2496. Maximum Value of a String in an Array

Easy Array String

Leetcode Link

Problem Description

The problem provides us with an array `strs` of alphanumeric strings, where an alphanumeric string is one that contains letters and/or numbers. The task is to determine the maximum value of any string in the array, with the 'value' being calculated differently based on the content of the string:

- If the string contains only digits, its value is the numeric value of the string when interpreted as a base 10 integer. For example, the string `"123"` has a value of 123.
- If the string contains any letters (thus it's not comprised of digits only), its value is the length of the string. For instance, the string `"abc123"` has a value of 6, since it contains three letters and three digits.

The goal is to iterate through each string in the array `strs`, calculate its value according to these rules, and return the highest value found.

Intuition

The intuition behind the solution is to iterate through each string in the array and apply the rules given to calculate the value for each string:

1. Define a function `f(s)` that takes a string `s` as input and returns:
   - The integer value of the string if the string is made up only of digits - we can use Python's built-in `int()` function for the conversion process.
   - The length of the string if it contains any non-digit characters - this can be done simply by using Python's `len()` function.
2. Then, use the built-in `max()` function to find the maximum value among all strings in the array. In Python, we can use a generator expression `(f(s) for s in strs)` to apply the function `f` to each string `s` in `strs` and find the maximum value.

The `all(c.isdigit() for c in s)` checks whether all characters `c` in string `s` are digits, which is needed to determine whether to interpret the string as a number or to use its length as the value.

By breaking down the problem into smaller parts and using Python's expressive features, we arrive at a concise and efficient solution.

Solution Approach

The solution's approach involves a single-pass algorithm leveraging basic list comprehension and utility functions in Python. Here's how it's implemented:

- The `Solution` class contains a method `maximumValue`, which is designed to process a list `strs` of alphanumeric strings and return the maximum calculated value among them.
- Within this method, we define an inner function `f(s)` that represents the rule-based evaluation of each string's value. It accepts a single string `s` as a parameter.
- The `f(s)` function uses a conditional expression using the generator expression `all(c.isdigit() for c in s)`. This evaluates to `True` if every character `c` in the string `s` is a digit (a numerical character between `'0'` and `'9'`), which Python's `str.isdigit()` method can check individually.
- If the string `s` consists solely of digits, `f(s)` converts it to an integer using `int(s)` and returns this integer value.
- If the string contains any non-digit characters, the `else` condition is triggered, and the length of the string is returned using `len(s)`.
- The main part of the `maximumValue` method consists of the `max()` function, which is a built-in Python function that finds the maximum item in an iterable. The iterable, in this case, is a generator expression that applies the `f(s)` function to every string `s` in the list `strs`.
- Each string is passed to the `f(s)` function, which calculates its value based on the rules provided. The `max()` function then compares these values, and the highest value is determined and returned as the result of the `maximumValue` method.

By using a helper function and a generator expression within the `max()` function, the solution elegantly handles the calculation of each string's value according to its characteristics (being a pure number or containing letters) and efficiently finds the maximum of these values.

Example Walkthrough

Let's consider the array `strs` contains the following alphanumeric strings:

```
["10", "abc", "456", "12345", "xyz89"]
```

Using the solution approach, let's iterate through each string and calculate its value:

1. The string `"10"` contains only digits, so according to the rules, its value is the numeric value of the string. Therefore, `f("10")` returns `int("10")` which is `10`.
2. The string `"abc"` contains letters, and so its value is the length of the string. Hence, `f("abc")` returns `len("abc")` which is `3`.
3. Similarly, `"456"` is also comprised of only digits, so `f("456")` evaluates to `int("456")`, giving the value `456`.
4. `"12345"` is another string of only digits, so its value is `f("12345")` which equals `int("12345")`, that is `12345`.
5. Lastly, the string `"xyz89"` contains letters as well as digits, so we only look at the length of the string. The value will be `f("xyz89")` which equals `len("xyz89")`, resulting in `5`.

Now we find the maximum value among all computed values. This is done by applying the `max()` function to the generator expression which evaluates `f(s)` for each string `s` in `strs`:

```
max_values = [f(s) for s in strs] # This will be [10, 3, 456, 12345, 5]
max_value = max(max_values) # This finds the max which is 12345

return max_value # The function returns 12345
```

So, the highest value found in the array `strs` is `12345`, which will be the output of the function `maximumValue`.

Python Solution

```
from typing import List

class Solution:
    def maximumValue(self, strs: List[str]) -> int:
        # This helper function calculates the value of a given string.
        def calculate_value(string: str) -> int:
            # If the string contains only digits, convert it to an integer.
            if all(char.isdigit() for char in string):
                return int(string)
            # Otherwise, return the length of the string.
            else:
                return len(string)

        # Calculate the values of all strings using the helper function
        # and return the maximum value found.
        return max(calculate_value(s) for s in strs)

```

Java Solution

```
class Solution {
    // Method to find the maximum value among all strings in the array
    public int maximumValue(String[] strings) {
        int ans = 0; // Initialize the answer to 0
        // Iterate through each string in the array
        for (String str : strings) {
            // Update the answer with the maximum value between the current answer and the value returned by function f
            ans = Math.max(ans, f(str));
        }
        return ans; // Return the maximum value found
    }

    // Helper function to calculate the value of a string based on given conditions
    private int f(String str) {
        int value = 0; // Initialize integer to store numeric value of the string
        // Iterate over each character in the string
        for (int i = 0, len = str.length(); i < len; ++i) {
            char ch = str.charAt(i); // Get the character at the current index
            // If the character is a letter, immediately return the length of the string as the value
            if (Character.isLetter(ch)) {
                return len;
            }
            // If the character is not a letter, accumulate its numeric value
            value = value * 10 + (ch - '0');
        }
        return value; // Return the numeric value of the string
    }
}

```

C++ Solution

```
#include <vector>    // Required for vector
#include <string>    // Required for string
#include <algorithm> // Required for std::max function

class Solution {
public:
    // Function to find the maximum value from a vector of strings
    int maximumValue(vector<string>& strs) {
        // Lambda function to calculate the numeric value of the string or
        // return the string size if it contains a non-digit character
        auto calculateValue = [](const string& str) {
            int numericValue = 0; // Initialize the numeric value to 0
            // Iterate through each character in the string
            for (const char& c : str) {
                // If the character is not a digit, return the size of the string
                if (!isdigit(c)) {
                    return static_cast<int>(str.size());
                }
                // Combine the previous value and the current digit to form the numeric value
                numericValue = numericValue * 10 + c - '0';
            }
            // Return the formed numeric value
            return numericValue;
        };

        int maxValue = 0; // Initialize the maximum value to 0
        // Iterate through each string in the input vector
        for (const auto& s : strs) {
            // Update maxValue with the greater value between itself and
            // the numeric value calculated for the current string
            maxValue = std::max(maxValue, calculateValue(s));
        }
        // Return the maximum value found
        return maxValue;
    }
};

```

Typescript Solution

```
/**
 * Calculates the maximum value from an array of strings where each string
 * is either a number represented as a string or a non-numeric string.
 *
 * @param {string[]} strings - An array of strings containing numbers or text.
 * @return {number} The maximum value found in the array.
 */
function maximumValue(strings: string[]): number {
    /**
     * Helper function that converts a string to a number if it is numeric,
     * otherwise returns the length of the string.
     *
     * @param {string} str - The input string to convert.
     * @return {number} The numeric value of the string or its length.
     */
    const convertStringToValue = (str: string): number => {
        // Check if the string is a valid number and convert, otherwise return its length
        return Number.isNaN(Number(str)) ? str.length : Number(str);
    };

    // Map each string to its numeric value and return the maximum from the resulting array
    return Math.max(...strings.map(convertStringToValue));
}

```

Time and Space Complexity

Time Complexity

The time complexity of the code is `O(nmk)`, where `n` is the number of strings in the input list `strs`, and `k` is the average length of the strings.

The `f` function consists of a comprehension `all(c.isdigit() for c in s)` that iterates over each character `c` in the string `s`, resulting in `O(k)` where `k` is the length of the string `s`. The `f` function is called once for each string in `strs`, leading to `n` calls in total.

Therefore, iterating over all `n` strings and checking each character for being a digit or not takes `O(nmk)` time.

Space Complexity

The space complexity of the code is `O(1)`.

The `f` function uses a comprehension that checks if each character is a digit, but it does not use any additional space that scales with the input size. The storage for temporary variables like the value of `int(s)` or `len(s)` does not depend on the size of the input list.

Since there is no dynamic data structure used that grows with the input, the space complexity remains constant, regardless of the size of `strs`.