1813. Sentence Similarity III

You are given two strings `sentence1` and `sentence2`, each representing a **sentence** composed of words. A sentence is a list of **words** that are separated by a **single** space with no leading or trailing spaces. Each word consists of only uppercase and lowercase English characters.

Two sentences `s1` and `s2` are considered **similar** if it is possible to insert an arbitrary sentence (*possibly empty*) inside one of these sentences such that the two sentences become equal. **Note** that the inserted sentence must be separated from existing words by spaces.

For example,

- `s1 = "Hello Jane"` and `s2 = "Hello my name is Jane"` can be made equal by inserting `"my name is"` between `"Hello"` and `"Jane"` `in s1.`
- `s1 = "Frog cool"` and `s2 = "Frogs are cool"` are **not** similar, since although there is a sentence `"s are"` inserted into `s1`, it is not separated from `"Frog"` by a space.

Given two sentences `sentence1` and `sentence2`,
return **true** if `sentence1` and `sentence2` are **similar**. Otherwise, return **false**.

Example 1:

Input: sentence1 = "My name is Haley", sentence2 = "My Haley"

Output: true

Explanation:

`sentence2` can be turned to `sentence1` by inserting "name is" between "My" and "Haley".

Example 2:

Input: sentence1 = "of", sentence2 = "A lot of words"

Output: false

Explanation:

No single sentence can be inserted inside one of the sentences to make it equal to the other.

Example 3:

Input: sentence1 = "Eating right now", sentence2 = "Eating"

Output: true

Explanation:

`sentence2` can be turned to `sentence1` by inserting "right now" at the end of the sentence.

Constraints:

- `1 <= sentence1.length, sentence2.length <= 100`
- `sentence1` and `sentence2` consist of lowercase and uppercase English letters and spaces.
- The words in `sentence1` and `sentence2` are separated by a single space.

1813. Sentence Similarity III

```
/*
   Arrays (two pointers)
   Time complexity: O(n+m)
   Space complexity: O(#words in sentence1+#words in sentence2)
*/
class Solution {
   public:
      void string2vector(std::string s, std::vector<std::string>& v){
         std::stringstream ss(s);
         std::string word;
         while(ss>>word) v.push_back(word);
      }

      bool areSentencesSimilar(string sentence1, string sentence2) {
         // To be sure that sentence1 is longer than sentence2
         if(sentence1.size()<sentence2.size()) swap(sentence1,sentence2);

         // Put words of each sentence in an array
         std::vector<std::string> v1,v2;
         string2vector(sentence1,v1);
         string2vector(sentence2,v2);

         int n=v1.size();
         int m=v2.size();

         // Check similarity
         int i=0,j=n-1; // sentence1 (v1)
         int k=0,l=m-1; // sentence2 (v2)
         while(k<m && v1[i]==v2[k]){
            i++;
            k++;
         }
         while(l>=0 && v1[j]==v2[l]){
            j--;
            l--;
         }

         return l<k;
      }
};
```

1813. Sentence Similarity III

```
/*
   Deque
   Time complexity: O(n+m)
   Space complexity: O(#words in sentence1+#words in sentence2)
*/
class Solution {
   public:
      void string2deque(std::string s, std::deque<std::string>& q){
         std::stringstream ss(s);
         std::string word;
         while(ss>>word) q.push_back(word);
      }

      bool areSentencesSimilar(string sentence1, string sentence2) {
         // To be sure that sentence1 is longer than sentence2
         if(sentence1.size()<sentence2.size()) swap(sentence1,sentence2);

         // Put words of each sentence in an deque
         std::deque<std::string> q1,q2;
         string2deque(sentence1,q1);
         string2deque(sentence2,q2);

         // Check
         while(!q2.empty() && q1.front()==q2.front()){
            q1.pop_front();
            q2.pop_front();
         }
         while(!q2.empty() && q1.back()==q2.back()){
            q1.pop_back();
            q2.pop_back();
         }

         return q2.empty();
      }
};
```