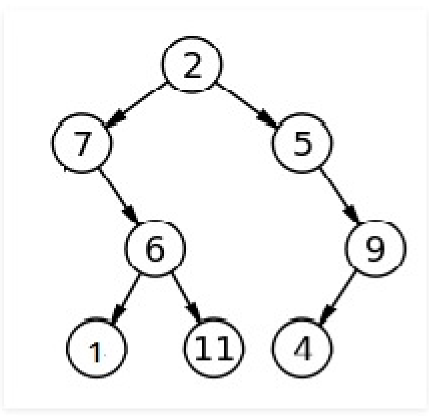
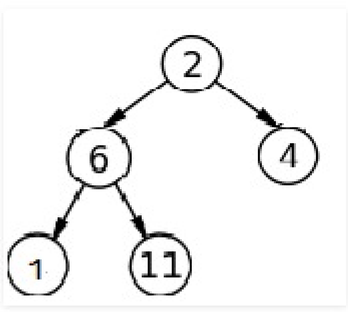
Given a binary tree, how do you remove all the half nodes?

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 Java Python

```
// Java program to remove half nodes
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    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(tree.root);

        NewRoot = tree.RemoveHalfNodes(tree.root);

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

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

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.