The Fake Geek's blog

Saturday, December 13, 2014

Scramble String

A very interesting problem. I like this recursion solution, very neat. Remember to check every boundary cases in order to reduce recursions

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*.

Update a DP solution. Consider a simple case "gre". The string may be split to "g" and "re", or "gr" and "e". So a scrambled string of "gre" can be "ger", "rge", "reg", "egr", "erg" or itself ("gre"). The DP solution uses an 3D matrix, scramble[k][i][j], the first dimension indicates the length of the substring, and the second and third dimension indicate the start index of first and second string, respectively (s1.substring(i, i + k) and s2.substring(j, j + k)). So similar to the recursion solution, we check every substring of s1 and s2 and construct the matrix.

```
public boolean isScramble(String s1, String s2) {
        if (s1.length() != s2.length())
            return false;
        if (s1.length() == 0 || s1.equals(s2))
            return true;

        int length = s1.length();
        boolean[][][] scramble = new boolean[length][length][length];

        for (int i = 0; i < length; i++) {
            for (int j = 0; j < length; j++) {
                scramble[0][i][j] = s1.charAt(i) == s2.charAt(j) ? true : false;
            }
        }

        for (int len = 2; len <= length; len++) { //the length of substring
            for (int i = 0; i <= length - len; i++) {
                for (int j = 0; j <= length - len; j++) {
                    boolean r = false;
                    for (int k = 1; k < len && r == false; k++) {
                        r = (scramble[k - 1][i][j] && scramble[len - k - 1][i + k][j + k
                        || (scramble[k - 1][i][j + len - k] && scramble[len - k - 1][i
                    }
                    scramble[len - 1][i][j] = r;
                }
            }
        }
        return scramble[length - 1][0][0];
    }
```

```
public class ScrambleString {
    public boolean isScramble(String s1, String s2) {
        if ((s1 == null && s2 != null) || (s1 != null && s2 == null))
            return false;
        if (s1.length() != s2.length())
            return false;
        if (s1.equals(s2))
            return true;
        int length = s1.length();
        int chars1 = 0;
        int chars2 = 0;

        // check if two strings are consisted by same characters
        for (int i = 0; i < length; i++)
        {
            chars1 += Character.getNumericValue(s1.charAt(i));
            chars2 += Character.getNumericValue(s2.charAt(i));
        }
        if (chars1 != chars2)
            return false;

        if (s1.length() == 1)
            return true;
        //the string must be swapped at certain position assume s2 is a scramble string
        //iterate through all positions and recursively check
        for (int i = 1; i < length; i++)
        {
            if (isScramble(s1.substring(0,i), s2.substring(0,i)) && isScramble(s1.substr
                return true;
            // if the string is reversed, the reversed one is also a scramble string
            if (isScramble(s1.substring(0,i), s2.substring(length - i)) && isScramble(s1
                return true;
        }
        return false;
    }
}
```

Update: 2015 - 01 -18
I did a performance test on "great" and "rgtae" for both methods:

```
****DP****
true
Running time: 161ms
Used memory in bytes: 507712
****Recursion****
true
Running time: 75ms
Used memory in bytes: 494824
```

Yeah, recursion is preferred. :)

Posted by Unknown at 1:33 PM

Labels: Dynamic Programming, Google, LeetCode, Strings

2 comments:

Unknown October 6, 2019 at 7:30 AM

why is the anagram test necessary??
shouldnot it be taken care of by the remaining code.

Reply

Unknown July 24, 2020 at 12:25 PM

Found a really good explanation of this question
https://youtu.be/uqRrb4t_ktk

Reply

About Me

Unknown
View my complete profile

Translate

Select Language
Powered by Google Translate

Blog Archive

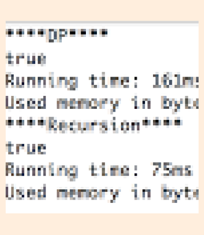
Popular Posts

Scala note 2: Functional Sets
Mathematically, we call the function which takes an integer as argument and which returns a boolean indicating whether the given integer be...


Scala note 4: Huffman Coding
This post is based on Coursera's Scala course homework for week 4 and week 5. The whole problem can be found here . Some notes: Ca...


Scala note 1: Pascal's Triangle, Parentheses Balancing and Counting Change
I decided to take a look on Scala, so that next time I read a source code, at least I know what I am reading. I choose Coursera's Func...

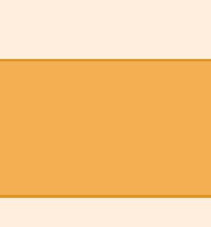
Natural Language Processing: the IMDB movie reviews
Natural language processing (NLP) relates to problems dealing with text problems, usually based on machine learning algorithms. Many machi...


Trapping Rain Water II
Given an m x n matrix of positive integers representing the height of each unit cell in a 2D elevation map, compute the volume of water i...

Longest Repeating Character Replacement
Given a string that consists of only uppercase English letters, you can replace any letter in the string with another letter at most k ti...

Scramble String
A very interesting problem. I like this recursion solution, very neat. Remember to check every boundary cases in order to reduce recursions ...

Split Array Largest Sum
Given an array which consists of non-negative integers and an integer m , you can split the array into m non-empty continuous subarrays...


Scala note 6: Bloxorz
Notes: val, lazy val and def: def expr = { val x = { print("x"); 1} lazy val y = { print("y"); 2} def z = ...

B-Tree Java implementation
Two days of coding. One afternoon to build a tree, three nights to destroy the tree in different ways. I finally make the code. Sort of.....