2109. Adding Spaces to a String

Medium Array String Simulation

Problem Description

The problem presents us with a string `s` that is 0-indexed, which means the first character of the string is considered to be at position 0. We are also given an integer array `spaces` containing indices of the string `s`. We are tasked with modifying the string `s` by adding spaces at specific positions before each character indicated by the `spaces` array. The key here is to insert each space before the character at the given index. For instance, if `s = "ExampleString"` and `spaces = [7]`, we should insert a space before the character at index 7 ('S'), resulting in "Example String".

To solve the problem, we have to iterate through the string, keep track of the indices where spaces need to be added, and make sure spaces are added before the characters at those indices.

Intuition

The intuition behind the solution is to create a new list to store the characters of the modified string including the spaces. We iterate through each character of the original string `s`, checking if the current index matches any element in the `spaces` array. If it does, we append a space to our answer list before appending the current character. The process is repeated for all characters in the original string. Since we're given that `spaces` is 0-indexed and every space is to be added before the character at the given index, we can iterate through `s` and `spaces` concurrently.

To implement this, we maintain two pointers:

1. `i` to iterate through the string `s`.
2. `j` to iterate through the `spaces` array.

Solution Approach

The solution approach involves iterating through the original string `s` while simultaneously keeping track of the space indices using the `spaces` array. The data structure used to store the result is a list, which is chosen due to its dynamic size and ease of insertion.

The algorithm follows these steps:

1. Initialize an empty list called `ans` which will hold the characters of the new string with spaces.
2. Use two pointers `i` to go through each character in the string `s`, and `j` to access elements in the `spaces` array.
3. Iterate over the string `s` with the help of the `enumerate` function which provides the index (`i`) and character (`c`) at each iteration.
4. For each character iteration, check if the `j` th index of the `spaces` array is present (i.e., `j < len(spaces)`) and equals the current string index `i`.
5. If the conditions are met, append a space (`' '`) to the `ans` list, indicating the point where a space needs to be added as per the `spaces` array.
6. Increment the `j` pointer after adding a space to move to the next element of the `spaces` array.
7. Append the current character `c` to the `ans` list.
8. After the iteration is over, join all the characters in the `ans` list to form the modified string with spaces added at the specified indices.
9. Return the joined string as the final result.

This approach ensures that each space is added in the right position before any corresponding character as specified by the indices in `spaces`. The use of a list for `ans` makes appending both spaces and characters straightforward as we iterate through the given string and indices.

The code associated with the approach is given in the reference solution:

```
class Solution:
    def addSpaces(self, s: str, spaces: List[int]) -> str:
        ans = []
        j = 0
        for i, c in enumerate(s):
            if j < len(spaces) and i == spaces[j]:
                ans.append(' ')
                j += 1
            ans.append(c)
        return ''.join(ans)
```

This approach is efficient due to the single pass over the string `s` and the direct mapping from the `spaces` array to the indices in the string.

Example Walkthrough

Let's go through an example to illustrate the solution approach. Consider the following input:

- String `s`: `"leetcode"`
- Spaces array `spaces`: `[4, 7]`

We want to modify the string by adding spaces at indices 4 and 7.

Here is a step-by-step walkthrough of the solution:

1. Initialize an empty list named `ans` for building the result with spaces.
2. Set two pointers, `i` at 0 to iterate through the string `s`, and `j` at 0 to iterate through the `spaces` array.
3. Start iterating over each character of `s`. For this example, let's go through each iteration:
   - i = 0, c = 'l', j = 0 (Space not needed, `ans` becomes: `['l']`)
   - i = 1, c = 'e', j = 0 (Space not needed, `ans` becomes: `['l', 'e']`)
   - i = 2, c = 'e', j = 0 (Space not needed, `ans` becomes: `['l', 'e', 'e']`)
   - i = 3, c = 't', j = 0 (Space not needed, `ans` becomes: `['l', 'e', 'e', 't']`)
   - i = 4, c = 'c', j = 0 (Space needed, `ans` becomes: `['l', 'e', 'e', 't', ' ']`)
     - Increment `j` to 1 since we added a space.
   - i = 4, c = 'c', j = 1 (After space was added, `ans` becomes: `['l', 'e', 'e', 't', ' ', 'c']`)
   - i = 5, c = 'o', j = 1 (Space not needed, `ans` becomes: `['l', 'e', 'e', 't', ' ', 'c', 'o']`)
   - i = 6, c = 'd', j = 1 (Space not needed, `ans` becomes: `['l', 'e', 'e', 't', ' ', 'c', 'o', 'd']`)
   - i = 7, c = 'e', j = 1 (Space needed, `ans` becomes: `['l', 'e', 'e', 't', ' ', 'c', 'o', 'd', ' ']`)
     - Increment `j` to 2 since we added a space.
   - i = 7, c = 'e', j = 2 (After space was added, `ans` becomes: `['l', 'e', 'e', 't', ' ', 'c', 'o', 'd', ' ', 'e']`)
4. After finishing the iteration, `ans` contains all the characters with spaces properly added: `['l', 'e', 'e', 't', ' ', 'c', 'o', 'd', ' ', 'e']`.
5. Convert the `ans` list into a string using `''.join(ans)`, which results in: `"leet code"`.

This is the final modified string, with spaces added at the right indices as per our `spaces` array. It accurately represents the original string with the required spaces inserted, demonstrating the effectiveness of the solution approach.

Solution Implementation

Python

```
class Solution:
    def addSpaces(self, s: str, spaces: List[int]) -> str:
        # Initialize an empty list to build the answer
        answer = []
        # Pointer to track the index within the 'spaces' list
        space_index = 0

        # Iterate over the characters and indices of the input string 's'
        for index, char in enumerate(s):
            # Check if the current index matches the next space position
            # and that we have not used all the provided spaces
            if space_index < len(spaces) and index == spaces[space_index]:
                # If so, append a space to the 'answer' list
                answer.append(' ')
                # Move the pointer to the next space position
                space_index += 1
            # Append the current character to the 'answer' list
            answer.append(char)

        # Join all elements of 'answer' to get the final string with spaces
        return ''.join(answer)
```

Java

```
class Solution {
    // Method that adds spaces into a string at specified indices.
    public String addSpaces(String s, int[] spaces) {
        // StrindBuilder is used for efficient string manipulation
        StringBuilder result = new StringBuilder();

        // Use two pointers: 'i' for string 's', and 'j' for the spaces array
        for (int i = 0, j = 0; i < s.length(); ++i) {
            // Check if we have more spaces to add and if the current position matches the next space position
            if (j < spaces.length && i == spaces[j]) {
                // If so, append a space to the result
                result.append(' ');
                // Move to the next position in the spaces array
                ++j;
            }
            // Append the current character from string 's'
            result.append(s.charAt(i));
        }

        // Return the modified string with spaces added
        return result.toString();
    }
}
```

C++

```
// The Solution class with a single method addSpaces to insert spaces into a string.
class Solution {
public:
    // Given a string `inputString` and a vector `spacePositions` containing positions at which spaces are to be inserted,
    // this function returns the string with spaces added at the specified positions.
    string addSpaces(string inputString, vector<int>& spacePositions) {
        // Initialize an empty string to store the result.
        string result = "";

        // Iterate over each character in `inputString`.
        // `charIndex` is the index of the current character.
        // `spaceIndex` is the index for iterating through `spacePositions`.
        for (int charIndex = 0, spaceIndex = 0; charIndex < inputString.size(); ++charIndex) {
            // If there are still positions left in `spacePositions` to process and
            // the current character index matches the current space position,
            // then add a space to `result` and move to the next space position.
            if (spaceIndex < spacePositions.size() && charIndex == spacePositions[spaceIndex]) {
                result += ' ';
                ++spaceIndex; // Move to the next position for a space.
            }
            // Add the current character from `inputString` to `result`.
            result += inputString[charIndex];
        }

        // Return the resulting string with spaces inserted.
        return result;
    }
};
```

TypeScript

```
/**
 * Inserts spaces into a string based on specified indices.
 *
 * @param {string} str - The original string without spaces.
 * @param {number[]} spaceIndices - An array of indices where spaces should be inserted.
 * @returns {string} - The string with inserted spaces.
 */
function addSpaces(str: string, spaceIndices: number[]): string {
    // Initialize an empty string to build the resulting string with spaces.
    let resultStr = '';

    // Initialize variables to keep track of current indices in the string (strIndex) and
    // space indices array (spaceIndex).
    let strIndex = 0;
    let spaceIndex = 0;

    // Iterate over each character in the original string.
    while (strIndex < str.length) {
        // If we have space indices left to process and the current string index matches
        // the next space index, add a space to the result string.
        if (spaceIndex < spaceIndices.length && strIndex === spaceIndices[spaceIndex]) {
            resultStr += ' ';
            // Move to the next space index after inserting a space.
            spaceIndex++;
        }
        // Add the current character from the original string to the result string.
        resultStr += str[strIndex];
        // Move to the next character index.
        strIndex++;
    }

    // Return the result string with spaces added.
    return resultStr;
}
```

```
class Solution:
    def addSpaces(self, s: str, spaces: List[int]) -> str:
        # Initialize an empty list to build the answer
        answer = []
        # Pointer to track the index within the 'spaces' list
        space_index = 0

        # Iterate over the characters and indices of the input string 's'
        for index, char in enumerate(s):
            # Check if the current index matches the next space position
            # and that we have not used all the provided spaces
            if space_index < len(spaces) and index == spaces[space_index]:
                # If so, append a space to the 'answer' list
                answer.append(' ')
                # Move the pointer to the next space position
                space_index += 1
            # Append the current character to the 'answer' list
            answer.append(char)

        # Join all elements of 'answer' to get the final string with spaces
        return ''.join(answer)
```

Time and Space Complexity

The given Python code snippet adds spaces in the specified indices of a string and is designed to have:

- Time Complexity: The time complexity of the code is `O(n)`, where `n` is the length of the string `s`. The `enumerate` function loops through the characters of the string `s` just once. For each character, the code checks if a space should be inserted before it, which is an `O(1)` operation since it just checks the current index against the current space index in `spaces`. The append operation for a list in Python has an average case time complexity of `O(1)`, so appending either a character or a space doesn't significantly affect the time complexity. `j` increments at most `len(spaces)` times, which is independent of the length of `s`, so it does not change the overall time complexity dominated by the length of `s`.
- Space Complexity: The space complexity is `O(n)`, where `n` is the length of the string `s`. This is because a list `ans` is used to build the output string with spaces. In the worst case, the length of `ans` is equal to the length of `s` plus the number of spaces to insert. Since the number of spaces is at most `n - 1` (a space after every character except the last one), the space complexity remains `O(n)` when considering both the input string and the additional spaces.