

Binary Tree Solutions

Solution 1:

Time Complexity : $O(h)$
Space Complexity: $O(1)$

```
import java.util.*;
class Solution{

    static class Node{
        int data;
        Node left;
        Node right;
    };

    static Node newNode(int data){
        Node temp = new Node();
        temp.data = data;
        temp.left = temp.right = null;
        return (temp);
    }

    static boolean isUnivalTree(Node root){

        if (root == null){
            return true;
        }

        if (root.left != null
            && root.data != root.left.data)
            return false;

        if (root.right != null
            && root.data != root.right.data)
            return false;

        return isUnivalTree(root.left)
            && isUnivalTree(root.right);
    }
}
```



priyanshi082005@gmail.com

```
}
```

```
public static void main(String[] args){
```

```
    Node root = newNode(1);  
    root.left = newNode(1);  
    root.right = newNode(1);  
    root.left.left = newNode(1);  
    root.left.right = newNode(1);  
    root.right.right = newNode(1);
```

```
    if (isUnivalTree(root)) {  
        System.out.print("YES");  
    }  
    else{  
        System.out.print("NO");  
    }  
}
```

Solution 2 :

Time Complexity : $O(n)$
Space Complexity: $O(n)$

```
class Node{  
    int data;  
    Node left, right;  
  
    public Node(int item)  
    {  
        data = item;  
        left = right = null;  
    }  
}
```

```
class Solution{  
    Node root;  
  
    void mirror(){  
        root = mirror(root);  
    }
```



priyanshi082005@gmail.com

```
Node mirror(Node node){  
    if (node == null)  
        return node;  
  
    /* do the subtrees */  
    Node left = mirror(node.left);  
    Node right = mirror(node.right);  
  
    /* swap the left and right pointers */  
    node.left = right;  
    node.right = left;  
  
    return node;  
}  
  
void inOrder(){  
    inOrder(root);  
}  
  
void inOrder(Node node){  
    if (node == null)  
        return;  
  
    inOrder(node.left);  
    System.out.print(node.data + " ");  
  
    inOrder(node.right);  
}  
  
public static void main(String args[]){  
    BinaryTree tree = new BinaryTree();  
    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);  
  
    System.out.println("Inorder traversal of input tree is :");  
    tree.inOrder();  
    System.out.println("");  
    tree.mirror();  
    System.out.println("Inorder traversal of binary tree is :");  
    tree.inOrder();  
}
```



priyanshi082005@gmail.com

}

Solution 3 :

Time Complexity : $O(n)$ Space Complexity: $O(1)$

```
class Solution {  
  
    static class Node {  
        int data;  
        Node left, right;  
    }  
  
    static Node newNode(int data){  
        Node newNode = new Node();  
        newNode.data = data;  
        newNode.left = null;  
        newNode.right = null;  
        return (newNode);  
    }  
  
    static Node deleteLeaves(Node root, int x){  
        if (root == null)  
            return null;  
        root.left = deleteLeaves(root.left, x);  
        root.right = deleteLeaves(root.right, x);  
  
        if (root.data == x && root.left == null && root.right == null) {  
            return null;  
        }  
        return root;  
    }  
  
    static void inorder(Node root){  
        if (root == null)  
            return;  
        inorder(root.left);  
        System.out.print(root.data + " ");  
        inorder(root.right);  
    }  
}
```



priyanshi082005@gmail.com

```
System.out.print(root.data + " ");
inorder(root.right);
}

public static void main(String[] args){
    Node root = newNode(10);
    root.left = newNode(3);
    root.right = newNode(10);
    root.left.left = newNode(3);
    root.left.right = newNode(1);
    root.right.right = newNode(3);
    root.right.right.left = newNode(3);
    root.right.right.right = newNode(3);
    deleteLeaves(root, 3);
    System.out.print("Inorder traversal after deletion : ");
    inorder(root);
}
}
```

Solution 4 :

Time Complexity : $O(n^2)$
Space Complexity: $O(n^2)$

```
import java.util.HashMap;
public class Solution {

    static HashMap<String, Integer> m;
    static class Node {
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            left = null;
            right = null;
        }
    }
}
```



priyanshi082005@gmail.com

```
static String inorder(Node node){  
    if (node == null)  
        return "";  
  
    String str = "(";  
    str += inorder(node.left);  
    str += Integer.toString(node.data);  
    str += inorder(node.right);  
    str += ")";  
  
    if (m.get(str) != null && m.get(str)==1 )  
        System.out.print( node.data + " ");  
  
    if (m.containsKey(str))  
        m.put(str, m.get(str) + 1);  
    else  
        m.put(str, 1);  
  
    return str;  
}  
  
static void printAllDups(Node root){  
    m = new HashMap<>();  
    inorder(root);  
}  
  
public static void main(String args[]){  
    Node root = null;  
    root = new Node(1);  
    root.left = new Node(2);  
    root.right = new Node(3);  
    root.left.left = new Node(4);  
    root.right.left = new Node(2);  
    root.right.left.left = new Node(4);  
    root.right.right = new Node(4);  
    printAllDups(root);  
}  
}
```

Solution 5 :

priyanshi082005@gmail.com

Time Complexity : O(n)

Space Complexity: O(1)

```
class Node {  
  
    int data;  
    Node left, right;  
  
    public Node(int item) {  
        data = item;  
        left = right = null;  
    }  
}  
  
class Res {  
    public int val;  
}  
  
class Solution {  
    Node root;  
    int findMaxUtil(Node node, Res res) {  
  
        if (node == null)  
            return 0;  
  
        int l = findMaxUtil(node.left, res);  
        int r = findMaxUtil(node.right, res);  
  
        int max_single = Math.max(Math.max(l, r) + node.data,  
                                  node.data);  
  
        int max_top = Math.max(max_single, l + r + node.data);  
  
        res.val = Math.max(res.val, max_top);  
  
        return max_single;  
    }  
  
    int findMaxSum() {
```



priyanshi082005@gmail.com

```
        return findMaxSum(root);
    }

    int findMaxSum(Node node) {

        Res res = new Res();
        res.val = Integer.MIN_VALUE;

        findMaxUtil(node, res);
        return res.val;
    }

    public static void main(String args[]) {
        Solution tree = new Solution();
        tree.root = new Node(10);
        tree.root.left = new Node(2);
        tree.root.right = new Node(10);
        tree.root.left.left = new Node(20);
        tree.root.left.right = new Node(1);
        tree.root.right.right = new Node(-25);
        tree.root.right.right.left = new Node(3);
        tree.root.right.right.right = new Node(4);
        System.out.println("maximum path sum is : " +
                           tree.findMaxSum());
    }
}
```

priyanshi082005@gmail.com