4.29 (7 votes)

Solution

Overview

The problem states that we need to split the entire string into groups such that each group other than the first group has `k` number of upper-case characters.

Be sure to communicate thoroughly with your interviewer to make sure you're covering all cases. In this problem, the constraints are thorough because there is no interviewer to communicate with. However, in an interview, there is a potential to ask a few follow-up questions from the interviewer, like:

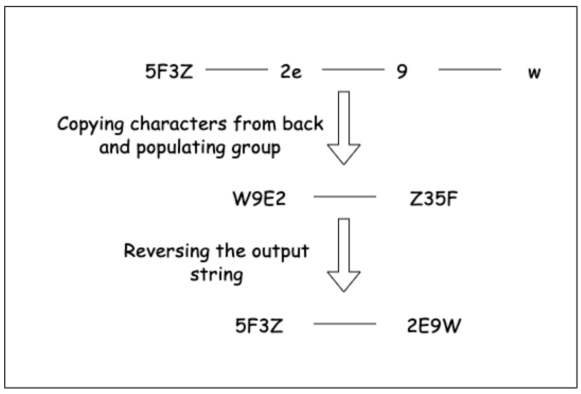
1. Can we have more numbers of groups in the output string as compared to the input string?
2. Can `k` be greater than the size of the input string?

Approach 1: Right to Left Traversal

Intuition

We need to form some groups in the string where each group has exactly the `k` characters in it except the first group (which can have `k` or fewer characters) and each group will be separated by a `-`.

Thus, the problem's main essence is finding how many alphanumeric characters will come in the first group! We can think of forming groups of size `k` from the end of the given string, and when the last group is left (which will be first in reality) it will automatically have `k` or fewer characters in it.



Using the above thought process, let's understand how to address this problem.

We can start traversing the string from the end so that we can form all the groups other than the first group in the size of `k` alphanumeric characters. While traversing from the end, we need to make sure that groups are formed in such a way that each group satisfies our problem's conditions. When we reach the start of the input string our output string will automatically be forming the group of size `k` alphanumeric characters leaving the first group with either equal to the size of `k` or lesser than the size of `k`. There's one scenario here where if all our groups including the first group are of size `k`, then `dash` gets inserted at the end of the string. Thus we need to make sure for such cases we should remove the last element from our answer string. However, our output string needs to be reversed since we were traversing the input string from the end.

Algorithm

1. Initialize:
   - `count` to `0`, which is used to count the number of characters in the current group.
   - `n` to input string length.
   - `ans` to an empty string, which is used to store the final result.
2. Now, iterate on the input string in reverse order:
   - We will skip `'-'` characters from the input string.
   - If the current character is not `'-'`, we include the current character in `ans` string and increment the current group size by incrementing `count` by `1`.

JavaScript Auto

```javascript
/**
 * @param {string} s
 * @param {number} k
 * @return {string}
 */
var licenseKeyFormatting = function(s, k) {
  const base = s.replaceAll('-', '').toUpperCase();
  const baseLength = base.length;
  let answer = '';

  if (baseLength % k) {
    answer += base.substr(0, baseLength % k);
    for (let i = 0; i < (baseLength - (baseLength % k)) / k; i++) {
      answer += '-' + base.substr(k * i + (baseLength % k), k);
    }
  } else {
    for (let i = 0; i < baseLength / k; i++) {
      answer += base.substr(k * i, k) + '-';
    }
    answer = answer.slice(0, answer.length - 1);
  }

  return answer;
};
```

Console Run Submit

- If `count` reaches `k`, it means we formed a group of size `k`, thus we can append a `'-'` in `ans` now, and reset `count` to start counting a new group.

3. After we finish traversing on the input string, we should check if the last character inserted wasn't a dash. If we find a dash we need to remove it from `ans` string.

4. Now that we formed all groups in reverse order, thus we need to reverse the `ans` string and then return it.

Implementation

C++ Java Python3 Copy

```cpp
class Solution {
public:
    string licenseKeyFormatting(string s, int k) {
        int count = 0;
        int n = s.length();
        string ans = "";
        for (int i = n - 1; i >= 0; i--) {
            if (s[i] != '-') {
                ans.push_back(toupper(s[i]));
                count++;
                if (count == k) {
                    ans.push_back('-');
                    count = 0;
                }
            }
        }
        if (ans.size() > 0 && ans.back() == '-') {
           ans.pop_back();
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }
};
```

Complexity Analysis

Let N be the size of the input array.

- Time Complexity: $O(N)$
  - We traverse on each input string's character once in reverse order which takes $O(N)$ time.
  - At the end, we reverse the `ans` thus iterating on it once, which also takes $O(N)$ time.
  - Thus, overall we take $O(N)$ time.
- Space Complexity: $O(1)$
  - We are not using any extra space other than the output string.

Approach 2: Left to Right Traversal

Intuition

To solve the problem, let's look at the inputs carefully,

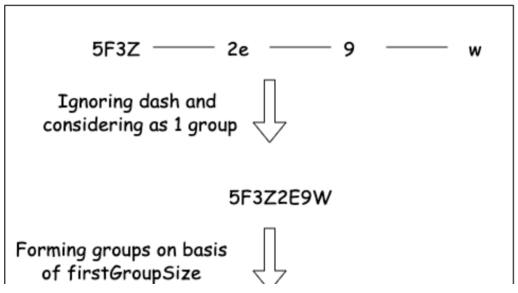
> We will be given an alphanumeric string which will have numbers, characters and dash.

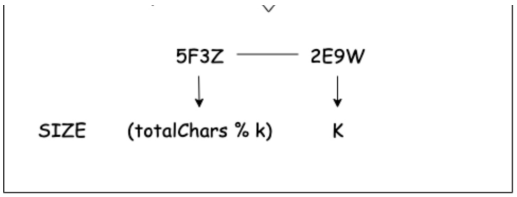
The problem states that we need to form equal groups of size `k` upper case characters other than the first group. For doing so, we need to first find the total number of alphanumeric characters in the input string.
And then the size of the first group will be decided on the basis of 2 factors:

1. Total count of alphanumeric characters in the string
2. Value of `k`

If we observe carefully, we just need to find how many characters will be left behind at last when we form groups from the end of the string, thus the size of the first group will be given by `total count of alphanumeric characters in string % value of k`.

In this approach, we will first populate the first group and then fill the characters in the remaining groups, whereas in the first approach we fill the first group in the end.





We can also have two cases where the size of `k` is equal to, or greater than the total count of alphanumeric characters of the input string. During such cases, our output string will only consist of 1 group.

Algorithm

By analysing the above observations, we can derive the following algorithm,

1. Initialize:
    - `totalChars` to `0`, which is used to count the number of characters in the input string excluding dash.
    - `count` to `0`, which is used to count the number of characters in the current group.
    - `sizeOfFirstGroup` to be populated which will store the result of `(totalChars % k)`.
    - `ans` to an empty string, which is used to store the final result
2. Now, iterate on the input string:
    - We will skip `'-'` characters from the input string to get the total count of characters in the input string.
    - Fill the first group by only copying `sizeOfFirstGroup` characters in the `ans` string and then break the loop.
    - Return the `ans` string if we reach the end of the loop.
    - Append the `ans` string with `-` in order to form the first group.
    - Continue iterating from the previous `i` till the end of the input string.
    - If the current character is not `'-'`, we include the current character in `ans` string and increment the current group size by incrementing `count` by `1`.
    - If `count` reaches `k`, it means we formed a group of size `k`, thus we can append a `'-'` in `ans` now, and reset `count` to start counting a new group.
3. After we finish traversing on the input string, we return `ans` string.

Implementation

C++ Java C# JavaScript Copy

```javascript
var licenseKeyFormatting = function(s, k) {
        let totalChars = 0;
        for (let i = 0; i < s.length; i++) {
            if (s.charAt(i) != '-') {
                totalChars++;
            }
        }
        let sizeOfFirstGroup = (totalChars % k);
        if (sizeOfFirstGroup == 0) {
            sizeOfFirstGroup = k;
        }
        let ans = "";
        let i = 0;
        let count = 0;

        while (i < s.length) {
            if (count == sizeOfFirstGroup) {
                count = 0;
                break;
            }
            if (s.charAt(i) != '-') {
                count++;
                ans += s.charAt(i).toUpperCase();
            }
            i++;
        }
        /* This case will only appear if value of k is greater than total number
```

Complexity Analysis

Let N be the size of the input array.

- Time Complexity: $O(N)$
    - We traverse on each input string's character once to get the count of `totalChars` which takes $O(N)$ time.
    - We traverse input string for the second time in order to correctly populate `ans` string in groups which again takes $O(N)$ time.
    - Thus, overall we take $O(N)$ time.
- Space Complexity: $O(1)$
    - We are not using any extra space other than the output string.

Share

Comments (2)

Sort by: Best

Type comment here... (Markdown supported)

Preview Comment

Article Commenting Rules

1. This comment section is for questions and comments regarding this **LeetCode article**. All posts must respect our LeetCode Community Rules.

2. Concerns about errors or bugs in the article, problem description, or test cases should be posted on LeetCode Feedback, so that our team can address them.

mkmoisen Jan 01, 2023

Approach 1 has O(N) space, not O(1), because we are building up a string of length N that we return.

Unless we assume that returning memory does not count towards the space complexity? I don't think that is a correct assumption.

4 Show 2 Replies Reply

vinodrammohan Dec 22, 2022

The key to solving this problem within the allotted time is to be able to come up with Approach 1. Though Approach 2 solves the problem it is hard to handle all edge cases, especially during the interview.

0 Reply

1