1816. Truncate Sentence

Easy Array String

Problem Description

This problem requires creating a function that takes a sentence 's' as a string, and an integer 'k'. Here, a sentence is defined as a list of words that are separated by a single space without any leading or trailing spaces, and words only consist of uppercase and lowercase English letters. The task is to truncate the sentence so that only the first 'k' words are left in the sentence. The expected return is the truncated sentence.

The function should handle different cases, such as:

- Sentences with exactly 'k' words shouldn't be altered.
- Sentences with more than 'k' words should be cut off right before the (k+1)-th word.
- If a sentence has fewer than 'k' words, it should be returned as is, since there are not enough words to truncate.

The critical aspect of the problem is to figure out how to count the words efficiently and accurately truncate the sentence at the correct position.

Intuition

The solution approach to this problem is quite straightforward and involves a simulation technique. Essentially, the solution simulates the process of reading the sentence word by word until 'k' words are counted, and then it truncates the sentence at that point. The intuition behind this approach is that counting spaces between words can be used to determine the word boundaries in the sentence.

Since we know that words are separated by spaces, we can iterate through the given sentence character by character, checking for spaces. Each time we encounter a space, it indicates that we have passed a word. We decrement 'k' since we want to stop after 'k' words. As long as 'k' is greater than zero, we continue our iteration. When 'k' reaches zero, it means we have encountered 'k' words and we can return the substring of the sentence up to the current character, excluding the current space.

If we finish iterating through the sentence and 'k' has not reached zero, this means the original sentence had fewer than 'k' words, and so we return the original sentence without any truncation.

Solution Approach

The implementation of the given solution follows a simple yet effective approach and uses basic programming constructs rather than complex data structures or patterns. The approach is based on the observation that words in a sentence are delimited by spaces. Thus, the main algorithmic steps are as follows:

1. Initialize a loop that iterates over the characters of the sentence `s`.
2. Check each character to determine if it's a space by using the condition `c == ' '`.
3. Each time a space is encountered, decrement the word count `k` by one, since a space signifies the end of a word.
4. Continuously check if `k` has reached zero within the loop. As soon as `k` is equal to zero, it indicates that the desired `k` words have been counted.
5. When `k` reaches zero, truncate the sentence by returning a substring of `s` from the beginning to the current index `i`, where `i` is the position right before the space that follows the `k-th` word. This is done by the expression `s[:i]`.
6. If `k` does not reach zero by the end of iteration (meaning the sentence has fewer than `k` words), then the whole sentence `s` is returned, as no truncation is needed.

The function doesn't require any additional data structures, as the input string `s` and integer `k` are sufficient to achieve the task. The for-loop and the if-condition inside it together form the core of the algorithm, simulating the reading process of the sentence. The enumeration of `s` is done using Python's `enumerate()` function, which provides a counter `i` along with each character `c`. This way, the current index is always available without the need for an extra counter variable.

In summary, this solution leverages the simple pattern that spaces separate words and iterates through the sentence once (linear time complexity O(n), where n is the number of characters in the sentence), decrementing `k` with each space until it finds the exact point to truncate the sentence, making it an efficient and straightforward approach to the problem.

Example Walkthrough

Consider the following example:

Let's say we have the sentence `s = "Hello LeetCode users are awesome"` and `k = 3`. We want to truncate the sentence such that only the first `k` words remain.

1. Start by examining each character of the sentence. Begin with `H` and increment the index `i` with each character.
2. Proceed until you encounter the first space after `Hello`, which indicates the end of word one. Since `k` is initially 3, after finding the first word we decrement `k` to 2.
3. Continue to the next word `LeetCode` and find the space following it, which signs the end of the second word. We decrement `k` to 1.
4. Move on to the third word `users` and find the space after it. With this, we have encountered three words in total and decrement `k` to 0.
5. Now that `k` has reached 0, we truncate the sentence at the current index, which is right before `are`. The index `i` at this point is the character space after `users`.
6. Return the substring from the start up to but not including this space: `s[:i]` which gives us `"Hello LeetCode users"` as the expected truncated sentence.

The process showcases the simulation approach in action and illustrates how the solution effectively counts words separated by spaces, truncates the sentence after the `k-th` word, and handles sentences regardless of their initial word count in relation to `k`.

Solution Implementation

Python

```python
class Solution:
    # Function to truncate the sentence up to the k-th space (word).
    def truncateSentence(self, sentence: str, k: int) -> str:
        # Iterate over each character in the sentence with its index.
        for index, char in enumerate(sentence):
            # Subtract from 'k' if the character is a space.
            k -= char == ' '
            # If k reaches 0, we've found the k-th space.
            if k == 0:
                # Return the substring of sentence up to the current index.
                return sentence[:index]
        # If the loop completes, there were fewer than k spaces, so return the entire sentence.
        return sentence

# Example usage:
# sol = Solution()
# truncated = sol.truncateSentence("Hello how are you Contestant", 4)
# print(truncated)  # Output: "Hello how are you"
```

Java

```java
class Solution {

    /**
     * Truncates a sentence to the first k words.
     *
     * @param sentence The original sentence.
     * @param wordCount The number of words to truncate the sentence to.
     * @return The truncated sentence.
     */
    public String truncateSentence(String sentence, int wordCount) {
        // Iterate over each character in the sentence
        for (int i = 0; i < sentence.length(); ++i) {
            // Check if the current character is a space, indicating a word boundary
            if (sentence.charAt(i) == ' ') {
                // Decrement the word count and check if we've hit the target word count
                if (--wordCount == 0) {
                    // Return the substring from the beginning to the word boundary
                    return sentence.substring(0, i);
                }
            }
        }
        // If the loop completes without returning, it means the sentence has fewer
        // or equal words than k, so return the original sentence as-is
        return sentence;
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to truncate a sentence to the first k words
    string truncateSentence(string sentence, int wordCount) {
        // Loop through each character in the sentence
        for (int i = 0; i < sentence.size(); ++i) {

            // Check if the current character is a space
            if (sentence[i] == ' ') {

                // Decrement the word count, and if it reaches zero, we've found the k-th word
                --wordCount;

                // If there are no more words left, return the substring from the beginning to this point
                if (wordCount == 0) {
                    return sentence.substr(0, i);
                }
            }
        }

        // If we never returned from the loop, we didn't reach k words, return the entire sentence
        return sentence;
    }
};
```

TypeScript

```typescript
/**
 * This function truncates a sentence after a given number of words.
 *
 * @param {string} sentence - The sentence to be truncated.
 * @param {number} wordCount - The number of words to keep in the truncated sentence.
 * @return {string} - The truncated sentence.
 */
function truncateSentence(sentence: string, wordCount: number): string {
    // Iterate through each character in the sentence
    for (let index = 0; index < sentence.length; ++index) {
        // Check if the current character is a space and decrement the word count
        if (sentence[index] === ' ' && --wordCount === 0) {
            // If the word count reaches zero, return the substring from start to current index
            return sentence.slice(0, index);
        }
    }
    // If the sentence has less words than the wordCount, return the original sentence
    return sentence;
}
```


```python
class Solution:
    # Function to truncate the sentence up to the k-th space (word).
    def truncateSentence(self, sentence: str, k: int) -> str:
        # Iterate over each character in the sentence with its index.
        for index, char in enumerate(sentence):
            # Subtract from 'k' if the character is a space.
            k -= char == ' '
            # If k reaches 0, we've found the k-th space.
            if k == 0:
                # Return the substring of sentence up to the current index.
                return sentence[:index]
        # If the loop completes, there were fewer than k spaces, so return the entire sentence.
        return sentence

# Example usage:
# sol = Solution()
# truncated = sol.truncateSentence("Hello how are you Contestant", 4)
# print(truncated)  # Output: "Hello how are you"
```


Time and Space Complexity

The time complexity of the code can be determined by analyzing the loop that iterates over each character of the string `s`. Since the loop goes through all characters up to `k` spaces or up to the end of the string, whichever comes first, the worst-case scenario is when there are no spaces in the first `k` characters or when `k` is larger than the total number of words. In either case, the loop will go through the entire string once, making the time complexity `O(n)`, where `n` is the length of the string `s`.

Regarding space complexity, the algorithm uses a fixed number of variables (`i`, `k`, and `c`) which occupy constant space. The output is a substring of the input string; therefore, the space taken by the input is not considered additional space used by the algorithm. Thus, the space complexity is `O(1)`, assuming the input string `s` is already given and does not count towards the space complexity of the algorithm. This reflects that apart from the input, the algorithm consumes a constant amount of space.