Scramble String Problem Documentation

Description

We can scramble a string s to get a string t using the following algorithm:

1. If the length of the string is 1, stop.
2. *If the length of the string is greater than 1, perform the following steps:*
    - Split the string into two non empty substrings at a random index. For a string s, divide it into x and y where s = x + y.
    - Randomly decide whether to swap the two substrings or to keep them in the same order. After this step, s may become s = x + y or s = y + x.
    - Apply step 2 recursively on each of the two substrings x and y.

Given two strings s1 and s2 of the same length, return true if s2 is a scrambled string of s1, otherwise return false.

Example 1

Input: s1 = "great", s2 = "rgeat"

Output: true

Explanation:

- *One possible scenario applied on s1 is:*
- "great" > "gr/eat" (divide at random index)
- "gr/eat" > "gr/eat" (random decision is not to swap the two substrings and keep them in order)
- "gr/eat" > "g/r / e/at" (apply the same algorithm recursively on both substrings, divide at random index each of them)

- "g/r / e/at" > "r/g / e/at" (random decision was to swap the first substring and to keep the second substring in the same order)
- "r/g / e/at" > "r/g / e/ a/t" (apply the algorithm recursively, divide "at" to "a/t")
- "r/g / e/ a/t" > "r/g / e/ a/t" (random decision is to keep both substrings in the same order)
- The algorithm stops now, and the resulting string is "rgeat" which is s2.
- Since one possible scenario led s1 to be scrambled to s2, we return true.

Example 2

Input: s1 = "abcde", s2 = "caebd"

Output: false

Example 3

Input: s1 = "a", s2 = "a"

Output: true

Constraints

- s1.length == s2.length
- 1 <= s1.length <= 30
- s1 and s2 consist of lowercase English letters.

Approach

The problem can be solved using a recursive approach with memoization to avoid redundant calculations. The key steps are:

1. **Base Case:**

- If the substrings of s1 and s2 being compared are equal, return true.

2. **Sorting Check:**

- If the sorted versions of the substrings are not equal, they cannot be scrambles of each other, so return false.

3. **Recursive Check:**

- Try splitting the substrings at every possible index and recursively check:
- Without swapping the parts.
- With swapping the parts.

Algorithm

1. Use a helper function dfs(i1, i2, length) to check if s1[i1:i1+length] is a scrambled string of s2[i2:i2+length].
2. **Base cases:**

- If the substrings are equal, return true.
- If the sorted substrings are not equal, return false.

3. **For each possible split index k from 1 to length 1:**

- Check the two cases: without swapping and with swapping.
- If either case returns true, return true.

4. If no valid split leads to a match, return false.

Explanation

- The dfs function is a depth first search that checks if the substring s1[i1:i1+length] is a scramble of s2[i2:i2+length].
- Memoization is used to store results of previously computed states to avoid redundant calculations.
- The base cases check for direct equality and sorted equality of substrings.
- The recursive part splits the substrings and checks both with and without swapping the parts.
- The outer function isScramble initiates the recursive check from the start of both strings.

This approach ensures that all possible scenarios are checked efficiently, leading to the correct result.