<https://leetcode.com/problems/find-the-divisibility-array-of-a-string/description/>

You are given a 0-indexed string word of length n consisting of digits, and a positive integer m.

The divisibility array div of word is an integer array of length n such that:

div[i] = 1 if the numeric value of word[0,...,i] is divisible by m, or

div[i] = 0 otherwise.

Return the divisibility array of word.

**Example 1:**

**Input:** word = "998244353", m = 3

**Output:** [1,1,0,0,0,1,1,0,0]

**Explanation:** There are only 4 prefixes that are divisible by 3: "9", "99", "998244", and "9982443".

**Example 2:**

**Input:** word = "1010", m = 10

**Output:** [0,1,0,1]

**Explanation:** There are only 2 prefixes that are divisible by 10: "10", and "1010".

**Constraints:**

1 <= n <= 10^5

word.length == n

word consists of digits from 0 to 9

1 <= m <= 10^9

**Attempt 1: 2024-01-21**

**Solution 1: Math (30 min)**

class Solution {

    public int[] divisibilityArray(String word, int m) {

        int n = word.length();

        int[] result = new int[n];

        // Initialize variable to store the modulo m of the number formed so far

        long num = 0;

        for(int i = 0; i < n; i++) {

            int digit = word.charAt(i) - '0';

            // Aggregate the number by shifting the previous number by one decimal place

            // and adding the new digit, then calculate modulo m of the new number

            num = (num \* 10 + digit) % m;

            // If the current aggregated number is divisible by m (modulo is 0)

            // then set the corresponding position in the result array to 1

            if(num == 0) {

                result[i] = 1;

            }

        }

        return result;

    }

}

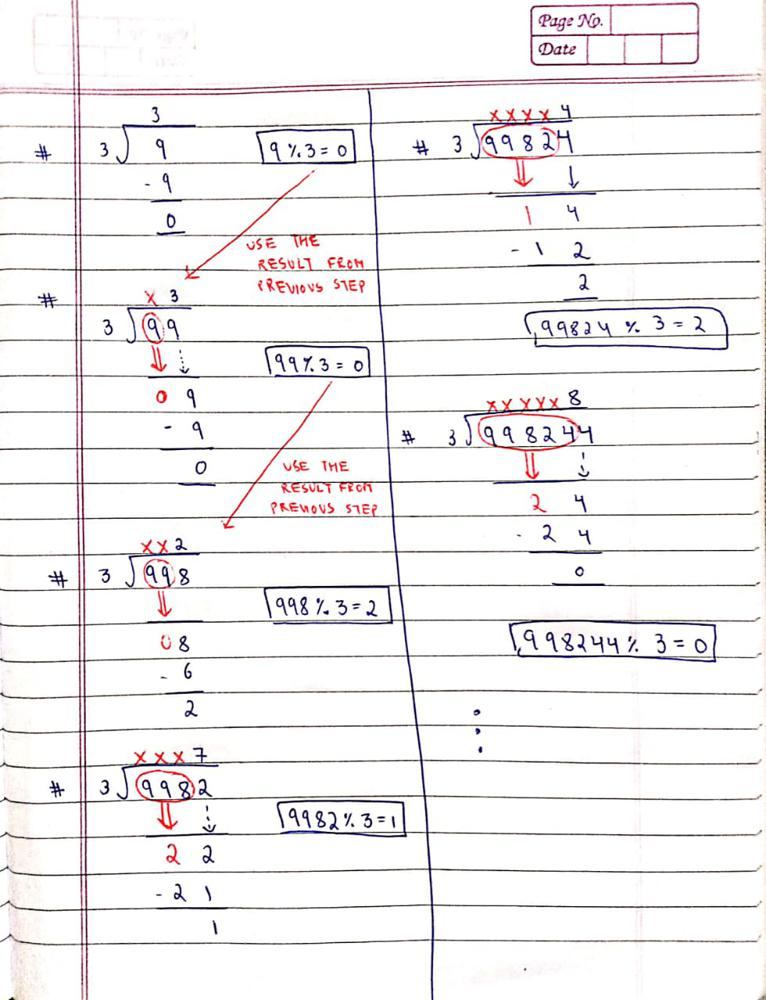
Time Complexity: O(N)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/find-the-divisibility-array-of-a-string/solutions/3230950/stepwise-explanation-with-images-easy-to-understand-simple-division-rule/>

<https://leetcode.com/problems/find-the-divisibility-array-of-a-string/solutions/3231219/explained-reminder-checking-very-simple-easy-to-understand-solution/>



**Approach**

Let say a number n when we devide it by m we get **q as quotient & r as reminder**.

So we can write => n = q\*m + r

Now add a new digit ( say d) to the the end.

This is equivalent to multiplying 10 with previous number and then add d to it.

So the new number can be written as n\*10 + d;

Now replace value of n from prvious equation, we get

= n\*10 + d

= (q\*m + r) \* 10 + d

= **10\*q\*m + 10\*r + d**

**In above result there are three terms about of which the first is always divisible by m. So the remaining part is 10r + d**

**Now we just need to check if 10 \* r + d is divisible by m or not.**

Keep continuing above for all digits in word.

**Refer to**

<https://algo.monster/liteproblems/2575>

**Problem Description**

In this problem, we have a string word which is made up of digit characters (0 to 9) and a positive integer m. The idea is to create a **divisibility array** div from this string. This array will have the same length as the word and each of its elements will either be a 1 or a 0. The rule for the divisibility array is as follows: if the number represented by the substring of word from index 0 to index i can be divided by m without any remainder, then div[i] will be 1. Otherwise, div[i] will be 0. The task is to return this divisibility array.

An example for clarity: If our word is "1234" and m=2, our div array should be [0, 1, 0, 0] because only the substring "12" is divisible by 2.

**Intuition**

The approach to solving this problem involves a consideration of how numbers in base 10 are constructed and how divisibility checks are performed. A key insight is that if we are to calculate whether various prefixes of a number are divisible by m, it's not efficient to compute the entire number from scratch at each step.

**Instead, we employ a modular arithmetic property that allows us to update our current value by taking into account only the newly added digit. Specifically, if we want to shift a number x by one decimal place and then add a digit d, this can be expressed as 10\*x + d. However, since we're only interested in divisibility by m, we can work with x and d modulo m to keep the numbers small and manageable.**

Respectively, with each new digit encountered in the word, we multiply our running tally x by 10 and add the numerical value of the current digit, always taking the modulo m after each operation. **This keeps x as the remainder of the number composed of digits seen so far divided by m.** If at any point x becomes 0, the number composed of digits up to that point is divisible by m, and we add 1 to our answer array; otherwise, we add 0.

We iterate over each character in the word, apply the process described above, and construct our divisibility array incrementally. This method is efficient and avoids redundant calculations, allowing us to get the answer in linear time with respect to the length of word.

**Solution Approach**

The provided solution uses a straightforward approach where no additional complex data structures or patterns are employed. The algorithm relies on basic arithmetic operations, specifically the modulus operation, and it follows the incremental construction philosophy:

Initialize an empty list ans which will eventually hold the resulting divisibility array.

Begin with a variable x set to 0. This variable represents the current numeric value as we process each character of the input string word, considering the modulo m.

Iterate over each character c in the string word. For each character,

Convert the character to its integer value.

Update x by multiplying it by 10 (shifting the number to the left by one decimal place) and adding the integer value of the current character to include it in our numeric value.

Perform the modulus operation with m to update x to contain the remainder of the new number modulo m.

Check if x is 0 after the modulus operation. If it is, append 1 to the list ans since the number composed of all digits up to this point is divisible by m. If x is not 0, append 0 to the list.

Proceed to the next character and repeat steps 3 and 4 until all characters are processed.

Once done, return the list ans as the final divisibility array.

The Python code provided efficiently implements this approach, using a loop to iterate over each character in word and modifying the variable x iteratively. The modulus operation is used to ensure that the numerical value considered at each step is within manageable bounds and directly corresponds to the divisibility condition.

**It's important to note that this approach, while simple, takes advantage of the modulus operation's property that (a \* b) % m = ((a % m) \* (b % m)) % m. This property allows us to keep intermediate values small and perform continuous divisibility checks without having to compute or store large numbers, hence maintaining a constant space complexity with respect to the value of the numbers involved.**

**Example Walkthrough**

Let's illustrate the solution approach using a small example. Suppose we have the string word = "2034" and the divisor m = 3. We want to generate a divisibility array div such that if the number represented by the substring of word from index 0 to index i is divisible by 3, then div[i] is 1; otherwise, it is 0.

Initialize an empty list ans to hold the resulting divisibility array.

Let variable x be the running total, initialized to 0.

We iterate over the characters in word. Initially, word[0] = '2'.

Convert character '2' to its integer value, which is 2.

Update x by calculating (10 \* x + 2) % 3 = (0 \* 10 + 2) % 3 = 2 % 3 = 2.

Since x is not 0, append 0 to the list ans.

The list ans is now [0] and x is 2.

The next character is '0'.

Convert '0' to integer 0.

Update x by calculating (10 \* x + 0) % 3 = (10 \* 2 + 0) % 3 = 20 % 3 = 2.

Since x is not 0, append 0 to list ans.

The list ans is now [0, 0] and x is 2.

The next character is '3'.

Convert '3' to integer 3.

Update x by calculating (10 \* x + 3) % 3 = (10 \* 2 + 3) % 3 = 23 % 3 = 2.

Since x is not 0, append 0 to list ans.

The list ans is now [0, 0, 0] and x is 2.

The last character is '4'.

Convert '4' to integer 4.

Update x by calculating (10 \* x + 4) % 3 = (10 \* 2 + 4) % 3 = 24 % 3 = 0.

Since x is 0, append 1 to list ans.

The final divisibility array ans is [0, 0, 0, 1], which corresponds to the fact that the substring "2034" is divisible by 3, but the substrings "2", "20", and "203" are not.

In this example, the divisibility array div for word = "2034" and m = 3 is [0, 0, 0, 1].

**Java Solution**

class Solution {

    public int[] divisibilityArray(String word, int m) {

        int wordLength = word.length(); // Get the length of the provided string

        int[] divisibility = new int[wordLength]; // Create an array to store divisibility results

        long numModM = 0; // Initialize variable to store the modulo m of the number formed so far

        // Iterate over each character in the string

        for (int i = 0; i < wordLength; ++i) {

            // Aggregate the number by shifting the previous number by one decimal place

            // and adding the new digit, then calculate modulo m of the new number

            numModM = (numModM \* 10 + (word.charAt(i) - '0')) % m;

            // If the current aggregated number is divisible by m (modulo is 0)

            // then set the corresponding position in the result array to 1

            if (numModM == 0) {

                divisibility[i] = 1;

            }

        }

        return divisibility; // Return the populated divisibility result array

    }

}

**Time and Space Complexity**

**Time Complexity**

The time complexity of the given code is O(n), where n is the length of the word string. This complexity arises because the code iterates over each character in the word string exactly once. Within the loop, it performs constant time operations: a multiplication, an addition, a modulo operation, and a conditional check. Since none of these operations depend on the size of the string, they don't add any additional factor to the complexity.

**Space Complexity**

The space complexity of the code is O(n), with n being the length of the word string. The additional space is used to store the ans list, which contains an integer for each character in the word. No other significant space-consuming structures or recursive calls are involved, so the space complexity is linear with respect to the input size.