824. Goat Latin

Easy String

Leetcode Link

Problem Description

The problem provides a playful language transformation challenge based on certain rules similar to Pig Latin, called "Goat Latin". The task involves applying these rules to a given sentence to transform each word and construct a final "Goat Latin" sentence. Here are the rules for this transformation:

1. If a word begins with a vowel (one of 'a', 'e', 'i', 'o', 'u'), attach "ma" to the end of the word. For instance, "apple" turns into "applema".
2. If a word starts with a consonant (a letter not a vowel), remove its first letter, place it at the end of the word, and then append "ma". So "goat" becomes "oatgma".
3. For each word, append a certain number of letter 'a's to its end, equal to the word's position in the sentence (1-indexed). That means, one 'a' for the first word, two 'a's for the second word, and so on.

The output should be the newly formed "Goat Latin" sentence as one string.

Intuition

To solve this problem effectively, the solution should systematically process the given sentence according to the rules. We can iterate through each word and apply the conversion rules step by step while maintaining the order and structure provided by the input sentence.

The intuition behind the implemented solution includes:

- **Splitting the sentence**: Breaking the input sentence into words to handle the conversions individually.
- **Enumerating the words**: Looping through the words with enumeration to keep track of the current word's index for the 'a' appending rule.
- **Characterizing words by their first letters**: Checking if the word starts with a vowel or consonant to apply the respective rule.
- **Appending 'ma' and 'a's**: Transforming each word by appending 'ma' and the appropriate number of 'a's based on the word's index in the sentence.
- **Joining the transformed words**: After processing all the words, the last step is to join them back into a single string with spaces to form the final sentence in "Goat Latin".

This approach allows us to systematically apply the transformation rules to the entire sentence, resulting in a solution that's simple, direct, and easy to comprehend.

Solution Approach

The implementation of the solution for converting a sentence to "Goat Latin" follows a simple yet efficient approach. Here are the steps taken in the code, corresponding to the algorithmic concepts used:

- **Splitting the Sentence**: We begin by splitting the input string `sentence` on spaces, which gives us access to the individual words in the sentence. This is done with `sentence.split()`. The result is a list of words, and this allows us to address each word individually, which is pivotal to applying the transformation rules of "Goat Latin".
- **Processing Each Word**: Using Python's `enumerate` function provides both the index and the value during the iteration, which is essential since the "Goat Latin" transformation rules require knowledge of the word's position in the sentence. This is important for the third rule, where we append an increasing number of 'a's.
- **Performing Conditional Transforms**: Inside the loop, we check the first letter of each word (lowercased to handle both uppercase and lowercase) to determine if it's a vowel or a consonant. This is done with: `word.lower()[0] not in ['a', 'e', 'i', 'o', 'u']`.
  - If the word begins with a consonant, we modify it by reordering the characters: the first letter is moved to the end, followed by appending "ma".
  - If the word begins with a vowel, we directly append "ma" to it. This is reflected by `word += 'ma'`.
- **Appending 'a's Based on Index**: After applying the above conditional transformation to a word, we append a sequence of 'a' characters to the end of the word. The number of 'a's to append is determined by the word's position in the list (1-indexed), which is readily available through the loop counter `i`. This operation is captured by `word += 'a' * (i + 1)`.
- **Aggregating the Results**: As each word is transformed according to the rules of "Goat Latin", it is appended to a list `ans`. This list serves as a collection of the individual transformed words.
- **Forming the Final Sentence**: Once all words have been transformed and collected, the final sentence is formed by joining the list `ans` back into a string with spaces between each word. This operation is performed with `' '.join(ans)`.

By using an array to collect the transformed words and only joining them into a sentence at the end, the solution avoids the inefficient practice of frequently concatenating strings, which is costly in terms of runtime. The use of enumerating words and straightforward string manipulation make the code readable and maintainable.

Example Walkthrough

Consider the following sentence: "I speak Goat Latin".

According to the rules and the intuition mentioned in the problem description, we'll process this sentence as follows:

1. "I" starts with a vowel. According to the first rule, we add "ma" resulting in "Ima". As it's the first word, we append one 'a' to it: "Ima".
2. "speak" starts with a consonant. We move the 's' to the end of the word and add "ma", thus obtaining "peaksma". It's the second word in the sentence, so we append two 'a's: "peaksmaa".
3. "Goat" starts with a consonant. Following the same rule as before, we get "oatGma". It's the third word, and accordingly, we append three 'a's: "oatGmaaaa".
4. "Latin" also starts with a consonant, so we move 'L' to the end, add "ma" and because it's the fourth word, we append four 'a's: "atinLmaaaaa".

The transformed words are then combined into a single "Goat Latin" sentence:

"I speak Goat Latin" -> "Ima peaksmaaa oatGmaaaa atinLmaaaaa".

Now let's put this example into the given markdown template:

```

```

Python Solution

```python
class Solution:
    def toGoatLatin(self, sentence: str) -> str:
        # Initialize an empty list to store the transformed words
        transformed_words = []

        # Split the sentence into words and iterate over them with their indices
        for index, word in enumerate(sentence.split()):
            # Check if the first letter of the word is a vowel
            # Both uppercase and lowercase vowels are checked
            if word.lower()[0] not in 'aeiou':
                # If it's a consonant, we convert the goat-latin word by simply appending 'ma'
                transformed_word = word + 'ma'
            else:
                # If it's a consonant, move the first letter to the end and then append 'ma'
                transformed_word = word[1:] + word[0] + 'ma'

            # Append 'a' repeated 'index+1' times, since index is 0-based we add 1
            transformed_word += 'a' * (index + 1)

            # Append the goat-latin word to the list of transformed words
            transformed_words.append(transformed_word)

        # Join all transformed words into a single string separated by spaces
        goat_latin_sentence = ' '.join(transformed_words)

        # Return the final goat-latin sentence
        return goat_latin_sentence
```

Java Solution

```java
class Solution {
    public String toGoatLatin(String sentence) {
        // Initialize an array list to hold the transformed words.
        List<String> transformedWords = new ArrayList<>();
        // Create a set containing vowels for both lowercase and uppercase for easy checking.
        Set<Character> vowels = new HashSet<>(
            Arrays.asList('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U')
        );
        // 'wordIndex' will be used to add increasing numbers of 'a' at the end of each word.
        int wordIndex = 1;

        // Split the sentence into words and iterate over each word.
        for (String word : sentence.split(" ")) {
            StringBuilder modifiedWord = new StringBuilder();

            // If the first character of the word is not a vowel, rotate the first character to the end.
            if (!vowels.contains(word.charAt(0))) {
                modifiedWord.append(word.substring(1));
                modifiedWord.append(word.charAt(0));
            } else {
                // If the first character is a vowel, leave the word unchanged.
                modifiedWord.append(word);
            }

            // Append "ma" to the end of the word.
            modifiedWord.append("ma");
            // Append 'a's' equal to 'wordIndex' number of times at the end of the word.
            for (int i = 0; i < wordIndex; ++i) {
                modifiedWord.append("a");
            }

            // Increment the word index for the next word.
            ++wordIndex;

            // Add the modified word to the list of transformed words.
            transformedWords.add(modifiedWord.toString());
        }

        // Join the list of transformed words into a single string with spaces and return.
        return String.join(" ", transformedWords);
    }
}
```

C++ Solution

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

/**
 * Transforms a sentence into "Goat Latin".
 *
 * Rules:
 * - If a word begins with a vowel (a, e, i, o, or u), append "ma" to the end of the word.
 * - If a word begins with a consonant, remove the first letter and append it to the end, then add "ma".
 * - Add one letter 'a' to the end of each word per its word index in the sentence, starting with 1.
 *
 * @param sentence The sentence to be transformed.
 * @return The transformed sentence in "Goat Latin".
 */
std::string toGoatLatin(const std::string& sentence) {
    std::istringstream iss(sentence);
    std::string word;
    std::vector<std::string> words;
    int index = 1;

    while (iss >> word) { // Split the sentence into words.
        // Check if the first character is a vowel (case-insensitive).
        if (std::string("aeiouAEIOU").find(word[0]) != std::string::npos) {
            // If the first letter is a vowel, append "ma" to the end of the word.
            word += "ma";
        } else {
            // If the first letter is a consonant, move it to the end and append "ma".
            word = word.substr(1) + word[0] + "ma";
        }

        // Add 'a' characters to the end of the word, repeated according to its word index.
        word += std::string(index, 'a');
        index++;

        // Append the transformed word to the result, followed by a space.
        goatLatinSentence += word + " ";
    }

    if (!goatLatinSentence.empty()) {
        // Remove the trailing space from the final result.
        goatLatinSentence.pop_back();
    }

    return goatLatinSentence;
}

/* Example usage */
int main() {
    std::string goatLatinSentence = toGoatLatin("I speak Goat Latin");
    // Expected output: "Ima peaksmaaa oatGmaaaa atinLmaaaaa"
    std::cout << goatLatinSentence << std::endl;
    return 0;
}
```

Typescript Solution

```typescript
/**
 * Transforms a sentence into "Goat Latin".
 *
 * Rules:
 * - If a word begins with a vowel (a, e, i, o, or u), append "ma" to the end of the word.
 * - If a word begins with a consonant, remove the first letter and append it to the end, then add "ma".
 * - Add one letter 'a' to the end of each word per its word index in the sentence, starting with 1.
 *
 * @param {string} sentence The sentence to be transformed.
 * @returns {string} The sentence transformed into "Goat Latin".
 */
function toGoatLatin(sentence: string): string {
    return sentence
        .split(' ') // Split the sentence into words.
        .map((word, index) => { // Transform each word.
            let transformedWord: string;

            // Check if the first character is a vowel, case-insensitive.
            if (/[aeiouAEIOU]/.test(word[0])) {
                transformedWord = word; // Begin with the word itself if it starts with a vowel.
            } else {
                // Move the first character to the end if it starts with a consonant.
                transformedWord = word.slice(1) + word[0];
            }

            // Append 'ma' and a number of 'a's corresponding to the word's index.
            return `${transformedWord}ma${'a'.repeat(index + 1)}`;
        })
        .join(' '); // Reassemble the sentence.
}

// Example usage:
// const goatLatinSentence = toGoatLatin("I speak Goat Latin");
// The expected output would be "Ima peaksmaaa oatGmaaaa atinLmaaaaa"
```

Time and Space Complexity

The provided Python function `toGoatLatin` converts a sentence into "Goat Latin", which is a playful variation of English made for the purposes of a coding question.

Time Complexity:

The time complexity of this function is $O(nm)$, where n is the number of words in the input sentence, and m is the average length of a word. The reasoning for this is as follows:

- Splitting the sentence into words involves iterating over every character in the string, which takes $O(s)$, where s is the length of the input string. Assuming s is comparable to nm, this component is $O(nm)$.
- The `for` loop runs once for every word, so it iterates n times.
- Inside the loop, checking `if word.lower()[0] not in ['a', 'e', 'i', 'o', 'u']` is $O(1)$ since it's a fixed set of comparisons for the first character only.
- The slicing and concatenation operations (`word[1:] + word[0]`, `word += 'ma'`, and `word += 'a' * (i + 1)`) are where the m factor comes from. These string operations are $O(m)$ since strings are immutable in Python, and creating a new string with the modifications requires $O(m)$ time.
- The final `' '.join(ans)` has to iterate over all the words and the additional characters added with `'ma'` and `'a' * (i + 1)`. The join operation takes $O(n + I[1 to n] of i)$, which simplifies to $O(n^2)$. However, because words can vary in length and this quadratic component is only related to the n characters added, which is small compared to the length of the whole words, this can still be considered $O(nm)$ for large inputs.

Space Complexity:

The time complexity relates to the amount of memory required for processing.

- Memory is required for the list of words produced by `sentence.split()`, which is $O(nm)$ since it stores each word of the sentence.
- Additional space is used to store the transformed words in the `ans` list. Since each word grows linearly with the number of words due to the addition of the 'ma' and 'a' characters, the space complexity due to transformed words is also $O(nm)$.
- Other operations use a constant amount of space and do not depend on the size of the input.

Therefore, the overall space complexity is $O(nm)$, where n is the number of words and m is the average length of the words.