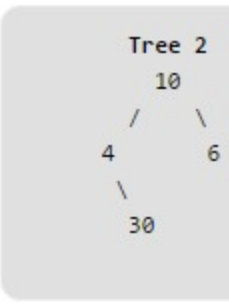
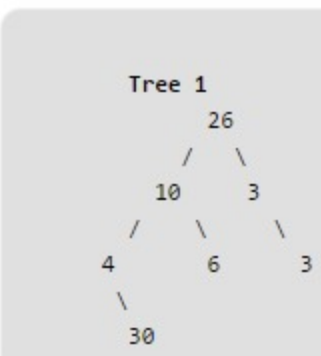
Check if a binary tree is subtree of another binary tree | Set 1

Given two binary trees, check if the first tree is subtree of the second one. A subtree of a tree T is a tree S consisting of a node in T and all of its descendants in T. The subtree corresponding to the root node is the entire tree; the subtree corresponding to any other node is called a proper subtree.

For example, in the following case, tree S is a subtree of tree T.





Solution: Traverse the tree T in preorder fashion. For every visited node in the traversal, see if the subtree rooted with this node is identical to S.

Following is the implementation for this.

C | Java | Python

```java
// Java program to check if binary tree is subtree of another binary tree

// A binary tree node
class Node
{
    int data;
    Node left, right, nextRight;

    Node(int item)
    {
        data = item;
        left = right = nextRight = null;
    }
}

class BinaryTree
{
    Node root1,root2;

    /* A utility function to check whether trees with roots as root1 and
       root2 are identical or not */
    boolean areIdentical(Node root1, Node root2)
    {

        /* base cases */
        if (root1 == null && root2 == null)
            return true;

        if (root1 == null || root2 == null)
            return false;

        /* Check if the data of both roots is same and data of left and right
           subtrees are also same */
        return (root1.data == root2.data
                && areIdentical(root1.left, root2.left)
                && areIdentical(root1.right, root2.right));
    }

    /* This function returns true if S is a subtree of T, otherwise false */
    boolean isSubtree(Node T, Node S)
    {
        /* base cases */
        if (S == null)
            return true;

        if (T == null)
            return false;

        /* Check the tree with root as current node */
        if (areIdentical(T, S))
            return true;

        /* If the tree with root as current node doesn't match then
           try left and right subtrees one by one */
        return isSubtree(T.left, S)
                || isSubtree(T.right, S);
    }

    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();

        // TREE 1
        /* Construct the following tree
              26
             /   \
            10     3
           /    \     \
          4      6      3
           \
            30  */

        tree.root1 = new Node(26);
        tree.root1.right = new Node(3);
        tree.root1.right.right = new Node(3);
        tree.root1.left = new Node(10);
        tree.root1.left.left = new Node(4);
        tree.root1.left.left.right = new Node(30);
        tree.root1.left.right = new Node(6);

        // TREE 2
        /* Construct the following tree
           10
         /    \
         4      6
          \
          30  */

        tree.root2 = new Node(10);
        tree.root2.right = new Node(6);
        tree.root2.left = new Node(4);
        tree.root2.left.right = new Node(30);

        if (tree.isSubtree(tree.root1, tree.root2))
            System.out.println("Tree 2 is subtree of Tree 1 ");
        else
            System.out.println("Tree 2 is not a subtree of Tree 1");
    }
}

// This code has been contributed by Mayank Jaiswal
```

Run on IDE

Output:

```
Tree 2 is subtree of Tree 1
```

Time Complexity: Time worst case complexity of above solution is O(mn) where m and n are number of nodes in given two trees.

We can solve the above problem in O(n) time. Please refer Check if a binary tree is subtree of another binary tree | Set 2 for O(n) solution.