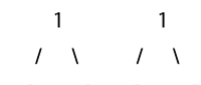
# Binary Tree

(For the below tasks, you may want to create a binary tree first and use the same tree for all of these tasks. All the methods as well as the tester statements should be written in one class. You do not need to write a different class for each method. You may also follow functional programming approach too. In that case you will not need to write any specific class. Just functions with proper parameters.)

1. Given two binary trees, find if both of them are identical or not.

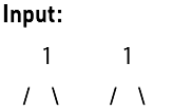
**Input:**



**Output:** Yes

**Explanation:** There are two trees both having 3 nodes and 2 edges, both trees are identical having the root as 1, left child of 1 is 2 and right child of 1 is 3.

**Input:**

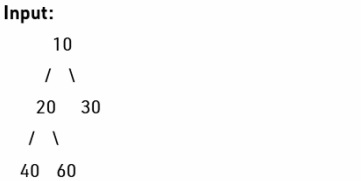


**Output:** No

**Explanation:** There are two trees both having 3 nodes and 2 edges, but both trees are not identical.

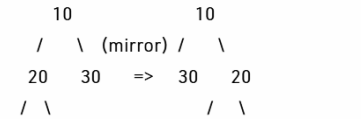
2. Given a binary tree, convert it into its mirror.

**Input:**



**Output:** 30 10 60 20 40

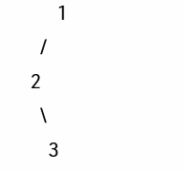
**Explanation:** The tree is



The inroder traversal of mirror is 30 10 60 20 40.

3. Given a binary tree, find if it is height balanced or not. A tree is height balanced if the difference between heights of left and right subtrees is not more than one for all nodes of the tree.

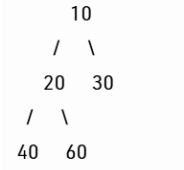
**Input:**



**Output:** 0

**Explanation:** The max difference in height of left subtree and right subtree is 2, which is greater than 1. Hence unbalanced

**Input:**

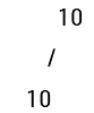


**Output:** 1

**Explanation:** The max difference in height of left subtree and right subtree is 1. Hence balanced.

4. Given a binary tree, check whether all of its nodes have the value equal to the sum of their child nodes.

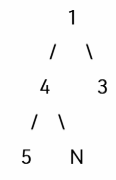
**Input:**



**Output:** 1

**Explanation:** Here, every node is sum of its left and right child.

**Input:**



**Output:** 0

**Explanation:** Here, 1 is the root node and 4, 3 are its child nodes. 4 + 3 = 7 which is not equal to the value of root node. Hence, this tree does not satisfy the given conditions.

5. Given a binary tree, find the largest value in each level.

**Input :**

**Output :** 4 9 7

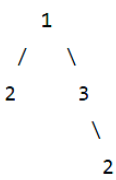
**Explanation :**

There are three levels in the tree:

1. {4}, max = 4
2. {9, 2}, max = 9
3. {3, 5, 7}, max=7

6. Given a binary tree, check if it has duplicate values.

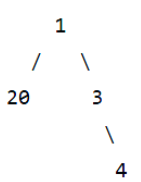
**Input : Root of below tree**



**Output : Yes**

**Explanation : The duplicate value is 2.**

**Input : Root of below tree**

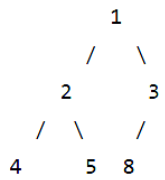


**Output : No**

**Explanation : There are no duplicates.**

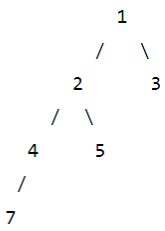
7. Given a root of a binary tree, and an integer k, print all the nodes which are at k distance from root. Distance is the number of edges in the path from the source node (Root node in our case) to the destination node.

   For example, in the below tree, 4, 5 & 8 are at distance 2 from root.

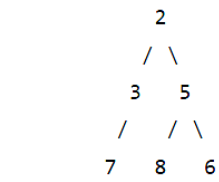



8. Given a binary tree and a key, write a function that prints all the ancestors of the node with the key in the given binary tree.

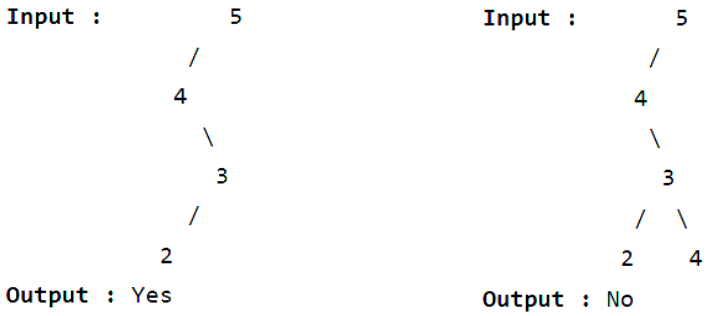
   For example, if the given tree is following Binary Tree and the key is 7, then your function should print 4, 2, and 1.



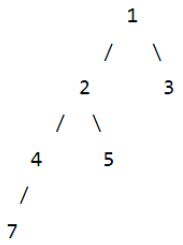

9. Given a binary tree, print all the nodes having exactly one child. Print "-1" if no such node exists.



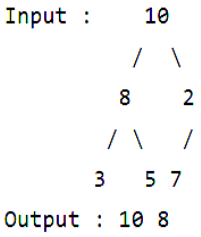

10. Given a binary tree, check whether it is a skewed binary tree or not. A skewed tree is a tree where each node has only one child node or none.



11. Given a binary tree, find its minimum depth. The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.
For example, the minimum depth of the below Binary Tree is 2.

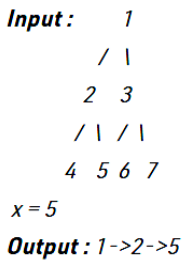


12. Given a binary tree, print all nodes that are full nodes. Full nodes are nodes which have both left and right children as non-empty.



**Advanced Problems:**

13. Given a binary tree with distinct nodes(no two nodes have the same data values). The problem is to print the path from root to a given node **x**. If node **x** is not present then print "No Path".



14. Evaluate a given binary expression tree representing algebraic expressions. A binary expression tree is a binary tree, where the operators are stored in the tree's internal nodes, and the leaves contain constants. Assume that each node of the binary expression tree has zero or two children. The supported operators are +, -, * and /.

For example, the value of the following expression tree is 28: