1417. Reformat The String

Easy String

Leetcode Link

Problem Description

The problem presents us with an input string `s` that contains only lowercase letters and digits. Our task is to rearrange the characters of `s` into a new string where letters and digits alternate, meaning no two letters or two digits are adjacent to each other. If such an arrangement is not possible, the function should return an empty string.

In essence, we're asked to create a pattern like `letter-digit-letter-digit` ... or `digit-letter-digit-letter` ... throughout the entire string. If there is a significant imbalance between the number of digits and letters (i.e., the count of one type exceeds the other by more than one), it is clear that reformatting in the desired way is impossible. This is because at some point, we would need to place two characters of the same type next to each other to use all characters, which is against the rules.

Intuition

The underlying intuition for the solution comes from the rule that no two adjacent characters can be of the same type (i.e., both digits or both letters). This naturally leads to the observation that if there are significantly more characters of one type than the other, the task is impossible. Specifically, if the difference in quantity between letters and digits is more than one, we cannot interleave them perfectly.

To arrive at the solution:

1. We first split the input string into two lists: one containing all the letters and the other containing all the digits.
2. We then check the lengths of these two lists. If the difference between their lengths is more than 1, we return an empty string since reformatting is impossible.
3. If the list of letters is shorter than the list of digits, we swap the lists to ensure that we start and end with the character type that has more occurrences.
4. We then initialize an empty list `ans` to build the reformatted string.
5. We iterate over both lists simultaneously, taking one letter and one digit at a time and appending them alternately to `ans`.
6. If there is one extra character (which will always be of the type we started with due to the earlier swapping), we append it to the end of `ans`.
7. Finally, we join `ans` into a string and return it as the result.

In this approach, we use a greedy strategy, always placing a character of the surplus type (if there is one) at the beginning and end of the final string and then filling the middle by alternating between letters and digits.

Solution Approach

The solution uses the following algorithmic concepts and data structures:

Algorithm:

The algorithm is a greedy one, placing characters in such a way that we adhere to the alternating letter-digit pattern until we run out of characters. This approach takes advantage of the fact that as long as the number of letters and numbers are within one count of each other, a valid sequence is always possible.

Data Structures:

Two lists, `a` and `b`, are used to separate the letters and digits from the input string, respectively. An additional list, `ans`, is used to build the final result.

Approach Steps:

1. **Separating Characters**: Using list comprehensions, we go through the string twice, once to collect all lowercase letters and add them to `a`, and another time for digits that are added to `b`.

```
a = [c for c in s if c.islower()]
b = [c for c in s if c.isdigit()]
```

2. **Checking Possibility**: We compute the difference between the lengths of `a` and `b` using `abs(len(a) - len(b))`. If this value is more than 1, we immediately return an empty string `''` as it's impossible to rearrange the characters into a valid sequence.

```
if abs(len(a) - len(b)) > 1:
    return ''
```

3. **Arranging the Characters**:

    - If there are more digits than letters, we swap `a` and `b` so that `a` always contains the longer list.
    - The zip function is used to iterate over pairs of characters from `a` and `b`. In each iteration, we take one character from each list and append them to `ans`, ensuring that they alternate in the final string.

```
if len(a) < len(b):
    a, b = b, a
ans = []
for x, y in zip(a, b):
    ans.append(x + y)
```

4. **Handling the Remainder**: If `a` is longer than `b` by one character, which can only happen if there were more letters than digits or vice versa but not more than one, we append the last remaining character from `a` to `ans`.

```
if len(a) > len(b):
    ans.append(a[-1])
```

5. **Creating the Final String**:

    - Lastly, we join the list `ans` into a string using `''.join(ans)` and return it.

```
return ''.join(ans)
```

In summary, the algorithm first checks for the possibility of the task, then creates a structure to hold the alternating characters, and finally constructs the required string in a greedy manner.

Example Walkthrough

Let's assume we have an input string `s` = "a1b2c".

Following the solution approach:

1. **Separating Characters**:

    - We create two lists, where `a = ['a', 'b', 'c']` extracts the letters and `b = ['1', '2']` extracts the digits.

```
a = ['a', 'b', 'c']
b = ['1', '2']
```

2. **Checking Possibility**:

    - We calculate the difference in lengths. Here, `len(a) = 3` and `len(b) = 2`. The difference is `1`, which is allowable.
    - Since the difference is not more than 1, we can proceed.

3. **Arranging the Characters**:

    - Since `a` is not shorter than `b`, there is no need to swap.
    - We iterate using `zip` to take one character from each list and create pairs:

```
ans = []
for x, y in zip(a, b):
    ans.append(x)
    ans.append(y)
# Now, ans = ['a', '1', 'b', '2']
```

4. **Handling the Remainder**:

    - Because `a` is longer than `b` by one, we have one extra character 'c' from `a`. We append it to the list `ans`.

```
ans.append('c')
# The updated ans is ['a', '1', 'b', '2', 'c']
```

5. **Creating the Final String**:

    - We join `ans` into a string, resulting in "a1b2c".

```
return ''.join(ans)
# Output string is "a1b2c"
```

In this example, the input string was successfully rearranged to follow the pattern of letters and digits alternating, and no additional characters were left unused, satisfying the problem's requirements.

Python Solution

```python
class Solution:
    def reformat(self, s: str) -> str:
        # Extract all alphabetic characters into the list 'alphabets'
        alphabets = [c for c in s if c.islower()]

        # Extract all numeric characters into the list 'digits'
        digits = [c for c in s if c.isdigit()]

        # Early return an empty string if the absolute difference in counts is greater than 1
        # This is because it would be impossible to alternate characters as required
        if abs(len(alphabets) - len(digits)) > 1:
            return ''

        # If there are fewer alphabets than digits, swap the lists to prepare for formatting
        if len(alphabets) < len(digits):
            alphabets, digits = digits, alphabets

        # Initialize an empty list to hold the reformatted string characters
        reformatted = []

        # Iterate over the characters in both lists simultaneously using zip
        for alpha_char, digit_char in zip(alphabets, digits):
            # Append alternated characters to the 'reformatted' list
            reformatted.append(alpha_char + digit_char)

        # If there's an extra character in 'alphabets', append it to the end
        if len(alphabets) > len(digits):
            reformatted.append(alphabets[-1])

        # Join all elements in 'reformatted' into a string and return
        return ''.join(reformatted)

```

Java Solution

```java
class Solution {
    public String reformat(String s) {
        // StringBuilder to hold digits
        StringBuilder digits = new StringBuilder();
        // StringBuilder to hold letters
        StringBuilder letters = new StringBuilder();

        // Separate digits and letters into different StringBuilder objects
        for (char c : s.toCharArray()) {
            if (Character.isDigit(c)) {
                digits.append(c);
            } else {
                letters.append(c);
            }
        }

        // Get the lengths of the two StringBuilder objects
        int digitCount = digits.length();
        int letterCount = letters.length();

        // If the count difference is more than 1, it's impossible to reformat
        if (Math.abs(digitCount - letterCount) > 1) {
            return "";
        }

        // StringBuilder to hold the final reformatted string
        StringBuilder reformatted = new StringBuilder();

        // Build the reformatted string by alternating between the groups
        for (int i = 0; i < Math.min(digitCount, letterCount); ++i) {
            // Check which group has more characters and append accordingly
            if (digitCount > letterCount) {
                reformatted.append(digits.charAt(i));
                reformatted.append(letters.charAt(i));
            } else {
                reformatted.append(letters.charAt(i));
                reformatted.append(digits.charAt(i));
            }
        }

        // Add the extra character at the end if counts are not equal
        if (digitCount > letterCount) {
            reformatted.append(digits.charAt(digitCount - 1));
        }
        if (digitCount < letterCount) {
            reformatted.append(letters.charAt(letterCount - 1));
        }

        // Return the final reformatted string
        return reformatted.toString();
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to reformat the string such that digits and letters alternate,
    // and if it's not possible, return an empty string.
    string reformat(string s) {
        string digits = "", letters = "";
        // Separate digits and letters into two different strings.
        for (char c : s) {
            if (isdigit(c)) {
                digits += c;
            } else {
                letters += c;
            }
        }

        int digitCount = digits.size(), letterCount = letters.size();

        // If the difference in count between digits and letters is more than 1,
        // it's not possible to alternate.
        if (abs(digitCount - letterCount) > 1) return "";

        string result = "";

        // Interleave the characters from both strings.
        for (int i = 0; i < min(digitCount, letterCount); ++i) {
            // If there are more digits, start with a digit; otherwise, start with a letter.
            if (digitCount > letterCount) {
                result += digits[i];
                result += letters[i];
            } else {
                result += letters[i];
                result += digits[i];
            }
        }

        // If there's an extra character in one of the strings, append it to the result.
        if (digitCount > letterCount) result += digits[digitCount - 1];
        if (digitCount < letterCount) result += letters[letterCount - 1];

        return result;
    }
};

```

Typescript Solution

```typescript
// Function to separate digits and alphabetic characters from a string.
function separateDigitsAndLetters(s: string): { digits: string; letters: string } {
    // Initialize empty strings for digits and letters.
    let digits = '';
    let letters = '';

    // Iterate through each character of the string.
    s.split('').forEach(char => {
        // If the character is a digit, add it to the digits string.
        // Otherwise, add it to the letters string.
        if (!isNaN(parseInt(char))) {
            digits += char;
        } else {
            letters += char;
        }
    });

    // Return an object containing separated digits and letters.
    return { digits, letters };
}

// Function to check if reformatting is possible based on the counts of digits and letters.
function canReformat(digitCount: number, letterCount: number): boolean {
    // Reformatting is not possible if the count difference is more than 1.
    return Math.abs(digitCount - letterCount) <= 1;
}

// Helper function to interleave characters from digits and letters.
function interleaveCharacters(
    digits: string,
    letters: string,
    longerCount: number,
    shorterCount: number
): string {
    let result = '';

    // Interleave characters from both strings until one of them runs out.
    for (let i = 0; i < shorterCount; ++i) {
        // Append characters from the string with more characters first.
        if (longerCount === digits.length) {
            result += digits[i];
        }
        // Append characters from the other string next.
        result += letters[i];
        // Append characters from the string with fewer characters last (if not yet appended).
        if (longerCount !== digits.length) {
            result += digits[i];
        }
    }

    // Add the last remaining character if there is an extra in one of the strings.
    if (longerCount !== shorterCount) {
        result += longerCount === digits.length ? digits[shorterCount] : letters[shorterCount];
    }

    return result;
}

// Main function to reformat the string such that digits and letters alternate.
function reformat(s: string): string {
    // Use the separateDigitsAndLetters function to get digits and letters.
    const { digits, letters } = separateDigitsAndLetters(s);

    // Get the counts of digits and letters.
    const digitCount = digits.length;
    const letterCount = letters.length;

    // If reformatting is not possible, return an empty string.
    if (!canReformat(digitCount, letterCount)) {
        return '';
    }

    // Determine which string has more characters.
    const longerCount = Math.max(digitCount, letterCount);
    const shorterCount = Math.min(digitCount, letterCount);

    // Use the interleaveCharacters helper function to create the reformatted string.
    return interleaveCharacters(digits, letters, longerCount, shorterCount);
}

```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the input string `s`. This is because the code iterates through the string twice: once to create the list of letters `a` and once to create the list of digits `b`. Each of these operations is `O(n)`. The `zip` operation inside the for loop and the conditionals outside and inside the loop do not change the overall time complexity, which remains linear with respect to the size of the input.

The space complexity is also `O(n)`. We create two lists, `a` and `b`, which together hold all characters from the string `s`. In the worst case, where all characters are either digits or letters, both lists would be about half the length of `s`, so the space used by these lists scales linearly with the length of `s`. The `ans` list will at most hold the same number of characters as `s`, further contributing to the linear space complexity. The space required for the variables and the single characters added in each iteration is negligible compared to the space used by the lists.