2278. Percentage of Letter in String

Easy String

Problem Description

In this problem, we are given a string `s` and a single character `letter`. Our goal is to determine what percentage of the characters in the string `s` are the same as the character `letter`. Once we have calculated this percentage, we are required to round it down to the nearest whole percent. Rounding down means that if the percentage is a decimal, we drop the decimal part and keep only the whole number part. For instance, if the percentage comes out to be 75.9, we round it down to 75.

Intuition

The process to find the solution is straightforward. To calculate the percentage of occurrences of `letter` in the string `s`, we can follow these steps:

1. Count the number of occurrences of `letter` in `s`. In Python, this can be done using the `count` method of a string, which counts the number of times a substring appears in the string.
2. To get the percentage, we divide the number of occurrences of `letter` by the total length of the string `s`. Since the length of the string represents 100%, dividing the occurrence count by the length gives us the required fraction.
3. To convert this fraction into a percentage value, we multiply it by 100.
4. Lastly, because we need to return the percentage rounded down to the nearest whole number, we perform integer division by using `//` operator in Python, which divides and then truncates the decimal part.

Solution Approach

The solution to this problem employs a very simple approach, which does not require complex algorithms or data structures. The steps taken in the implementation are a direct translation of the intuitive approach mentioned earlier, and here is how they translate into code:

1. Use the `count` method on the string `s` to find the number of times the character `letter` appears. The `count` method iterates over the string and increments a counter every time it encounters `letter`. The method signature `s.count(letter)` performs this operation.
2. Multiply the count obtained by 100 to convert the count into a percentage. This step involves basic arithmetic multiplication, as we want to represent the count as a percentage of the total number of characters in the string.
3. Perform integer division using the `//` operator between the product obtained in the previous step and the length of the string `s`. Integer division will divide the two numbers and return the quotient without any fractional part, effectively rounding down to the nearest whole number. The length of the string is obtained with `len(s)`, which is a built-in function that returns the number of characters in a string.
4. The result of this integer division is the percentage of characters in `s` that are the same as `letter`, rounded down to the nearest whole number, which we return as the final answer.

In Python, this entire process is succinctly written in a single line as `return s.count(letter) * 100 // len(s)` within the `percentageLetter` method of the `Solution` class. This one-liner is an efficient way to perform the described steps due to Python's concise syntax for string manipulation and arithmetic operations.

No additional data structures are necessary for this approach, and the pattern used is a simple computation based on counting and arithmetic operations.

Example Walkthrough

Consider the string `s = "aaabbc"` and the character `letter = "a"`. We want to calculate what percentage of the characters in `s` are the same as `letter`, then round this down to the nearest whole percent.

1. We count the number of times `letter` appears in `s`. Using `s.count(letter)`, we find that "a" appears 3 times.
2. Next, we determine the percentage. We multiply the count (3) by 100, giving us 300.
3. Then, we divide this product by the total number of characters in `s`, which is 6 (found using `len(s)`). To ensure we round down to the nearest whole number, we use floor division: `300 // 6`.
4. The result is 50. Therefore, 50% of the characters in the string "aaabbc" are "a".

Using the code `return s.count(letter) * 100 // len(s)`, this calculation is condensed into one line, and when we pass our example through this expression, it would return the integer 50, denoting that 50 percent of the letters in "aaabbc" are "a" rounded down to the nearest whole number.

Solution Implementation

Python

```python
class Solution:
    def percentageLetter(self, s: str, letter: str) -> int:
        # Calculate the number of occurrences of the specified letter in the string
        count_of_letter = s.count(letter)

        # Calculate the total length of the string to determine the percentage base
        total_length = len(s)

        # Compute the percentage of the specified letter in the string
        # The use of '//' ensures the result is an integer (floor division)
        percentage = (count_of_letter * 100) // total_length

        # Return the computed percentage
        return percentage
```

Java

```java
class Solution {
    public int percentageLetter(String s, char letter) {
        // Initialize counter for occurrences of the given letter
        int count = 0;

        // Convert the string to a character array for iteration
        char[] characters = s.toCharArray();

        // Iterate over all characters in the string
        for (char c : characters) {
            // Check if the current character matches the target letter
            if (c == letter) {
                // Increment the count if it matches
                count++;
            }
        }

        // Calculate the percentage of the letter in the string
        // Multiply by 100 first to avoid integer division truncation
        int percentage = (count * 100) / s.length();

        // Return the calculated percentage
        return percentage;
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to calculate the percentage of a specific letter in a string.
    int percentageLetter(string s, char letter) {
        int count = 0; // Initialize a variable to count occurrences of 'letter'.

        // Iterate over each character in the string 's'.
        for (char& currentChar : s) {
            // Increment count if the current character matches 'letter'.
            count += (currentChar == letter);
        }

        // Calculate and return the percentage of the letter in the string.
        // Multiply count by 100 before dividing to avoid integer division issues.
        return (count * 100) / s.size();
    }
};
```

TypeScript

```typescript
// Given a string "s" and a character "letter", this function calculates the percentage of
// 'letter' occurrences within "s" and returns the floor value of the percentage.

// Define the function with its parameters and return type
function percentageLetter(s: string, letter: string): number {
    // Initialize a variable to keep track of the occurrences of "letter"
    let letterCount = 0;

    // Store the length of the input string for later use
    let totalCharacters = s.length;

    // Iterate over each character in the provided string
    for (let currentChar of s) {
        // If the current character is the same as the "letter" we're looking for, increment the count
        if (currentChar === letter) letterCount++;
    }

    // Calculate the percentage of the "letter" in the string
    // flooring it to get the lower integer bound.
    let percentage = Math.floor((letterCount / totalCharacters) * 100);

    // Return the calculated percentage
    return percentage;
}
```

```python
class Solution:
    def percentageLetter(self, s: str, letter: str) -> int:
        # Calculate the number of occurrences of the specified letter in the string
        count_of_letter = s.count(letter)

        # Calculate the total length of the string to determine the percentage base
        total_length = len(s)

        # Compute the percentage of the specified letter in the string
        # The use of '//' ensures the result is an integer (floor division)
        percentage = (count_of_letter * 100) // total_length

        # Return the computed percentage
        return percentage
```

Time and Space Complexity

Time Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the string `s`. This is because `s.count(letter)` iterates over each character in the string to count the occurrences of `letter`, which requires a single pass through the string.

Space Complexity

The space complexity is `O(1)` as the space used does not depend on the size of the input string `s`. Only a fixed number of variables (for the count and the result of the calculation) are used, which occupy constant space.