482. License Key Formatting

Easy String

Leetcode Link

Problem Description

You are provided with a string `s` representing a license key. This string only contains alphanumeric characters (letters and numbers) and dashes `-`. The string is split into `n + 1` segments by `n` dashes. You are also given an integer `k`.

The goal is to reformat the license key in such a way that each segment contains exactly `k` characters, except possibly for the first segment, which can be shorter but must contain at least one character. Additionally, each group must be separated by a single dash, and all lowercase letters should be converted to uppercase.

The task is to transform the string `s` into a new string that meets these requirements and to return the reformatted license key.

Intuition

To solve this problem, the solution has to process the string and reformat it according to the given rules. Here's the step-by-step reasoning behind the solution approach:

1. First, we simplify our given string by removing all the dashes from it. This allows us to start with only alphanumeric characters, making it easier to handle the grouping.
2. We convert all lowercase letters to uppercase since the final output should be in uppercase, as specified in the rules.
3. We need to deal with the requirement that the first group can be shorter than the others. To manage this, we calculate the length of the first group, which is the remainder of the length of the string after removing all the dashes, modulo `k`. If this remainder is zero, it means the string's length is a multiple of `k`, so the first group should also be `k` characters long.
4. We iterate over the characters of the now dash-less and uppercase string, adding each character to a results list.
5. While doing this, we keep count of the number of characters added since the last dash or the start of the string. When this count reaches the length of the current group (which starts as the length of the first group), we add a dash to the results list (except after the final character).
6. After adding a dash, we reset the count and also set the group length to `k` for all groups that will be processed after the first.
7. Once we finish processing all characters, we join the characters in the results list into a single string, which is our reformatted license key.

By following these steps, we arrive at a reformatted string that satisfies all the specified conditions for the license key formatting problem.

Solution Approach

The implementation of the reformatted license key solution uses basic string manipulation techniques and list operations to conform to the desired formatting rules. Here's a breakdown of how the code achieves this:

1. `s = s.replace('-', '').upper()`: The first line of the method removes all dashes from the input string `s` using `replace('-', '')` and converts all lowercase letters to uppercase with `upper()`. This simplifies the string so that we only deal with alphanumeric characters.
2. `res = []`: A new list `res` is initialized to keep track of the characters for the final result. Lists in Python provide efficient append operations which are used later in the code to construct the reformatted string step by step.
3. `cnt = (len(s) % k) or k`: This line sets the initial group's length. If there is a remainder when dividing the string's length by `k`, this remainder will determine the length of the first group; otherwise, the first group will also be `k` characters long. This is handled by using the logical `or` which returns `k` if `len(s) % k` is zero.
4. `t = 0`: A variable `t` is initialized to keep a count of how many characters have been processed since the last dash was added to `res` or since the start.
5. The code then iterates over the characters in the processed string:
   - `for i, c in enumerate(s)`: The `enumerate` function is used to get both the index `i` and the character `c` during iteration.
   - `res.append(c)`: Each character is appended to `res`.
   - `t += 1`: The counter `t` is incremented with each character added.
6. The following conditional block checks if the current group is complete:
   - `if t == cnt`: If `t` equals the current group length `cnt`, it means a group is complete, and a dash should be added, unless it's the last group.
   - Inside the conditional block:
     - `t = 0`: The counter is reset for the next group.
     - `cnt = k`: From now on, every group will have `k` characters as the first group condition has been satisfied.
     - `if i != len(s) - 1`: A dash is appended to `res` only if the current character is not the last one to prevent a trailing dash.
7. Finally, `''.join(res)`: The contents of the list `res` are combined using `join` to form the reformatted license key as a single string.

The `enumerate` function aids in keeping track of both the index and the character simultaneously; conditional statements control the logic for when to add dashes and reset the counter, and the list's property of dynamic resizing helps efficiently build the output string without worrying about preallocating the exact size of the resultant string.

Example Walkthrough

Let's illustrate the solution approach with a small example. Suppose the input string `s` is "2-4A0r7-4k" and the integer `k` is 4.

1. First, we remove all the dashes from `s` and convert it to uppercase: "24A0R74K".
2. Then we initialize an empty list `res` to hold the reformatted characters, and we calculate the length of the first group. The length of "24A0R74K" is 8 and `k` is 4. So, the length for the first group will be `len(s) % k`, which is 0. Since the remainder is 0, our first group will also be 4 characters long (equal to `k`).
3. We start iterating over "24A0R74K". Initialize `t` to 0 for the count of characters.
4. As we iterate, we append each character to `res` and increment `t`.
5. After adding 4 characters ("24A0"), since `t` equals `cnt` (both are 4), we append a dash to `res` given that we are not at the end of the string. We reset `t` to 0 and set `cnt` to `k` for all following groups.
6. We continue appending characters to `res` and once `t` again equals `k`, we append another dash. This process continues until all characters have been processed.
7. We join the elements of `res` with `''` to get our final reformatted string. In this case, "24A0-R74K".

Through these steps, following the solution approach, the input string is reformatted to "24A0-R74K", where each group has exactly `k` characters, except the first one (if needed), and all characters are in uppercase, separated by dashes. This satisfies all the conditions for the problem.

Python Solution

```python
class Solution:
    def licenseKeyFormatting(self, s: str, k: int) -> str:
        # Remove all the hyphens from the string and convert to uppercase.
        s = s.replace('-', '').upper()

        # Initialize an empty list to store the formatted license key.
        formatted_license_key = []

        # Calculate the number of characters before the first dash.
        first_group_length = (len(s) % k) or k

        # Initialize a count variable to keep track of the characters added in the current group.
        count = 0

        # Iterate over the characters in the modified string.
        for index, char in enumerate(s):
            # Append the current character to the formatted_license_key list.
            formatted_license_key.append(char)
            count += 1

            # Check if the current group has reached its required length.
            if count == first_group_length:
                # Reset count for the next group.
                count = 0
                # Update first_group_length to k as all subsequent groups should be of length k.
                first_group_length = k
                # Append a dash if the current character is not the last one.
                if index != len(s) - 1:
                    formatted_license_key.append('-')

        # Join the list into a string and return the formatted license key.
        return ''.join(formatted_license_key)
```

Java Solution

```java
class Solution {
    public String licenseKeyFormatting(String s, int k) {
        // Remove all hyphens and convert to upper case letters
        s = s.replace("-", "").toUpperCase();

        // StringBuilder to hold the formatted license key
        StringBuilder formattedKey = new StringBuilder();

        // Initialize count for tracking group sizes
        int count = 0;

        // Calculate the initial size for the first group if it's not of length k
        int firstGroupLength = s.length() % k;
        if (firstGroupLength == 0) {
            firstGroupLength = k; // If modulus is 0, the first group is of full length k
        }

        // Iterate over each character in the stripped license key
        for (int i = 0; i < s.length(); ++i) {
            // Append the current character to the StringBuilder
            formattedKey.append(s.charAt(i));
            ++count; // Increment the character count for the current group

            // If the current count reaches the firstGroupLength or k,
            // it indicates the end of a group
            if (count == firstGroupLength) {
                count = 0; // Reset the count for the next group
                firstGroupLength = k; // All subsequent groups will be of length k

                // Append a hyphen if this is not the last character
                if (i != s.length() - 1) {
                    formattedKey.append('-');
                }
            }
        }

        // Convert the StringBuilder to a String and return the formatted license key
        return formattedKey.toString();
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    string licenseKeyFormatting(string s, int K) {
        // Remove hyphens and convert characters to uppercase
        string formattedString = "";
        for (char c : s) {
            if (c != '-') { // Skip hyphens
                if ('a' <= c && c <= 'z') { // Convert lowercase to uppercase
                    c -= 'a' - 'A';
                }
                formattedString += c;
            }
        }

        // Calculate the size of the first group of characters
        int firstGroupSize = formattedString.size() % K;
        if (firstGroupSize == 0) {
            firstGroupSize = K; // If the entire string is a multiple of K, use K instead
        }

        // Initialize counter
        int counter = 0;

        // Build the resulting formatted key string
        string result = "";
        for (int i = 0; i < formattedString.size(); ++i) {
            result += formattedString[i]; // Add the next character to the result
            counter++; // Increment the counter

            // Check if we reach the end of a group
            if (counter == firstGroupSize) {
                counter = 0; // Reset the counter for the next group
                firstGroupSize = K; // After the first group, all groups will be of size K
                if (i != formattedString.size() - 1) { // If this isn't the last character
                    result += "-"; // Add a hyphen
                }
            }
        }

        return result; // Return the resulting formatted license key
    }
};
```

Typescript Solution

```typescript
// Function to format the license key
function licenseKeyFormatting(S: string, K: number): string {
    // Remove hyphens and convert characters to uppercase
    let formattedString = S.replace(/-/g, '').toUpperCase();
    for (let c of S) {
        if (c !== '-') { // Skip hyphens
            if ('a' <= c && c <= 'z') { // Convert lowercase to uppercase
                c = c.toUpperCase();
            }
            formattedString += c;
        }
    }

    let formattedStringArray: string = formattedString.split('');

    // Calculate the size of the first group of characters
    let firstGroupSize: number = formattedString.length % K;
    if (firstGroupSize === 0) {
        firstGroupSize = K; // If the entire string is a multiple of K, use K instead
    }

    // Initialize counter
    let counter: number = 0;

    // Build the resulting formatted key string
    let resultArray: string[] = [];
    for (let i = 0; i < formattedString.length; ++i) {
        resultArray.push(formattedString[i]); // Add the next character to the result
        counter++; // Increment the counter

        // Check if we reach the end of a group
        if (counter === firstGroupSize) {
            counter = 0; // Reset the counter for the next group
            firstGroupSize = K; // After the first group, all groups will be of size K
            if (i !== formattedString.length - 1) { // If this isn't the last character
                resultArray.push("-"); // Add a hyphen
            }
        }
    }

    return resultArray.join(''); // Return the resulting formatted license key
}
```

Time and Space Complexity

Time Complexity

The time complexity of the function can be analyzed by looking at the number of operations it performs relative to the size of the input, `n`. The function consists of the following main steps:

1. Removing dashes and converting the string to upper case: Both of these operations (`s.replace('-', '')` and `s.upper()`) iterate over the entire string once, which has a time complexity of `O(N)` where `N` is the length of the string `s`.
2. The loop that builds the final formatted string: This loop goes through each character in the string `s` one time, and the operations inside the loop are constant-time operations. Therefore, this loop also has a time complexity of `O(N)`.

Since both steps are sequentially executed and both have a linear complexity in terms of `N`, the overall time complexity is `O(N) + O(N)`, which simplifies to `O(N)`.

Space Complexity

The space complexity of the function is determined by the amount of additional memory used as the size of the input varies. In this case:

1. The `res` list is the main additional data structure which holds the reformatted license key. At most, this will hold the same number of alphanumeric characters as the original `s` with the addition of the dashes necessary for formatting. In the worst-case scenario, when no dashes are to be removed from the input, the length of `res` would be `len(s) + len(s)//k`, considering an additional dash `len(s)//k` times. Therefore, the space complexity is `O(N + N/k)` which is equivalent to `O(N)` where `N` is the length of the string `s`.
2. The `cnt` and `t` variables are integer counters with constant space, so they contribute `O(1)`.

Combining the above points, the total space complexity would be `O(N)` for the `res` array and `O(1)` for the other variables, leading to an overall space complexity of `O(N)`.