572. Subtree of Another Tree

May 6, 2017 | 124.3K views

Average Ranking: 2.45 (118 votes)

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**.

Solution

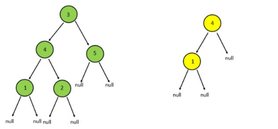
Approach #1 Using Preorder Traversal [Accepted]

Algorithm

We can find the preorder traversal of the given tree s and t, given by, say $s_{preorder}$ and $t_{preorder}$ respectively(represented in the form of a string). Now, we can check if $t_{preorder}$ is a substring of $s_{preorder}$.

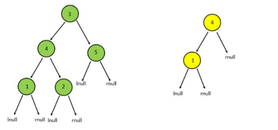
But, in order to use this approach, we need to treat the given tree in a different manner. Rather than assuming a *null* value for the children of the leaf nodes, we need to treat the left and right child as a *lnull* and *rnull* value respectively. This is done to ensure that the $t_{preorder}$ doesn't become a substring of $s_{preorder}$ even in cases when t isn't a subtree of s.

You can also note that we've added a '#' before every considering every value. If this isn't done, the trees of the form `s:[23, 4, 5]` and `t:[3, 4, 5]` will also give a true result since the preorder string of the `t("23 4 lnull rnull 5 lnull rnull")` will be a substring of the preorder string of `s("3 4 lnull rnull 5 lnull rnull")`. Adding a '#' before the node's value solves this problem.



Preorder$_s$ "#3 #4 #1 null null #2 null null #5 null null" Preorder$_t$ "#4 #1 null null null"

Preorder, substring of Preorder, but t isn't a subtree of s



Preorder$_s$ "#3 #4 #1 lnull rnull #2 lnull rnull #5 lnull rnull" Inorder$_t$ "#4 #1 lnull rnull rnull"

Preorder, not a substring of Preorder,

Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    HashSet < String > trees = new HashSet < >();
    public boolean isSubtree(TreeNode s, TreeNode t) {
        String treeS = preorder(s, true);
        String treeT = preorder(t, true);
        return treeS.indexOf(treeT) >= 0;
    }
    public String preorder(TreeNode t, boolean left) {
        if (t == null) {
            if (left)
                return "lnull";
            else
                return "rnull";
        }
        return "#"+t.val + " " + preorder(t.left, true)+ " "+preorder(t.right, false);
    }
}
```

Complexity Analysis

- Time complexity : $O(m^2 + n^2 = m * n)$. A total of n nodes of the tree s and m nodes of tree t are traversed. Assuming string concatenation takes $O(k)$ time for strings of length k and `indexOf` takes $O(m * n)$.
- Space complexity : $O(max(m, n))$. The depth of the recursion tree can go upto n for tree t and m for tree s in worst case.

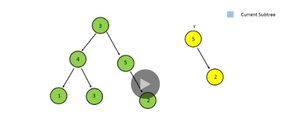
Approach #2 By Comparison of Nodes [Accepted]

Algorithm

Instead of creating an inorder traversal, we can treat every node of the given tree t as the root, treat it as a subtree and compare the corresponding with the given subtree s for equality. For checking the equality, we can compare the all the nodes of the two subtrees.

For doing this, we make use a function `traverse(s,t)` which traverses over the given tree s and treats every node as the root of the subtree currently being considered. It also checks the two subtrees currently being considered for their equality. In order to check the equality of the two subtrees, we make use of `equals(x,y)` function, which takes x and y, which are the roots of the two subtrees to be compared as the inputs and returns True or False depending on whether the two are equal or not. It compares all the nodes of the two subtrees for equality. Firstly, it checks whether the roots of the two trees for equality and then calls itself recursively for the left subtree and the right subtree.

The following animation depicts an abstracted view of the process:



Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isSubtree(TreeNode s, TreeNode t) {
        return traverse(s,t);
    }
    public boolean equals(TreeNode x,TreeNode y)
    {
        if(x==null && y==null)
            return true;
        if(x==null || y==null)
            return false;
        return x.val==y.val && equals(x.left,y.left) && equals(x.right,y.right);
    }
    public boolean traverse(TreeNode s,TreeNode t)
    {
        return s!=null && ( equals(s,t) || traverse(s.left,t) || traverse(s.right,t));
    }
}
```

Complexity Analysis

- Time complexity : $O(m * n)$. In worst case(skewed tree) `traverse` function takes $O(m * n)$ time.
- Space complexity : $O(n)$. The depth of the recursion tree can go upto n. n refers to the number of nodes in a.

Rate this article:

Previous Next

Comments: 97

Sort By

Type comment here... (Markdown is supported)

Preview Post

leetcodefan ★ 1947 December 30, 2018 10:04 AM

Approach 1 is brilliant! However, it is kind of misleading to rate this problem as easy since Approach 1 deserves a medium

158 Share Reply

SHOW 4 REPLIES

SohaDhamib ★ 55 December 23, 2017 2:07 PM

I dont get why did we use lnull and rnull.
#sR1null null null is not a substring of #2#4#1null null#2 null null #5 null null there is an extra null in the first string ??

57 Share Reply

palaa ★ 321 July 29, 2018 8:52 AM

we don't need to mark null as lnull and null, please fix

71 Share Reply

SHOW 5 REPLIES

alexanin ★ 47 May 24, 2018 5:33 AM

At approach #1 solution example code line "HashSet < String > trees = new HashSet < > ();" is unnecessary. Plz, fix it)

40 Share Reply

christpm ★ 1451 April 13, 2018 8:52 AM

Agree with others here about Approach #1 that:

1. The optimal runtime (using StringBuilder) is O(m+n+mn). O(m) and O(n) for generating the string for each tree, and O(mn) for the substring search.
2. The optimal space complexity is O(m+n), required to store each of the generated strings.

46 Share Reply

SHOW 2 REPLIES

luckygirl ★ 64 October 22, 2017 6:52 AM

How do you get O(m^2 + n^2 = m * n)? Preorder traversal should have linear time complexity. So it should be (m + n = m*n), isn't it?

21 Share Reply

SHOW 4 REPLIES

hello_world_cn ★ 885 January 4, 2018 10:08 AM

boolean left, is unnecessary for approach 1.

```
public class Solution {
    HashSet < String > trees = new HashSet < > ();
    boolean isSubtree(TreeNode s, TreeNode t) {
```

Read More

12 Share Reply

SHOW 3 REPLIES

Ellest ★ 122 January 15, 2018 9:32 AM

@luckygirl - I think the author of this article may of taken string concatenation into account. I agree it should be O(m + n) for the serialization (preorder traversal) given we're optimizing the concatenation (i.e. StringBuilder in Java, string array then concatenation at end for Python, etc.)

Note that comparison can also be reduced to O(m) as we could use KMP string comparison to check if

Read More

10 Share Reply

staymagnum ★ 7 October 7, 2017 9:48 PM

@vinod23
I find that's not necessary to differ lnull and rnull, both deal with null is ok.

7 Share Reply

SHOW 1 REPLY

GNG ★ 6 October 31, 2018 7:40 AM

Shouldn't the time complexity of Approach 2 be O(min(m, n)) instead of O(m*n)?

4 Share Reply

SHOW 1 REPLY