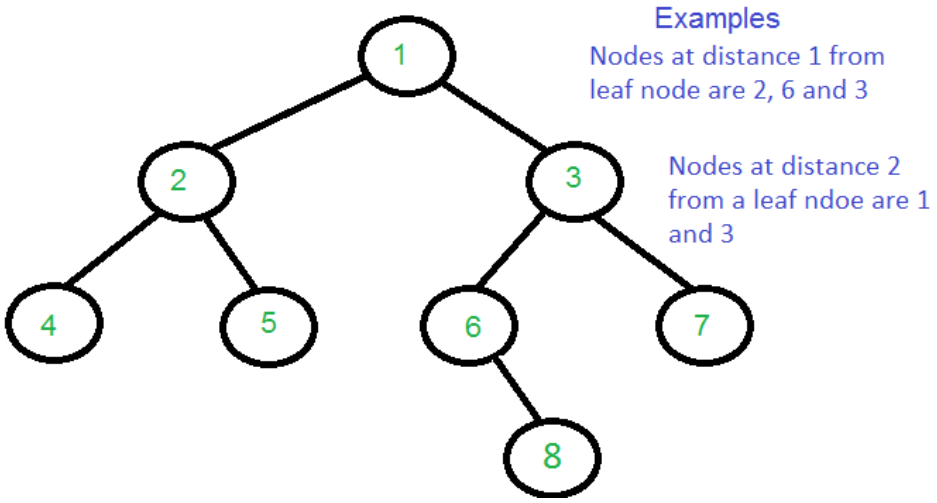
Print all nodes that are at distance k from a leaf node

Given a Binary Tree and a positive integer k, print all nodes that are distance k from a leaf node.

Here the meaning of distance is different from previous post. Here k distance from a leaf means k levels higher than a leaf node. For example if k is more than height of Binary Tree, then nothing should be printed. Expected time complexity is O(n) where n is the number nodes in the given Binary Tree.



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

The idea is to traverse the tree. Keep storing all ancestors till we hit a leaf node. When we reach a leaf node, we print the ancestor at distance k. We also need to keep track of nodes that are already printed as output. For that we use a boolean array visited[].

C++

```
/* Program to print all nodes which are at distance k from a leaf */
#include <iostream>
using namespace std;
#define MAX_HEIGHT 10000

struct Node
{
```

```
    int key;
    Node *left, *right;
};

/* utility that allocates a new Node with the given key  */
Node* newNode(int key)
{
    Node* node = new Node;
    node->key = key;
    node->left = node->right = NULL;
    return (node);
}

/* This function prints all nodes that are distance k from a leaf node
   path[] --> Store ancestors of a node
   visited[] --> Stores true if a node is printed as output.  A node may be k
                distance away from many leaves, we want to print it once */
void kDistantFromLeafUtil(Node* node, int path[], bool visited[],
                          int pathLen, int k)
{
    // Base case
    if (node==NULL) return;

    /* append this Node to the path array */
    path[pathLen] = node->key;
    visited[pathLen] = false;
    pathLen++;

    /* it's a leaf, so print the ancestor at distance k only
       if the ancestor is not already printed  */
    if (node->left == NULL && node->right == NULL &&
        pathLen-k-1 >= 0 && visited[pathLen-k-1] == false)
    {
        cout << path[pathLen-k-1] << " ";
        visited[pathLen-k-1] = true;
        return;
    }

    /* If not leaf node, recur for left and right subtrees */
    kDistantFromLeafUtil(node->left, path, visited, pathLen, k);
    kDistantFromLeafUtil(node->right, path, visited, pathLen, k);
}

/* Given a binary tree and a nuber k, print all nodes that are k
   distant from a leaf*/
void printKDistantfromLeaf(Node* node, int k)
{
    int path[MAX_HEIGHT];
    bool visited[MAX_HEIGHT] = {false};
    kDistantFromLeafUtil(node, path, visited, 0, k);
```

```
}

/* 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 << "Nodes at distance 2 are: ";
    printKDistantfromLeaf(root, 2);

    return 0;
}
```

Java

```
// Java program to print all nodes at a distance k from leaf
// A binary tree node
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    static Node root;

    /* This function prints all nodes that are distance k from a leaf node
     path[] --> Store ancestors of a node
     visited[] --> Stores true if a node is printed as output.  A node may
     be k distance away from many leaves, we want to print it once */
    void kDistantFromLeafUtil(Node node, int path[], boolean visited[],
                              int pathLen, int k)
    {
        // Base case
```

```
        if (node == null)
            return;

        /* append this Node to the path array */
        path[pathLen] = node.data;
        visited[pathLen] = false;
        pathLen++;

        /* it's a leaf, so print the ancestor at distance k only
         if the ancestor is not already printed  */
        if (node.left == null && node.right == null
           && pathLen - k - 1 >= 0 && visited[pathLen - k - 1] == false)
        {
            System.out.print(path[pathLen - k - 1] + " ");
            visited[pathLen - k - 1] = true;
            return;
        }

        /* If not leaf node, recur for left and right subtrees */
        kDistantFromLeafUtil(node.left, path, visited, pathLen, k);
        kDistantFromLeafUtil(node.right, path, visited, pathLen, k);
    }

    /* Given a binary tree and a nuber k, print all nodes that are k
     distant from a leaf*/
    void printKDistantfromLeaf(Node node, int k)
    {
        int path[] = new int[1000];
        boolean visited[] = new boolean[1000];
        kDistantFromLeafUtil(node, path, visited, 0, k);
    }

    // Driver program to test the above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();

        /* Let us construct the tree shown in above diagram */
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.right.left.right = new Node(8);

        System.out.println(" Nodes at distance 2 are :");
        tree.printKDistantfromLeaf(root, 2);
    }
```

```
}

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

Output:

```
Nodes at distance 2 are: 3 1
```

Time Complexity: Time Complexity of above code is O(n) as the code does a simple tree traversal.

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

79 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)

3.4 Average Difficulty : **3.4/5.0**
Based on **9** 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.

132 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

Gourav Goswami • 2 months ago

```
#include "stdafx.h"

#include <iostream>

using namespace std;

#define MAX_HEIGHT 10000

#include<conio.h>

struct Node

{

int key;

Node *left, *right;

};

Node* newNode(int key)
```

see more

• Reply • Share ›

Hiccup • 5 months ago

@GeeksForGeeks
Somewhat better Code which uses only path array.

https://app.box.com/s/797q6l0c...

• Reply • Share ›

xudifsd • 5 months ago

it's not necessary to use 2 arrays, just do not print on leaf node, see https://gist.github.com/xudifs...

• Reply • Share ›

nitin • 8 months ago

Better solution, does not require flags to store path and isVisited information.

```
int kDistantFromLeafUtil(Node* root, int k)
{
if (root==NULL)
return 0;
if (root->left == NULL && root->right == NULL)
return 0;

int d1 = kDistantFromLeafUtil(root->left, k);
int d2 = kDistantFromLeafUtil(root->right, k);

if (d1+1 == k || d2+1 ==k) {
cout<<root->key<<" ";
}

int min = d1;
if (min > d2)
min = d2;

return min+1;
}
```

• Reply • Share ›

Gaurav Narula ➚ nitin • 3 months ago

A node may have different distance from root for right and left subtrees. Won't work for k = 2 in the given example

• Reply • Share ›

xudifsd ➚ nitin • 5 months ago

I don't think this is correct if k > 2

• Reply • Share ›

Sushanto • 8 months ago

There's no need to remember the path.

We can use Bit technic.

1. Say k=3,int dist=1<<k 2.="" we="" pass="" 'dist'="" to="" the="" leaf="" node="" recursively="" and="" wait="" for="" its="" return="" value(="" i.e,="" val="recur(root->leftChild) | recur(root->righChild))
3. if root is a leaf node, return dist>>1(i.e, we decrease the level)
4. if root is not a leaf node, we get 'val' from its leftchild and rightchild,
4.1. if val&1==true, then print the node and then return val>>1(decrease level to ancestor)

For the example above:
k=2, so dist= 0100
Node 1: Not leaf, Val=recur(2,0100) | recur(3,0100)
Node2: Not leaf, val=recur(4,0100) | recur(5,0100)
Node3: Not leaf, val=recur(6,0100) | recur(7,0100)
Node4: Leaf, return 0010(0100>>1)
Node5: Leaf, return 0010
Node6: Not leaf, val=recur(NULL, 0100) | recur(8,0100)

see more

• Reply • Share ›

Abhishek • 9 months ago

This can even be easily done by iterative approach by performing a Breadth first search and maintaining parent array.

1) Perform BFS.
2) When pushing the node into queue
update its parent.
if v is the dequed element and we are examining the left u1and right u2 children of node v, then

parent[u1] = v;
parent[u2] = v;

when deque-ing an element from queue.
e.g if V is dequeued from queue,
then check if its left and right child are null.
then

print the parent from parent array (parent[]) which is at distance k from V.

```
int dist = K(given);
int p =V;
while(dist--)
{
p = parent[p]; // V is dequeued node.
}
```

}
p would be the node at distance k from V. i.e leaf node. Hence Print it.
cout<<p<<endl; recursive="" calls="" are="" always="" expensive.!!!="">

• Reply • Share ›

Anup Rai • 9 months ago

In this question are we assuming that we only have to take the distance upwards because there may be cases where the leaf at right subtree is at a distance k-1 from the root that would make the distance of the leaf from the left child of the root to be a valid candidate to be printed. In the solution provided it is not considered to be a candidate.

• Reply • Share ›

vergil ➚ Anup Rai • 9 months ago

yes...it is mentioned at the start of the topic that we only consider above

• Reply • Share ›

SiSi • 9 months ago

without using boolean... jst print the keys of the map used here

```
void kdistanceNodes(struct Node *root ,int *v,int in,int k,map<int,int> &mp)
{
if(root==NULL)
return;
v[in]=root->data;
in++;
if(root->left==NULL && root->right==NULL)
{
mp[v[in-k-1]]++;
return;
}
kdistanceNodes(root->left,v,in,k,mp);
kdistanceNodes(root->right,v,in,k,mp);
}
```

• Reply • Share ›

OURJAY • 10 months ago

here k is less than height of the tree right???

• Reply • Share ›

Santosh Testuser • a year ago

```
int distanceFromLeaf(Node node,int distance)

{

if(node==null)
```

if(node == null)

{

return -1;

}

if(node.left==null && node.right==null)

{

return 0;

}

int lh=1+distanceFromLeaf(node.left,distance) ;

see more

• Reply • Share ›

neelabhsingh • a year ago

Please see following link , It is explain very clearly..
http://algorithmanddatastructu...

• Reply • Share ›

Guest • a year ago

@GeeksforGeeks your approach only prints ancestors of a leaf at distance k. it won't print nodes in the subtree of leaf!!

try this case:-

output should be 1,6,8 instead of 1,8

Node * root = newNode(1);

root->right = newNode(3);

root->right->left = newNode(6);

root->right->right = newNode(7);

root->right->left->right = newNode(8);

root->right->left->right->right = newNode(18);

root->right->left->right->right->right = newNode(20);

so, only approach left is this http://www.geeksforgeeks.org/p...

• Reply • Share ›

vergil Guest • 9 months ago

subtree of leaves....what does that mean...?

• Reply • Share ›

Zeng Qiang • a year ago

My solution using Java. the HashSet is used to store all possible height of the root. This is a bottom up solution, but iterate the hashset cost a lot of time.

```
public static HashSet<integer> printLevelK(TreeNode root,int k){

HashSet<integer> res = new HashSet<integer>();

res.add(0);

if(root == null) return res;

HashSet<integer> left = printLevelK(root.left,k);

HashSet<integer> right = printLevelK(root.right,k);

if(left.contains(k) || right.contains(k)) System.out.print(root.val+" ");

for(int x:left) res.add(x+1);

for(int x:right) res.add(x+1);

return res;

}
```

correct me if I am wrong!

• Reply • Share ›

Ankit • a year ago

shouldn't the condition be pathlen-k-1 == 0??

• Reply • Share ›

Mr. Lazy Ankit • a year ago

No! the condition is right as we only need to check if the index of the element at k distance away from the current leaf node is valid (greater or equal to 0) i.e, this element exists and is stored in the path array.

• Reply • Share ›

lucy Mr. Lazy • 8 months ago

is it require " visited[pathLen] = false;"

• Reply • Share ›

Guest • a year ago

Why do we need two lists (path & visited) ?
Check this solution -

```
bool BinaryTree::printNodeAtKDist(BinaryTree* pRoot, const int nDist, string& str, int&
printLevel)
{
if (NULL == pRoot) return 0;
if (NULL == pRoot->leftTree && NULL == pRoot->rightTree) {
printLevel = 1;
return 1;
} else {
printNodeAtKDist(pRoot->leftTree, nDist, str, printLevel);
printNodeAtKDist(pRoot->rightTree, nDist, str, printLevel);
} //else

if (printLevel && nDist > printLevel)
printLevel++;

if (printLevel && nDist == printLevel) {
```

see more

• Reply • Share ›

Guest ➚ Guest • a year ago

Boolian array is required to the cases when duplicates are represent and k distance from the leaves. For e.g. if 3 is the root in the above diagram also is 3 is the right child of root than ans is 3 but would print two times 3 3 so visited array is used to handle this.

• Reply • Share ›

DarkCoder • a year ago

How about performing a BFS and storing the levels in the matrix in the reversely. Now, just print the rows at a distance of k to the leaf nodes.

• Reply • Share ›

Nayan • a year ago

I have simple recursive solution which uses concept similar to that of printing kth node from end of linked list. Idea is to keep passing k to all node and when get out of recursion decrement k. so check for k = 0 just after recursively calling for left and right child.

check my code out at http://ideone.com/MxqFdx

2 • Reply • Share ›

The_Geek Nayan • a year ago

for k=3, your code not working.

• Reply • Share ›

Guest Nayan • a year ago

Oh, well, you this its that simple, huh?

```
node *root = new node(1);
root->left = new node(2);
root->left->left = new node(3);
root->left->left = new node(4);
root->left->right = new node(5);
root->left->right->left = new node(6);
root->left->right->right = new node(7);
root->left->right->left->left = new node(8);
```

For k=4, let me know what your code prints out and what should be the actual output.

1 • Reply • Share ›

Guest Guest • a year ago

Actually:

```
node *root = new node(1);
root->left = new node(2);
root->left->left = new node(3);
root->left->left->left = new node(4);
root->left->left->right = new node(5);
root->left->left->right->left = new node(6);
root->left->left->right->right = new node(7);
root->left->left->right->left->left = new node(8);
```

1 • Reply • Share ›

mercs Nayan • a year ago

Wow nice code!!

Thanx!

• Reply • Share ›

Guest • a year ago

@GeeksforGeeks

Is this the right approach to this problem? For example if k = 2, we have to print node 6 also, don't we? Since 6 is at distance 2 from leaf node 7?

1 • Reply • Share ›

AllergicToBitches Guest • a year ago

Here k distance from a leaf means k levels higher than a leaf node. Read the 2nd para

• Reply • Share ›

ryan • 2 years ago

@GeeksforGeeks
simple and robust code u can check in all cases

```
int printkdist(nodeptr node,int level,int dist)
{
if(node==NULL)
return 0;
if(!node->left&&!node->right)
return level;
int lef=0,rgt=0;
if(node->left)
lef=printkdist(node->left,level+1,dist);
if(node->right)
rgt=printkdist(node->right,level+1,dist);
int max=lef>rgt?lef:rgt;
if(max-level==dist)
printf("%d\n",node->data);
int min=lef>rgt?rgt:lef;
return min;
}
```

1 • Reply • Share ›

avik_d • 2 years ago

shouldn't the condition be pathlen-k-1 == 0??

• Reply • Share ›

Ankit avik_d • a year ago

I have the same question :-\

• Reply • Share ›

<HoldOnLife!#> • 2 years ago

cant we use the approach of print given node distance from target node , and replace target node by leaf nodes ?

• Reply • Share ›

code12 • 2 years ago

@gfg

Why node with key value 6 is not there in output for the input distance 2. It is at the 2 unit

Why node with key value 6 is not there in output for the input distance 2. It is at the 2 unit distance from leaf node with key value 7.

• Reply • Share ›

Ashwini Singh ➦ code12 • 2 years ago

@code12

We have to only consider bottom-up distance from the leaves...

• Reply • Share ›

Guest • 2 years ago

@GeeksforGeeks this does not work when k>height of tree.

In your given example above: if k=3, then your program prints only 1 while it should also print 3 as it as at k distance from 4 and 5.

1 • Reply • Share ›

Abhinav Bhardwaj • 2 years ago

```
HashSet<integer> myset=new HashSet<integer>();
public void printNodesKdistanceFormLEaf(Node root,ArrayList<integer> arr,int k)
{
if(root==null)
return;

if(root.leftChild==null && root.rightChild==null)
{
printArray(arr,k);
return;
}

arr.add(root.data);
printNodesKdistanceFormLEaf(root.leftChild,arr,k);
printNodesKdistanceFormLEaf(root.rightChild,arr,k);

arr.remove(arr.size()-1);
}

public void printArray(ArrayList<integer> arr,int k)
{

if(!myset.contains(arr.get(arr.size()-k)))
System.out.print(arr.get(arr.size()-k)+" ");
myset.add(arr.get(arr.size()-k));

}
```

• Reply • Share ›

DS+Algo • 2 years ago

Actually the output will be "1 3" not "3 1"

• Reply • Share ›

Kyoo Sn M M • 2 years ago

This is my approach with hashTables if the order of the nodes does not matter

```
public static void printkDistance(Node root,int k){
HashSet<integer> hs= new HashSet<>();
printKDistance(root,hs,1,k);
}

public static void printKDistance(Node root,HashSet<integer> hs, int c, int k)
{
if(root==null) return;
//if is leaf and exist a positive level parent then add the level in the path to be printed
if(root.left==null && root.right==null && c>k){
hs.add(c-k);
return;
}
printKDistance(root.left,hs,c+1,k);
printKDistance(root.right,hs,c+1,k);
//after visit the sub nodes verify if it needs to be printed and remove it
if(hs.contains(c)){
System.out.println(root.v);
hs.remove(c);
}
}
```

• Reply • Share ›

Jun • 2 years ago

Hello.
I have a confusion here...
If we analyse the solution...,let k=2
In the given example...,
1. root=1. so we insert it in the pat array and set visted as false.pathlen=1
2. then it gies to root->left and insery 2 in the path array and visted =false.pathlen=2
3.Now again it goes to root->left and insert 4 into the array and set visited=false,pathlen=3.
Since, it is leaf node and it satisfies the if condition, the path[pathlen-k-1] will be printed.
The PROBLEM is...
NOW, when we go to root->right, then we insert 5 into the path array and set false.
path={1,2,4,5}, pathlen=4,k=2
so pathlen-k-1=1..

i.e it will print 2....??????
But technically it shud print 1 na....
I think i am missing out something...
Please someone point out the mistake....

Thanx in advance :)

• Reply • Share ›

DS+Algo Jun • 2 years ago

@sunshine

Actually the pathlen is passed as call by value, So every call of function will have its own pathlen.. Try to understand following with open mind...:
when 2 is inserted, path ={1,2} and pathlen will be 2 for this function instance where root is 2.

then 4 is inserted , path = {1,2,4} and pathlen will be 4 for this function instance where root is 4.
Now since 4 is leaf, it will print path[pathlen-k-1] i.e path[4-2-1]=2.
Now it returns to the function instance with root=2.
This function instance still has pathlen=2
So when it calls the function with root as root->right,
5 wil be inserted at path[pathlen]=path[2]
So the path[] for this function instance will be {1,2,5}
Hope you understood, @sunshine

if you still have doubts, then welcome

• Reply • Share ›

Abhishek Chauhan • 2 years ago

A nlogn approach with no extra space other than one created by recursive steps.
Although here redundant prints will come.

```
void print_leaf_knode_aux(node *head, node *root, int k, int path, int *max_path)
{
if (!root || !head || (*max_path > 0))
return;

if (root == head)
*max_path = path;

print_leaf_knode_aux(head, root->left, k, path + 1, max_path);
print_leaf_knode_aux(head, root->right, k, path + 1, max_path);

if (k == (*max_path - path))
printf("%d ", root->data);
```

}

void print_leaf_knode(node *head, node *root, int k)

see more

• Reply • Share ›

Nihal Sharma • 2 years ago

public void printAtLevel(Node node, int k) {
if (node == null)
return;
if (k == 1)
{
System.out.println(node.getData());
return;
}
printAtLevel(node.getLeft(), k - 1);
printAtLevel(node.getRight(), k - 1);
}

• Reply • Share ›

AlienOnEarth • 2 years ago

GeeksforGeeks:

This is another approach to print nodes at k distance from leaf. This solution has o(1) space complexity and o(n) time complexity and is similar to Print nodes at k distance from target node. Below is the source code in C. I request 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 ›

Programmer ➚ AlienOnEarth • 2 years ago

your code print print node more than 1 time...

▲ | ▼ • Reply • Share ›

Abhishek Chauhan ➚ AlienOnEarth • 2 years ago

Elegant approach , but it does not work.
You are loosing information for the leafs once you are moving up.
INPUT: 5 2 7 1 3 6 8 4 9
Try for above bst with input k = 3.
Expected output should be 5 but your approach does not give any output in this case.

▲ | ▼ • Reply • Share ›

Guest ➚ AlienOnEarth • 2 years ago

Can you please explain the complexity of

if (isLeaf(root))

{

return 0;

}

▲ | ▼ • Reply • Share ›

AlienOnEarth ➚ Guest • 2 years ago

Its constant. isLeaf means checking whether the given node has children or not.

▲ | ▼ • Reply • Share ›

zzpp ➚ AlienOnEarth • 2 years ago

I think your solution will print out the same node several times. For example, a root with a left child and a right child, k = 1. Then the root will be printed out twice.

▲ | ▼ • Reply • Share ›

Load more comments