1678. Goal Parser Interpretation

Easy String

Leetcode Link

Problem Description

In the given problem, you are provided with a string named `command` which contains certain patterns, including the single character `"G"`, the substring `"()"`, and the substring `"(al)"`. The aim is to convert this string into a new string where each of those patterns is replaced with a specific string as follows:

- `"G"` should remain the same, being interpreted as `"G"`.
- `"()"` should be interpreted (replaced) with `"o"`.
- `"(al)"` should be interpreted (replaced) with `"al"`.

After converting each pattern in the `command` string to its corresponding interpretation, you are required to concatenate them back together in the same order to form the final interpreted string.

Intuition

To visualize the solution to the problem, imagine you are reading the `command` string character by character. There are a few key observations that can help us devise the algorithm:

1. Whenever you encounter the character `"G"`, it's straightforward: you just need to include this character as it is in the output.
2. When you encounter the character `"("`, you need to determine which pattern it represents. This requires looking at the subsequent character: - If the next character is `")"`, then we can infer that the substring `"()"` is complete, and therefore it should be interpreted as `"o"`. - If the next character is not `")"`, then we can assume the next few characters form the pattern `"(al)"`, which should be interpreted as `"al"`.

The provided Python function `interpret` realizes this logic by iterating over each character in the `command` string. An empty list called `ans` is initialized to store each interpreted character or substring. The function examines each character and, based on the rules above, appends the correct string to `ans`.

Eventually, the list `ans` is concatenated into a single output string using the `join` method, which is then returned as the result. This approach is efficient since it avoids creating and concatenating strings within the loop, instead accumulating the final result in a list and joining it all together at the end.

Solution Approach

The solution uses a simple iterative algorithm to traverse the command string and a list to accumulate the results progressively. Here's a step-by-step explanation:

1. **Initializing an Empty List**: The list `ans` is used to accumulate interpreted characters or strings.
2. **Iterating Over the Command**: The command string `command` is iterated character by character using a `for` loop.
3. **Interpreting 'G'**: When a 'G' is encountered, it's directly appended to `ans` because 'G' translates to 'G' without any modification.
4. **Identifying Parens Patterns**: If the character is `'('`, it is a signal that `')'` or `'al)'` will follow.
   - If the next character is `')'`, then `"()"` is recognized, and `'o'` is appended to `ans`.
   - Otherwise, `"(al)"` is recognized. Since the current character is `'('` and following characters will make up the `al)` part, `'al'` is appended to `ans`. This step is facilitated by the ability to look ahead in the string at index `i + 1`.
5. **Ignoring Other Characters**: During the interpretation, any character that is not 'G' or '(' does not trigger any action because they are parts of patterns already addressed when the '(' was encountered.
6. **Building the Final String**: After the loop completes, `ans` contains all the interpreted strings, in order. The `join` function is then used to concatenate these parts into a single string.

Data Structure:

- A list, `ans`, is used to store the translated pieces.

Algorithm:

- Iterative character traversal checks for specific patterns but is optimized to not include redundant checks. For example, no check is made for the closing parenthesis `')'` or the letters of `"al"` after the opening parenthesis `'('` has been encountered and the appropriate interpretation has been made.

Instead of string concatenation within the loop, which can be expensive due to string immutability in Python, appending to a list and then joining is a common pattern used to build strings efficiently.

Example Walkthrough

Let's take a simple example to illustrate how the solution works using the steps outlined in the solution approach.

Example `command` string: `"G()(al)G"`

Expected output: `"GoalG"`

Now, let's walk through the steps of the solution using this example:

1. **Initializing an Empty List (`ans`)**: Create an empty list named `ans` to hold parts of the interpreted string.
2. **Iterating Over the Command:**
   - Start iterating character by character:
     - Index 0: `'G'` is found.
     - Index 1: `'('` is found.
     - Index 2: `')'` is found.
     - Index 3: `'('` is found.
     - Index 4: `'l'` is found.
     - Index 5: `')'` is found.
     - Index 6: `'G'` is found.
3. **Interpreting 'G'**:
   - When we see the first `'G'` at index 0, we append `'G'` to `ans`, which now looks like this: `['G']`.
4. **Identifying Parens Patterns**:
   - At Index 1, `'('` is encountered, so we check the following character:
     - Index 2 shows that the next character is `')'`, so we append `'o'` to `ans`, resulting in: `['G', 'o']`.
   - We move past the `')'`, since it's been interpreted as part of `"()"`.
   - At Index 3, although we encounter an `'a'`, we know it must be part of `"(al)"` since there was no closing parenthesis just before it. This suggests that the whole `"(al)"` pattern was completed in previous steps, so we skip checking characters that are part of identified patterns.
   - At Index 6, we see another `'G'`, and we append it to `ans` to get: `['G', 'o', 'al', 'G']`.
5. **Ignoring Other Characters**:
   - We don't need to worry about characters at indexes 3, 4, and 5 (`'al)'`) because they were already interpreted when we processed the `'('` at index 1.
6. **Building the Final String**:
   - Finally, we join the list `ans` to get `"GoalG"` which is the expected output.

By following the steps of the proposed algorithm, we've translated the `command` string from `"G()(al)G"` to `"GoalG"` using the replacement rules given in the problem description. This example demonstrates how each character in the command string is processed to form the interpreted string using a list to build the result efficiently.

Python Solution

```python
class Solution:
    def interpret(self, command: str) -> str:
        # Initialize an empty list to build the answer string
        answer = []

        # Enumerate through each character in the command string
        for index, char in enumerate(command):
            # If the character 'G' is encountered, append it to the answer list
            if char == 'G':
                answer.append(char)
            # If the character '(' is encountered, check the next character
            elif char == '(':
                # If the next character is ')', it represents 'o', append 'o' to the answer list
                if command[index + 1] == ')':
                    answer.append('o')
                # Otherwise, it's the start of '(al)', append 'al' to the answer list
                else:
                    answer.append('al')

        # Join the elements in the answer list to create the final string
        return ''.join(answer)

```

Java Solution

```java
class Solution {
    public String interpret(String command) {
        // Initialize a StringBuilder to construct the interpreted string
        StringBuilder interpreted = new StringBuilder();

        // Iterate over each character in the input command string
        for (int i = 0; i < command.length(); ++i) {
            // Get the current character in the command string
            char currentChar = command.charAt(i);

            // Check if the current character is 'G'
            if (currentChar == 'G') {
                // Append 'G' to the interpreted string
                interpreted.append(currentChar);
            } else if (currentChar == '(') {
                // Check if the next character is ')', which implies this is an "o"
                if (command.charAt(i + 1) == ')') {
                    interpreted.append("o");
                    ++i; // Increment index to skip the next character as it's already processed
                } else {
                    // If it's not ")", then it must be "(al)" based on problem statement
                    interpreted.append("al");
                    i += 3; // Increment index to skip the next three characters "(al)"
                }
            }
        }

        // Convert StringBuilder to String and return the interpreted string
        return interpreted.toString();
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to interpret the command string.
    string interpret(string command) {
        string answer; // This will hold our interpreted result.

        // Loop through each character in the command string.
        for (int i = 0; i < command.size(); ++i) {
            char c = command[i]; // Current character at position i.

            // If the current character is 'G', add it directly to the answer.
            if (c == 'G') {
                answer += c;
            }
            // If the current character is '(', we need to check the next character.
            else if (c == '(') {
                // Make sure we do not go out of bounds.
                if (i + 1 < command.size()) {
                    // If the next character is ')', it represents "o".
                    if (command[i + 1] == ')') {
                        answer += "o";
                        ++i;  // Skip the next character as we have processed it.
                    }
                    // If the next character is not ')', we assume it's the start of "al".
                    else {
                        answer += "al";
                        i += 3;  // Skip the next three characters ("al").
                    }
                }
            }
            // If we have an unexpected character, continue to the next iteration.
        }

        // Return the interpreted command.
        return answer;
    }
};

```

Typescript Solution

```typescript
// Function to interpret commands replacing "()" with "o" and "(al)" with "al"
function interpret(command: string): string {
  const commandLength = command.length;
  const interpretedCommandArray: string[] = [];

  // Loop through each character in the command string
  for (let index = 0; index < commandLength; index++) {
    const currentChar = command[index];

    // If the character is 'G', just add it to the array
    if (currentChar === 'G') {
      interpretedCommandArray.push(currentChar);
    // Check the character after '(', if it's ')', then it's 'o',
    // otherwise it's the start of "al"
    } else if (currentChar === '(') {
      interpretedCommandArray.push(command[index + 1] === ')' ? 'o' : 'al');
      // If 'al' was added, skip the next three characters 'a', 'l', and ')'
      if (command[index + 1] !== ')') {
        index += 3;
      }
    }
  }

  // Join the array into a string and return it
  return interpretedCommandArray.join('');
}

```

Time and Space Complexity

The time complexity of the code is `O(n)` where `n` is the length of the input string `command`. This is because the function iterates through each character in the string exactly once.

The space complexity of the code is also `O(n)` because it constructs a list `ans` that, in the worst case scenario, contains a separate element for each character in the input string when the input comprises only 'G's.