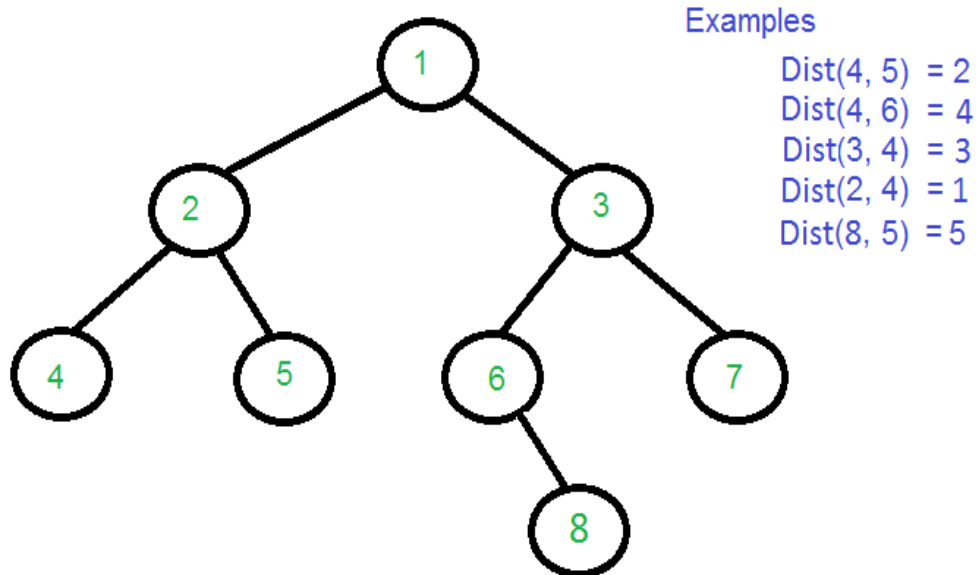


## Find distance between two given keys of a Binary Tree

Find the distance between two keys in a binary tree, no parent pointers are given. Distance between two nodes is the minimum number of edges to be traversed to reach one node from other.



**We strongly recommend to minimize the browser and try this yourself first.**

The distance between two nodes can be obtained in terms of **lowest common ancestor**. Following is the formula.

$$\text{Dist}(n1, n2) = \text{Dist}(\text{root}, n1) + \text{Dist}(\text{root}, n2) - 2 * \text{Dist}(\text{root}, \text{lca})$$

'n1' and 'n2' are the two given keys

'root' is root of given Binary Tree.

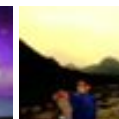
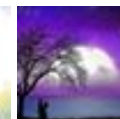
Google™ Custom Search



GeeksforGeeks



52,731 people like GeeksforGeeks.



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

'lca' is lowest common ancestor of n1 and n2  
Dist(n1, n2) is the distance between n1 and n2.

Following is C++ implementation of above approach. The implementation is adopted from last code provided in [Lowest Common Ancestor Post](#).

```
/* Program to find distance between n1 and n2 using one traversal */
#include <iostream>
using namespace std;

// A Binary Tree Node
struct Node
{
    struct Node *left, *right;
    int key;
};

// Utility function to create a new tree Node
Node* newNode(int key)
{
    Node *temp = new Node;
    temp->key = key;
    temp->left = temp->right = NULL;
    return temp;
}

// Returns level of key k if it is present in tree, otherwise returns -1
int findLevel(Node *root, int k, int level)
{
    // Base Case
    if (root == NULL)
        return -1;

    // If key is present at root, or in left subtree or right subtree,
    // return true;
    if (root->key == k)
        return level;

    int l = findLevel(root->left, k, level+1);
    return (l != -1)? l : findLevel(root->right, k, level+1);
}

// This function returns pointer to LCA of two given values n1 and n2.
// It also sets d1, d2 and dist if one key is not ancestor of other
// d1 --> To store distance of n1 from root
// d2 --> To store distance of n2 from root
```

# ITT Tech - Official Site

[itt-tech.edu](http://itt-tech.edu)

Associate, Bachelor Degree  
Programs Browse Programs Now &  
Learn More.

## Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)

```

// lvl --> Level (or distance from root) of current node
// dist --> To store distance between n1 and n2
Node *findDistUtil(Node* root, int n1, int n2, int &d1, int &d2,
                  int &dist, int lvl)
{
    // Base case
    if (root == NULL) return NULL;

    // If either n1 or n2 matches with root's key, report
    // the presence by returning root (Note that if a key is
    // ancestor of other, then the ancestor key becomes LCA
    if (root->key == n1)
    {
        d1 = lvl;
        return root;
    }
    if (root->key == n2)
    {
        d2 = lvl;
        return root;
    }

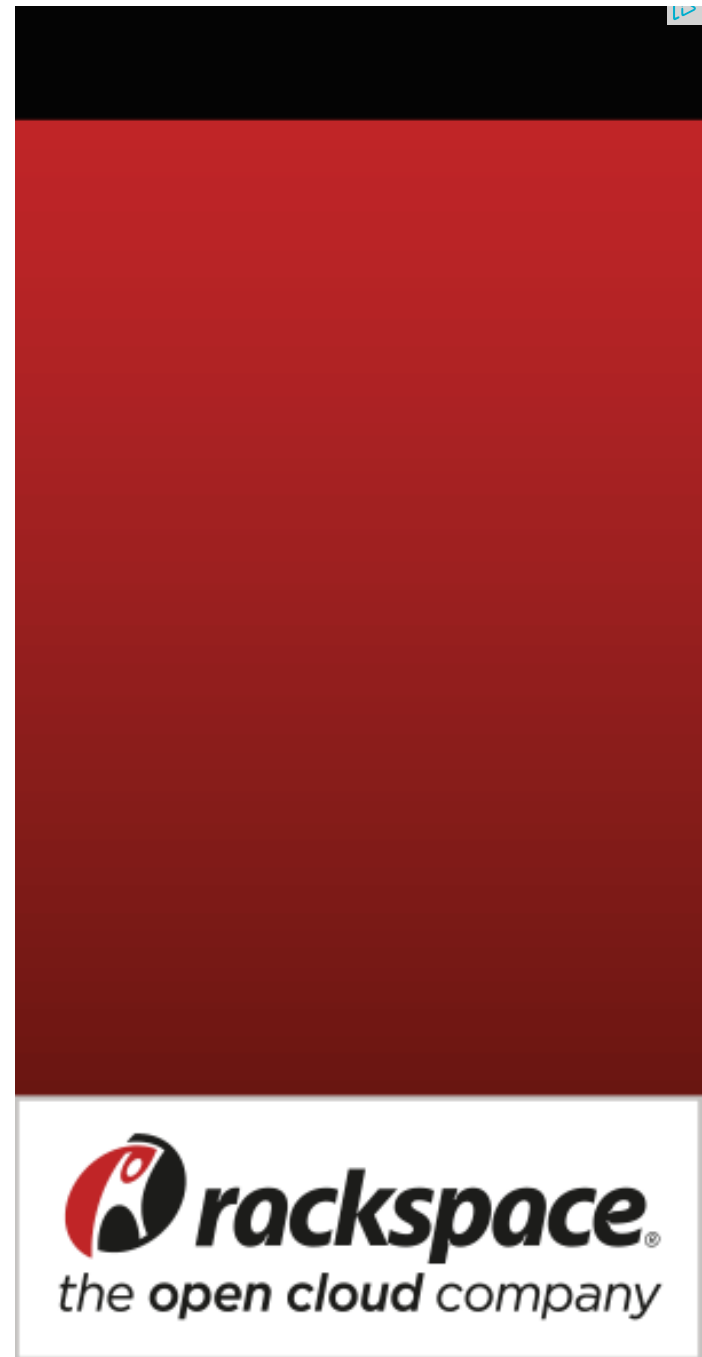
    // Look for n1 and n2 in left and right subtrees
    Node *left_lca = findDistUtil(root->left, n1, n2, d1, d2, dist, lvl+1);
    Node *right_lca = findDistUtil(root->right, n1, n2, d1, d2, dist, lvl+1);

    // If both of the above calls return Non-NULL, then one key
    // is present in once subtree and other is present in other,
    // So this node is the LCA
    if (left_lca && right_lca)
    {
        dist = d1 + d2 - 2*lvl;
        return root;
    }

    // Otherwise check if left subtree or right subtree is LCA
    return (left_lca != NULL)? left_lca: right_lca;
}

// The main function that returns distance between n1 and n2
// This function returns -1 if either n1 or n2 is not present in
// Binary Tree.
int findDistance(Node *root, int n1, int n2)
{
    // Initialize d1 (distance of n1 from root), d2 (distance of n2
    // from root) and dist(distance between n1 and n2)
    int d1 = -1, d2 = -1, dist;

```



## Recent Comments

affiszerv Your example has two 4s on row 3, that's why it..

Backtracking | Set 7 (Sudoku) · 27 minutes ago

**RVM** Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 47 minutes ago

**Vishal Gupta** I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 47 minutes ago

**@meya** Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node \*head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 2 hours ago

AdChoices

► [Binary Tree](#)

► [Graph C++](#)

► [Java Tree](#)

AdChoices

```
Node *lca = findDistUtil(root, n1, n2, d1, d2, dist, 1);

// If both n1 and n2 were present in Binary Tree, return dist
if (d1 != -1 && d2 != -1)
    return dist;

// If n1 is ancestor of n2, consider n1 as root and find level
// of n2 in subtree rooted with n1
if (d1 != -1)
{
    dist = findLevel(lca, n2, 0);
    return dist;
}

// If n2 is ancestor of n1, consider n2 as root and find level
// of n1 in subtree rooted with n2
if (d2 != -1)
{
    dist = findLevel(lca, n1, 0);
    return dist;
}

return -1;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree given in the above example
    Node * root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    cout << "Dist(4, 5) = " << findDistance(root, 4, 5);
    cout << "\nDist(4, 6) = " << findDistance(root, 4, 6);
    cout << "\nDist(3, 4) = " << findDistance(root, 3, 4);
    cout << "\nDist(2, 4) = " << findDistance(root, 2, 4);
    cout << "\nDist(8, 5) = " << findDistance(root, 8, 5);
    return 0;
}
```


Output:

```
Dist(4, 5) = 2
Dist(4, 6) = 4
Dist(3, 4) = 3
Dist(2, 4) = 1
Dist(8, 5) = 5
```

**Time Complexity:** Time complexity of the above solution is  $O(n)$  as the method does a single tree traversal.

Thanks to **Atul Singh** for providing the initial solution for this post.

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

- [▶ Java to C++](#)
- [▶ Distance Formula](#)
- [▶ Find Distance](#)
- AdChoices 
- [▶ Find Distance](#)
- [▶ Tree Root](#)
- [▶ Nodes](#)



## Related Tpoics:

- [Print a Binary Tree in Vertical Order | Set 2 \(HashMap based Method\)](#)
- [Print Right View of a Binary Tree](#)
- [Red-Black Tree | Set 3 \(Delete\)](#)
- [Construct a tree from Inorder and Level order traversals](#)

- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree



18



3



1

Writing code in comment? Please use [ideone.com](https://ideone.com) and share the link here.

21 Comments

GeeksforGeeks

Sort by Newest ▼



Join the discussion...



**prashant jha** · 3 days ago

find the lowest common ancestors of the given two nodes and then find the dis

```
#include<iostream>
```

```
using namespace std;
```

```
#define max 999
```

```
struct tnode
```

```
{
```

```
tnode* lchild;
```

```
int data;
```

```
tnode* rchild;
```

```
tnode(int d)
```

{

see more

^ | v • Reply • Share ›



**joud zouzou** • 20 days ago

Anything wrong with this solution?:

/\*

a: first node

b: second node

l\_a: level of first node

l\_b: level of second node

ans: distance between node a and node b

\*/

while(a!=b)

{

if (l\_b > l\_a) // move to parent of b

{

l\_b--;

b/=2;

ans++;

}

else if (l\_a > l\_b) // move to parent of a

{

see more

^ | v • Reply • Share ›



**sunil** • a month ago

This is the same question as the diameter question

^ | v • Reply • Share ›

piyush ag

a month ago



**piyush.ag** · a month ago

A better way ::

```

int kDistanceChildNode(Node *root , Node *node , int level) {
if (root == NULL)
return -1;

if (root == node)
return level;

int l = kDistanceChildNode(root->left , node , level+1);

if (l!=-1)
return l;

else
return kDistanceChildNode(root->right , node , level+1);
}

Node *lca(Node *root , Node *node1 , Node *node2) {
if (root == NULL || l )

```

[see more](#)

^ | v · Reply · Share ›



**prashant jha** · a month ago

find lca and get the distance from lca to both nodes  
below standard ques

^ | v · Reply · Share ›



**gaurav sachdeva** · 2 months ago

I found a better way to do this. Idea is keep trying to find the nos from left and r  
either no is found.

Writing pseudo code:-

```

function distance(Node root, Node N1, int n1, int n2, int count) {

```



```

function distance(Node root, Node n1, int n1, int n2, int count) {
    if(!N) return 0; if(N->val == n1 || N->val == n2) return 1;
    int left = distance(root, N->left, a, b, count);
    int right = distance(root, N->right, a, b, count);
    if(left+right == 1) {
        //here 1st no is found and we are searching for 2nd
        count++;
        return 1;
    }
    if(left + right == 2) {
        //here second no is found
        return count+1;
    }
    return 0;
}

```

And we can call distance like:-

```
print(distance(root,root,n1,n2,0));
```

I suppose this is way easier and is  $O(n)$ .

Please update incase I missed something.

^ | v • Reply • Share ›



**Aiden** • 2 months ago

if both nodes have same val,the method above will not work

^ | v • Reply • Share ›



**Atul Anand** • 2 months ago

another method :-

1)first find LCA of n1 and n2.

2)a=find distance of n1 from lca.

3)b=find distance of n2 from lca.

4) return a+b;

^ | v • Reply • Share ›



**new\_coder** • 2 months ago

Please let me know if this algo makes any sense.

We can do level order traversal to find the difference between the keys..

Algo:-

1. If root is one of the two keys. Find the other key in left subtree and right subtree and the other key between them.

2. Else: Traverse left subtree and right subtree.

If both the keys are in the same subtree then check their levels.

If they are on the same level, return distance=2.

else return abs(difference in their levels).

else (if both the keys are in different subtree)

calculate at which level they are and return (level\_leftSubtree + level\_rightSubtree)

Take root at level=0

Thanks.

^ | v • Reply • Share ›



**zealfire** • 2 months ago

can the formula be this: if both nodes are left and right child of lca then add distance. if they are on same side of lca then then : |distance of one node from lca - distance of other node from lca|  
this will be right or wrong

^ | v • Reply • Share ›



**Guest** • 2 months ago

```
int distance( tree* root , int p1 , int p2 )
```

```
{
```

```

if ( !root )
return 0;
int d1;
int d2;
bool p1_found = find_distance( root , p1, &d1);

if( !p1_found)
return 0; /// node does not exist
bool p2_found = find_disatnce( root , p2 , &d2 );
if( !p2_found)
return 0; // the value does not exist in tree..

d1 = d1 - d2;
if ( d1 )
return d1;
return (-d1);
,

```

[see more](#)

^ | v • Reply • Share ›



**micintosh** • 2 months ago

If there exists duplicates in the tree, the algorithm does not work well; consider

```

Node * root = newNode(1);
root->left = newNode(2);
root->right = newNode(2);
root->left->left = newNode(4);
root->left->right = newNode(5);
root->right->left = newNode(6);
root->right->right = newNode(7);
root->right->left->right = newNode(8);
cout << "Dist(2, 8) = " << findDistance(root, 2, 8);

```

Dist(2, 8) should be 2 or 4; but the algorithm above returns 3.

The reason is that "If both of the above calls return Non-NULL, then one key is present in other" is wrong.

To correct the algorithm, you may need to first get lca, and then return findLev

^ | v • Reply • Share ›



**Rahul** • 2 months ago

If we modify the node structure a little bit, we can make this problem more eas

So Here are my changes

1. in node sturcture add level and pointer to parent. while creating node assign

2. search for both keys, store the reference in curr1 and curr2.

3. If both nodes curr1 and curr2 are not null, it means both node exist, otherwis exists.

3. take a new variable dist as 0.

4. Now run following code

```
temp1= curr1;
```

```
temp2 = curr2;
```

```
while(curr1-> level != curr2->level)
```

---

see more

^ | v • Reply • Share ›



**Guest** • 2 months ago



If we modify the node structure a little bit, we can make this problem more eas

So Here are my changes

1. in node sturcture add level and pointer to parent. while creating node assign
2. search for both keys, store the reference in curr1 and curr2.
3. If both nodes curr1 and curr2 are not null, it means both node exist, otherwis exists.

3. take a new variable dist as 0.

4. Now run following code

```
temp1= curr1;
temp2 = curr2;
while(curr1-> level != curr2->level)
{
if(curr1->level > curr2->level)
curr1 = curr1->parent;
else
curr2 = curr2->parent;
dist++;
}
```

5. finally return distance = (temp1->level) + (temp2->level) + dist

Let me know if you find any problem with this approach.

^ | v • Reply • Share ›



**blackball** • 2 months ago

Because there's only one path from one key to another, so we could just trave when we found one of the keys, and end the counting when we found the othe

```
#include <iostream>
```

```
using namespace std;
```

```
struct TreeNode {
```

int main()

```
int val;
```

```
TreeNode *left, *right;
```

```
explicit TreeNode(int v):val(v),left(NULL),right(NULL){}
```

```
};
```

```
class Solution {
```

```
public:
```

[see more](#)

^ | v • Reply • Share ›



**xxmajia** → blackball • 2 months ago

yes, but instead of adding one more field in the data structure, you can parent pointer, and it can be generate at  $O(N)$  time with extra  $O(N)$  spa

^ | v • Reply • Share ›



**Vinay Singh** • 2 months ago

i didn't understand this code...can anyone simplify this problem.

^ | v • Reply • Share ›



**varun** • 2 months ago

nice post

^ | v • Reply • Share ›



**sumit** • 2 months ago

```
#include <cstdlib>
```

```
#include <iostream>
```

```
#include <conio.h>
```

```
using namespace std;
```

```
using namespace std;
```

```
struct tree
```

```
{
```

```
struct tree* left;
```

```
int item;
```

```
struct tree* right;
```

```
};
```

```
tree* newNode(int key)
```

```
{
```

```
tree *temp = new tree;
```

```
temp->item = key;
```

```
temp->left = temp->right = NULL;
```

```
return temp;
```

[see more](#)

^ | v • Reply • Share ›



**xxmajia** • 2 months ago

Thanks for sharing

To improve this algo, i think we can use the LCA algo using extra space to rec

^ | v • Reply • Share ›



**Atul** • 2 months ago

Instead of calculating dist every time, do the following using flag passed by ref

```
if (left_lca && right_lca)
```

```
{
```

```
if(flag)
```

```
{
```

```
dist = d1 + d2;
```

```
flag=false;
```

```
return root;
```

```
return root;  
}  
else  
return root;  
}
```

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

**Contact Us!**

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team