Sentence Similarity III

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	1813
tags	Streams String Manipulation
language	C++
solved on	@06/10/2024
link	https://leetcode.com/problems/sentence-similarity-iii/description/
Completion	☑

Intuition

The problem can be solved by breaking the sentences into words and then comparing them from both ends (left to right and right to left) to find if sentence1 is a subsequence of sentence2 or vice versa.

Approach

1. Split both sentences into words.
2. Ensure that `sentence1` has fewer or equal words compared to `sentence2` by swapping them if necessary.
3. Compare the words starting from the beginning of both lists.
4. Compare the words starting from the end of both lists.
5. If all words of the smaller sentence match the corresponding words of the larger sentence in either the left or right segments, the sentences are considered similar.

Complexity

Time Complexity:

- Splitting the sentences into words takes **O(n + m)**, where **n** is the length of `sentence1` and **m** is the length of `sentence2`.
- Comparing the words from both ends takes **O(min(k1, k2))**, where **k1** and **k2** are the number of words in `sentence1` and `sentence2`, respectively.

Thus, the overall time complexity is **O(n + m)**, as splitting the sentences dominates the word comparison.

Space Complexity:

- The space complexity is **O(k1 + k2)** for storing the split words from both sentences, where **k1** and **k2** are the number of words in `sentence1` and `sentence2`, respectively.

Code

```
class Solution {
public:
    vector<string> splitIntoWords(string& sentence) {
```

```
        stringstream stream(sentence);
        vector<string> words;
        string word;
        while (stream >> word) {
            words.push_back(word);
        }
        return words;
    }

    bool areSentencesSimilar(string sentence1, string sentence2) {
        vector<string> words1 = splitIntoWords(sentence1);
        vector<string> words2 = splitIntoWords(sentence2);

        int size1 = words1.size();
        int size2 = words2.size();

        if (size1 > size2) {
            swap(size1, size2);
            swap(words1, words2);
        }

        int leftIndex = 0;
        int rightIndex1 = size1 - 1;
        int rightIndex2 = size2 - 1;

        while (leftIndex < size1 && words1[leftIndex] == words2[leftIndex]) {
            leftIndex++;
        }

        while (rightIndex1 >= 0 && words1[rightIndex1] == words2[rightIndex2]) {
            rightIndex1--;
            rightIndex2--;
        }

        return rightIndex1 < leftIndex;
    }
};
```