

# 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  $s_1$ , we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of  $s_1 = \text{"great"}$ :

```
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  $\text{"gr"}$  and swap its two children, it produces a scrambled string  $\text{"rgeat"}$ .

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

We say that  $\text{"rgeat"}$  is a scrambled string of  $\text{"great"}$ .

Similarly, if we continue to swap the children of nodes  $\text{"eat"}$  and  $\text{"at"}$ , it produces a scrambled string  $\text{"rgtae"}$ .

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

We say that  $\text{"rgtae"}$  is a scrambled string of  $\text{"great"}$ .

Given two strings  $s_1$  and  $s_2$  of the same length, determine if  $s_2$  is a scrambled string of  $s_1$ .

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", "erg" or itself ("gre"). The DP solution uses a 3D matrix,  $\text{scramble}[k][i][j]$ , 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 ( $s_1.substring(i + k)$  and  $s_2.substring(j + k)$ ). So similar to the recursion solution, we check every substring of  $s_1$  and  $s_2$  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 + k - 1][j]));
                }
                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.substring(i, length), s2.substring(i, length)))
                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.substring(i, length), s2.substring(0, length - i)))
                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??  
shouldn't 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/uqRb4t\\_ktk](https://youtu.be/uqRb4t_ktk)

Reply

Newer Post

Home

Older Post

Subscribe to: Post Comments (Atom)

## About Me

Unknown

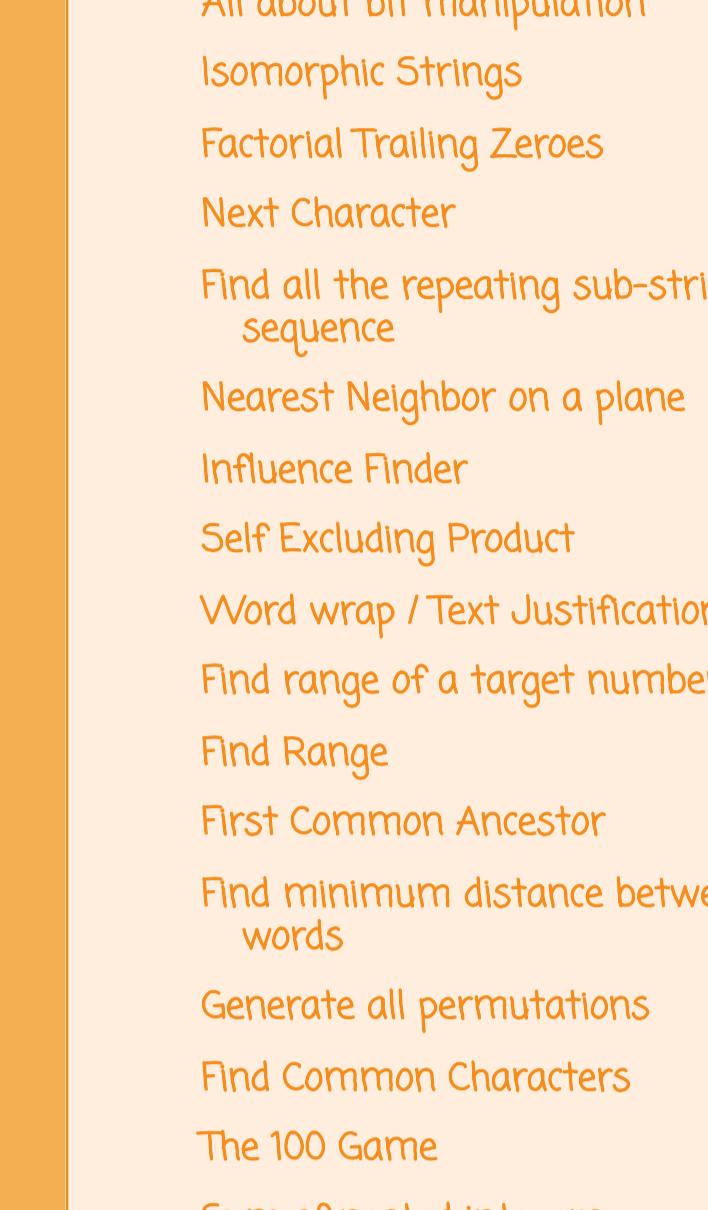
[View my complete profile](#)

## Translate

Select Language

Powered by [Google Translate](#)

## AdSense



## Blog Archive

- 2017 (1)
- 2016 (183)
- 2015 (189)
- ▼ 2014 (101)
  - ▼ December (94)
    - [Binary Search Tree Iterator](#)
    - [All about bit manipulation](#)
    - [Isomorphic Strings](#)
    - [Factorial Trailing Zeros](#)
    - [Next Character](#)
    - [Find all the repeating sub-string sequence](#)
    - [Nearest Neighbor on a plane](#)
    - [Influence Finder](#)
    - [Self Excluding Product](#)
    - [WVord wrap / Text Justification](#)
    - [Find range of a target number](#)
    - [Find Range](#)
    - [First Common Ancestor](#)
    - [Find minimum distance between words](#)
    - [Generate all permutations](#)
    - [Find Common Characters](#)
    - [The 100 Game](#)
    - [Sum of nested integers](#)
    - [Longest palindromic subsequence of an array](#)
    - [Longest Palindromic Substring - DP & O\(n\) solution](#)
    - [Writing Comparators](#)
    - [Find pythagorean triples](#)
    - [Hash table Java implementation](#)
    - [One Edit Distance & Edit Distance](#)
    - [Missing Ranges](#)
    - [Two Sum with sorted and unsorted input array](#)
    - [Excel Sheet Column Title](#)
    - [Majority Element I & II](#)
    - [Longest Valid Parentheses](#)
    - [First Missing Positive](#)
    - [Next Permutation](#)
    - [Maximum Subarray, the Divide and Conquer method](#)
    - [Valid number](#)
    - [Spiral Matrix](#)
    - [Permutation Sequence](#)
    - [Rotate List](#)
    - [Minimum Window Substring](#)
    - [Maximal Rectangle](#)
    - [Trapping Rain Water](#)
    - [2014, Another Year, ends, life is, not OK](#)
    - [Largest Rectangle in Histogram](#)
    - [Interleaving String](#)
    - [Recover Binary Search Tree](#)
    - [Longest Substring with At Most Two Distinct Charac...](#)
    - [Pow \(x, n\)](#)
    - [Sqrtn\(\)](#)
    - [Fraction to Recurring Decimal](#)
    - [Convert Sorted Array / Linked List to BST](#)
    - [Flatten Binary Tree to Linked List](#)
    - [Distinct Subsequences](#)
    - [Populating Next Right Pointers in Each Node I & II](#)
    - [Surrounded Regions](#)
    - [WVord Ladder I & II](#)
    - [Best Time to Buy and Sell Stock](#)
    - [Sum Root to Leaf Numbers](#)
    - [Compare Version Numbers](#)
    - [Longest Consecutive Sequence](#)
    - [Hash map for deep copy](#)
    - [Palindrome Partitioning I & II](#)
    - [Gas Station](#)
    - [Single number I & II & III](#)
    - [N-Queens](#)
    - [LRU Cache](#)
    - [Maximum Gap](#)
    - [Check if a string is a number](#)
    - [Find possible triangle triplets](#)
    - [Binary Tree Upside down](#)
    - [All the Tree Traversal you need \(not recursively\)](#)
    - [Insertion Sort List](#)
    - [Reverse Nodes in K-Group](#)
    - [PriorityQueue and binary heap](#)
    - [Merge K sorted lists](#)
    - [Regular Expression Matching](#)
    - [Unique Binary Search Tree I and II](#)
    - [Word Break ... II](#)
    - [Word Break](#)
    - [Scramble String](#)
    - [WVildcard Matching](#)
    - [Sets: HashSet, TreeSet and LinkedHashSet](#)
    - [From Queue to other basic concepts \(Interface and ...\)](#)
    - [Symmetric Tree](#)
    - [Balanced Binary Tree](#)
    - [Add Interval to an array of intervals](#)
    - [Max points on a line](#)
    - [Count and Say](#)
    - [Find the sum of a binary tree](#)
    - [Some basic concepts: class, object, reference, met...](#)
    - [Find Peak Element](#)
    - [Linked List Cycle](#)
    - [Evaluate Reverse Polish Notation](#)
    - [Intersection of Two Linked Lists](#)
    - [Subsets... II](#)
    - [Binary Tree Postorder Traversal - Iterative](#)
    - [Binary Tree Maximum Path Sum](#)
- October (1)
- September (6)

## 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  $m \times n$  matrix of positive integers representing the height of each unit cell in a 2D elevation map, compute the volume of water L...

[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  $k$ ...

[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...

Awesome Inc. theme. Theme images by molotovcocktail. Powered by [Blogger](#).