09/08/2017

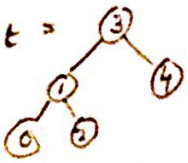
Assumptions made

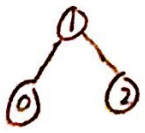
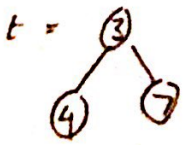
1. Tree is a binary tree
2. If S or T == Null or is empty return true
3. Binary tree can be of any size
4. function returns a bool value (True, false)
5. Tree T will not be a Null tree

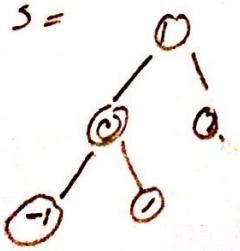
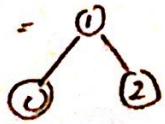
Test Cases

S = Null t : a tree with n elements → True

S =  t =  (S and t are the same) → True

S =  t =  (T contains s elements and structure) → True

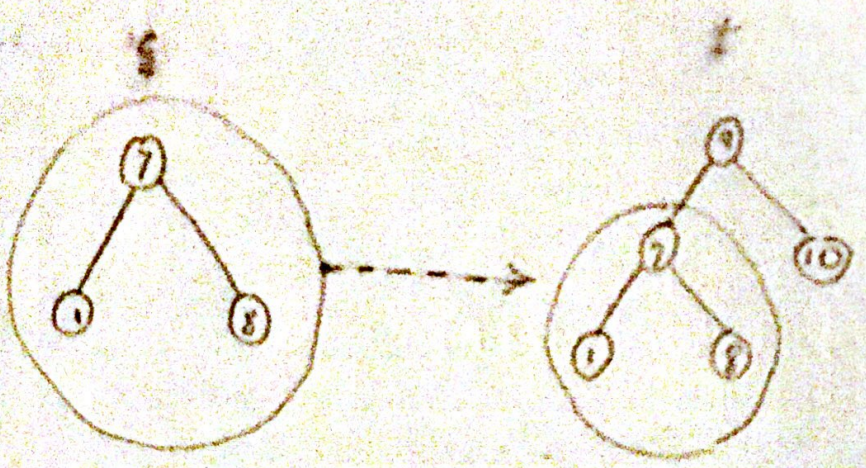
S =  t =  (T does not contain s elements or structure → False

S =  t =  (size of s > t) return false

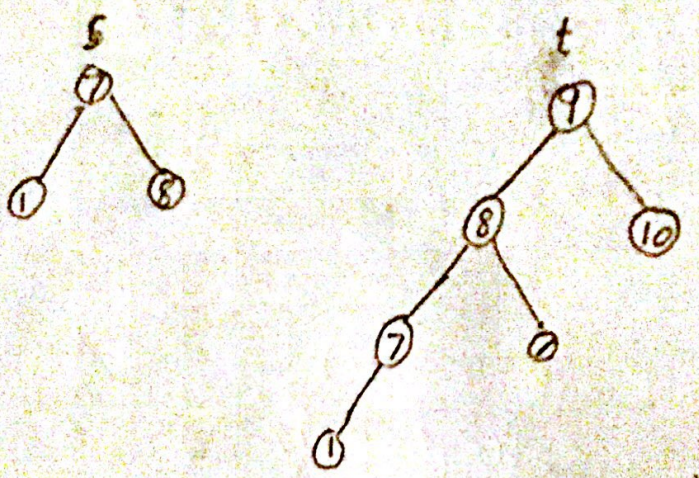
Examples

TEBOTI

Case 1 True (contains elements in the same order or structure)

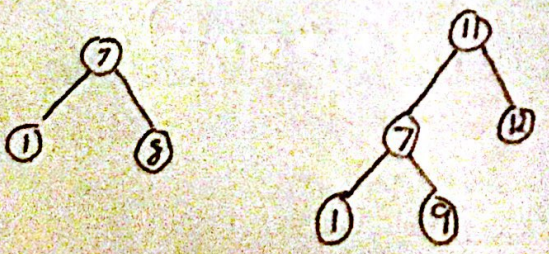


Case 2 False (contains elements but not in the same order or structure)



Elements {1, 8, 7} are present in both s and t

Case 3 False (Does not contain all elements)



Binary tree

```
template <typename T>
struct TreeNode
{
    T info;
    TreeNode<T> *left;
    TreeNode<T> *right;

    TreeNode(T i, TreeNode<T> *l, TreeNode<T> *r)
    : info(i), left(l), right(r)
};


template <typename T>
bool Contains_Sub_tree(TreeNode<T> *S, TreeNode<T> *t)
{
    if(t == nullptr || S == nullptr)
        return true;
    if(t->info == S->info)
    {
        bool Left = true;
        bool right = true;
        if(S->left != nullptr && t->left != nullptr)
        {
            Left = Contains_Sub_tree(S->left, t->left);
        }
        if(S->right != nullptr && t->right != nullptr)
        {
            right = Contains_Sub_tree(S->right, t->right);
        }
        return Left && right;
    }
    else
        return false
}
```

Optimize

Check if size of S is Larger than t if given size of both trees

If not Compute using regular function