2490. Circular Sentence

Easy String

Problem Description

A sentence is defined as a list of words that are separated by a single space, with no leading or trailing spaces. Words are made up of uppercase and lowercase English letters, and it's important to note that uppercase and lowercase letters are distinct from one another.

A sentence is considered "circular" if it meets two conditions:

1. The last character of a word is the same as the first character of the following word.
2. The last character of the last word is the same as the first character of the first word.

The task is to determine if a given sentence meets these criteria for being a circular sentence. In other words, we need to check if the sentence forms a continuous loop where the end of one word connects to the beginning of the next, and finally, the sentence loops from the end back to the beginning.

Intuition

To determine if a sentence is circular, a straightforward approach is adopted:

1. First, check if the first character of the sentence is the same as the last character. This fulfills the second condition for the sentence to be circular, forming the loop from end to beginning.
2. Then, inspect each space in the sentence to ensure that it is the separator between two words where the last character of the word before the space matches the first character of the word after the space. This is the first condition needed for a circular sentence.
3. Loop through each character in the sentence, and at every space character, compare the characters immediately before and after the space. If they are the same, we proceed; if not, the sentence cannot be circular.
4. If all spaces in the sentence have matching characters before and after, and the first and last characters of the sentence are the same, the sentence is circular. Otherwise, it is not.

Solution Approach

The solution provided uses a simple yet effective approach to determine whether the given sentence is circular. The key aspects of the implementation are:

1. **Checking the first and last characters**: The condition `s[0] == s[-1]` checks whether the first character of the sentence (`s[0]`) is the same as the last character (`s[-1]`). This is essential for the sentence to loop back to the beginning after the last word.
2. **Iterating through the sentence**: We then utilize the `enumerate()` function, which allows us to loop through each character in the sentence while having access to both the character itself (`c`) and its index (`i`).
3. **List comprehension with `all()`**: A list comprehension is used in conjunction with the `all()` function. `all()` will return `True` only if all elements in the iterable are `True`. This is perfect for checking if every space in the sentence is followed by a word that starts with the same character that ends the previous word.
4. **Condition within list comprehension**: The condition `c != " " or s[i - 1] == s[i + 1]` within the list comprehension serves a dual purpose. It checks that:
   - For elements that are not spaces (where `c != " "`), we do not need to perform any action, so these cases should always return `True` by default to pass the `all()` condition.
   - For spaces (which means `c is " "`), it checks if the character before the space (`s[i - 1]`) is the same as the character after the space (`s[i + 1]`), thereby adhering to the rule that the last character of one word must match the first character of the next word.
5. **Combining the conditions**: The above two aspects are combined using an `and` operator to ensure that the sentence is circular only if both conditions are satisfied.

This approach is efficient because it only involves a single traversal of the sentence and does not require any extra data structures, thus keeping the space complexity to O(1). The code simplicity and the absence of nested loops also ensure that the time complexity is kept to O(n), where `n` is the length of the input sentence.

Example Walkthrough

Let's go through an example to illustrate the solution approach.

Consider the sentence: `"cat toy yak kite energy"`

To determine if this sentence is circular according to the description, we need to apply the provided solution.

1. First, we check if the sentence starts and ends with the same character, which are "c" and "y" in this case. Since they are not the same, we could conclude the sentence is not circular. However, for the sake of demonstration, we'll continue with the next steps as if they were the same to illustrate the full process.
2. Now we'll iterate through the sentence using `enumerate()` to examine each character and its position.
3. Our list comprehension will iterate over each character `c` along with its index `i`.
4. For each character that is not a space, we consider it valid as it doesn't contribute to the condition for circularity.
5. For each space character, we check the condition `(s[i - 1] == s[i + 1])`.

- After "cat", the space is followed by "toy". The last character of "cat" is 't' and the first character of "toy" is 't', so this condition is met.
- Following "toy", the space is followed by "yak". The last character of "toy" is 'y' and the first character of "yak" is 'y', so this condition is met as well.
- After "yak", the space is followed by "kite". The last character of "yak" is 'k' and the first character of "kite" is 'k', meeting the condition once again.
- Finally, after "kite", the last character 'e' should match the first character of "energy" 'e', which it does.

6. If the first and last characters matched and all the conditions for spaces were true as per the steps above, we could say the sentence is circular. The code would look like this:

```
sentence = "cat toy yak kite energy"
is_circular = sentence[0] == sentence[-1] and all(c != " " or sentence[i - 1] == sentence[i + 1] for i, c in enumerat
```

Since the first character 'c' doesn't match the last character 'y', `sentence[0] == sentence[-1]` evaluates to `False` and therefore `is_circular` would be `False` even though every other condition is met. As such, the sentence `"cat toy yak kite energy"` is not circular.

This demonstrates how the provided solution approach can be applied to any sentence to efficiently determine its circularity.

Solution Implementation

Python

```python
class Solution:
    def is_circular_sentence(self, sentence: str) -> bool:
        """
        Determine if the given sentence is circular.

        A sentence is considered circular if:
        1. The first and last characters are the same (ignoring spaces), and
        2. For every space in the sentence, the characters immediately before
           and after the space must be the same.

        Parameters:
        sentence (str): The input sentence to be evaluated.

        Returns:
        bool: True if the sentence is circular, False otherwise.
        """

        # Check if the first character (ignoring leading spaces) is the same as the
        # last character (ignoring trailing spaces)
        # and iterate through the sentence checking the condition for circularity
        return sentence[0] == sentence[-1] and all(
            char != " " or sentence[index - 1] == sentence[index + 1]
            for index, char in enumerate(sentence)
        )

# Example usage:
# sol = Solution()
# print(sol.is_circular_sentence("radar"))  # This would return True
# print(sol.is_circular_sentence("hello"))  # This would return False
```

Java

```java
class Solution {

    /**
     * Checks if the given sentence is circular. A sentence is considered circular
     * if the first and last characters are the same and each space is both preceded
     * and succeeded by the same character.
     *
     * @param sentence The input sentence as a string.
     * @return true if the sentence is circular, otherwise false.
     */
    public boolean isCircularSentence(String sentence) {
        int sentenceLength = sentence.length();

        // Check if the first and last characters are the same.
        // If not, the sentence cannot be circular.
        if (sentence.charAt(0) != sentence.charAt(sentenceLength - 1)) {
            return false;
        }

        // Iterate through the characters of the sentence
        for (int i = 1; i < sentenceLength; ++i) {
            // Check if there is a space and if it is not flanked
            // by the same character on both sides. If not, return false.
            if (sentence.charAt(i) == ' ' && sentence.charAt(i - 1) != sentence.charAt(i + 1)) {
                return false;
            }
        }

        // If all checks are passed, the sentence is circular.
        return true;
    }
}
```

C++

```cpp
#include <string> // Include string library to use the string class
using namespace std;

class Solution {
public:
    // Function to check if the sentence is circular.
    // A circular sentence must start and end with the same character, and after every
    // space character, the next character must be the same as the last character before the space.
    bool isCircularSentence(string sentence) {
        int length = sentence.size(); // Get the size of the string

        // Check if the first and last character of the sentence is not the same
        if (sentence[0] != sentence[length - 1]) {
            return false; // If they are not the same, the sentence is not circular
        }

        // Iterate over the string starting from the second character
        for (int i = 1; i < length; ++i) {
            // Check if the current character is a space and the character before it
            // is not the same as the character after it
            if (sentence[i] == ' ' && sentence[i - 1] != sentence[i + 1]) {
                return false; // If they are not the same, the sentence is not circular
            }
        }

        // If all checks pass, the sentence is circular
        return true;
    }
};
```

TypeScript

```typescript
function isCircularSentence(sentence: string): boolean {
    // Get the length of the sentence
    const sentenceLength = sentence.length;

    // Check if the first and last character of the sentence are the same
    if (sentence[0] !== sentence[sentenceLength - 1]) {
        return false; // not circular if they differ
    }

    // Loop through the sentence to check each character
    for (let index = 1; index < sentenceLength; ++index) {
        // Check if current character is a space
        if (sentence[index] === ' ' && sentence[index - 1] !== sentence[index + 1]) {
            // Return false if characters around the space are not the same
            return false;
        }
    }

    // Return true if all checks passed, the sentence is circular
    return true;
}
```

```python
class Solution:
    def is_circular_sentence(self, sentence: str) -> bool:
        """
        Determine if the given sentence is circular.

        A sentence is considered circular if:
        1. The first and last characters are the same (ignoring spaces), and
        2. For every space in the sentence, the characters immediately before
           and after the space must be the same.

        Parameters:
        sentence (str): The input sentence to be evaluated.

        Returns:
        bool: True if the sentence is circular, False otherwise.
        """

        # Check if the first character (ignoring leading spaces) is the same as the
        # last character (ignoring trailing spaces)
        # and iterate through the sentence checking the condition for circularity
        return sentence[0] == sentence[-1] and all(
            char != " " or sentence[index - 1] == sentence[index + 1]
            for index, char in enumerate(sentence)
        )

# Example usage:
# sol = Solution()
# print(sol.is_circular_sentence("radar"))  # This would return True
# print(sol.is_circular_sentence("hello"))  # This would return False
```

Time and Space Complexity

The time complexity of the provided code is `O(n)` where `n` is the length of the string `sentence`. This is because the code iterates through each character in the string exactly once (via the `enumerate` function in the `all()`), and each operation inside the loop is `O(1)`, including checking equality of characters.

The space complexity of the code is `O(1)`. No additional space that scales with the size of the input string is required, as the evaluation is done in-place and the `all()` function does not create a new list or use any additional space that would depend on the length of the input string.