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.

```
Tree 2
  10
  /   \
4      6
 \
  30
```

```
Tree 1
      26
     /   \
   10     3
  /   \     \
4      6      3
 \
  30
```

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

```
#include <stdio.h>
#include <stdlib.h>
```

```
/* A binary tree node has data, left child and right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* A utility function to check whether trees with roots as root1 and
   root2 are identical or not */
bool areIdentical(struct node * root1, struct 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 */
bool isSubtree(struct node *T, struct 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);
}


/* Helper function that allocates a new node with the given data
```

```
   and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node =
        (struct node*)malloc(sizeof(struct node));
    node->data  = data;
    node->left  = NULL;
    node->right = NULL;
    return(node);
}

/* Driver program to test above function */
int main()
{
    // TREE 1
    /* Construct the following tree
              26
            /   \
          10     3
        /    \     \
      4      6      3
       \
        30
    */
    struct node *T        = newNode(26);
    T->right              = newNode(3);
    T->right->right       = newNode(3);
    T->left               = newNode(10);
    T->left->left         = newNode(4);
    T->left->left->right  = newNode(30);
    T->left->right        = newNode(6);

    // TREE 2
    /* Construct the following tree
          10
        /    \
      4      6
       \
        30
    */
    struct node *S    = newNode(10);
    S->right          = newNode(6);
    S->left           = newNode(4);
    S->left->right    = newNode(30);


    if (isSubtree(T, S))
        printf("Tree 2 is subtree of Tree 1");
    else
        printf("Tree 2 is not a subtree of Tree 1");
```

```
    getchar();
    return 0;
}
```

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 {

    static Node root1,root2;

    /* A utility function to check whether trees with roots as root1 and
     root2 are identical or not */
    boolean areIdentical(Node node1, Node node2) {

        /* base cases */
        if (node1 == null && node2 == null) {
            return true;
        }

        if (node1 == null || node2 == null) {
            return false;
        }

        /* Check if the data of both roots is same and data of left and right
         subtrees are also same */
        return (node1.data == node2.data
                && areIdentical(node1.left, node2.left)
                && areIdentical(node1.right, node2.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(root1, 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
```

Python

```
# Python program to check binary tree is a subtree of
# another tree

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# A utility function to check whether trees with roots
# as root 1 and root2 are indetical or not
def areIdentical(root1, root2):

    # Base Case
    if root1 is None and root2 is None:
        return True
    if root1 is None or root2 is None:
        return False

    # Check fi the data of both roots is same and data of
    # left and right subtrees are also same
    return (root1.data == root2.data and
            areIdentical(root1.left , root2.left)and
            areIdentical(root1.right, root2.right)
            )

# This function returns True if S is a subtree of T,
# otherwise False
def isSubtree(T, S):
```

```
    # Base Case
    if S is None:
        return True

    if T is None:
        return True

    # 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 subtreee one by one
    return isSubtree(T.left, S) or isSubtree(T.right, S)


# Driver program to test above function

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

T = Node(26)
T.right = Node(3)
T.right.right  = Node(3)
T.left = Node(10)
T.left.left = Node(4)
T.left.left.right = Node(30)
T.left.right = Node(6)

""" TREE 2
     Construct the following tree
           10
         /    \
       4      6
        \
         30
    """
S = Node(10)
S.right = Node(6)
S.left = Node(4)
S.left.right = Node(30)
```

```
if isSubtree(T, S):
    print "Tree 2 is subtree of Tree 1"
else :
    print "Tree 2 is not a subtree of Tree 1"

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

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.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

172 Comments Category: Trees

Related Posts:

- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree

- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?
- Closest leaf to a given node in Binary Tree

(Login to Rate and Mark)

2.6 Average Difficulty : **2.6/5.0**
Based on **22** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

prashant jha

```
#include
#define size 50
using namespace std;
struct tnode
{
tnode* lchild;
int data;
tnode* rchild;
tnode(int d)
{
lchild=NULL;
data=d;
rchild=NULL;
}
};
void create(tnode* &root,int d)
{
int n;
if(d==0)
return;
root=new tnode(d);
cout<<"enter the left child of "<data<>n;
create(root->lchild,n);
cout<<"enter thr right child of "<data<>n;
create(root->rchild,n);
}
int check(tnode* t1,tnode* t2)
{
if((!t1)&&(!t2))
return 1;
if(t1->data!=t2->data)
return 0;
if(((t1)&&(!t2))||((t2)&&(!t1)))
return 0;
int k1=check(t1->lchild,t2->lchild);
int k2=check(t1->rchild,t2->rchild);
if((k1)&&(k2))
return 1;
return 0;
}
void fun(tnode* t1,tnode* t2,int &flag)
{
if(!t1)
return;
if(t1->data==t2->data)
```

```
{
int k=check(t1,t2);
if(k)
flag=1;
return;
}
fun(t1->lchild,t2,flag);
fun(t1->rchild,t2,flag);
}
int main()
{
tnode* t1,*t2;
t1=t2=NULL;
int n,flag=0;
cout<>n;
create(t1,n);
cout<>n;
create(t2,n);
fun(t1,t2,flag);
if(flag)
cout<<"it is the subtree.n";
else
cout<<"it is not subtree.n";
return 0;
```

alien

This solution will take o(n^2) in worst case. Is there any o(n) solution?

kingfed

The above code doesnt work in the case when we simply add a extra node to S.
S->left->right->left = newNode(24);
or infact any node to a leaf of S even now S is a subtree of T.
What I mean is that in the code checking if two subtrees are identical would not suffice . Kindly resolve my doubt

Archit

```
#include
#include
using namespace std;
struct node
{
int data;
node *left;
node *right;
};
node *newNode(int data)
{
node *tmp=new node;
tmp->data=data;
tmp->left=NULL;
tmp->right=NULL;
return tmp;
}
int isIdentical(node *T,node *S)
{
if(S==NULL&&T==NULL)
return 1;
if(S==NULL)
return 1;
if(T==NULL)
return 0;
if(T->data!=S->data)
return 0;
return(isIdentical(T->left,S->left)&&isIdentical(T->right,S->right));
}
int isSubtree(node *T,node *S)
{
if(T==NULL&&S==NULL)
return 1;
if(T==NULL||S==NULL)
return 0;
if(T->data==S->data)
{
```

```
if(isIdentical(T,S))
return 1;
}
return(isSubtree(T->left,S)||isSubtree(T->right,S));
}
int main()
{
struct node *T = newNode(26);
T->right = newNode(3);
T->right->right = newNode(3);
//T->left = newNode(10);
//T->left->left = newNode(10);
//T->left->left->left = newNode(4);
//T->left->left->right = newNode(6);
//T->left->left->left->right=newNode(30);
T->right->right=newNode(10);
T->right->right->left=newNode(4);
T->right->right->right=newNode(6);
T->right->right->left->right=newNode(30);
struct node *S = newNode(10);
S->right = newNode(6);
S->left = newNode(4);
S->left->right = newNode(30);
if( isSubtree(T, S) )
printf("Tree S is subtree of tree T");
else
printf("Tree S is not a subtree of tree T");
return 0;
}
```

Archit

```
#include
#include
using namespace std;
struct node
{
int data;
node *left;
node *right;
};
node *newNode(int data)
{
node *tmp=new node;
tmp->data=data;
tmp->left=NULL;
tmp->right=NULL;
return tmp;
}
int isIdentical(node *T,node *S)
{
if(S==NULL&&T==NULL)
return 1;
if(S==NULL)
return 1;
if(T==NULL)
return 0;
if(T->data!=S->data)
return 0;
return(isIdentical(T->left,S->left)&&isIdentical(T->right,S->right));
}
int isSubtree(node *T,node *S)
{
if(T==NULL&&S==NULL)
return 1;
if(T==NULL||S==NULL)
return 0;
if(T->data==S->data)
{
if(isIdentical(T,S))
return 1;
}
```

```
return(isSubtree(T->left,S)||isSubtree(T->right,S));
}
int main()
{
struct node *T = newNode(26);
T->right = newNode(3);
T->right->right = newNode(3);
//T->left = newNode(10);
//T->left->left = newNode(10);
//T->left->left->left = newNode(4);
//T->left->left->right = newNode(6);
//T->left->left->left->right=newNode(30);
T->right->right=newNode(10);
T->right->right->left=newNode(4);
T->right->right->right=newNode(6);
T->right->right->left->right=newNode(30);
struct node *S = newNode(10);
S->right = newNode(6);
S->left = newNode(4);
S->left->right = newNode(30);
if( isSubtree(T, S) )
printf("Tree S is subtree of tree T");
else
printf("Tree S is not a subtree of tree T");
return 0;
}
```

sp
With just one traversal you cannot reconstruct a tree. So only preorder is not sufficient to determine if the tree is subtree or not.
http://stackoverflow.com/questions/12880718/how-many-traversals-need-to-be-known-to-construct-a-bst
sassy
In your code checking first if the data at the node = data at the root of the smaller tree and then recursing if they are equal would reduce the no. of recursion calls massively.
Guru
Please suggest comments over this code

```
public boolean isSubTree(BinaryTreeNode T, BinaryTreeNode S) {
if (S == null) {
return true;
}
if (T == null) {
return false;
}
if (T.getData() == S.getData()) {
return (isSubTree(T.getLeft(), S.getLeft()) && isSubTree(
T.getRight(), S.getRight()));
} else {
return (isSubTree(T.getLeft(), S) || isSubTree(T.getRight(), S));
}
}
```

vamsi varanasi
What if we could find the root of the tree S in the Tree T and then check if the trees are identical it would reduce the complexity of checking for each and every node in tree T, assuming there are no duplicate values. would this approach work or am I missing something.
pavansrinivas
Using level order in JAVA
Time complexity(O(m+n))
Space Complexity(O(m+n))

```
boolean isSubtree(Node r2){
Node temp1 = root;
Queue q1 = new LinkedList();
q1.add(temp1);
while(!q1.isEmpty()){
temp1 = q1.poll();
if(temp1.i==r2.i){
return areIdentical(temp1, r2);
}
if(temp1.left!=null){
q1.add(temp1.left);
}
```

```
if(temp1.right!=null){
q1.add(temp1.right);
}
}
return false;
}
private boolean areIdentical(Node t1, Node t2){
Queue q1 = new LinkedList();
Queue q2 = new LinkedList();
Node temp1 = t1;
Node temp2 = t2;
q1.add(t1);
q2.add(t2);
while(!q1.isEmpty()&!q2.isEmpty()){
temp1 = q1.poll();
temp2 = q2.poll();
if(temp1.i!=temp2.i){
return false;
}
else if(temp1.left==null&&temp2.left!=null||temp1.right==null&&temp2.right!=null){
return false;
}
if(temp1.left!=null){
q1.add(temp1.left);
}
if(temp1.right!=null){
q1.add(temp1.right);
}
if(temp2.left!=null){
q2.add(temp2.left);
}
if(temp2.right!=null){
q1.add(temp2.right);
}
}
if(q1.isEmpty()&&q2.isEmpty()){
return true;
}
else if(q1.isEmpty()&&!q2.isEmpty()){
return false;
}
else{
return true;
}
}
```

Sambhav Sharma
In function areIdentical
shouldn't the following code
return (root1->data == root2->data && areIdentical(root1->left, root2->left) && areIdentical(root1->right, root2->right));
be changed to this:
if(root1->data = root2->data)
{
if(areIdentical(root1->left,root2->left))
return areIdentical(root1->right,root2->right);
else
return 0;
}
else
return 0;
Maybe may understanding of the return statement is not clear, but I feel it will call for all the three statements and then do the AND of their results…
Kk
The && operator does not work that way. I think the name of the concept is short-circuit evaluation, where it will evaluate root1->data == root2->data first. If this is false, then it will not call areIdentical on root1 and root2's children.
Vivek
hi geeksforgeeks..
below is an optimized solution in O(n)..(not O(m*n).)
where is the size of the larger tree.

please go through this solution.

```
#include
#include
/* A binary tree node has data, left child and right child */
struct node
{
int data;
struct node* left;
struct node* right;
};
int check_subtree(struct node *root1, struct node *root2, int *count)
{
if(!root1 && !root2)
return 1;
if(!root1 || !root2)
return 0;
if(*count == 0)
{
if(root1->data == root2->data )
{
*count= *count +1;
return 1 && check_subtree(root1->left,root2->left, count) && check_subtree(root1-
>right, root2->right, count);
}
else
return check_subtree(root1->left, root2, count) || check_subtree(root1->right, root2,
count);
}
else if(*count > 0)
{
if(root1->data == root2->data)
{
*count= *count +1;
return 1 && check_subtree(root1->left, root2->left, count) && check_subtree(root1-
>right, root2->right, count);
}
else
return 0;
}
}
int isSubtree(struct node *root1, struct node *root2)
{
int count = 0;
return check_subtree(root1, root2, &count);
}
struct node* newNode(int data)
{
struct node* node =
(struct node*)malloc(sizeof(struct node));
node->data = data;
node->left = NULL;
node->right = NULL;
return(node);
}
/* Driver program to test above function */
int main()
{
/* Construct the following tree
26
/
10 3
/
4 6 3
30
*/
struct node *T = newNode(26);
T->right = newNode(3);
T->right->right = newNode(3);
T->left = newNode(10);
T->left->left = newNode(4);
```

```
T->left->left->right = newNode(30);
T->left->right = newNode(6);
/* Construct the following tree
10
/
4 6
30
*/
struct node *S = newNode(10);
S->right = newNode(6);
S->left = newNode(4);
S->left->right = newNode(30);
int count =0;
if( isSubtree(T, S) )
printf("Tree S is subtree of tree T");
else
printf("Tree S is not a subtree of tree T");
getchar();
return 0;
}
```

dag
I think the complexity will be exponential because you are checking each permutation in recursive call.
Vivek
complexity is not exponential as each node is visited only once
Kk
You are not resetting the count. If you have a partial match, you should reset the count before returning false.
Anyways, the reason g4g wrote the code in that particular way is to avoid maintaining another state variable.
That is why they split the responsibility as:
1) Checking if subtrees rooted at root1 and root2 are identical
2) checking if T contains S as a subtree.
nik
U are assuming that no value is repeating..which isn't right
DarkProtocol
1) Check if root of subtree(S) found in Tree(T), if not found return false,or else proceed
2) take two pointers, p1 points to root of S and p2 pointing to T, Check isidentical(p1,p2).
Bohemia
How's this one ?
1)get the Inorder of BST1=In1[]
2)get the Inorder of BST2=In2[]
3)Find occurence of In1[1] in In2[] (Binary Search) and Match the subsequent Values of Nodes: IF ALL MATCHED return TRUE else FALSE
should be O(n) n is no of nodes in larger BST
Mammamia
Approach won't work in all cases, as inorder of two different trees can be same.
xxmajia
you need at least 2 order to construct a tree,
so you can get inorder of both, then preorder or postorder of them again, then check 4 arrays
Bohemia
yes ! if inorder and either pre or post order (one of these two) are found in the bigger tree's corresponding traversals, we can say that the smaller tree is contained in the larger tree .
zedus
Hmmm… I think there's an issue with this solution.
Inside the "areIdentical" function,
there's
——
if(root1 == NULL || root2 == NULL)
return false;
——
which isn't correct imho.
the big tree that we are testing "against" may contain MORE than the tree we are searching for.
for example, it's fine if the tree "pattern" contains NULL at a certain node,
but the big tree contains a real node there.
We should return true in that case.
Might be an issue of definition, but that's the only definition of subtree that i know.
Whiskers
what if we do a traversal to find the root of S in T and then do a kind of inorder traversal to check ?
T=node of where it the data equal to data of the root in tree S
visit(t,s)
{
if(t->data!=s->data)

```
return 0;
if(s==null && t==null)
return 1;
if(t==null)
return 0;
if(s==null)
return true;
if(visit(t->l,s->l))
return (visit(t->r,s->r));
}
}
main()
{
if(visit(t,s))
cout<<"its a subtree";
else
cout<<"no subtree";
}
```

vivek
Code is doing the Pre Order traversal not In order Travel, correct?

```
/* Paste your code here (You may delete these lines if not writing code) */
```

GeeksforGeeks
Thanks for pointing this out. We have updated explanation. Keep it up!
abhishek08aug
Intelligent 😃
vaibhavbright
This solution takes O(n^2) time in worst case. We can simply store inorder and preorder traversals of both the trees and use KMP algorithm for pattern matching. Thus it would have time complexity of O(n).
vaibhavbright
okay.. the current solution is taking O(n * m) time with the two trees of n & m no. of nodes
akshat gupta
——— T————————— S
——— 0————————— 12
———/—\————————/—\
——0——12————————-11—9
——/-.—/—\————————/
——.–. 11—9————————8
——.–.———/—\————————— –
——.–.——8——1———————————
T
Preorder: ….. 12,11,9,8,1
Inorder: …… 11,12,8,9,1
S
Preorder: 12,11,9,8
Inorder: 11,12,8,9
your Idea will Show S as a subtree of T
But,the tree structure Conradicts it..
Hence,it FAILS
Siddharth Bora
I think we can do this by KMPing on all the 3 orders – inorder, preorder, postorder.
In that way this should succeed
Jatin Kumar
No, I guess it won't work. For example the bigger tree is node(a, node(a, a, a), node(a, a, a)) and the smaller tree is simply node(a, null, node(a, null, a))
Doctor

```
/* Paste your code here (You may delete these lines if not writing code) */

int areIdentical(node* root1,node* root2)
{
if(root1==NULL&&root2==NULL)
return 1;
if(root1==NULL||root2==NULL)
return 0;
return (root1->data==root2->data
&& areIdentical(root1->lchild,root2->lchild)|| areIdentical(root1->rchild,root2->rchild));
}
/*
```

26 26 10
/ \ \ / \
10 29 or 29 or 4 16
/ \ / \ / \
4 16 28 35 28 35
\
8
T s1 s2
I think s1 and s2 are subtree of T:If yes then your code doesnot give write answer and suggested code for right answer is shown above.
*/
Pavan Dittakavi
How about the below piece of code, this is just a pseudo code, but it should work. If you guys find anything wrong or require any clarifications..please drop by 🙂

```
logical isSubset( root , root' )
{
  if( root == NULL && root' == NULL )
  return true;


  if( root == NULL || root' == NULL )
  return true;

  if( root->data == root'->data )
  return true && isSubset( root->left,root'->left ) && isSubset(root->right,root'->right );

  return isSubset( root->left,root' ) || isSubset( root->right,root' );
}
}
```

Pawan
```
 if( root->data == root'->data )
  return true && isSubset( root->left,root'->left ) && isSubset(root->right,root'->right );
```

yes, this sounds correct to me, because and operator will not check the further condition if the first condition itself is wrong.
Sorry I could not understand the below condition. Why we require this one?
return isSubset(root->left,root') || isSubset(root->right,root');

```
/* Paste your code here (You may delete these lines if not writing code) */
```

rocky
(1) Do serialization of both the trees (using preorder traversal). O(n)
(2) Then apply KMP algorithm to find if a substring exists. O(n)
Total Time Complexity : O(n)

```
void serialize(tree *t, string &s) {
       if(!t) { s = s + "#"; return; }
       s = s + t->val + '0';
       serialize(t->left, s);
       serialize(t->right, s);
       return;
}
```

Bharti
Why do we need to go through all the nodes and not stop at first match. As we also have the addresses of the nodes , we can always compare the addresses. For eg: lets say value at root of S is 10, now we started traversing T(any method) and as soon as we find 10 we will check whether pointers current.left and current.right matches to nodes S.left and S.right.
Moreover, solution suggested by you can be used to check whether a tree (S here) is equivalent to any subtree of tree T or not.
mccullum
i got the same doubt if a tree is a subtree why not just compare pointers!! traverse T and look for a node with same pointer as S..!!
Himanshu
I think I have a better time complexity algo.
If the bigger tree has n elements and smaller(potential subtree of the bigger one) has m elements, then the algo stated here is of O(m*n) time complexity.
Now what we can do is define a modified pre-order as follows:

print root
if(left) : go to left recursively
else : print ‘L'(or any other thing to notify left null reached) //for leaf
if(right) : go to right recursively
else : print ‘R'(any other thing to notify right null reached) //for leaf
This makes sure that no two different trees will have same modified-pre order output
or looking it the other way , given a modified pre-order output I can uniquely define my binary tree.
So algo is as follows :
1) Store the modified pre-order of both the trees inside a string. space will be O(m) & O(n)
2)Do a Knuth Morris Pratt (KMP) algo to find is smaller string a sub-string of bigger.
Thus the algo is basically convert each tree as a string and finding if smaller is sub-string of bigger or not. time complexity O(m+n)
Please let me know if I am making any mistake any where 🙂
dejavu
/* Consider the following trees
26
/ \
10 3
/ \ pre-order: 26 10 4 3 8 5 7
4 8
/ \
5 7
and
26
/
10 pre-order: 26 10 4 3
/ \
4 3
Although the smaller string is a sub-string of the bigger one, the Tree is not sub-tree of another.
But I think it will work with BST.
dejavu
err. the trees did not appear properly.
tree 1:
root = 26
left child of root = 10
left child of 10 = 4
right child of 26 = 3
tree 2:
root = 26
left child of root = 10
left child of 10 = 4
right child of 10 = 3
the two trees are not same but their pre-order traversals are same.
Himanshu
You probably did not get the meaning of “modified” pre-order, as I have written in my post, the mod pre-order of 1st would be : 26 10 4 L R R 3 L 8 5 L R 7 L R
and for the second would be :
26 10 4 L R 3 L R R
So the smaller is not sub tree of bigger.
If however your second tree was :
root = 3
right child of root = 8
left child of 8 = 5
right child of 8 = 7
The “modified” pre-order would then be :
3 L 8 5 L R 7 L R
Now this clearly is a substring of the main tree.
brinpage
@himanshu: just consider this case
4
/ \
3 5
/
6
(436LR5LR)
and
3
/ \
6 5
(36LR5LR)

but its not a subtree of the first one.
brinpage
i was wrong.. for first tree its
436LRR5LR..
i think ur approach is CORRECT!
James Bond
One small note : the characters representing L & R here should be different from any data in the trees considered.
James Bond
Also, inorder and postorder should also work.
Tushar
This approach is perfect. Moderator pls put this as a standard solution
Kartik
@Tushar: Thanks for providing inputs. We will take look into this approach and add it to the original post.
Anuj
I am not able to make out correct output in this case:
tree 1:
root = 1
L(1) = 2
R(1) = 3
L(2) = 4
R(2) = 5
L(4) = 6
R(4) = 7
L(5) = 8
R(5) = 9
Tree 2:
root = 2
L(2) = 4
L(4) = 6
R(5) = 7
I think the output as per your suggested algo would be:
tree 1:
1246LR7LR58LR9LR3LR
tree 2:
246LR7LRR
These doesn't match…please explain if there is any mistake

```
/* Paste your code here (You may delete these lines if not writing code) */
```

aygul
These should not match!
tree2 is not a sub tree of tree1. Because
"A subtree of a tree T is a tree S consisting of a node in T and all of its descendants in T."
tree2 is consistinf of a node in tree 1 but not all ofits descandants…
http://www.cse.iitb.ac.in/~saha/ subhasish
If the subtree appears at the bottom of original tree then this (or any traversal) will work. But if the subtree appears in the internal structure of the original tree, then this will not work.
Srava Vurapalli

```
bool BinarySearchTree::ISSubTree(BinarySearchTree& rootMainTree, BinarySearchTree& rootSubTree)
{
if(rootMainTree.Root() == NULL || rootSubTree.Root() == NULL)
{
return false;
}
Node* p = rootMainTree.Search(rootSubTree.Root()->value);
if(p == NULL)
{
return false;
}
else
{
return Compare(p,rootSubTree.Root());
}
}
bool BinarySearchTree::Compare(Node* rootMainTree, Node* rootSubTree)
{
if(rootMainTree == NULL && rootSubTree == NULL)
{
return true;
```

```
}
else if( (rootMainTree == NULL || rootSubTree)
{
return false;
}
else
{
if(rootMainTree->value == rootSubTree->value)
{
return BinarySearchTree::Compare(rootMainTree->left,rootSubTree->left) &&
BinarySearchTree::Compare(rootMainTree->right,rootSubTree->right);
}
return false;
}
```

Sravan Vurapalli

```
bool BinarySearchTree::ISSubTree(BinarySearchTree& rootMainTree, BinarySearchTree& rootSubTree)
{
if(rootMainTree.Root() == NULL || rootSubTree.Root() == NULL)
{
return false;
}
//Search for the element and stop if you find that if first tree.
Node* p = rootMainTree.Search(rootSubTree.Root()->value);
if(p == NULL)
{
return false;
}
else
{
return Compare(p,rootSubTree.Root());
}
}
bool BinarySearchTree::Compare(Node* rootMainTree, Node* rootSubTree)
{
if(rootMainTree == NULL && rootSubTree == NULL)
{
return true;
}
else if( (rootMainTree == NULL || rootSubTree == NULL)
{
return false;
}
else
{
if(rootMainTree->value == rootSubTree->value)
{
return BinarySearchTree::Compare(rootMainTree->left,rootSubTree->left) &&
BinarySearchTree::Compare(rootMainTree->right,rootSubTree->right);
}
return false;
}
}
```

Thanks And Regards,
Sravan Vurapalli.

Algorist

Please find below my code, and I find it really clean. Please let me know on this.
Also, i have one query and that is if the small tree is NULL(i.e. no nodes) and Big tree contains some nodes, then the the function should return true or false.. Please comment.
Thanks.

```
int isSubTree(Node* bRoot, Node* sRoot)
{
if (bRoot == NULL && sRoot == NULL) //If both the trees are NULL
return 1;
if (bRoot == NULL && sRoot != NULL) //If Big Tree is NULL and Small Tree is not NULL
return 0;
if (bRoot != NULL && sRoot == NULL) // If Big Tree has some nodes, and small tree is null, then small tree is a subtree of that kind of trees.
return 1;
if(bRoot->info == sRoot->info) // if same nodes found, then left and right subtrees should also be a SUBTREE
```

```
return isSubTree(bRoot->left, sRoot->left) && isSubTree(bRoot->right, sRoot->right);
//Else Either left or right of that node is a SUBTREE
return (isSubTree(bRoot->left, sRoot) || isSubTree(bRoot->right, sRoot));
}
```

viji

```
#include "malloc.h"
#include "stdio.h"
#include "conio.h"
struct node
{
    node *left;
    node *right;
    int data;
};
void insert(node **p, int d)
{
    if(*p == NULL)
    {
        *p = (node *)malloc(sizeof(node));
        (*p)->left = NULL;
        (*p)->right = NULL;
        (*p)->data = d;
    }
    else
    {
        if((*p)->data >= d)
            insert(&((*p)->left), d);
        else
            insert(&((*p)->right), d);
    }
}
void check(node *p, node *q, int *is)
{
    if(*is)
    {
    if((p == NULL) && (q == NULL))
    {
        return;
    }
    else if((p == NULL) || (q==NULL))
    {
        *is = 0;
        return;
    }
    if(p->data == q->data)
    {
        check(p->left, q->left, is);
        check(p->right, q->right, is);
    }
    else
    {
        *is = 0;
    }
    }
}
void sub_tree(node *p, node *q)
{
    int issubtree = 1;
    if(p == NULL)
        return;
    if(p->data == q->data)
    {
        check(p, q, &issubtree);
        if(issubtree == 1)
        {
            printf("the tree is subtree");
            return;
        }
```

```
    }
    else
    {
        sub_tree(p->left, q);
        sub_tree(p->right, q);
    }
}
int main(int argc, _TCHAR* argv[])
{

    node *p = NULL;
    node *q = NULL;
    insert(&p, 7);
    insert(&p, 4);
    insert(&p, 10);
    insert(&p, 3);
    insert(&p, 6);
    insert(&p, 9);
    insert(&p, 11);
    insert(&p, 5);
    insert(&q, 10);
    insert(&q, 9);
    insert(&q, 11);
    sub_tree(p, q);
    return 0;
}
```

http://ahmetalpbalkan.com ahmet alp balkan
Another awesome solution, write preorder traversal of a tree as “string” (I am serious) and then do it for parent tree to. then use String.contains method!
Liked it? (:
Mohit Ahuja
That might work for Binary Search Trees but not for Binary Trees:for e.g.-
S=3
\
3
and
T= 3
/
3
would generate same string on preorder traversal but they surely S is not a subtree of T.
viresh
u could improvise the code by finding the root of subtree in the given tree and then using isIndentical() function…
Dreamer
This logic seems correct to me. Any reason why can’t we use this.
guest
The algorithm’s time complexity is linear to the number of nodes. Another linear algorithm can be printing nodes of both trees in preorder into two arrays and check if one array if subarray of another using dynamic programming.
kartik
Time complexity of the solution given in post is O(mn) where m is the number of nodes in S and n is the number of nodes in T.
Time complexity of the inorder and postorder matching solution is O(n), but this solution doesn’t handle the case of duplicates. See following comments SDiZ and me. Please correct me if I am wrong.
SDiZ
I guess you can print both trees in both inorder and postorder, and check if those of tree S are substring of tree T.
generate inorder/postorder – O(n)
substring using KMP – O(n)
SDiZ
provided that no tree nodes have duplicated label, that is.
Your tree have two “3”, so it does not work.
Sorry.
Bugaboo
Why would a duplicated node label fail? Woudn’t a string matching algorithm work nevertheless?
kartik
Consider the following example.
Main Tree T

```
            /   \
          10     3
        /    \     \
      4       6      3
       \
        30
```

Tree S

```
    3
   /
  3
```

The inorder and postorder traversals of S are substrings of inorder and postorder traversals of T, but S is not a subtree of T.
If we don't have duplciates then this is a nice O(N) approach.
MB
kartik,
Even without duplicates the inorder and postorder traversal strings will fail, in some cases.
You should use the preorder and inorder traversals to test for equivalence.
The constraint of no duplicates still applies. Using preorder and inorder still fails with duplicates.
MB
SDiZ,
You can't use the inorder and postorder traversal strings of a binary tree to test for equivalence. Consider the following binary trees.
Tree T

```
        26
        /
       10
       / \
      4   6
      /
     30
```

Tree S

```
       10
       / \
      4   6
```

```
Tree T                        Tree S
inorder:   30,4,10,6,26     4,10,6
postorder: 30,4,6,10,26     4,6,10
```

A substring comparison is going to return true, which is not correct. Tree S is not a subtree of Tree T. You need to use the preorder and inorder traversals. Plus, as already noted, each node needs to be uniquely labeled. You can't have duplicate values in the binary tree.

```
Tree T                    Tree S
preorder: 26,10,4,30,6    10,4,6
```

So a substring search of the preorder string of T with the preorder string of S correctly returns false.
MB
Your areIdentical(struct node * root1, struct node *root2) function has a subtle bug in it. The easiest way to see it is to build a truth table:
r1 r2 action
———————————————
1 true true continue
2 false true return false – r1 has terminated before r2
3 true false return true – r2 has terminated before r1
4 false false return true – r2 and r1 have terminated
Your logic covers cases 1,2 and 4 but misses the third case. This can happen when the r1 tree has more nodes, but all of the nodes of r2 have matched up to that point, so you have a match, the r1 tree just has more nodes, which is ok. It is also seems this would be the more common case. Your logic is only going to match sub-trees that have the same number of leaf nodes.

```
bool areIdentical(struct node * root1, struct node *root2) {
    if (!root2) return true;
    if (!root1) return false

    return (root1->data == root2->data   &&
            areIdentical(root1->left, root2->left) &&
            areIdentical(root1->right, root2->right) );
}
```

http://geeksforgeeks.org/ Sandeep
@MB: I think the following condition handles the case 3 as well.

```
    if(root1 == NULL || root2 == NULL)
        return false;
```

Could you please provide an example tree for which it doesn't seem to work.
MB
It doesn't take a complex tree to show where this fails. For case 3, root1=true and root2=false, which should return true, not false as you have indicated.

```
/*
example tree that returns false incorrectly,
the root2 tree is obviously a subtree of  the  root1 tree.

       root1          root2
       A              A
      /
     B
*/
if (root1==NULL || root2==NULL)
  return false;
```

http://geeksforgeeks.org/ Sandeep
@MB: Please take a closer look at the definition of subtree. In your example, root2 is not a subtree of root1.
MB
Sandeep, the definition says,
"A subtree of a tree T is a tree S consisting of a node in T and all of its descendants in T."
The root node of root2, or A, and all of its descendants are in the root1 tree, which makes it a subtree of root1.
What am I missing? How is the tree root2 not a subtree of root1?
kartik
@MB: root2 doesn't contain the descendant B of A in root1. So it is not a subtree. Following are the only possible subtrees of root1.
A
/
B
B
NULL (Empty Tree)
MB
Kartik,
If you have the set {A,B}, you're saying that {B} and {null} are subsets but {A} isn't a subset?
That is not correct,{A},{B} and {null} are all subsets of the set {A,B}.
Which is similar to saying that if you have the string "AB" The string "B" is a substring of "AB" but the string "A" is not a substring of "AB"
Mmmmmm, interesting.
Bugaboo
Sorry, if my previous post did not have the figure right. This is what I meant:
– Tree 'T' has 'A' as root and 'B' as its left child
– Tree 'S' has 'A' as root and 'B' as its right child
kartik
@MB: Subsets and subtrees are different things. Sets don't have structure or hierarchy, but trees do have.
And if you look at the definition, it says
"A subtree of a tree T is a tree S consisting of a node in T **and all of its descendants in T**."
In case of "A", its descendant "B" is not present, so only "A" cannot be a subtree. I hope I clarified this time 🙂
MB
Kartik,
You're correct sets don't have structure or hierarchy, but they do have well defined rules about what is and isn't a subset.
Let's try this. The definition says:
Let T == root1 and S == root2
Break it down statement by statement:
"A subtree of a tree T is a tree S consisting of a node in T"
Does the node A in S exist in T? YES
"and all of its descendants in T"
Do all of the descendants of A in S exist in T? YES
So root2 is a subtree of root1.
Based on that simple definition I don't see how you could logically come to any other conclusion.
I guess will just have to agree to disagree. Best of luck to you.
kartik
@MB:
Read the definition following way:
"A subtree of a tree T is a tree S consisting of a node of T and all of the node's descendants which are there in T"
In the example, S contains A but not descendatnts of A which are there in T.

MB
Kartik
I took a closer look at this and I now concur with you. I came across a couple of alternate definitions that helped me to understand. The first is, “Two trees are equivalent if they both have the same topology and if the objects contained in corresponding nodes are equal” and “The collection of nodes that become unreachable from the root when an edge is cut is called a subtree. In addition the entire tree is considered as a subtree. As a consequence there is one subtree associated with each node in the tree.”
One interesting thing came from this. I don’t think you can use a preorder and postorder traversal string to compare binary trees for equivalence. You need the preorder and inorder, plus the nodes need to be uniquely labeled, no duplicate keys
MB
Kartik,
What I intended to say was that was that I don’t think you can use the inorder and postorder traversals to test binary trees for equivalence, you need to use the preorder and inorder traversals.
Bugaboo
I am confused. Does a sub-tree mean it preserves the structure as well?

```
   A (T)        A (S)
  /               \
 B                 B
```

Is ‘S’ a sub-tree of ‘T’?
– The above code would say it is not (which I feel is correct)
– Doing an in-order and pre-order traversal would say it is (which I feel is incorrect)
MB
Bugaboo,
Yes, subtrees preserve structure, they are topologically equivalent.
You apparently didn’t do the traversals correctly.
Tree T
preorder: a,b
inorder: b,a
Tree S
preorder: a,b
inorder: a,b
Once you have the correct traversals it’s obvious that Tree S is not a subtree of Tree T.