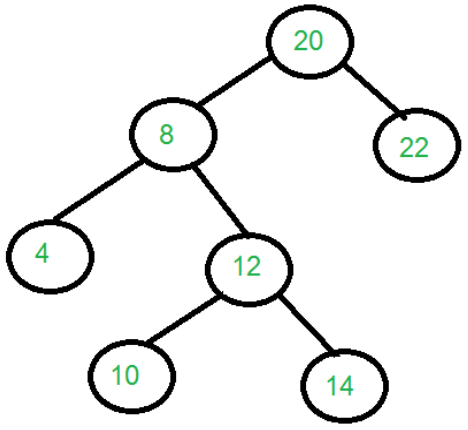
Print all nodes at distance k from a given node

Given a binary tree, a target node in the binary tree, and an integer value k, print all the nodes that are at distance k from the given target node. No parent pointers are available.

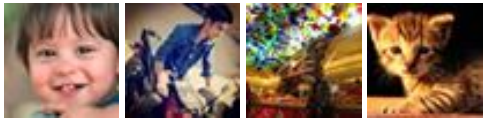



```
Consider the tree shown in diagram

Input: target = pointer to node with data 8.
       root = pointer to node with data 20.
       k = 2.
Output : 10 14 22

If target is 14 and k is 3, then output
should be "4 20"
```

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

There are two types of nodes to be considered.
1) Nodes in the subtree rooted with target node. For example if the target node is 8 and k is 2, then such nodes are 10 and 14.
2) Other nodes, may be an ancestor of target, or a node in some other subtree. For target node 8 and k is 2, the node 22 comes in this category.

Finding the first type of nodes is easy to implement. Just traverse subtrees rooted with the target node and decrement k in recursive call. When the k becomes 0, print the node currently being traversed (See this for more details). Here we call the function as *printkdistanceNodeDown()*.



How to find nodes of second type? For the output nodes not lying in the subtree with the target node as the root, we must go through all ancestors. For every ancestor, we find its distance from target node, let the distance be d, now we go to other subtree (if target was found in left subtree, then we go to right subtree and vice versa) of the ancestor and find all nodes at k-d distance from the ancestor.

Following is C++ implementation of the above approach.

```
#include <iostream>
using namespace std;

// A binary Tree node
struct node
{
    int data;
    struct node *left, *right;
};

/* Recursive function to print all the nodes at distance k in the
   tree (or subtree) rooted with given root. See  */
void printkdistanceNodeDown(node *root, int k)
{
    // Base Case
    if (root == NULL || k < 0)  return;

    // If we reach a k distant node, print it
    if (k==0)
    {
        cout << root->data << endl;
        return;
    }

    // Recur for left and right subtrees
    printkdistanceNodeDown(root->left, k-1);
    printkdistanceNodeDown(root->right, k-1);
}

// Prints all nodes at distance k from a given target node.
// The k distant nodes may be upward or downward.  This function
// Returns distance of root from target node, it returns -1 if target
// node is not present in tree rooted with root.
int printkdistanceNode(node* root, node* target , int k)
{
    // Base Case 1: If tree is empty, return -1
    if (root == NULL) return -1;
```


Popular Posts

```
    // If target is same as root.  Use the downward function
    // to print all nodes at distance k in subtree rooted with
    // target or root
    if (root == target)
    {
        printkdistanceNodeDown(root, k);
        return 0;
    }

    // Recur for left subtree
    int dl = printkdistanceNode(root->left, target, k);

    // Check if target node was found in left subtree
    if (dl != -1)
    {
         // If root is at distance k from target, print root
         // Note that dl is Distance of root's left child from target
         if (dl + 1 == k)
            cout << root->data << endl;

         // Else go to right subtree and print all k-dl-2 distant node
         // Note that the right child is 2 edges away from left child
         else
            printkdistanceNodeDown(root->right, k-dl-2);

         // Add 1 to the distance and return value for parent calls
         return 1 + dl;
    }

    // MIRROR OF ABOVE CODE FOR RIGHT SUBTREE
    // Note that we reach here only when node was not found in left su
    int dr = printkdistanceNode(root->right, target, k);
    if (dr != -1)
    {
         if (dr + 1 == k)
            cout << root->data << endl;
         else
            printkdistanceNodeDown(root->left, k-dr-2);
         return 1 + dr;
    }

    // If target was neither present in left nor in right subtree
    return -1;
}

// A utility function to create a new binary tree node
```

```
node *newnode(int data)
{
    node *temp = new node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    /* Let us construct the tree shown in above diagram */
    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);
    node * target = root->left->right;
    printkdistanceNode(root, target, 2);
    return 0;
}
```

Output:

```
4
20
```

Time Complexity: At first look the time complexity looks more than O(n), but if we take a closer look, we can observe that no node is traversed more than twice. Therefore the time complexity is O(n).

This article is contributed by **Prasant Kumar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recent Comments

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

Backtracking | Set 7 (Sudoku) · 25 minutes ago

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

Flipkart Interview | Set 6 · 45 minutes ago

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

Software Engineering Lab, Samsung Interview | Set 2 · 45 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

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

Writing code in comment? Please use **ideone.com** and share the link here.

14 Comments GeeksforGeeks

Sort by Newest

prashant jha • 3 days ago

push the ancestors of the node in the stack and get the distances from each a
node

```
#include<iostream>
#define size 50
using namespace std;
struct tnode
{
tnode* lchild;
int data;
tnode* rchild;
tnode(int d)
{
lchild=NULL;
data=d;
rchild=NULL;
}
};
```

see more

• Reply • Share ›

AlienOnEarth • 4 days ago

GeeksforGeeks:

This is another approach to print nodes at k distance from leaf. This solution h
time complexity and is similar to Print nodes at k distance from target node. B
you to kindly consider this.

```
int printkdistanceNode(struct node *root, int k)
{
// Base Case 1: If tree is empty, return -1
if (root == NULL) return -1;
if (isLeaf(root))
{
return 0;
}
```

see more

• Reply • Share ›

Ankit Jain • 14 days ago

```
/*Print vertical tree*/
#include<stdio.h>
#include<stdlib.h>

struct BTree
{
int data;
struct BTree *left;
struct BTree *right;
int flag;
};
struct BTree* insert(struct BTree *root,int data)
{
struct BTree *temp=root;
if(temp==NULL)
```

```
{
temp=(struct BTree*)malloc(sizeof(struct BTree));
temp->data=data;
```

see more

Reply • Share ›

algo1 • 16 days ago

Someone please help me understand this

Reply • Share ›

Ravi Kiran • 22 days ago

->find the height of given node
->height of given node -/+ height of other node = distance
then print node

```
void printNodeatDistanceK(bnode* root,int givenNodesHeight,int level,int distar
{
if(root== NULL)
return;

if((level-givenNodesHeight == distance) || (level+givenNodesHeight == distanc
{
//print node
}

printNodeatDistanceK(root->left,givenNodesHeight,level+1,distance);
printNodeatDistanceK(root->right,givenNodesHeight,level+1,distance);

}
```

see more

Hiccup • a month ago

---

tree.h

```
#include <iostream>

#ifndef TREE_H

#define TREE_H

using namespace std;

struct Node {

int key;

Node *left , *right;

Node() : key(-1), left(NULL), right(NULL) {
```

see more

• Reply • Share ›

Siva Krishna • a month ago

We can do like this...
Dist(a, b) = distance between two nodes a, b
lca(a, b) = least common ancestor of a, b

Dist(Node, target) = Dist(root, Node) + Dist(root, target) - 2 * Dist(root, lca(No

for every node if Dist(Node, target) == k then print Node

Reply • Share ›

Guest ➚ Siva Krishna • a month ago

What is a and b here. How do you find it?

Reply • Share ›

Siva Krishna ➚ Guest • a month ago

a and b can be any nodes. Dist(root, Node) can be find by trave
LCA of two nodes can be found using any standard approach.

Reply • Share ›

Jothi • a month ago

Thanks for posting this Solution!
I have a question...
int dl = printkdistanceNode(root->left, target, k);

I dont understand this part. dl will be always (-1) or 0 in the recursive calls. So
target from the node. For example, If k=2 and the target is two levels below roo
k) would be true, right? Please correct be if I am wrong.

Reply • Share ›

GeeksforGeeks Mod ➚ Jothi • a month ago

Please take a closer look, the function also returns dl+1 and dr+1 that
than 1, if the ancestor is higher than 1 edge.

1 Reply • Share ›

piyush.ag • a month ago

This one is iterative approach.

```
#include <iostream>

#include <stack>
```

```
#include <stdlib.h>

using namespace std;

struct node {

int data;

struct node* left;

struct node* right;

};

typedef struct node Node;
```

see more

Reply • Share ›

Kiran • a month ago

Good one, thanks

Reply • Share ›

raja • a month ago

good one

Reply • Share ›

Subscribe Add Disqus to your site