# 2520. Count the Digits That Divide a Number





**Leetcode Link** 

## **Problem Description**

Given an integer num, the task is to find out how many of its own digits can divide the number itself without leaving a remainder. To clarify, for a digit d in num to be a valid divisor, the number num divided by this digit d should result in a whole number, meaning that num % d should equal 0. It is important to note that the integer num will not contain the digit 0, ensuring we avoid division by zero. You're expected to count all such digits that divide num and return this count as your answer.

#### Intuition

The basic idea for solving this problem is straightforward: go through each digit in the integer num and check if num can be divided by that digit without any remainder. We keep a count of all such digits that meet the criteria.

To implement this, we can use a while loop to iterate over the digits of num. In each iteration, we:

- 1. Obtain the last digit of num by using the divmod operation with 10, which gives us both the quotient and the remainder. The remainder here is the last digit which we will check for divisibility.
- 2. If num is divisible by this digit (checked by num % val == 0), we increment our answer count.
- 3. We continue this process until we've checked all digits of num.

By following this process, we can count and return the number of digits in num that divide num itself.

## **Solution Approach**

The implementation of the provided solution makes use of a simple while loop and basic arithmetic operations to solve the problem. This approach does not require additional data structures, thus maintaining a low space complexity. Here's how the solution is implemented, step-by-step:

- 1. Initialize ans as 0 which will hold our count of valid digits and x as num to begin processing the digits.
- 2. Use a while loop that continues as long as x is greater than 0. This loop will help us process each digit from right to left until we have iterated through all the digits of num.
- 3. Inside the loop, the divmod(x, 10) function is used to divide x by 10. divmod returns two values: the quotient and the remainder. The quotient is stored back in x for the next iteration, effectively removing the last digit from x. The remainder, which is the last digit of x, is stored in val.
- 4. An inline conditional statement checks if the current digit val divides num evenly (num % val == 0). If the condition is true, it increments the ans counter by 1.
- 5. Once all digits have been processed and the loop ends, the function returns the value of ans, which now represents the total count of digits in num that can divide it without a remainder.

By iterating through each digit and checking divisibility, the algorithm ensures that all possible divisors are considered. This approach is an example of enumeration, where every element in a set (in this case, every digit of an integer) is counted for its validity against a specific condition (divisibility). Hence, every digit is enumerated to determine whether it divides the original number.

This solution is efficient as it processes each digit once and performs a constant number of operations per digit, leading to a time complexity that is linear with respect to the number of digits in num.

# Example Walkthrough

Let's walk through a concrete example to illustrate the solution approach. Consider the integer num = 252. We need to find out how many of its digits can divide the number itself without leaving a remainder.

Following the solution approach step by step:

- 1. Initialize ans = 0 as the count of valid digits that divide num and x = num which in our case is x = 252.
- 2. Set up a while loop that continues while x > 0. Beginning with x = 252, we will iterate through each digit.

The first iteration goes as follows: 3. We use divmod(x, 10) to divide x by 10. For the first iteration, divmod(252, 10) returns (25, 2). We update x to 25 and val to 2. 4. We check if num % val == 0. Since 252 % 2 == 0, the condition is true, and we increment ans to 1.

The second iteration: 3. x has been updated to 25, and we perform divmod(25, 10) which gives us (2, 5). Now, x = 2 and val = 5. 4. Again, we check num % val. Here, 252 % 5 == 2, leaving a remainder. Therefore, we do not increment ans.

The third iteration: 3. Now with x = 2, when we use divmod(2, 10) we get (0, 2). We update x to 0 for the next iteration (which won't happen since x is now 0), and val = 2. 4. Checking num % val one last time, we find that 252 % 2 == 0. Since it divides evenly, we increment ans to 2.

In this example, two of the digits (2 and 2) can divide 252 without a remainder. So the function would return 2.

5. The while loop ends because x is now 0, and we have processed all digits. We return the final count ans, which is 2.

## Python Solution 1 class Solution:

```
def countDigits(self, num: int) -> int:
           # Initialize 'count' to keep track of the count of digits
           count = 0
           # Make a copy of the input number for manipulation
           temp_num = num
           # Iterate until the copied number is reduced to 0
           while temp num:
               # Divide 'temp_num' by 10 to isolate the rightmost digit ('digit')
11
12
               # and reduce 'temp_num' for the next iteration
               temp_num, digit = divmod(temp_num, 10)
13
14
               # Avoid a division by zero and check if the digit divides 'num' without a remainder
15
               if digit != 0 and num % digit == 0:
16
                   # If the digit divides 'num', increment the 'count'
17
                   count += 1
18
19
20
           # Finally, return the count of such digits
21
           return count
22
Java Solution
```

#### class Solution { public int countDigits(int num) {

```
int count = 0; // Initialize count of digits
           // Loop through each digit of the number
           for (int current = num; current > 0; current /= 10) {
               int digit = current % 10; // Get the last digit
               // Check if digit is non-zero and perfectly divides the num
               if (digit != 0 && num % digit == 0) {
                   count++; // Increment count if condition is met
11
12
13
14
           return count; // Return the final count
15
16
17 }
18
C++ Solution
```

#### 2 public: // Function to count the number of digits in 'num' that are divisors of 'num' int countDigits(int num) {

1 class Solution {

```
int count = 0; // Variable to store the count of digits that are divisors of 'num'
           for (int current = num; current > 0; current /= 10) {
               int digit = current % 10; // Extract the rightmost digit of 'current'
               // Avoid division by zero error when the digit is 0
               if (digit == 0) continue;
10
11
               // Check if 'num' is divisible by 'digit'
12
               if (num % digit == 0) {
13
                   ++count; // Increment the count if 'digit' is a divisor of 'num'
14
15
16
17
           return count; // Return the final count of divisor digits
18
19 };
20
Typescript Solution
```

### for (let current = num; current; current = Math.floor(current / 10)) { // Iterate over each digit of the number let digit = current % 10; // Extract the current digit

```
// Check if the original number is divisible by the current digit
           // and ensure not to divide by 0
           if (digit !== 0 && num % digit === 0) {
               count++; // Increment count if the number is divisible by the digit
10
11
12
       return count; // Return the total count of divisible digits
13 }
14
```

number of digits in a number increases with the logarithm of the number, hence the logarithmic complexity.

it only uses a fixed number of integer variables (ans, x, and val) regardless of the input size.

let count = 0; // Initialize a variable to count the digits that meet the criteria

# Time and Space Complexity

function countDigits(num: number): number {

The time complexity of the code is O(log num). This is because the while loop runs as many times as there are digits in num. The

The space complexity of the code is 0(1). The space used by the algorithm does not depend on the size of the input number num, as