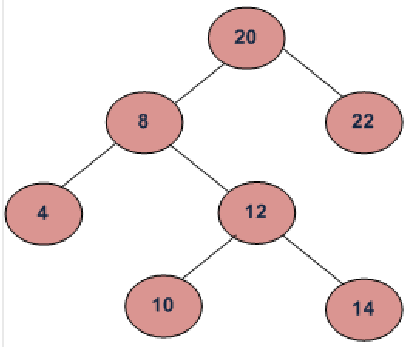
Lowest Common Ancestor in a Binary Search Tree.

Given values of two nodes in a Binary Search Tree, write a c program to find the **L**owest **C**ommon **A**ncestor (LCA). You may assume that both the values exist in the tree.

The function prototype should be as follows:

```
struct node *lca(node* root, int n1, int n2)
n1 and n2 are two given values in the tree with given root.
```



For example, consider the BST in diagram, LCA of 10 and 14 is 12 and LCA of 8 and 14 is 8.

Following is definition of LCA from Wikipedia:
Let T be a rooted tree. The lowest common ancestor between two nodes n1 and n2 is defined as the lowest node in T that has both n1 and n2 as descendants (where we allow a node to be a descendant of itself).

The LCA of n1 and n2 in T is the shared ancestor of n1 and n2 that is located farthest from the root. Computation of lowest common ancestors may be useful, for instance, as part of a procedure for determining the distance between pairs of nodes in a tree: the distance from n1 to n2 can be computed as the distance from the root to n1, plus the distance from the root to n2, minus twice the distance from the root to their lowest common ancestor. (Source Wiki)

Solutions:
If we are given a BST where every node has **parent pointer**, then LCA can be easily determined by traversing up using parent pointer and printing the first intersecting node.

We can solve this problem using BST properties. We can **recursively traverse** the BST from root. The main idea of the solution is, while traversing from top to bottom, the first node n we encounter with value between n1 and n2, i.e., n1 < n < n2 or same as one of the n1 or n2, is LCA of n1 and n2 (assuming that n1 < n2). So just recursively traverse the BST in, if node's value is greater than both n1 and n2 then our LCA lies in left side of the node, if it's is smaller than both n1 and n2, then LCA lies on right side. Otherwise root is LCA (assuming that both n1 and n2 are present in BST)

C

```c
// A recursive C program to find LCA of two nodes n1 and n2.
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node* left, *right;
};

/* Function to find LCA of n1 and n2. The function assumes that both
   n1 and n2 are present in BST */
struct node *lca(struct node* root, int n1, int n2)
{
    if (root == NULL) return NULL;

    // If both n1 and n2 are smaller than root, then LCA lies in left
    if (root->data > n1 && root->data > n2)
        return lca(root->left, n1, n2);

    // If both n1 and n2 are greater than root, then LCA lies in right
    if (root->data < n1 && root->data < n2)
        return lca(root->right, n1, n2);

    return root;
}

/* Helper function that allocates a new node with the given data.*/
struct node* newNode(int data)
{
    struct node* node = (struct node*)malloc(sizeof(struct node));
    node->data  = data;
    node->left  = node->right = NULL;
    return(node);
}

/* Driver program to test lca() */
int main()
{
    // Let us construct the BST shown in the above figure
    struct node *root        = newNode(20);
    root->left               = newNode(8);
    root->right              = newNode(22);
    root->left->left         = newNode(4);
    root->left->right        = newNode(12);
    root->left->right->left  = newNode(10);
    root->left->right->right = newNode(14);

    int n1 = 10, n2 = 14;
    struct node *t = lca(root, n1, n2);
    printf("LCA of %d and %d is %d \n", n1, n2, t->data);

    n1 = 14, n2 = 8;
    t = lca(root, n1, n2);
    printf("LCA of %d and %d is %d \n", n1, n2, t->data);

    n1 = 10, n2 = 22;
    t = lca(root, n1, n2);
    printf("LCA of %d and %d is %d \n", n1, n2, t->data);

    getchar();
    return 0;
}
```

Run on IDE

Java

```
// Recursive Java program to print lca of two nodes

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;

    /* Function to find LCA of n1 and n2. The function assumes that both
     n1 and n2 are present in BST */
    Node lca(Node node, int n1, int n2) {
        if (node == null) {
            return null;
        }

        // If both n1 and n2 are smaller than root, then LCA lies in left
        if (node.data > n1 && node.data > n2) {
            return lca(node.left, n1, n2);
        }

        // If both n1 and n2 are greater than root, then LCA lies in right
        if (node.data < n1 && node.data < n2) {
            return lca(node.right, n1, n2);
        }

        return node;
    }

    public static void main(String args[]) {

        // Let us construct the BST shown in the above figure
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(20);
        tree.root.left = new Node(8);
        tree.root.right = new Node(22);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(12);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(14);

        int n1 = 10, n2 = 14;
        Node t = tree.lca(root, n1, n2);
        System.out.println("LCA of " + n1 + " and " + n2 + " is " + t.data);

        n1 = 14;
        n2 = 8;
        t = tree.lca(root, n1, n2);
        System.out.println("LCA of " + n1 + " and " + n2 + " is " + t.data);

        n1 = 10;
        n2 = 22;
        t = tree.lca(root, n1, n2);
        System.out.println("LCA of " + n1 + " and " + n2 + " is " + t.data);

    }
}

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

Run on IDE

Python

```
# A recursive python program to find LCA of two nodes
# n1 and n2

# 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

# Function to find LCA of n1 and n2. The function assumes
# that both n1 and n2 are present in BST
def lca(root, n1, n2):

    # Base Case
    if root is None:
        return None

    # If both n1 and n2 are smaller than root, then LCA
    # lies in left
    if(root.data > n1 and root.data > n2):
        return lca(root.left, n1, n2)

    # If both n1 and n2 are greater than root, then LCA
    # lies in right
    if(root.data < n1 and root.data < n2):
        return lca(root.right, n1, n2)

    return root

# Driver program to test above function

# Let us construct the BST shown in the figure
root = Node(20)
root.left = Node(8)
root.right = Node(22)
root.left.left = Node(4)
root.left.right = Node(12)
root.left.right.left = Node(10)
root.left.right.right = Node(14)

n1 = 10 ; n2 = 14
```

```
t = lca(root, n1, n2)
print "LCA of %d and %d is %d" %(n1, n2, t.data)

n1 = 14 ; n2 = 8
t = lca(root, n1, n2)
print "LCA of %d and %d is %d" %(n1, n2 , t.data)

n1 = 10 ; n2 = 22
t = lca(root, n1, n2)
print "LCA of %d and %d is %d" %(n1, n2, t.data)

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

Run on IDE

Output:

```
LCA of 10 and 14 is 12
LCA of 14 and 8 is 8
LCA of 10 and 22 is 20
```

Time complexity of above solution is O(h) where h is height of tree. Also, the above solution requires O(h) extra space in function call stack for recursive function calls. We can avoid extra space using **iterative solution**.

```
/* Function to find LCA of n1 and n2. The function assumes that both
   n1 and n2 are present in BST */
struct node *lca(struct node* root, int n1, int n2)
{
    while (root != NULL)
    {
         // If both n1 and n2 are smaller than root, then LCA lies in left
        if (root->data > n1 && root->data > n2)
           root = root->left;

        // If both n1 and n2 are greater than root, then LCA lies in right
        else if (root->data < n1 && root->data < n2)
           root = root->right;

        else break;
    }
    return root;
}
```

Run on IDE

See this for complete program.

You may like to see below articles as well :

Lowest Common Ancestor in a Binary Tree

LCA using Parent Pointer

Find LCA in Binary Tree using RMQ

Exercise

The above functions assume that n1 and n2 both are in BST. If n1 and n2 are not present, then they may return incorrect result. Extend the above solutions to return NULL if n1 or n2 or both not present in BST.

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

189 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- 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?

(Login to Rate and Mark)

2.1 Average Difficulty : **2.1/5.0**
Based on **22** vote(s)

Add to TODO List

Mark as DONE

Like Share 28 people like this.

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

189 Comments **GeeksforGeeks** 1 Login

Recommend 5 Share Sort by Newest

Join the discussion…

Namita Rane • a month ago

We need to consider another corner case.
if (root == n1 || root == n2)
return root;

• Reply • Share ›

Vivek Anand Sampath • a month ago

Java code for find LCA in BST which also checks if the passed in values are actually present.

http://code.geeksforgeeks.org/...

• Reply • Share ›

.NetGeek • 2 months ago

C# Code to find LCA : http://ideone.com/teVS9k

1 • Reply • Share ›

Rahul Devgan • 2 months ago

There is a problem with this code.
it gives null pointer exception when tried with 25 & 30.

Kindly correct :)

• Reply • Share ›

Rahul Shukla • 3 months ago

http://www.ideserve.co.in/lear...
Here is a nice detailed explanation of the algorithm in Java.

3 • Reply • Share ›

Sathiyaseelan • 3 months ago

Cool one.. Simple and effective use of BST property :)

• Reply • Share ›

Shiv Singh • 4 months ago

when it is given n1 and n2 both are present in tree than why they check for (root == NULL) return;

• Reply • Share ›

Saurabh Vats • 4 months ago

At first place, you people should not have modified the arrays and trees categories. If at all then put this one in BST.
Also please do further categorizations only after making some kind of poll from users.

1 • Reply • Share ›

coderr • 6 months ago

```
struct node *lca(struct node* root, node* n1, node * n2)
{
if (root == NULL || !n1 || !n2) return NULL;

if (root->data > n1->data && root->data > n2->data)
return lca(root->left, n1, n2);

if (root->data < n1->data && root->data < n2->data)
return lca(root->right, n1, n2);

return root;
}
```

• Reply • Share ›

Asutosh Satapathy • 6 months ago

'the first node n we encounter with value between n1 and n2' : I think it's not the first node but the last node we encounter. For example, let's insert a node with value 9 between 8 and 20 in the diagram above. In this case, the algorithm will give 9 as the answer which is wrong.

• Reply • Share ›

anonymous → Asutosh Satapathy • 6 months ago

if you insert 9 between 8 and 20 in given binary search tree, then it will no longer remaing binary search tree.

• Reply • Share ›

Asutosh Satapathy → anonymous • 6 months ago

True. I missed that. Thanks for clarifying.

• Reply • Share ›

PrateekSingh Chauhan • 7 months ago

y lca of 8 and 14 is 8??

2 • Reply • Share ›

Abhijit Patra → PrateekSingh Chauhan • 2 months ago

According to definition of the lowest common ancestor (LCA) of two nodes 'n1' and 'n2' in a tree is the lowest node that has both 'n1' and 'n2' as descendants. Each node is considered descendant of itself .So if we will take an example of 8 and 14 , then 8 is as considered own descendant of 8 and for 14 , we can take three descendant 8,12,and 14 itself .
8-->8
14->8,12,14
so common descendant is 8 right ,Hope u will understand the logic.
1 • Reply • Share ›

Srinivas Kothuri PrateekSingh Chauhan • 5 months ago
typo
• Reply • Share ›

devakar verma • 7 months ago
@GeeksforGeeks, @Author
As you are assuming n1 is greater than n2.
Then at time of comparison in if cases, if root->data is greater than n2 it is obvious that it is also greater than n1. In the same way if root->data is less than n1 then it is obvious it is also less than n2. So, please rectify the conditions.
• Reply • Share ›

Bewkoof_coder • 9 months ago
code with only one comparison..with O(h) complexity.
condition's.1)inser the elements first lower then higher one.2)elements must belongs to the tree

```
void LCA(struct node *start,int low,int high){

if(start->data > high)

LCA(start->left,low,high);

else if(start->data < low)

LCA(start->right,low,high);

else

printf("LCA of %d and %d is %d ",low,high,start->data);

}
```

• Reply • Share ›

Dman • 9 months ago
instead of doing two comparisons at every recursive step, we can implement it by using

one comparison only if we we know which key is minimum.

This is implemented in following code.

Edge case is also considered that is if one or more key is not present.

https://ideone.com/Ep1i5l

1 • Reply • Share ›

Bewkoof_coder Dman • 9 months ago

cool.. that is right

• Reply • Share ›

kartikeya • 9 months ago

what is wrong in this code??

struct node *lca(struct node* root, int n1, int n2)

{

if(root==NULL)

return NULL;

if(root->data==n1 || root->data==n2)

return root;

struct node * l,*r;

l=lca(root->left,n1,n2);

r=lca(root->right,n1,n2);

if(l && r)

see more

• Reply • Share ›

Ajcoo • 9 months ago

Isn't the lowest common ancestor of 8 and 14 is 20?

• Reply • Share ›

Rashmi Dewangan Ajcoo • 9 months ago

yeah it shud be according to the word "ancestor" bt here its coded to give ans as 8.

Ih has been mentioned -" we allow a node to be a descendant of itself"

1 • Reply • Share ›

kolo • 9 months ago
i alsu think iterative is best than recursive onez ..

-kolo(polo polo)
roll no.420420
brazil
• Reply • Share ›

Abhishek kolo • 6 months ago
really !! why ??? Recursive makes the implementation so easy , no ??
• Reply • Share ›

KURT WAGNER • 10 months ago
the iterative one is much better
1 • Reply • Share ›

Ankit Marothi • 10 months ago
Here 11 and 13 are not present in the three. Still the LCA for them is printed as 12. Will tweak the code and post it shortly.
• Reply • Share ›

Siya Ankit Marothi • 10 months ago
Because algo is comparing values. If you want that there should be LCA for tree node just check whether that particular node is present or not?
• Reply • Share ›

Karthick Shanmugam • a year ago
The below code works for null values for n1 and n2 as well as finds the lca value

struct node *lca(struct node* root, int n1, int n2){

int l=0;r=0;lca_val=NULL;

while (root != NULL)

{

// If both n1 and n2 are smaller than root, then LCA lies in left

if (root->data > n1 && root->data > n2 && lca_val==NULL)

root = root->left;

// If both n1 and n2 are greater than root, then LCA lies in right else if (root->data < n1 && root->data < n2 && lca_val==NULL)

root = root->right;

see more

• Reply • Share ›

Yagyesh • a year ago

This code works good...check it out...
http://ideone.com/8NNQGv

• Reply • Share ›

Piyush • a year ago

Why lca of 10 & 14 is 12 and not 8?

• Reply • Share ›

kingbhanu Piyush • a year ago

Its because we want to find the lowest node i.e farthest from the root ..not nearest to it

1 • Reply • Share ›

helper • a year ago

my c code for this:
http://ideone.com/G8kzhF

1 • Reply • Share ›

Anil Kumar K K • a year ago

@geeksforgeeks, I think few cases are not handled here
1. What if n1 and n2 are not present ?
2. If n1 or n2 is equal to "node", their ancestor will be parent of "node" right ?

1 • Reply • Share ›

Nishit Anil Kumar K K • a year ago

Anil,
Please read the entire article. The author did mention that he did not cover all the cases and suggested ways on how to handle them.

2 • Reply • Share ›

ryan • 2 years ago

```
int lowestcmnans(nodeptr root,int a,int b)
{
if(root==NULL)
return 0;
int lef=0,rig =0,curr=0;
if(root->left)
lef=lowestcmnans(root->left,a,b);
if(root->right)
rig=lowestcmnans(root->right,a,b);
if(root->data==a||root->data==b)
{
curr=1;
if(lef+rig+curr==2)
{
printf("%d\n",root->data);
return 0;
}
}
else return (rig||lef||curr);
}
```

1 • Reply • Share ›

Scala • 2 years ago

very nice explanation . very simple.

1 • Reply • Share ›

Thanos • 2 years ago

Extend the above solutions to return NULL if n1 or n2 or both not present in BST, using iterative.

---

```
struct node *lca(struct node* root, int n1, int n2)

{

int v1 = 0, v2 = 0;

struct node* p;

while (root)

{

if (root->data > n1 && root->data > n2)

root = root->left;

else if (root->data <n1 &&="" root-="">data < n2)
```

see more

• Reply • Share ›

Joe ➦ Thanos • 2 years ago

You don't need to, because loop will terminate when root reaches null. So if p and q ae not found, it returns null.

• Reply • Share ›

abhinav benarsi • 2 years ago

```
struct node *lca(struct node *root,struct node *n1,struct node *n2)
{
while(1)
{
if((n1->data>root->data&&n2->data<root->data)||
(n1->data<root->data)&&(n2->data>root->data))
return root;
if(n1->data<root->data)
root=root->left;
else
root=root->right;
}
}
```

the above algorithm is also working for skew trees

1 • Reply • Share ›

DS+Algo • 2 years ago

This code returns NULL if n1 or n2 or both not present in BST.

http://ideone.com/lFLgrc

```
bool isPresent(node *root,int key)
{
if(!root)
return false;
while(root)
{
if(root->key>key)
root=root->left;
else if(root->key<key) root="root-">right;
else
break;
}
return root?true:false;
}
```

see more

1 • Reply • Share ›

Jun ➦ DS+Algo • 2 years ago
Perfect...but for this we need to traverse the tree again..can't we do it better??
• Reply • Share ›

Kerem Sahin ➦ Jun • a year ago
I think that it can be done somewhat better. Instead of checking whether n1 and n2 is present in the tree at the start of the algorithm, we can check their existence after finding LCA. However, this also becomes O(h).
• Reply • Share ›

Deepesh Panjabi • 2 years ago
http://ideone.com/1j9OAs
1 • Reply • Share ›

tweety • 2 years ago
extended solution if atleast one node is not present in tree
please check and let me know if something is wrong ??

```
// A recursive C program to find LCA of two nodes n1 and n2.
#include <stdio.h>
#include <stdlib.h>
//#include<conio.h>
struct node
{
int data;
struct node* left, *right;
};

/* Function to find LCA of n1 and n2. The function assumes that both
n1 and n2 are present in BST */
struct node *lca(struct node* root, int n1, int n2)
{
static int v1=0,v2=0,f;
```

see more

3 • Reply • Share ›

Stack ➦ tweety • a year ago
I guess the worst case time complexity of this code is O(n^2). The code works fine though.
• Reply • Share ›

Jun ➦ tweety • 2 years ago

Pefect...As far as I tested..The code works perfectly fine...Thanx....:)

^ | v • Reply • Share ›

sarunac • 2 years ago

If I build the tree like below,

```
struct node *root = newNode(20);
root->left = newNode(10);
root->right = newNode(22);
root->left->left = newNode(8);
root->left->left->left = newNode(4);
root->left->left->right = newNode(12);
root->left->left->right->right = newNode(14);
```

Pictorially, I have just taken the 10 and inserted it to be a parent of 8 and left child of 20.

This is still a valid BST and the LCA for this tree, according to the above algorithm seems to return 10 , instead of 8. Am I overlooking some very important property of BST or LCA? It would be really helpful, if anyone could shed some light on this.

^ | v • Reply • Share ›

kradmour ➦ sarunac • 2 years ago

You can't have 12 and 14 in the left sub tree of 10. That violates the structural property of the BST.

1 ^ | v • Reply • Share ›

sarunac ➦ kradmour • 2 years ago

No, 12 and 14 are on the right subtrees of 10. My tree is the same as the one told in the question, but with just 10 inserted between 8 and 20.

^ | v • Reply • Share ›

DS+Algo ➦ sarunac • 2 years ago

12 and 14 are in right subtree of 8
and 8 is in left subtree of 10,
So basically 12 and 14 are in left subtree of 10
But in BST every node in left subtree must be less than root and every node in right subtree must be larger than root.

And remember there is only one BST present for a group of nodes and i.e. its inorder must be in ascending order

Hope u understood.
otherwise feel free to ask

^ | v • Reply • Share ›

Load more comments