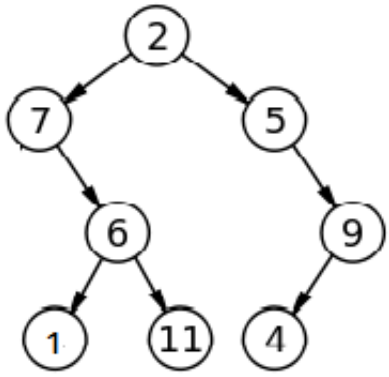
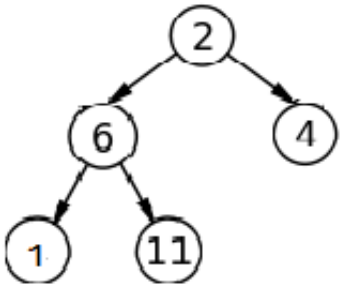
Given a binary tree, how do you remove all the half nodes?

Given A binary Tree, how do you remove all the half nodes (which has only one child)? Note leaves should not be touched as they have both children as NULL.

For example consider the below tree.



Nodes 7, 5 and 9 are half nodes as one of their child is Null. We need to remove all such half nodes and return the root pointer of following new tree.



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

The idea is to use post-order traversal to solve this problem efficiently. We first process the left children, then right children, and finally the node itself. So we form the new tree bottom up, starting from the leaves towards the root. By the time we process the current node, both its left and right subtrees were already processed.

Below is the implementation of this idea.

C

```c
// C program to remove all half nodes
#include <stdio.h>
#include <stdlib.h>

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

struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = node->right = NULL;
    return(node);
}

void printInoder(struct node*root)
{
    if (root != NULL)
    {
        printInoder(root->left);
        printf("%d ",root->data);
        printInoder(root->right);
    }
}

// Removes all nodes with only one child and returns
// new root (note that root may change)
struct node* RemoveHalfNodes(struct node* root)
{
    if (root==NULL)
        return NULL;

    root->left  = RemoveHalfNodes(root->left);
    root->right = RemoveHalfNodes(root->right);

    if (root->left==NULL && root->right==NULL)
        return root;

    /* if current nodes is a half node with left
       child NULL left, then it's right child is
       returned and replaces it in the given tree */
    if (root->left==NULL)
    {
        struct node *new_root = root->right;
        free(root); // To avoid memory leak
        return new_root;
    }


    /* if current nodes is a half node with right
       child NULL right, then it's right child is
       returned and replaces it in the given tree  */
    if (root->right==NULL)
    {
        struct node *new_root = root->left;
```

```
        free(root); // To avoid memory leak
        return new_root;
    }

    return root;
}

// Driver program
int main(void)
{
    struct node*NewRoot=NULL;
    struct node *root = newNode(2);
    root->left        = newNode(7);
    root->right       = newNode(5);
    root->left->right = newNode(6);
    root->left->right->left=newNode(1);
    root->left->right->right=newNode(11);
    root->right->right=newNode(9);
    root->right->right->left=newNode(4);

    printf("Inorder traversal of given tree \n");
    printInoder(root);

    NewRoot = RemoveHalfNodes(root);

    printf("\nInorder traversal of the modified tree \n");
    printInoder(NewRoot);
    return 0;
}
```

Run on IDE

Java

```
// Java program to remove half nodes
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;

    void printInorder(Node node) {
        if (node != null) {
            printInorder(node.left);
            System.out.print(node.data + " ");
            printInorder(node.right);
        }
    }

    // Removes all nodes with only one child and returns
    // new root (note that root may change)
    Node RemoveHalfNodes(Node node) {
        if (node == null) {
```

```
            return null;
        }

        node.left = RemoveHalfNodes(node.left);
        node.right = RemoveHalfNodes(node.right);

        if (node.left == null && node.right == null) {
            return node;
        }

        /* if current nodes is a half node with left
         child NULL left, then it's right child is
         returned and replaces it in the given tree */
        if (node.left == null) {
            Node new_root = node.right;
            return new_root;
        }

        /* if current nodes is a half node with right
         child NULL right, then it's right child is
         returned and replaces it in the given tree  */
        if (node.right == null) {
            Node new_root = node.left;
            return new_root;
        }

        return node;
    }

    // Driver program
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        Node NewRoot = null;
        tree.root = new Node(2);
        tree.root.left = new Node(7);
        tree.root.right = new Node(5);
        tree.root.left.right = new Node(6);
        tree.root.left.right.left = new Node(1);
        tree.root.left.right.right = new Node(11);
        tree.root.right.right = new Node(9);
        tree.root.right.right.left = new Node(4);

        System.out.println("the inorder traversal of tree is ");
        tree.printInorder(root);

        NewRoot = tree.RemoveHalfNodes(root);

        System.out.print("\nInorder traversal of the modified tree \n");
        tree.printInorder(NewRoot);
    }
}
```

Run on IDE

Python

```
# Python program to remove all half nodes

# A binary tree node
class Node:
    # Constructor for creating a new node
    def __init__(self , data):
```

```
        self.data = data
        self.left = None
        self.right = None

# For inorder traversal
def printInorder(root):
    if root is not None:
        printInorder(root.left)
        print root.data,
        printInorder(root.right)

# Removes all nodes with only one child and returns
# new root(note that root may change)
def RemoveHalfNodes(root):
    if root is None:
        return None

    # Recur to left tree
    root.left = RemoveHalfNodes(root.left)

    # Recur to right tree
    root.right = RemoveHalfNodes(root.right)

    # if both left and right child is None
    # the node is not a Half node
    if root.left is None and root.right is None:
        return root

    # If current nodes is a half node with left child
    # None then it's right child is returned and
    # replaces it in the given tree
    if root.left is None:
        new_root = root.right
        temp = root
        root = None
        del(temp)
        return new_root

    if root.right is None:
        new_root = root.left
        temp = root
        root = None
        del(temp)
        return new_root

    return root

# Driver Program
root = Node(2)
root.left = Node(7)
root.right = Node(5)
root.left.right = Node(6)
root.left.right.left = Node(1)
root.left.right.right = Node(11)
root.right.right = Node(9)
root.right.right.left = Node(4)

print "Inorder traversal of given tree"
printInorder(root)

NewRoot = RemoveHalfNodes(root)

print "\nInorder traversal of the modified tree"
printInorder(NewRoot)
```

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

Run on IDE

Output:

```
Inorder traversal of given tree
7 1 6 11 2 5 4 9
Inorder traversal of the modified tree
1 6 11 2 4
```

Time complexity of the above solution is O(n) as it does a simple traversal of binary tree.

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

38 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)

2.5 Average Difficulty : **2.5/5.0**
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.

38 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

flavio • 11 days ago

root->left = RemoveHalfNodes(root->left);
root->right = RemoveHalfNodes(root->right);

I don't get why he does that. Can someone explain to me please?

• Reply • Share ›

Rakesh Venkatesh flavio • 5 days ago

this will traverse down till the leaf and while backtracking it will remove the half nodes and adjusts the pointers accordingly

• Reply • Share ›

Ashish • 5 months ago

Java implementation of same as above:

```
private static Node removeHalfNodes(Node root) {
if(root == null){
return null;
}
if(root.getLeft()==null && root.getRight()==null){
return root;
}
root.setLeft(removeHalfNodes(root.getLeft()));
root.setRight(removeHalfNodes(root.getRight()));
if(root.getLeft()==null || root.getRight()==null){
return root.getLeft()!=null?root.getLeft():root.getRight();
}
```

```
return root;
}
```

 • Reply • Share ›

ranjan • 7 months ago

```
// C program to remove all half nodes

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node* left, *right;

};

struct node* newNode(int data)

{

struct node* node = (struct node*)
```

see more

 • Reply • Share ›

ranjan ranjan • 7 months ago

This is another variant which works:)

 • Reply • Share ›

Prashant Awasthi • 7 months ago

Anyone checked the above code .. it's not working ... output is same as before removing the nodes and after removing the nodes ...

 • Reply • Share ›

Billionaire • 7 months ago

Awesome post order solution!

1 • Reply • Share ›

Vivek Sharma • 8 months ago

By using post order traversal we can solve this problem we first process left child then right child and finally the node.

1 • Reply • Share ›

Ajcoo ➚ Vivek Sharma • 7 months ago

That is what is being done in above method !

▲ ▼ • Reply • Share ›

abhishek rai • 8 months ago

```
void Half_Node (struct node *root) {
if( root == NULL ) return;
Half_Node ( root->left );
Half_Node ( root->right );
struct node *temp;
if ( root->left == NULL && root->right != NULL )
{ temp = root;
root = root->right; free( temp ); }
if ( root->left != NULL && root-> right == NULL )
{ temp = root;
root = root->left; free(temp); }
} ..... // Kindly have a look :)
```

▲ ▼ • Reply • Share ›

Om Prakash Gupta • 8 months ago

http://ideone.com/nwHUrw

▲ ▼ • Reply • Share ›

Dman • 9 months ago

Easy Implementation . Don't assign root->left and root->right so early. Instead store the return values in some temp variables and after that do assignment based on requirements.

https://ideone.com/EdSofX

▲ ▼ • Reply • Share ›

Satinder Singh • 9 months ago

Using pre order traversal--

```
void RemoveHalfNodes(struct node* root)

{

if (root==NULL)

return;

int p=1;
```

while(p)

{

if (root->left==NULL &&root->right!=NULL)

{

node *temp;

see more

• Reply • Share ›

praveen kumar Satinder Singh • 9 months ago

This solution may fail when you have the partial tree in the third level.
for eg. for the below tree the above solution will not work.
A
/ \
B C
/ \
D E
/
F

• Reply • Share ›

praveen kumar praveen kumar • 9 months ago

Sorry i didnt see the pre-order traversal .Yes if you run this function for the nodes this will work.

• Reply • Share ›

Abhishek • 10 months ago

How to format code in this comment section?

• Reply • Share ›

Abhishek • 10 months ago

Here is my solution (http://ideone.com/iBNZnH):

struct node * RemoveHalfNodes(struct node * root)

{

if(root == NULL) return root;

if(root->left != NULL && root->right != NULL) {

root->left = RemoveHalfNodes(root->left);

```
root->right = RemoveHalfNodes(root->right);

return root;

}

if(root->left == NULL && root->right == NULL) return root;

if(root->left != NULL ) {
```

see more

• Reply • Share ›

mama chaturi • a year ago

you need to fix image if its binary tree then left node is less and right node is greater then root .

• Reply • Share ›

nikhil jain • a year ago

We can also do it by using level order traversal.
By following below step:-
1.Check whether root is half node or not,if not then find that node which is full node.
2.put that node in queue.
3.dequeue that node & find the other full node in left sub tree & make them left child & enqueue it.
4.find the other full node in right subtree make them right child & enqueue it
5.Repeat 3 & 4 untill we get the leaf.

Basically i am enqueue only that node which is full.
please tell me If i am wrong,otherwise this solution can create trouble in any interview :P

2 • Reply • Share ›

Shaveen Kumar • a year ago

```
void prune(Node* root)
{
if(root == NULL)
return;

if(root->left != NULL && root->right != NULL)
{
prune(root->left);
prune(root->right);
}
if (root->left == NULL)
{
root = root->right;
```

```
prune(root);
}
if (root->right == NULL)
{
root = root->left;
prune(root);
}
}
```

• Reply • Share ›

Jerry Goyal • a year ago

"convert a binary tree to full binary tree" could be a better title.

6 • Reply • Share ›

varun • a year ago

What are we trying to do in below lines.... Sorry i am java guy so trying to put the solution in Java

```
struct node *new_root = root->right;

free(root); // To avoid memory leak

return new_root;
```

• Reply • Share ›

varun varun • a year ago

Never mind... I got that... here is the Java version

```
public Node removeAllHalfNodesFromBinaryTree(Node root) {

        if (root != null) {

            Node leftNode = root.left = removeAllHalfNodesFromBinar

            Node rightNode = root.right = removeAllHalfNodesFromBir

            if (leftNode == null && rightNode == null) {

                return root;

            }

            if (leftNode == null) {
```

see more

• Reply • Share ›

Amit Dawle • a year ago

Few lines in haskell:

```
module EmptyNodes where

data Tree a = EmptyTree | Node a (Tree a) (Tree a) deriving (Show)

halfNode :: Tree t -> Tree t
halfNode EmptyTree = EmptyTree
halfNode (Node a EmptyTree EmptyTree ) = Node a EmptyTree EmptyTree

halfNode (Node _ left EmptyTree ) = halfNode left
halfNode (Node _ EmptyTree right ) = halfNode right
halfNode (Node a left right ) = Node a (halfNode left) (halfNode right)

> let tree = Node 2 ( Node 7 EmptyTree (Node 6 (Node 1 EmptyTree EmptyTree) (Node 11 EmptyTree EmptyTree) ) ) (Node 5 EmptyTree (Node 9 (Node 4 EmptyTree EmptyTree) EmptyTree))

> halfNode tree

Node 2 (Node 6 (Node 1 EmptyTree EmptyTree) (Node 11 EmptyTree EmptyTree)) (Node 4 EmptyTree EmptyTree)
```

• Reply • Share ›

Piyush • a year ago

This solves it.

```
Node removeHalf(Node head){
if( head == NULL || (head->left == NULL && head->right == NULL) )
return head;
head->left = removeHalf(head->left);
head->right = removeHalf(head->right);
if (head->left != NULL && head->right != NULL)
return head;
else
return (head->left!= NULL)? head->left : head->right;
}
```

• Reply • Share ›

Lovekesh Bazaria • a year ago

How about this?

http://ideone.com/i6RbYh

• Reply • Share ›

Guest • a year ago

Pass root node to this function -

```
Node _func(Node root)
{
if(root==NULL) return NULL;

if ((root->left!=NULL && root->right!=NULL) || (root->left==NULL && root->right==NULL) )
{
//these are good nodes i.e. with zero or 2 childs,no removal
//so, fixate their childs first
root->left = _func(root->left);
root->right = _func(root->right);
//and since they're good, they are valid child to their parent
return root;
}
else
{
//these are bad nodes, with 1 child, and will be removed

//so every bad node will remove itself from between and give next node, right or left,
whichever is present , a chance to get attached as child node*/
if(root->left)
return _func(root->left);
else
return = _func(root->right)
}
}
```

• Reply • Share ›

vishnu • a year ago

Is it correct?

http://ideone.com/DcySU2

1 • Reply • Share ›

acemrek • a year ago

We can also do it with keeping parent.

http://ideone.com/SSzvMV

• Reply • Share ›

Koustav Chatterjee • a year ago

My Java Code : http://ideone.com/HpMcSm

My Java Code : http://ideone.com/HpMcOm

• Reply • Share ›

Howard Sueing III • a year ago

Can't we just do a breadth first search traversal on this tree, visiting each node in that manner. For each visit we would check for a the scenario where there is half node. If so, set the parent node equal to which ever child node is there as a leaf. e.g.

```
public void removeHalfNode(Node root) {
if (root == null) return;
Queue queue = new Queue();
visit(root);
root.visited = true;
queue.enqueue(root);
while (!queue.isEmpty()) {
Node top = queue.dequeue();
for (Node r: top.adjacent()) {
if (!r.visisted) {
visit(r);
r.visited = true;
queue.enqueue(r);
}
```

see more

• Reply • Share ›

Pr ➚ Howard Sueing III • a year ago

Or post-order, or in-Order

• Reply • Share ›

Pr ➚ Howard Sueing III • a year ago

You could use a BFS. But it uses an additional memory. Which is unnecessary when you have an option of not using it with pre-order traversal.

• Reply • Share ›

Rajeev ➚ Howard Sueing III • a year ago

Did you consider the scenario, when root itself is half node. Your function is void hence to me it seems you are assuming that root will not change.

• Reply • Share ›

Vibhor Garg • a year ago

we can use preorder traversal also ?

• Reply • Share ›

go4gold ➦ Vibhor Garg • a year ago

yes you can do . go through the below code

```
void removehalfnode(node *p,node *parent,int direction)

{

if(p == NULL)

return;

if(p->left!=NULL && p->right == NULL)

{

if(direction==1)

parent->right = p->left;

else

parent->left = p->left;
```

see more

1 ▲ | ▼ • Reply • Share ›

wei su ➦ Vibhor Garg • a year ago

how?

▲ | ▼ • Reply • Share ›

Vibhor Garg ➦ wei su • a year ago

Like This :-
http://ideone.com/4elfgC

You can refer the actual code here.

▲ | ▼ • Reply • Share ›