Scramble String

Given a string *s1*, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of *s1* = `"great"`:

```
    great
   /    \
  gr    eat
 / \    /  \
g   r  e   at
           / \
          a   t
```

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node `"gr"` and swap its two children, it produces a scrambled string `"rgeat"`.

```
    rgeat
   /    \
  rg    eat
 / \    /  \
r   g  e   at
           / \
          a   t
```

We say that `"rgeat"` is a scrambled string of `"great"`.

Similarly, if we continue to swap the children of nodes `"eat"` and `"at"`, it produces a scrambled string `"rgtae"`.

```
    rgtae
   /    \
  rg    tae
 / \    /  \
r   g  ta   e
       / \
      t   a
```

We say that `"rgtae"` is a scrambled string of `"great"`.

Given two strings *s1* and *s2* of the same length, determine if *s2* is a scrambled string of *s1*.

Solution 1

Assume the strings are all lower case letters

```
class Solution {
public:
    bool isScramble(string s1, string s2) {
        if(s1==s2)
            return true;

        int len = s1.length();
        int count[26] = {0};
        for(int i=0; i<len; i++)
        {
            count[s1[i]-'a']++;
            count[s2[i]-'a']--;
        }

        for(int i=0; i<26; i++)
        {
            if(count[i]!=0)
                return false;
        }

        for(int i=1; i<=len-1; i++)
        {
            if( isScramble(s1.substr(0,i), s2.substr(0,i)) && isScramble(s1.subst
r(i), s2.substr(i)))
                return true;
            if( isScramble(s1.substr(0,i), s2.substr(len-i)) && isScramble(s1.sub
str(i), s2.substr(0,len-i)))
                return true;
        }
        return false;
    }
};
```

written by raychan original link here

Solution 2

The example shows the case where left child ALWAYS has equal or one-less characters than right child. But since "abb" is a scramble of "bab", as suggested by a test case, strings are not always partitioned in the way as the example implies.

However, if the answer is Yes, I think scrambles just become permutations. Isn't it?

So I am so confused what is expected...

Thanks!

written by diaz900 original link here

Solution 3

```
public class Solution {
    public boolean isScramble(String s1, String s2) {
        if (s1.equals(s2)) return true;

        int[] letters = new int[26];
        for (int i=0; i<s1.length(); i++) {
            letters[s1.charAt(i)-'a']++;
            letters[s2.charAt(i)-'a']--;
        }
        for (int i=0; i<26; i++) if (letters[i]!=0) return false;

        for (int i=1; i<s1.length(); i++) {
            if (isScramble(s1.substring(0,i), s2.substring(0,i))
             && isScramble(s1.substring(i), s2.substring(i))) return true;
            if (isScramble(s1.substring(0,i), s2.substring(s2.length()-i))
             && isScramble(s1.substring(i), s2.substring(0,s2.length()-i))) retur
n true;
        }
        return false;
    }
}
```

written by baifriend original link here