1796. Second Largest Digit in a String

Easy Hash Table String

Leetcode Link

Problem Description

The problem presents a scenario where we are given an alphanumeric string `s`. An alphanumeric string is composed of both lowercase English letters (such as 'a' to 'z') and numerical digits (from '0' to '9'). The task is to find the second largest numerical digit within the string and return its value. If the string does not have a second largest digit, because there is either no digit at all or there is only one unique digit, then the function should return `-1`.

Example Cases

- Given `s = "abc123"`, the function should return `2` because the numerical digits in the string are `1`, `2`, and `3`, and the second largest is `2`.
- Given `s = "ck077"`, the function should return `0` as `0` and `7` are the only digits, where `7` is the largest and `0` is the second largest.
- Given `s = "abc"`, the function should return `-1` because there are no numerical digits at all.

Intuition

To solve this problem, we can scan through the string character by character, checking for numeric digits. As we find each numeric digit, we compare it to two variables that are keeping track of the largest and second-largest digits found so far, let's call them `a` and `b`, respectively.

When a digit is encountered:

1. If it's larger than `a`, the current largest, we update the second-largest `b` to be the old largest `a` and then set `a` to this new largest value.
2. If it's not larger than `a` but is larger than `b` and not equal to `a`, it becomes the new second-largest, so we update `b`.

We initialize `a` and `b` to `-1` to indicate that we haven't found any digits yet. This also serves as our return value in case we find less than two unique numeric digits. After processing all characters, `b` will hold our answer, being the second-largest unique digit, or remain `-1` if there isn't one.

Solution Approach

The reference solution provided uses a simple, single-pass approach to iterate over the string and keep track of the two largest distinct digits found.

Let's go through the approach step by step:

1. Two variables `a` and `b` are initialized to `-1`. These will keep track of the largest (`a`) and the second largest (`b`) numerical digits found in the string so far.
2. We start a loop to iterate over each character `c` of the string `s`.
3. For each character, we check if it is a digit using `c.isdigit()`. If it isn't a digit, we simply continue to the next iteration, ignoring all non-digit characters.
4. If it is a digit, we convert it to an integer `v` using `int(c)`.
5. Now, we use two `if` conditions to update our two variables:
   - **First condition**: If the integer `v` is greater than our current largest `a`, it means we've found a new largest digit. So, we need to update both `a` and `b`:
     - `b` becomes the old largest digit (as it is now the second largest).
     - `a` is updated to be `v`, the new largest digit.
   - **Second condition**: If `v` is not greater than `a` but is greater than `b` and also not equal to `a` (we are looking for the second largest unique digit, distinct from the largest), then we update `b` to be `v`.
6. After the loop is finished, the variable `b` will contain the second largest digit if it exists, or `-1` if it doesn't.

No complex data structures or algorithms are needed here; the solution is straightforward and executed in a single pass, making it efficient with a time complexity of `O(n)`, where `n` is the length of the string.

Example Walkthrough

Let's take the string `s = "e4k95b2"` to illustrate the solution approach. We want to find the second largest numerical digit within this string.

1. We start by initializing two variables, `a` and `b`, to `-1`.
2. We begin iterating over each character of the string.
   - `e`: This is not a digit, so we ignore it.
   - `4`: This is a digit, so we check our conditions. Since `4` is greater than `a` which is `-1`, we update `a` to `4` and `b` remains `-1`.
   - `k`: Not a digit, we ignore it.
   - `9`: This is a digit, and it's greater than `a` which is currently `4`. We assign `a`'s value to `b`, so `b` becomes `4`, then we update `a` to `9`.
   - `5`: This is a digit, but it's smaller than `a` (`9`) and greater than `b` (`4`). Here, `b` is updated to `5`.
   - `b`: Not a digit, ignored.
   - `2`: This is a digit; however, it is smaller than both `a` and `b`, so we don't make any changes.
3. Having finished the loop, we've now determined that the largest digit in the string is `9` and the second largest is `5`.
4. We return the value of `b`, which is `5`, as the final answer.

By following these steps, we found that the second largest digit in the string `s = "e4k95b2"` is `5`. No further passes or complex operations were needed, and each step was simple and based only on comparison logic.

Python Solution

```python
class Solution:
    def secondHighest(self, s: str) -> int:
        # Initialize the highest and second highest numbers to -1
        highest = second_highest = -1

        # Loop through each character in the string
        for char in s:
            # Check if the character is a digit
            if char.isdigit():
                # Convert the character to an integer
                value = int(char)

                # Update the highest and second highest values if necessary
                if value > highest:
                    # If the value is greater than the current highest,
                    # update the highest and bump the old highest to second highest
                    highest, second_highest = value, highest
                elif second_highest < value < highest:
                    # If the value is between second highest and highest,
                    # update the second highest
                    second_highest = value

        # Return the second highest value found, or -1 if not found
        return second_highest

```

Java Solution

```java
class Solution {
    public int secondHighest(String s) {
        // Initialize first highest and second highest numbers to -1, assuming they don't exist.
        int firstHighest = -1, secondHighest = -1;

        // Iterate over the characters of the string.
        for (int i = 0; i < s.length(); ++i) {
            char currentChar = s.charAt(i);

            // Check if the current character is a digit.
            if (Character.isDigit(currentChar)) {
                int numericValue = currentChar - '0'; // Convert char to int value.

                // If the numeric value is greater than the first highest number,
                // update the second highest to be the old first highest,
                // and the first highest to be the new numeric value.
                if (numericValue > firstHighest) {
                    secondHighest = firstHighest;
                    firstHighest = numericValue;
                } else if (numericValue > secondHighest && numericValue < firstHighest) {
                    // If the numeric value is not the highest, but greater than the second highest,
                    // update the second highest.
                    secondHighest = numericValue;
                }
            }
        }

        // Return the second highest number found, or -1 if no such number exists.
        return secondHighest;
    }
}

```

C++ Solution

```cpp
#include <string> // Include the string header for using the string class

class Solution {
public:
    // Function to find the second highest digit in a string
    int secondHighest(string str) {
        int highest = -1;  // Initialize the highest digit as -1
        int secondHighest = -1; // Initialize the second highest digit as -1

        // Loop over each character in the string
        for (const char& ch : str) {
            // Check if the character is a digit
            if (isdigit(ch)) {
                int digitValue = ch - '0'; // Convert char to int digit

                // If the digit is greater than the current highest digit
                if (digitValue > highest) {
                    secondHighest = highest; // Update second highest to the old highest
                    highest = digitValue; // Update highest to the new digit
                } // Check if the digit is between the highest and second highest
                else if (digitValue > secondHighest && digitValue < highest) {
                    secondHighest = digitValue; // Update the second highest
                }
            }
        }
        return secondHighest; // Return the second highest digit found
    }
};

```

Typescript Solution

```typescript
// Function to find the second highest digit in a given string.
function secondHighest(s: string): number {
    let highestDigit = -1;     // Initialize the highest digit as -1.
    let secondHighestDigit = -1; // Initialize the second highest digit as -1.

    // Iterate through each character in the string.
    for (const character of s) {
        // Check if the current character is a digit.
        if (character >= '0' && character <= '9') {
            // Convert the character to its corresponding numerical value.
            const digit = character.charCodeAt(0) - '0'.charCodeAt(0);

            // Check if we've found a new highest digit.
            if (highestDigit < digit) {
                // Update highest and second highest digits.
                secondHighestDigit = highestDigit;
                highestDigit = digit;
            } else if (highestDigit !== digit && secondHighestDigit < digit) {
                // Update the second highest digit if the digit is
                // greater than the current second highest and not equal to the highest.
                secondHighestDigit = digit;
            }
        }
    }

    // Return the second highest digit found in the string.
    return secondHighestDigit;
}

```

Time and Space Complexity

Time Complexity

The time complexity of the provided function is `O(n)`, where `n` is the length of the input string `s`. This is because the function iterates over each character in the string exactly once. During each iteration, the operations performed (such as checking if the character is a digit, converting to an integer, comparing integer values, and assigning variables) are all constant time operations, hence they do not affect the overall linear complexity.

Space Complexity

The space complexity of the code is `O(1)`, as the space used does not scale with the input size. Only a fixed number of integer variables `a` and `b` are used to store the highest and second highest digits found, regardless of the size of the string.