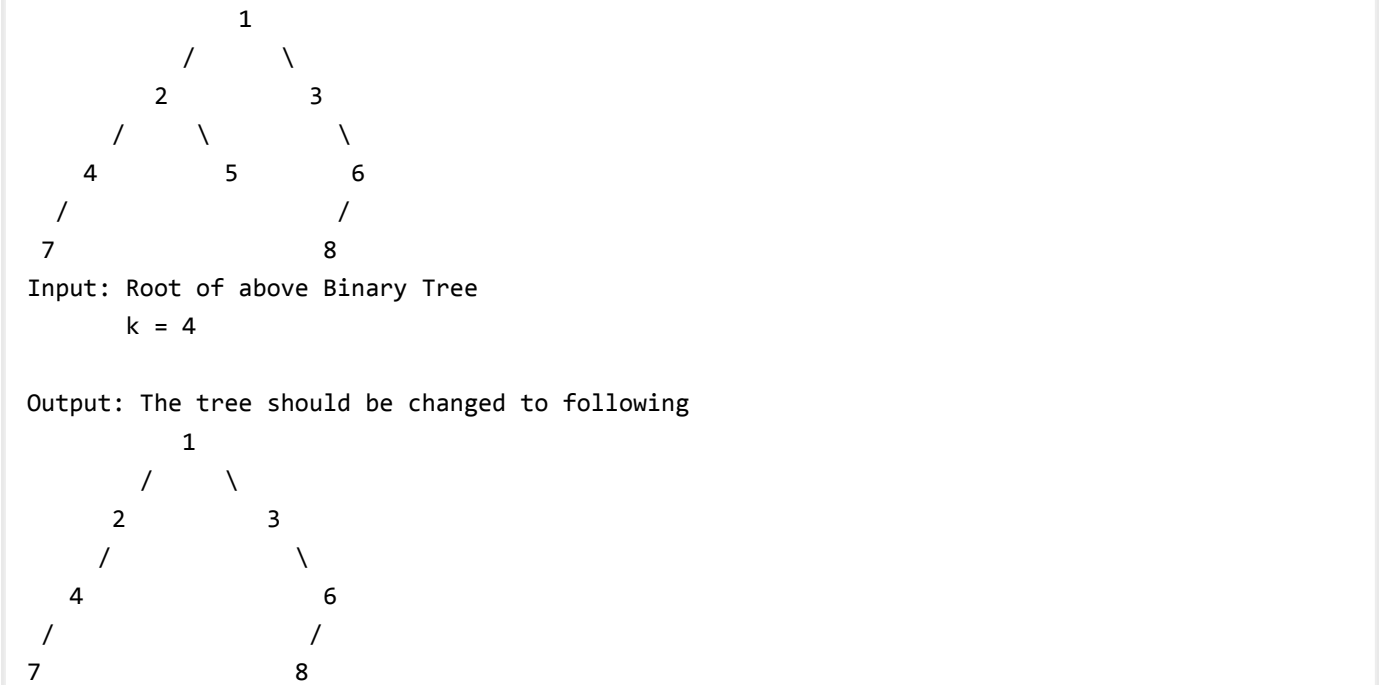
Remove nodes on root to leaf paths of length < K

Given a Binary Tree and a number k, remove all nodes that lie only on root to leaf path(s) of length smaller than k. If a node X lies on multiple root-to-leaf paths and if any of the paths has path length >= k, then X is not deleted from Binary Tree. In other words a node is deleted if all paths going through it have lengths smaller than k.

Consider the following example Binary Tree




```
Input: Root of above Binary Tree
       k = 4

Output: The tree should be changed to following
          1
        /    \
      2        3
     /           \
   4               6
 /                 /
7                 8
There are 3 paths
i) 1->2->4->7      path length = 4
ii) 1->2->5        path length = 3
iii) 1->3->6->8    path length = 4
There is only one path " 1->2->5 " of length smaller than 4.
The node 5 is the only node that lies only on this path, so
node 5 is removed.
Nodes 2 and 1 are not removed as they are parts of other paths
of length 4 as well.
```

```
If k is 5 or greater than 5, then whole tree is deleted.

If k is 3 or less than 3, then nothing is deleted.
```

We strongly recommend to minimize your browser and try this yourself first

The idea here is to use post order traversal of the tree. Before removing a node we need to check that all the children of that node in the shorter path are already removed.

There are 2 cases:

i) This node becomes a leaf node in which case it needs to be deleted.

ii) This node has other child on a path with path length >= k. In that case it needs not to be deleted.

The implementation of above approach is as below :

C/C++

```
// C++ program to remove nodes on root to leaf paths of length < K
#include<iostream>
using namespace std;

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

//New node of a tree
Node *newNode(int data)
{
    Node *node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Utility method that actually removes the nodes which are not
// on the pathLen >= k. This method can change the root as well.
Node *removeShortPathNodesUtil(Node *root, int level, int k)
{
    //Base condition
    if (root == NULL)
        return NULL;

    // Traverse the tree in postorder fashion so that if a leaf
    // node path length is shorter than k, then that node and
    // all of its descendants till the node which are not
    // on some other path are removed.
    root->left = removeShortPathNodesUtil(root->left, level + 1, k);
    root->right = removeShortPathNodesUtil(root->right, level + 1, k);

    // If root is a leaf node and it's level is less than k then
    // remove this node.
    // This goes up and check for the ancestor nodes also for the
    // same condition till it finds a node which is a part of other
    // path(s) too.
    if (root->left == NULL && root->right == NULL && level < k)
    {
        delete root;
```

```
        return NULL;
    }

    // Return root;
    return root;
}

// Method which calls the utitlity method to remove the short path
// nodes.
Node *removeShortPathNodes(Node *root, int k)
{
    int pathLen = 0;
    return removeShortPathNodesUtil(root, 1, k);
}

//Method to print the tree in inorder fashion.
void printInorder(Node *root)
{
    if (root)
    {
        printInorder(root->left);
        cout << root->data << " ";
        printInorder(root->right);
    }
}

// Driver method.
int main()
{
    int k = 4;
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->left->left->left = newNode(7);
    root->right->right = newNode(6);
    root->right->right->left = newNode(8);
    cout << "Inorder Traversal of Original tree" << endl;
    printInorder(root);
    cout << endl;
    cout << "Inorder Traversal of Modified tree" << endl;
    Node *res = removeShortPathNodes(root, k);
    printInorder(res);
    return 0;
}
```

Run on IDE

Java

```
// Java program to remove nodes on root to leaf paths of length < k

/* Class containing left and right child of current
 node and key value*/
class Node {

    int data;
    Node left, right;

    public Node(int item) {
        data = item;
```

```
        left = right = null;
    }
}

class BinaryTree {

    static Node root;

    // Utility method that actually removes the nodes which are not
    // on the pathLen >= k. This method can change the root as well.
    Node removeShortPathNodesUtil(Node node, int level, int k) {
        //Base condition
        if (node == null) {
            return null;
        }
        // Traverse the tree in postorder fashion so that if a leaf
        // node path length is shorter than k, then that node and
        // all of its descendants till the node which are not
        // on some other path are removed.
        node.left = removeShortPathNodesUtil(node.left, level + 1, k);
        node.right = removeShortPathNodesUtil(node.right, level + 1, k);

        // If root is a leaf node and it's level is less than k then
        // remove this node.
        // This goes up and check for the ancestor nodes also for the
        // same condition till it finds a node which is a part of other
        // path(s) too.
        if (node.left == null && node.right == null && level < k) {
            return null;
        }

        // Return root;
        return node;
    }

    // Method which calls the utitlity method to remove the short path
    // nodes.
    Node removeShortPathNodes(Node node, int k) {
        int pathLen = 0;
        return removeShortPathNodesUtil(node, 1, k);
    }

    //Method to print the tree in inorder fashion.
    void printInorder(Node node) {
        if (node != null) {
            printInorder(node.left);
            System.out.print(node.data + " ");
            printInorder(node.right);
        }
    }

    // driver program to test for samples
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        int k = 4;
        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.left.left.left = new Node(7);
        tree.root.right.right = new Node(6);
        tree.root.right.right.left = new Node(8);
        System.out.println("The inorder traversal of original tree is : ");
        tree.printInorder(root);
```

```
        Node res = tree.removeShortPathNodes(root, k);
        System.out.println("");
        System.out.println("The inorder traversal of modified tree is : ");
        tree.printInorder(res);
    }
}

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

Run on IDE

Output:

```
Inorder Traversal of Original tree
7 4 2 5 1 3 8 6
Inorder Traversal of Modified tree
7 4 2 1 3 8 6
```

Time complexity of the above solution is O(n) where n is number of nodes in given Binary Tree.

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

35 Comments Category: Trees Tags: tree-traversal

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)

3.6 Average Difficulty : **3.6/5.0**
Based on **8** 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.

35 Comments GeeksforGeeks 1 Login

Recommend 5 Share Sort by Newest

Join the discussion…

AllergicToBitches • 2 months ago

Slight optimization, if level >= k, we can skip level +1 and above -

```
Node *removeShortPathNodesUtil(Node *root, int level, int k)
{
if (root == NULL)
return NULL;

if(level >= k)
return root;

//rest of the code
}
```

• Reply • Share ›

Balu • 4 months ago

```
removeReachableLeafs(Node root, int k){
if(root == null) return null;
if(k>1&& isLeaf(root) return null;
root.left = removeReachableLeafs(root.left, k-1);
root.right = removeReachableLeafs(root.right, k-1);
return root;
```

}

Please review

1 • Reply • Share ›

Ashish • 5 months ago

The solution for this question will be similar to the solution for http://www.geeksforgeeks.org/r..., the only change being that we need to increase sum by 1 at each node as compared to increasing the sum by the node's data in that question.

• Reply • Share ›

InnocentHacker • 6 months ago

sorry for typo errors

```
public static BinaryTreeNode removeShortPathNodes(BinaryTreeNode root, int k){
return removeShortPathNodesHelper(root,k,1);
}

public static BinaryTreeNode removeShortPathNodesHelper(BinaryTreeNode root, int
k,int level){

if(root==null){
return null;
}
if(level>=k){
return root;
}
root.setLeft(removeShortPathNodesHelper(root.getLeft(),k,level+1));
root.setRight(removeShortPathNodesHelper(root.getRight(),k,level+1));
if(root.getLeft()!=null||root.getRight()!=null){
return root;
}
return null;
}
```

• Reply • Share ›

InnocentHacker • 6 months ago

```
public class BinaryTreeUtil{

public static BinaryTreeNode removeShortPathNodes(BinaryTreeNode root, int k){
return removeShortPathNodesHelper(root,k,1);
}
public static BinaryTreeNode removeShortPathNodesHelper(BinaryTreeNode root, int
k,int level){
if(root==null){
return null:
```

```
}
if(level>=k){
return root;
}
root.setLeft(removeShortPathNodes(root.getLeft(),k,level+1)){
root.setRight(removeShortPathNodes(root.getRight(),k,level+1)){
if(root.getLeft()!=null||root.setRight()!=null){
return root;
}
return null;
}
}
```

• Reply • Share ›

pramod • 7 months ago

=======java=====

```
static int k=1;

public boolean remove(Node n,int count){

if(n==null) return false;

count=count+1;

boolean left=remove(n.left,count);

boolean right=remove(n.right,count);

if(left) n.left=null;

if(right) n.right=null;

if(n.left==null&&n.right==null&&count<k){ return="" true;="" }else{="" return="" false;="" }="" }="">
```

• Reply • Share ›

Gaurav • 8 months ago

I guess we can optimize this. In case level is greater than k, we don't need to visit the nodes

so we can use like this

```
if(level<k) {="" root-="">left = removeShortPathNodesUtil(root->left, level + 1, k);

root->right = removeShortPathNodesUtil(root->right, level + 1, k);

}
```

}

If we check the number of recursive calls, it would we less in this case(13 instead of 17 recursive calls)

1 • Reply • Share ›

Karan Kapoor • 8 months ago

similar to the previous post of removing nodes when node does not lie in any path >=k

• Reply • Share ›

Arun Dixit • 8 months ago

int removeNode(Node root, int level, int k){
if(root==null)
return level;

int lh = removeNode(root.left,level+1,k);
int rh = removeNode(root.right,level+1,k);

int x = max(lh,rh);

if(root.left==null && root.right==null && x<k) root="null;" return="" x;="" }="">

• Reply • Share ›

saumik chaurasia • 9 months ago

TreeNode removeNodes(TreeNode node, int k,int len) {

if(node == null)

return null;

node.leftChild = removeNodes(node.leftChild, k,len+1) ;

node.rightChild = removeNodes(node.rightChild, k,len+1);

if(node.leftChild == null && node.rightChild == null && len < k){

System.out.println("Length " + len);

node = null;

}

return node;

}

• Reply • Share ›

Dman • 9 months ago

In the given code, if there exists no such path with length >=k, even then root pointer is pointing to some garbage location. Hence, one contition should be added in the end that is

if(res==NULL)
root=NULL;

or the other method is pass address of pointers in the recursive function code of which is given here : https://ideone.com/1bnDuo

1 • Reply • Share ›

Ayush Dman • 9 months ago

well i guess this part covers that case too..

```
if (root->left == NULL && root->right == NULL && level < k)
{
delete root;
return NULL;
}
```

• Reply • Share ›

DS+Algo • 9 months ago

No need to traverse the whole tree. i.e. when we reach to k-th node, we can return back instead of going further till leaf node.
Solution here: http://ideone.com/f4rr3z

1 • Reply • Share ›

OURJAY • 10 months ago

if level of leave is <k then="" delete="" leave.="">

• Reply • Share ›

Enkesh Gupta • 10 months ago

A simple Function-

```
struct node *remove(struct node *root,int k){
if(root==NULL)
return root;
root->left=remove(root->left,k-1);
root->right=remove(root->right,k-1);
if(!root->left && !root->right){
if(k>1){
free(root);
return NULL;
}
}
return root;
```

}

• Reply • Share ›

Sanjeev Dhankhar • 10 months ago

Level order traversal or using stack should be the better approach here...

• Reply • Share ›

sanju • a year ago

http://ideone.com/vyLo9T No need of two variables k and level.only k will do I guess

• Reply • Share ›

Bhargava • a year ago

May be this is of some helpful !

void removeK(TREE root,int count,int k)

{

TREE temp=NULL;

if(root==NULL){

return;

}

removeK(root->llink,count+1,k);

removeK(root->rlink,count+1,k); //Post-order Traversal

if((root->llink!=NULL && count<(k-1) && root->llink->llink==NULL && root->llink->rlink==NULL)){

see more

• Reply • Share ›

Guest • a year ago

don't know why it's not printing first node on my local machine . ideone however gives correct answer

Node *removeShortPathNodes(Node *root, int k){

if(root == NULL || k<=0) return root; // no modification required

if(!root->left && !root->right && k>0){

delete root;

```
return NULL;

}

root->left = removeShortPathNodes(root->left, k-1);

root->right = removeShortPathNodes(root->right, k-1);

return root;

}
```

http://ideone.com/iobQ23

42 ▲ ▼ • Reply • Share ›

AllergicToBitches Guest • 2 months ago
Ever heard of memory leak dude?? Doing preorder traversal to delete!! huh.
▲ ▼ • Reply • Share ›

Billionaire Guest • 7 months ago
Why so many upvotes on this one?

This is obviously wrong! You should delete its children before delete itself! i.e. post order
2 ▲ ▼ • Reply • Share ›

AllergicToBitches Billionaire • 2 months ago
Its 42 guest votes... seems he only upvoted his answer 42 times. lol
▲ ▼ • Reply • Share ›

DS+Algo Guest • 9 months ago
Ideone is not giving correct answer also.
Your first if condition should be
if(root==NULL || k<=1) return root; //Consider the case when only root is there then k=1 should not delete it

See here:http://ideone.com/zEymcL
▲ ▼ • Reply • Share ›

thevagabond85 • a year ago
that obscure int pathLen = 0;
▲ ▼ • Reply • Share ›

Abhinay Madasu • a year ago
Java Solution : http://ideone.com/T5fqBM

Java Solution : http://ideone.com/Yo1gBM

• Reply • Share ›

ayush katara • a year ago

can be done easily using level order traversal

• Reply • Share ›

ssk ayush katara • a year ago

Can you explain the algorithm?

1 • Reply • Share ›

Bhagwat Singh • a year ago

Hi
We can optimise this code .
As we are going to the depth >k which is not required .
whenever we get the path length == k return from there no need to go further because we only have to check the path with path length < k.

One change and one code addition will optimise the code:

add if(level==k) return root;

and change the if (root->left == NULL && root->right == NULL && level < k) to if (root->left == NULL && root->right == NULL) only.

1 • Reply • Share ›

SHAILIN DESAI • a year ago

It can still be optimized further. The above program runs upto the leaf node even if that path length is greater than K.
The following C program will travel one path only till k and if the node has any one children then it will return back.
The time complexity will be O(k). (Not sure though)

```
Node *removeShortPathNodes(Node *root, int k)
{

if(root == NULL)
{
return NULL;
}
else if ( k < 2)
{
return root;
}
if(root->left != NULL)
```

see more

• Reply • Share ›

pavan8085 • a year ago

A simple implementation in C

```
int delete_nodes ( node_t **root, int k )
{

static int level = 1;
int max_path =0;

if ( *root == NULL )
{
/* The non-existent children of leaf nodes are at
* level -1
*/
return -1;
}

level++;

max_path = 1 + max(delete_nodes(&((*root)->left), k),
```

see more

• Reply • Share ›

chicken • a year ago

typo-> "pre-order traversal"

1 • Reply • Share ›

Debanjan Chanda • a year ago

Problem source?

• Reply • Share ›

Kumar Gautam ➚ Debanjan Chanda • a year ago

It was asked recently in a flipkart telephonic round.

• Reply • Share ›

d_dll ➚ Kumar Gautam • 9 months ago

thanks

• Reply • Share ›

Debanjan Chanda ➚ Kumar Gautam • a year ago

Thanks.

• Reply • Share ›