1592. Rearrange Spaces Between Words

Easy String

Leetcode Link

Problem Description

In this problem, you are given a string `text` which consists of some words separated by some number of spaces. Each word is composed of lowercase English letters. The aim is to rearrange the spaces in such a way that there is an equal number of spaces between every pair of adjacent words, and the number of spaces between words should be as large as possible. If it is not possible to distribute spaces equally between words, any extra spaces should be added to the end of the result string. It's important to note that the resulting string should have the same length as the input string `text`, meaning that no spaces are added or removed beyond the initial allocation in `text`.

Intuition

Upon examining the problem, the first step is to determine how many spaces there are and how many words there are in the input string `text`. This is important because in order to distribute the spaces evenly, you need to know these counts.

The solution involves a few key steps:

- Count the total number of spaces in the string, which can be simply done using the string method `count()`.
- Split the string into words, using the string method `split()`, to find out the number of individual words.

Now, there are two scenarios to consider:

1. If there's only one word in the `text`, you cannot distribute spaces between words, so you just need to append all the spaces at the end of this single word.
2. If there is more than one word, calculate how many spaces should be distributed between each pair of words. This is done by dividing the total number of spaces by one less than the number of words (since spaces go between words). The modulus operator helps calculate the number of extra spaces that cannot be evenly distributed and thus should be placed at the end.

In mathematical terms, if `cnt` is the number of spaces and `m` is the number of gaps between words (one less than the number of words), then each gap will have `cnt // m` spaces, and the end of the string will have `cnt % m` spaces left over.

Finally, the solution will be to join the words with the calculated number of spaces between them and append any remaining spaces at the end.

Solution Approach

The implementation of the solution is straightforward and employs basic string manipulation methods available in Python. The solution does not resort to complex algorithms or data structures, as the problem itself is addressed with simple arithmetic and string operations. Here is the step-by-step breakdown of the solution approach:

1. First, we count the total number of spaces in the input string `text` using the `count()` method. This is stored in the variable `cnt`.

   ```python
   cnt = text.count(' ')
   ```

2. Next, we need to split `text` into words by using the `split()` method which by default splits by whitespace. The resulting list of words is stored in the variable `words`.

   ```python
   words = text.split()
   ```

3. We calculate the number of gaps between the words as one less than the number of words. This is because spaces are only between words and not after the last word (unless there are extra spaces that couldn't be distributed evenly).

   ```python
   m = len(words) - 1
   ```

4. The algorithm considers a special case where there is only one word. If `m` is 0 (i.e., no gaps because there's just one word), then all spaces should be appended to this single word and returned.

   ```python
   if m == 0:
       return words[0] + ' ' * cnt
   ```

5. If there is more than one word, we have to calculate the number of spaces to place between each word. We do this by integer division of the total spaces by the number of gaps `m`. This gives us the evenly distributed spaces between words.

   ```python
   ' ' * (cnt // m)
   ```

6. We also calculate the remainder of the spaces which will be added at the end of the string by using the modulus operator `cnt % m`. This will give us the extra spaces that cannot be evenly divided between words.

   ```python
   ' ' * (cnt % m)
   ```

7. Lastly, we join the words with calculated spaces between them and append any extra spaces at the end to get the final rearranged string. This is done by joining the list of words with the string of spaces corresponding to `cnt // m`, then appending the leftover spaces.

   ```python
   return (' ' * (cnt // m)).join(words) + ' ' * (cnt % m)
   ```

This concise yet effective approach guarantees that spaces are maximized between words, and any that cannot be evenly distributed are placed at the end, adhering to the problem's conditions, while ensuring the resulting string is the same length as the given `text`.

Example Walkthrough

Let's consider the input string `text` as "a b c d ". We can walk through the solution approach step-by-step using this example:

1. First, we count the number of spaces in the input string. The `text` "a b c d " has 6 spaces.

2. Next, we split `text` into words, resulting in `["a", "b", "c", "d"]`. We have 4 words here.

3. Now we calculate the number of gaps between the words. Since we have 4 words, there will be `4 - 1 = 3` gaps between them.

4. Since we have 3 gaps and 6 spaces, and we want the maximum number of evenly distributed spaces between words, we divide the number of spaces by the number of gaps. So, `6 // 3 = 2` spaces should be between each word.

5. We calculate the remainder of the spaces by using the modulus operator. There are no extra spaces since `6 % 3 = 0`.

6. Finally, we join the words with the calculated number of spaces between them, which in this case is 2 spaces. So the final string will be `"a  b  c  d"`, and there are no extra spaces left to append at the end.

By following the approach, we have restructured the original `text` to have an equal number of spaces between each pair of adjacent words and managed to keep the string length unchanged.

Python Solution

```python
class Solution:
    def reorderSpaces(self, text: str) -> str:
        # Count the number of spaces in the input text
        space_count = text.count(' ')

        # Split the text by spaces to get words
        words = text.split()

        # Calculate the number of spaces to be inserted between words
        num_spaces_between_words = len(words) - 1

        # If there's only one word, we append all spaces after the word
        if num_spaces_between_words == 0:
            return words[0] + ' ' * space_count

        # Compute the number of spaces to distribute evenly between words
        spaces_to_distribute = space_count // num_spaces_between_words
        # And compute the remaining spaces to put at the end
        remaining_spaces = space_count % num_spaces_between_words

        # Join the words with evenly distributed spaces and append the remainder at the end
        return (' ' * spaces_to_distribute).join(words) + ' ' * remaining_spaces

```

Java Solution

```java
class Solution {
    public String reorderSpaces(String text) {
        // Count the total spaces in the input text
        int spaceCount = 0;
        for (char c : text.toCharArray()) {
            if (c == ' ') {
                spaceCount++;
            }
        }

        // Split the text into words, ignoring multiple spaces between words
        String[] wordsArray = text.trim().split("\\s+"); // Trim helps to avoid empty strings at the start and end

        // Filter out any empty strings from the words array and add them to a list
        List<String> wordsList = new ArrayList<>();
        for (String word : wordsArray) {
            if (!word.isEmpty()) {
                wordsList.add(word);
            }
        }

        // Determine the number of gaps between words (slots for spaces)
        int numberOfGaps = wordsList.size() - 1;

        // Handle edge case with only one word by appending all spaces at the end
        if (numberOfGaps == 0) {
            return wordsList.get(0) + " ".repeat(spaceCount);
        }

        // Calculate spaces to distribute between words and at the end
        int spacesBetweenWords = spaceCount / numberOfGaps;
        int extraSpacesAtEnd = spaceCount % numberOfGaps;

        // Join the words with evenly distributed spaces
        String evenlySpacedText = String.join(" ".repeat(spacesBetweenWords), wordsList);

        // Add leftover spaces to the end of the text
        evenlySpacedText += " ".repeat(extraSpacesAtEnd);

        // Return the formatted text
        return evenlySpacedText;
    }
}

```

C++ Solution

```cpp
#include <string>
#include <sstream>
#include <vector>

std::string reorderSpaces(std::string text) {
    // Count the total number of spaces in the text
    int spaceCount = 0;
    for (char ch : text) {
        if (ch == ' ') {
            spaceCount++;
        }
    }

    // Split the text into words
    std::vector<std::string> words;
    std::istringstream stream(text);
    std::string word;

    while (stream >> word) {
        words.push_back(word);
    }

    int numWords = words.size();

    // If there is only one word, append all the spaces at the end
    if (numWords == 1) {
        return words[0] + std::string(spaceCount, ' ');
    }

    // Calculate the number of spaces to distribute between words
    int spacesBetweenWords = spaceCount / (numWords - 1);

    // Calculate the number of spaces to be added at the end
    int trailingSpacesCount = spaceCount % (numWords - 1);

    // Construct the string with evenly distributed spaces
    std::string result;
    for (int i = 0; i < numWords; ++i) {
        result += words[i];
        if (i < numWords - 1) { // No need to add spaces after the last word
            result += std::string(spacesBetweenWords, ' ');
        }
    }

    // Append the trailing spaces
    result += std::string(trailingSpacesCount, ' ');

    return result;
}

```

Typescript Solution

```typescript
function reorderSpaces(text: string): string {
    // Count the total number of spaces in the text
    let spaceCount = 0;
    for (const char of text) {
        if (char === ' ') {
            spaceCount++;
        }
    }

    // Split the text into words, ignoring multiple spaces
    const words = text.trim().split(/\s+/);
    const numWords = words.length;

    // If there is only one word, append all the spaces at the end
    if (numWords === 1) {
        return words[0] + ' '.repeat(spaceCount);
    }

    // Calculate the number of spaces to distribute between words
    const spacesBetweenWords = Math.floor(spaceCount / (numWords - 1));

    // Calculate the number of spaces to be appended to the end of the result
    const trailingSpacesCount = spaceCount % (numWords - 1);

    // Join the words with the spaces in-between and append trailing spaces
    return words.join(' '.repeat(spacesBetweenWords)) + ' '.repeat(trailingSpacesCount);
}

```

Time and Space Complexity

The time complexity of the code is `O(n)` where `n` is the length of the input string `text`. This time complexity stems from the two main operations in the function:

1. Counting the number of spaces with `text.count(' ')`, which iterates over each character in the input string once.

2. Splitting the string into words with `text.split()`, which also iterates over each character in the input string to find word boundaries and split the words accordingly.

Each of these operations is linear with respect to the length of the input string, and since they are not nested, the overall time complexity remains linear.

The space complexity of the code is also `O(n)`. This is because the `words` list is created by splitting the input text, and in the worst case (when all characters in the input are words and no multiple consecutive spaces), this list could contain a copy of every character in the input string. Additionally, during the join operation, a new string which is roughly the same size as the input string is created to accommodate the reordered words and spaces.