1417. Reformat The String String **Leetcode Link** Easy

The problem presents us with an input string s that contains only lowercase letters and digits. Our task is to rearrange the characters

Problem Description

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

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

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.

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

Intuition

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.

The underlying intuition for the solution comes from the rule that no two adjacent characters can be of the same type (i.e., both

end of ans.

interleave them perfectly.

- 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.
- 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

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

Solution Approach

The solution uses the following algorithmic concepts and data structures:

of the final string and then filling the middle by alternating between letters and digits.

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

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:

Data Structures:

each other, a valid sequence is always possible.

them to a, and another time for digits that are added to b. 1 a = [c for c in s if c.islower()] 2 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.

1. Separating Characters: Using list comprehensions, we go through the string twice, once to collect all lowercase letters and add

3. Arranging the Characters:

3 ans = []

1 if abs(len(a) - len(b)) > 1:

return ''

• 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. 1 if len(a) < len(b):

If there are more digits than letters, we swap a and b so that a always contains the longer list.

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

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.

1 if len(a) > len(b):

5. Creating the Final String:

1 return ''.join(ans)

Following the solution approach:

1. Separating Characters:

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

2 b = ['1', '2']

2. Checking Possibility:

a, b = b, a

4 for x, y in zip(a, b):

ans.append(x + y)

ans.append(a[-1])

finally constructs the required string in a greedy manner. **Example Walkthrough**

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

 \circ We calculate the difference in lengths. Here, len(a) = 3 and len(b) = 2. The difference is 1, which is allowable.

In summary, the algorithm first checks for the possibility of the task, then creates a structure to hold the alternating characters, and

3. Arranging the Characters: Since a is not shorter than b, there is no need to swap.

4. Handling the Remainder:

1 ans.append('c')

1 return ''.join(ans)

Python Solution

1 class Solution:

12

14

15

16

17

18

19

20 21

22

23

24

25

26

27

28

29

30

31

32

41

42

43

44

45

46

48

49

50

51

53

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

52 }

C++ Solution

1 class Solution {

2 public:

2 # Output string is "a1b2c"

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

5 # Now, ans = ['a', '1', 'b', '2']

2 # The updated ans is ['a', '1', 'b', '2', 'c']

characters were left unused, satisfying the problem's requirements.

Extract all alphabetic characters into the list 'alphabets'

Extract all numeric characters into the list 'digits'

Since the difference is not more than 1, we can proceed.

We iterate using zip to take one character from each list and create pairs:

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

5. Creating the Final String: We join ans into a string, resulting in "a1b2c".

If there are fewer alphabets than digits, swap the lists to prepare for formatting

Initialize an empty list to hold the reformatted string characters

Iterate over the characters in both lists simultaneously using zip

If there's an extra character in 'alphabets', append it to the end

Append alternated characters to the 'reformatted' list

Join all elements in 'reformatted' into a string and return

// Add the extra character at the end if counts are not equal

reformatted.append(digits.charAt(digitCount - 1));

reformatted.append(letters.charAt(letterCount - 1));

// Function to reformat the string such that digits and letters alternate,

// Separate digits and letters into two different strings.

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

if (abs(digitCount - letterCount) > 1) return "";

// Interleave the characters from both strings.

if (digitCount > letterCount) {

result += digits[i];

for (int i = 0; i < min(digitCount, letterCount); ++i) {</pre>

// If the difference in count between digits and letters is more than 1,

// If there are more digits, start with a digit; otherwise, start with a letter.

if (digitCount > letterCount) {

if (digitCount < letterCount) {</pre>

return reformatted.toString();

string reformat(string s) {

for (char c : s) {

} else {

string result = "";

if (isdigit(c)) {

digits += c;

letters += c;

// it's not possible to alternate.

// Return the final reformatted string

// and if it's not possible, return an empty string.

string digits = "", letters = "";

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

In this example, the input string was successfully rearranged to follow the pattern of letters and digits alternating, and no additional

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 10 if abs(len(alphabets) - len(digits)) > 1: 11 return ''

alphabets, digits = digits, alphabets

for alpha_char, digit_char in zip(alphabets, digits):

reformatted.append(alpha_char + digit_char)

if len(alphabets) < len(digits):</pre>

if len(alphabets) > len(digits):

return ''.join(reformatted)

public String reformat(String s) {

reformatted.append(alphabets[-1])

reformatted = []

alphabets = [c for c in s if c.islower()]

def reformat(self, s: str) -> str:

Java Solution

class Solution {

```
// StringBuilder to hold digits
           StringBuilder digits = new StringBuilder();
           // StringBuilder to hold letters
            StringBuilder letters = new StringBuilder();
 6
           // Separate digits and letters into different StringBuilder objects
           for (char c : s.toCharArray()) {
9
                if (Character.isDigit(c)) {
10
                    digits.append(c);
11
12
                } else {
                    letters.append(c);
13
14
15
16
17
           // Get the lengths of the two StringBuilder objects
            int digitCount = digits.length();
18
19
            int letterCount = letters.length();
20
21
           // If the count difference is more than 1, it's impossible to reformat
           if (Math.abs(digitCount - letterCount) > 1) {
23
                return "";
24
25
26
           // StringBuilder to hold the final reformatted string
27
            StringBuilder reformatted = new StringBuilder();
28
29
           // Build the reformatted string by alternating between the groups
30
            for (int i = 0; i < Math.min(digitCount, letterCount); ++i) {</pre>
                // Check which group has more characters and append accordingly
31
                if (digitCount > letterCount) {
32
33
                    reformatted.append(digits.charAt(i));
                    reformatted.append(letters.charAt(i));
34
                } else {
35
36
                    reformatted.append(letters.charAt(i));
                    reformatted.append(digits.charAt(i));
37
38
39
40
```

40 41 42 }; 43

69

70

71

72

73

74

75

76

77

78

79

80

if (!canReformat(digitCount, letterCount)) {

// 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);

return '';

```
29
                    result += letters[i];
30
               } else {
31
                    result += letters[i];
32
                    result += digits[i];
33
34
35
36
           // If there's an extra character in one of the strings, append it to the result.
37
           if (digitCount > letterCount) result += digits[digitCount - 1];
38
            if (digitCount < letterCount) result += letters[letterCount - 1];</pre>
39
           return result;
Typescript Solution
  1 // 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 = '';
  6
         // Iterate through each character of the string.
         s.split('').forEach(char => {
  8
             // If the character is a digit, add it to the digits string.
  9
 10
             // Otherwise, add it to the letters string.
 11
             if (!isNaN(parseInt(char))) {
                 digits += char;
 12
 13
             } else {
 14
                 letters += char;
 15
 16
         });
 17
 18
         // Return an object containing separated digits and letters.
         return { digits, letters };
 19
 20
 21
    // 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.
 24
 25
         return Math.abs(digitCount - letterCount) <= 1;</pre>
 26 }
 27
     // Helper function to interleave characters from digits and letters.
     function interleaveCharacters(
 30
         digits: string,
 31
         letters: string,
         longerCount: number,
 33
         shorterCount: number
     ): string {
 35
         let result = '';
 36
 37
         // Interleave characters from both strings until one of them runs out.
         for (let i = 0; i < shorterCount; ++i) {</pre>
 38
 39
             // Append characters from the string with more characters first.
             if (longerCount === digits.length) {
 40
 41
                 result += digits[i];
 42
 43
             // Append characters from the other string next.
             result += letters[i];
 44
 45
             // Append characters from the string with fewer characters last (if not yet appended).
             if (longerCount !== digits.length) {
 46
 47
                 result += digits[i];
 48
 49
 50
 51
         // Add the last remaining character if there is an extra in one of the strings.
 52
         if (longerCount !== shorterCount) {
 53
             result += longerCount === digits.length ? digits[shorterCount] : letters[shorterCount];
 54
 55
 56
         return result;
 57 }
 58
    // 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.
 61
 62
         const { digits, letters } = separateDigitsAndLetters(s);
 63
 64
         // Get the counts of digits and letters.
 65
         const digitCount = digits.length;
 66
         const letterCount = letters.length;
 67
 68
         // If reformatting is not possible, return an empty string.
```

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 0(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.