572 Subtree of Another Tree

2018年4月15日 19:41

Given two non-empty binary trees **s** and **t**, check whether tree **t** has exactly the same structure and node values with a subtree of **s**. A subtree of **s** is a tree consists of a node in **s** and all of this node's descendants. The tree **s** could also be considered as a subtree of itself.

Example 1:

Given tree s:

```
    3
   / \
  4   5
 / \
1   2
```

Given tree t:

```
  4
 / \
1   2
```

Return **true**, because t has the same structure and node values with a subtree of s.

Example 2:

Given tree s:

```
    3
   / \
  4   5
 / \
1   2
   /
  0
```

Given tree t:

```
  4
 / \
1   2
```

Return **false**.

来自 <https://leetcode.com/problems/subtree-of-another-tree/description/>

给定两个非空二叉树 **s** 和 **t**，检验 **s** 中是否包含和 **t** 具有相同结构和节点值的子树。**s** 的一个子树包括 **s** 的一个节点和这个节点的所有子孙。**s** 也可以看做它自身的一棵子树。

Solution for Python3:

```python
class Solution1:
    def isSubtree(self, s, t):
        """
        :type s: TreeNode
        :type t: TreeNode
        :rtype: bool
        """
        if self.isSame(s, t):
           return True
        if not s:
           return False
        return self.isSubtree(s.left, t) or self.isSubtree(s.right, t)

    def isSame(self, s, t):
       if not (s and t):
           return s == t
       return s.val == t.val and self.isSame(s.left, t.left) and 
self.isSame(s.right, t.right)

class Solution2:
    def isSubtree(self, s, t):
        """
        :type s: TreeNode
        :type t: TreeNode
        :rtype: bool
        """
        from hashlib import sha256
        def hash(x):
            sha = sha256()
            sha.update(x.encode("utf8"))
            return sha.hexdigest()
        def merkle(node):
            if not node:
                return '#'
            m_left = merkle(node.left)
            m_right = merkle(node.right)
            node.merkle = hash(m_left + str(node.val) + m_right)
            return node.merkle
        merkle(s)
        merkle(t)
        def dfs(node):
            if not node:
                return False
            return node.merkle == t.merkle or dfs(node.left) or dfs(node.right)
        return dfs(s)

class Solution3:
    def isSubtree(self, s, t):
        """
        :type s: TreeNode
        :type t: TreeNode
        :rtype: bool
        """
        def convert(r):
            return '*' + str(r.val) + '*' + convert(r.left) + convert(r.right) if r else '#'
        return convert(t) in convert(s)
```

Solution for C++:

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution1 {
public:
    bool isSubtree(TreeNode* s, TreeNode* t) {
        if (!s)
            return false;
        if (isSame(s, t))
```

```
            return true;
        return isSubtree(s->left, t) || isSubtree(s->right, t);
    }
    bool isSame(TreeNode* s, TreeNode* t) {
        if (!s && !t)
            return true;
        if (!s || !t)
            return false;
        if (s->val != t->val)
            return false;
        return isSame(s->left, t->left) && isSame(s->right, t->right);
    }
};
class Solution2 {
public:
    bool isSubtree(TreeNode* s, TreeNode* t) {
        string ss = convert(s), tt = convert(t);
        return strstr(ss.c_str(), tt.c_str());
    }
    string convert(TreeNode* s) {
        if (!s)
            return "#";
        return "*" + to_string(s->val) + "*" + convert(s->left) + convert(s->
right);
    }
};
```

Appendix:

C++ 字符串匹配判断字符串中是否含有某个子字符串

1) 原始字符串均为char*类型
   a. char *ori = "abcdefg"
   b. string child = "cde"
   c. string oristring = ori
   d. oristring.find(child) < oristring.length() true表示含有

2) 原始字符串为string类型
   a. string ori = "abcdefg"
   b. string child = "cde"
   c. strstr(ori.c_str(), child.c_str()) 没有找到返回NULL