string, return false
```

return true;

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

if(digit < counts.length) { // Ensure the digit is within the array bounds to avoid overwriting other indices 11 12 counts[digit]++; 13 14

9

function digitCount(num: string): boolean {

const counts = new Array(10).fill(0);

const digit = Number(num[i]);

for (let i = 0; i < length; i++) {

const length = num.length;

// Get the length of the input number string

// Initialize an array to count the occurrences of each digit (0 to 9)

// Fill the 'counts' array with the frequency of each digit in 'num'

```
15
       // Iterate over the number string and decrement the count for the digit at each index
16
       for (let i = 0; i < length; i++) {
17
           const countIndex = Number(num[i]);
18
           if(countIndex < counts.length) { // Check if index is within bounds to avoid negative indexing</pre>
19
20
               counts[countIndex]--;
21
22
23
24
       // Check if all counts have returned to zero
       return counts.every((count) => count === 0);
25
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.
```

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.

• 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.

- 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.

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).

lazily. The overall space complexity of the code is therefore the maximum of the individual space complexities, which is 0(1) since both the

• 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, 0(1).

Counter object and the all function use constant space in the context of this problem.

• The all function does not require additional space proportional to the input size since it evaluates the generator expression