3039. Apply Operations to Make String Empty

Medium Array Hash Table Counting Sorting

Problem Description

The task is to repeatedly delete the first occurrence of each alphabet character in a string `s` from 'a' to 'z', until the string becomes empty. At each stage, only one occurrence of each letter is removed, if that letter is present in the string. This operation is carried out repeatedly. The goal is to find the string `s` right before the last round of the operation is applied. In simpler terms, we want to know what `s` looks like when it contains just enough characters for one last complete pass (removing 'a' to 'z' sequence). For example, if we begin with `s = "aabcbbca"`, after performing these operations consecutively, prior to the final operation that empties it, the string is reduced to `"ba"`.

Intuition

To compute the string right before the last operation, the core idea relies on the observation that only the characters that have the highest frequency will be potentially left before the final removal (because all others will get eliminated in earlier rounds). So our approach involves two main steps:

1. Count the frequency of each character in the string. The character(s) with the maximum frequency will determine the number of operations needed before the string becomes empty, as they define the 'pace' at which the string is reduced through each operation set.
2. For each character that has the maximum frequency, we need to check if its current position in the string corresponds to the last occurrence of that character. If both conditions hold true for a character (maximum frequency and the position is the last occurrence), it will survive until right before the final operation.

To implement this idea:

- Use a data structure like a hash table or Counter (in Python) to record the number of occurrences of each character in the string. Determine the maximum occurrence count `mx`.
- Create another hash table to record the last occurrence index of each character in the string.
- Iterate through the string and for each character, check if the number of occurrences is equal to `mx` and its index is the last occurrence. If so, this character is part of the string right before the last deletion operation.
- Combine all such characters that meet the criteria to form the desired result.

By following this solution approach, we can ensure that only the characters that could possibly remain till the penultimate operation are included in the final string.

Solution Approach

The provided solution uses the `Counter` class from the Python collections module to efficiently count the occurrences of each character in the string. An additional dictionary, `last`, is created to store the index of the last occurrence of each character in the string. This approach leverages hash tables (dictionaries in Python), which offer efficient `O(1)` average time complexity for lookup, insert, and update operations. This is vital for keeping the overall solution efficient.

The implementation steps are as follows:

1. **Count Occurrences**: A Counter object, `cnt`, captures the frequency of each character by iterating over the string once (`O(n)` time complexity, where `n` is the length of `s`).
2. **Find Maximum Frequency**: The `most_common` method of the Counter object is then used to find character frequency, `mx`, that occurs most often (`O(k)` time complexity, where `k` is the number of distinct characters in the string; typically `k <= 26`).
3. **Record Last Index**: A dictionary, `last`, maps each character to the last index at which it appears in the string. This also involves iterating over the string once.
4. **Construct Result String**: Finally, the program iterates through `s` again and includes a character `c` in the result only if `c` meets both of the following conditions:

- `cnt[c] == mx`: The occurrence count of `c` matches the maximum frequency. This ensures we only consider characters that can last until just before the final operation.
- `last[c] == i`: The current index `i` is the last occurrence index of `c`. This condition ensures that for a character to be included in the final string, it must be the last one of its kind within `s`.

These combined conditions ensure we only append to the resulting string those characters that will be removed in the last operation. This guarantees the string constructed will be the exact string available right before the last operation is performed.

By iterating through the characters and checking these conditions, the solution constructs the answer in a single pass, i.e., in `O(n)` time complexity, which makes the overall algorithm run in linear time with respect to the length of the input string.

Example Walkthrough

Let's illustrate the solution approach using the example string `s = "aabcddcbba"`.

1. **Count Occurrences**: We first count the occurrences of each character using a Counter object.
   - `{'a': 3, 'b': 3, 'c': 2, 'd': 2}`
2. **Find Maximum Frequency**: Using the `most_common` method of the Counter object, we find the maximum occurrence frequency, `mx`.
   - `mx = 3` (both 'a' and 'b' occur 3 times)
3. **Record Last Index**: We record the last index where each character appears in the string.
   - `{'a': 8, 'b': 9, 'c': 7, 'd': 4}`
4. **Construct Result String**: We iterate through `s` from left to right, checking if each character meets the conditions to be appended to the result.
   - First 'a' at index 0: `cnt['a'] == mx` is `True`, but `last['a'] == 0` is `False`. We do not include this 'a'.
   - Second 'a' at index 1: `cnt['a'] == mx` is `True`, but `last['a'] == 1` is `False`. We do not include this 'a'.
   - Third 'a' at index 8: `cnt['a'] == mx` is `True` and `last['a'] == 8` is also `True`. We include this 'a'.
   - Apply similar logic to other characters.

Result would include the third 'a' and the second 'b' because those are the last occurrences and they also have the maximum frequency (`mx`). No other character satisfies both conditions, so they will all have been removed before the penultimate operation.

Hence, right before the last round that removes 'a' to 'z', string `s` looks like `"ab"`.

Solution Implementation

Python

```python
from collections import Counter

class Solution:
    def lastNonEmptyString(self, s: str) -> str:
        # Create a counter object to count occurrences of each character in the string.
        char_count = Counter(s)

        # Find the maximum count of any character in the string.
        max_count = char_count.most_common(1)[0][1]

        # Create a dictionary to record the last known index of each character.
        last_index = {char: idx for idx, char in enumerate(s)}

        # Build the result string comprising characters with the maximum count and
        # only include the character if it's the last occurrence in the string.
        result = "".join(char for idx, char in enumerate(s) if char_count[char] == max_count and last_index[char] == idx)

        return result
```

Java

```java
class Solution {
    public String lastNonEmptyString(String s) {
        // Create an array to count occurrences of each letter
        int[] count = new int[26];
        // Create an array to keep track of the last occurrence index of each letter
        int[] lastIndex = new int[26];
        int length = s.length();
        // mx represents the maximum occurrences of any character
        int maxOccurrences = 0;

        // Loop through the string to fill count and lastIndex arrays
        for (int i = 0; i < length; ++i) {
            int charIndex = s.charAt(i) - 'a';
            count[charIndex]++;
            // Update maximum occurrences found so far
            maxOccurrences = Math.max(maxOccurrences, count[charIndex]);
            // Update the last occurrence index of the current character
            lastIndex[charIndex] = i;
        }

        // StringBuilder to construct the final answer
        StringBuilder answer = new StringBuilder();

        // Loop through the string to find out the characters to append to the answer
        for (int i = 0; i < length; ++i) {
            int charIndex = s.charAt(i) - 'a';
            // Include the character if it occurs the maximum number of times
            // and the current index is the last occurrence of that character
            if (count[charIndex] == maxOccurrences && lastIndex[charIndex] == i) {
                answer.append(s.charAt(i));
            }
        }

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

C++

```cpp
class Solution {
public:
    // Function to find the last sequence of max repeated characters
    string lastNonEmptyString(string str) {
        // Array to store the count of each alphabet
        int count[26] = {0};
        // Array to store the index of last occurrence of each alphabet
        int lastOccurrence[26] = {0};
        int stringLength = str.size();
        int maxCount = 0; // Variable to store the maximum count found so far

        // Loop to count occurrences and to find the last position of each character
        for (int i = 0; i < stringLength; ++i) {
            int charIndex = str[i] - 'a'; // Convert character to index (0-25)
            // Update the occurrence count for this character
            maxCount = max(maxCount, ++count[charIndex]);
            // Update the last position of occurrence for this character
            lastOccurrence[charIndex] = i;
        }

        // String to store the answer
        string result;
        // Loop to build the answer string with characters of max count and are at their last occurrence
        for (int i = 0; i < stringLength; ++i) {
            int charIndex = str[i] - 'a';
            // Check if the current character has the max count and it is the last occurrence
            if (count[charIndex] == maxCount && lastOccurrence[charIndex] == i) {
                result.push_back(str[i]);
            }
        }

        // Return the resulting string
        return result;
    }
};
```

TypeScript

```typescript
// Returns the last string comprised of the most frequently occurred character in the input string `s`,
// considering only its last occurrence when multiple characters occur with the same maximum frequency.
function lastNonEmptyString(s: string): string {
    // Initialize an array `count` with 26 zeroes to store the frequency of each lowercase alphabet letter.
    const count: number[] = Array(26).fill(0);

    // Initialize an array `lastIndex` with 26 zeroes to remember the last occurrence index of each letter.
    const lastIndex: number[] = Array(26).fill(0);

    // Get the length of the input string.
    const lengthOfString = s.length;

    // Initialize `maxFrequency` to keep track of the highest frequency of any character in `s`.
    let maxFrequency = 0;

    // Iterate through the input string to populate `count` and `lastIndex` arrays.
    for (let i = 0; i < lengthOfString; ++i) {
        // Calculate the index corresponding to the current character (assuming 'a' to 'z' characters).
        const charIndex = s.charCodeAt(i) - 97;

        // Increment the count for this character and update `maxFrequency` if necessary.
        maxFrequency = Math.max(maxFrequency, ++count[charIndex]);

        // Update the last occurrence index for this character.
        lastIndex[charIndex] = i;
    }

    // Initialize an array `resultStrings` to hold characters that meet the criteria for output.
    const resultStrings: string[] = [];

    // Iterate over the input string to determine the result characters.
    for (let i = 0; i < lengthOfString; ++i) {
        // Calculate the index as before to access the count and last occurrence index.
        const charIndex = s.charCodeAt(i) - 97;

        // Check if the current character's count matches `maxFrequency` and the character is the last occurrence.
        if (count[charIndex] === maxFrequency && lastIndex[charIndex] === i) {
            // If the character meets the criteria, add the character to the result array.
            resultStrings.push(String.fromCharCode(charIndex + 97));
        }
    }

    // Join the array of result characters into a string and return the result.
    return resultStrings.join("");
}
```

```python
from collections import Counter

class Solution:
    def lastNonEmptyString(self, s: str) -> str:
        # Create a counter object to count occurrences of each character in the string.
        char_count = Counter(s)

        # Find the maximum count of any character in the string.
        max_count = char_count.most_common(1)[0][1]

        # Create a dictionary to record the last known index of each character.
        last_index = {char: idx for idx, char in enumerate(s)}

        # Build the result string comprising characters with the maximum count and
        # only include the character if it's the last occurrence in the string.
        result = "".join(char for idx, char in enumerate(s) if char_count[char] == max_count and last_index[char] == idx)

        return result
```

Time and Space Complexity

The time complexity of the provided code is `O(n)` where `n` is the length of the input string `s`. This is because the code iterates over the string multiple times independently: first, to count the occurrences of each character using `Counter`, and then to create a dictionary with the index of the last occurrence of each character. Finally, it iterates over the string again to build the result string.

The space complexity is `O(|Σ|)` where `|Σ|` is the size of the character set which, in the case of lowercase English letters, is 26. The space used by the `Counter` object and the last occurrence dictionary both depend on the number of different characters in the string, not the size of `s` itself.