# Subtree of Another Tree

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 1

```java
public boolean isSubtree(TreeNode s, TreeNode t) {
    return serialize(s).contains(serialize(t));
}

public String serialize(TreeNode root) {
    StringBuilder res = new StringBuilder();
    serialize(root, res);
    return res.toString();
}

private void serialize(TreeNode cur, StringBuilder res) {
    if (cur == null) {res.append(",#"); return;}
    res.append("," + cur.val);
    serialize(cur.left, res);
    serialize(cur.right, res);
}
```

written by compton_scatter original link here

## Solution 2

Traverse tree `s`, when `s.val` == `t.val`, start a recursive call to validate if they are the same.

```java
public class Solution {
    public boolean isSubtree(TreeNode s, TreeNode t) {
        if (s == null || t == null) return false;

        if (s.val == t.val) {
            if (isSame(s, t)) return true;
        }

        return isSubtree(s.left, t) || isSubtree(s.right, t);
    }

    private boolean isSame(TreeNode s, TreeNode t) {
        if (s == null && t == null) return true;
        if (s == null || t == null) return false;

        if (s.val != t.val) return false;

        return isSame(s.left, t.left) && isSame(s.right, t.right);
    }
}
```

written by shawngao original link here

## Solution 3

```
public class Solution {
    public boolean isSubtree(TreeNode s, TreeNode t) {
        List<TreeNode> tLikes = new ArrayList<>(); //there maybe multiple nodes that
have the same value as t's
        find(s, t, tLikes);
        if(tLikes.isEmpty()) {
            return false;
        }

        List<Integer> tSubtree = new ArrayList<>();
        inOrder(t, tSubtree);
        for(TreeNode tLike : tLikes) {
            List<Integer> tLikeSubtree = new ArrayList<>();

            inOrder(tLike, tLikeSubtree);
            if(tLikeSubtree.equals(tSubtree)) { //2 subtrees are alike if they have
the same inOrder traversal
                return true;
            }
        }

        return false;
    }

    private void find(TreeNode s, TreeNode t, List<TreeNode> list) { //return all no
des in s with the same val as t
        if(s == null) {
            return;
        }

        if(s.val == t.val) {
            list.add(s);
        }

        find(s.left, t, list);
        find(s.right, t, list);
    }

    private void inOrder(TreeNode n, List<Integer> list) {
        if(n == null) {
            return;
        }
        inOrder(n.left, list);
        list.add(n.val);
        inOrder(n.right, list);
    }
}
```

written by soumyadeep2007 original link here