Binary Tree Data Structure

[**Data Structure and Algorithms Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=gfg&utm_medium=header+link+click&utm_campaign=dsa+course+tracker&utm_term=dsa+course+promo&utm_content=binary-tree-lp)

[**Practice Problems on Binary Tree !**](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions&utm_source=gfg&utm_medium=header+link+click&utm_campaign=practice+tracker&utm_term=practice+promo&utm_content=binary-tree-lp)

[**Recent Articles on Binary Tree !**](https://www.geeksforgeeks.org/tag/binary-tree/?utm_source=gfg&utm_medium=header+link+click&utm_campaign=recent+article+tracker&utm_term=recent+article+tracker&utm_content=binary-tree-lp)

[**What is Binary Tree Data Structure?**](https://www.geeksforgeeks.org/introduction-to-binary-tree-data-structure-and-algorithm-tutorials/)

*Binary Tree is defined as a tree data structure where each node has at most 2 children. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.*



**Binary Tree Representation**

A Binary tree is represented by a pointer to the topmost node (commonly known as the “root”) of the tree. If the tree is empty, then the value of the root is NULL. Each node of a Binary Tree contains the following parts:

1. Data
2. Pointer to left child
3. Pointer to right child

**Basic Operation On Binary Tree:**

* Inserting an element.
* Removing an element.
* Searching for an element.
* Traversing the tree.

**Auxiliary Operation On Binary Tree:**

* Finding the height of the tree
* Find the level of a node of the tree
* Finding the size of the entire tree.

**Topic :**

* [Introduction](https://www.geeksforgeeks.org/binary-tree-data-structure/?ref=ghm#Introduction)
* [Basic Operation](https://www.geeksforgeeks.org/binary-tree-data-structure/?ref=ghm#basicop)
* [Traversals](https://www.geeksforgeeks.org/binary-tree-data-structure/?ref=ghm#traversals)
* [Standard Problems on Binary Trees](https://www.geeksforgeeks.org/binary-tree-data-structure/?ref=ghm#standard)

**Introduction :**

1. [Introduction to Binary Tree – Data Structure and Algorithm Tutorials](https://www.geeksforgeeks.org/introduction-to-binary-tree-data-structure-and-algorithm-tutorials/)
2. [Properties of Binary Tree](https://www.geeksforgeeks.org/binary-tree-set-2-properties/)
3. [Types of Binary Tree](https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/)
4. [Applications, Advantages and Disadvantages of Binary Tree](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-tree/)
5. [Binary Tree (Array implementation)](https://www.geeksforgeeks.org/binary-tree-array-implementation/)
6. [Complete Binary Tree](https://www.geeksforgeeks.org/complete-binary-tree/)
7. [Perfect Binary Tree](https://www.geeksforgeeks.org/perfect-binary-tree/)

**Basic Operations on Binary Tree:**

1. [Insertion in a Binary Tree](https://www.geeksforgeeks.org/insertion-binary-tree/)
2. [Deletion in a Binary Tree](https://www.geeksforgeeks.org/deletion-binary-tree/)
3. [Enumeration of Binary Trees](https://www.geeksforgeeks.org/enumeration-of-binary-trees/)
4. [Tree Traversals (Inorder, Preorder and Postorder)](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/)

**Some other important Binary Tree Traversals :**

1. [Level Order Tree Traversal](https://www.geeksforgeeks.org/level-order-tree-traversal/)
2. [Level order traversal in spiral form](https://www.geeksforgeeks.org/level-order-traversal-in-spiral-form/)
3. [Reverse Level Order Traversal](https://www.geeksforgeeks.org/reverse-level-order-traversal/)
4. [BFS vs DFS for Binary Tree](https://www.geeksforgeeks.org/bfs-vs-dfs-binary-tree/)
5. [Inorder Tree Traversal without Recursion](https://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion/)
6. [Morris traversal for Preorder](https://www.geeksforgeeks.org/morris-traversal-for-preorder/)
7. [Iterative Preorder Traversal](https://www.geeksforgeeks.org/iterative-preorder-traversal/)
8. [Iterative Postorder Traversal Using Two Stacks](https://www.geeksforgeeks.org/iterative-postorder-traversal/)
9. [Diagonal Traversal of Binary Tree](https://www.geeksforgeeks.org/diagonal-traversal-of-binary-tree/)
10. [Boundary Traversal of binary tree](https://www.geeksforgeeks.org/boundary-traversal-of-binary-tree/)

[More >>](https://www.geeksforgeeks.org/binary-tree-data-structure/binary-tree-traversal/)

**Must solve Standard Problems on Binary Tree Data Structure:**

* **Easy**
  1. [Calculate depth of a full Binary tree from Preorder](https://www.geeksforgeeks.org/calculate-depth-full-binary-tree-preorder/)
  2. [Construct a tree from Inorder and Level order traversals](https://www.geeksforgeeks.org/construct-tree-inorder-level-order-traversals/)
  3. [Check if a given Binary Tree is SumTree](https://www.geeksforgeeks.org/check-if-a-given-binary-tree-is-sumtree/)
  4. [Check if two nodes are cousins in a Binary Tree](https://www.geeksforgeeks.org/check-two-nodes-cousins-binary-tree/)
  5. [Check if removing an edge can divide a Binary Tree in two halves](https://www.geeksforgeeks.org/check-if-removing-an-edge-can-divide-a-binary-tree-in-two-halves/)
  6. [Check whether a given binary tree is perfect or not](https://www.geeksforgeeks.org/check-weather-given-binary-tree-perfect-not/)
  7. [Check if a Binary Tree contains duplicate subtrees of size 2 or more](https://www.geeksforgeeks.org/check-binary-tree-contains-duplicate-subtrees-size-2/)
  8. [Check if two trees are Mirror](https://www.geeksforgeeks.org/check-if-two-trees-are-mirror/)
  9. [Foldable Binary Trees](https://www.geeksforgeeks.org/foldable-binary-trees/)
  10. [Symmetric Tree (Mirror Image of itself)](https://www.geeksforgeeks.org/symmetric-tree-tree-which-is-mirror-image-of-itself/)
  11. [Write Code to Determine if Two Trees are Identical](https://www.geeksforgeeks.org/write-c-code-to-determine-if-two-trees-are-identical/)
  12. [Subtree with given sum in a Binary Tree](https://www.geeksforgeeks.org/subtree-given-sum-binary-tree/)
  13. [Succinct Encoding of Binary Tree](https://www.geeksforgeeks.org/succinct-encoding-of-binary-tree/)
  14. [Write a program to Calculate Size of a tree](https://www.geeksforgeeks.org/write-a-c-program-to-calculate-size-of-a-tree/)
  15. [Diameter of a Binary Tree](https://www.geeksforgeeks.org/diameter-of-a-binary-tree/)
  16. [Get Level of a node in a Binary Tree](https://www.geeksforgeeks.org/get-level-of-a-node-in-a-binary-tree/)
* **Medium**
  1. [Find all possible binary trees with given Inorder Traversal](https://www.geeksforgeeks.org/find-all-possible-trees-with-given-inorder-traversal/)
  2. [Populate Inorder Successor for all nodes](https://www.geeksforgeeks.org/populate-inorder-successor-for-all-nodes/)
  3. [Construct Complete Binary Tree from its Linked List Representation](https://www.geeksforgeeks.org/given-linked-list-representation-of-complete-tree-convert-it-to-linked-representation/)
  4. [Minimum swap required to convert binary tree to binary search tree](https://www.geeksforgeeks.org/minimum-swap-required-convert-binary-tree-binary-search-tree/)
  5. [Convert a given Binary Tree to Doubly Linked List | Set 1](https://www.geeksforgeeks.org/in-place-convert-a-given-binary-tree-to-doubly-linked-list/)
  6. [Convert a tree to forest of even nodes](https://www.geeksforgeeks.org/convert-tree-forest-even-nodes/)
  7. [Flip Binary Tree](https://www.geeksforgeeks.org/flip-binary-tree/)
  8. [Print root to leaf paths without using recursion](https://www.geeksforgeeks.org/print-root-leaf-path-without-using-recursion/)
  9. [Check if given Preorder, Inorder and Postorder traversals are of same tree](https://www.geeksforgeeks.org/check-if-given-preorder-inorder-and-postorder-traversals-are-of-same-tree/)
  10. [Check whether a given Binary Tree is Complete or not | Set 1 (Iterative Solution)](https://www.geeksforgeeks.org/check-if-a-given-binary-tree-is-complete-tree-or-not/)
  11. [Check if a binary tree is subtree of another binary tree | Set 2](https://www.geeksforgeeks.org/check-binary-tree-subtree-another-binary-tree-set-2/)
  12. [Find largest subtree sum in a tree](https://www.geeksforgeeks.org/find-largest-subtree-sum-tree/)
  13. [Maximum sum of nodes in Binary tree such that no two are adjacent](https://www.geeksforgeeks.org/maximum-sum-nodes-binary-tree-no-two-adjacent/)
  14. [Lowest Common Ancestor in a Binary Tree | Set 1](https://www.geeksforgeeks.org/lowest-common-ancestor-binary-tree-set-1/)
  15. [Height of a generic tree from parent array](https://www.geeksforgeeks.org/height-generic-tree-parent-array/)
  16. [Find distance between two given keys of a Binary Tree](https://www.geeksforgeeks.org/find-distance-two-given-nodes/)
* **Hard**
  1. [Modify a binary tree to get Preorder traversal using right pointers only](https://www.geeksforgeeks.org/modify-binary-tree-get-preorder-traversal-using-right-pointers/)
  2. [Construct Full Binary Tree using its Preorder traversal and Preorder traversal of its mirror tree](https://www.geeksforgeeks.org/construct-full-binary-tree-using-preorder-traversal-preorder-traversal-mirror-tree/)
  3. [Construct a special tree from given preorder traversal](https://www.geeksforgeeks.org/construct-a-special-tree-from-given-preorder-traversal/)
  4. [Construct tree from ancestor matrix](https://www.geeksforgeeks.org/construct-tree-from-ancestor-matrix/)
  5. [Construct the full k-ary tree from its preorder traversal](https://www.geeksforgeeks.org/construct-full-k-ary-tree-preorder-traversal/)
  6. [Construct Binary Tree from String with bracket representation](https://www.geeksforgeeks.org/construct-binary-tree-string-bracket-representation/)
  7. [Convert a Binary Tree into Doubly Linked List in spiral fashion](https://www.geeksforgeeks.org/convert-a-binary-tree-into-doubly-linked-list-in-spiral-fashion/)
  8. [Convert a Binary Tree to a Circular Doubly Link List](https://www.geeksforgeeks.org/convert-a-binary-tree-to-a-circular-doubly-link-list/)
  9. [Convert Ternary Expression to a Binary Tree](https://www.geeksforgeeks.org/convert-ternary-expression-binary-tree/)
  10. [Check if there is a root to leaf path with given sequence](https://www.geeksforgeeks.org/check-root-leaf-path-given-sequence/)
  11. [Remove all nodes which don’t lie in any path with sum>= k](https://www.geeksforgeeks.org/remove-all-nodes-which-lie-on-a-path-having-sum-less-than-k/)
  12. [Maximum spiral sum in Binary Tree](https://www.geeksforgeeks.org/maximum-spiral-sum-in-binary-tree/)
  13. [Sum of nodes at k-th level in a tree represented as string](https://www.geeksforgeeks.org/sum-nodes-k-th-level-tree-represented-string/)
  14. [Sum of all the numbers that are formed from root to leaf paths](https://www.geeksforgeeks.org/sum-numbers-formed-root-leaf-paths/)
  15. [Merge Two Binary Trees by doing Node Sum (Recursive and Iterative)](https://www.geeksforgeeks.org/merge-two-binary-trees-node-sum/)
  16. [Find root of the tree where children id sum for every node is given](https://www.geeksforgeeks.org/find-root-tree-children-id-sum-every-node-given/)

**Quick Links :**

* [‘Practice Problems’ on Trees](https://practice.geeksforgeeks.org/topics/Tree/)
* [‘Quizzes’ on Binary Trees](https://www.geeksforgeeks.org/data-structure-gq/binary-trees-gq/)
* [‘Videos’ on Trees](https://www.youtube.com/watch?v=IpyCqRmaKW4&list=PLqM7alHXFySHCXD7r1J0ky9Zg_GBB1dbk)

**Calculate depth of a full Binary tree from Preorder**

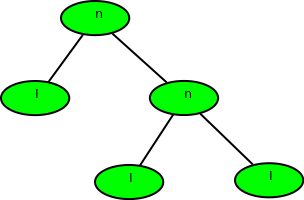
Given preorder of a binary tree, calculate its [depth(or height)](https://www.geeksforgeeks.org/write-a-c-program-to-find-the-maximum-depth-or-height-of-a-tree/) [starting from depth 0]. The preorder is given as a string with two possible characters.

1. ‘l’ denotes the leaf
2. ‘n’ denotes internal node

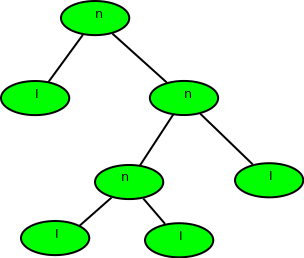
The given tree can be seen as a full binary tree where every node has 0 or two children. The two children of a node can ‘n’ or ‘l’ or mix of both.

**Examples :**

Input : nlnll  
Output : 2  
Explanation :



Input : nlnnlll  
Output : 3



[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Preorder of the binary tree is given so traverse

Also, we would be given a string of char (formed of ‘n’ and ‘l’), so there is no need to implement tree also.

The recursion function would be:

1. Base Case: return 0; when tree[i] = ‘l’ or i >= strlen(tree)
2. find\_depth( tree[i++] ) //left subtree
3. find\_depth( tree[i++] ) //right subtree

Where i is the index of the string tree.

**Implementation:**

#Python program to find height of full binary tree

# using preorder

# function to return max of left subtree height

# or right subtree height

**def** findDepthRec(tree, n, index) :

**if** (index[0] >**=** n **or** tree[index[0]] **==** 'l'):

**return** 0

    # calc height of left subtree (In preorder

    # left subtree is processed before right)

    index[0] **+=** 1

    left **=** findDepthRec(tree, n, index)

    # calc height of right subtree

    index[0] **+=** 1

    right **=** findDepthRec(tree, n, index)

**return** (max(left, right) **+** 1)

# Wrapper over findDepthRec()

**def** findDepth(tree, n) :

    index **=** [0]

**return** findDepthRec(tree, n, index)

# Driver program to test above functions

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    tree**=** "nlnnlll"

    n **=** len(tree)

    print(findDepth(tree, n))

# This code is contributed by SHUBHAMSINGH10

**Output**

3

**Time Complexity: O(N)**

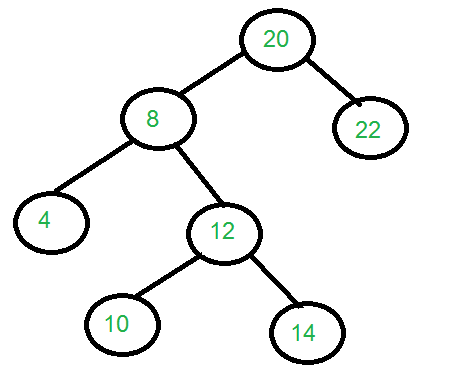
**Auxiliary Space: O(1)**

*From <*[*https://www.geeksforgeeks.org/calculate-depth-full-binary-tree-preorder/*](https://www.geeksforgeeks.org/calculate-depth-full-binary-tree-preorder/)*>*

**Construct a tree from Inorder and Level order traversals | Set 1**

1. Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
2. Last Updated : 26 Jul, 2022
3. Read
4. Discuss(120)
5. Courses
6. Practice
7. Video

Given inorder and level-order traversals of a Binary Tree, construct the Binary Tree. Following is an example to illustrate the problem.



Input: Two arrays that represent Inorder  
 and level order traversals of a   
 Binary Tree  
in[] = {4, 8, 10, 12, 14, 20, 22};  
level[] = {20, 8, 22, 4, 12, 10, 14};

Output: Construct the tree represented   
 by the two arrays.  
 For the above two arrays, the   
 constructed tree is shown in   
 the diagram on right side

*From <*[*https://www.geeksforgeeks.org/construct-tree-inorder-level-order-traversals/*](https://www.geeksforgeeks.org/construct-tree-inorder-level-order-traversals/)*>*

**Check if a given Binary Tree is SumTree**

Write a function that returns true if the given Binary Tree is SumTree else false. A SumTree is a Binary Tree where the value of a node is equal to the sum of the nodes present in its left subtree and right subtree. An empty tree is SumTree and the sum of an empty tree can be considered as 0. A leaf node is also considered as SumTree.

Following is an example of SumTree.

26  
 / \  
 10 3  
 / \ \  
 4 6 3

Recommended Problem

Sum Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+3 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/sum-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.5L

**Method 1 (Simple)**

Get the sum of nodes in the left subtree and right subtree. Check if the sum calculated is equal to the root’s data. Also, recursively check if the left and right subtrees are SumTrees.

1. C++
2. C
3. Java
4. Python3
5. C#
6. Javascript

# Python3 program to implement

# the above approach

# A binary tree node has data,

# left child and right child

**class** node:

**def** \_\_init\_\_(self, x):

        self.data **=** x

        self.left **=** None

        self.right **=** None

# A utility function to get the sum

# of values in tree with root as root

**def** sum(root):

**if**(root **==** None):

**return** 0

**return** (sum(root.left) **+**

            root.data **+**

            sum(root.right))

# returns 1 if sum property holds

# for the given node and both of

# its children

**def** isSumTree(node):

    # ls, rs

    # If node is None or it's a leaf

    # node then return true

**if**(node **==** None **or**

      (node.left **==** None **and**

       node.right **==** None)):

**return** 1

    # Get sum of nodes in left and

    # right subtrees

    ls **=** sum(node.left)

    rs **=** sum(node.right)

    # if the node and both of its children

    # satisfy the property return 1 else 0

**if**((node.data **==** ls **+** rs) **and**

        isSumTree(node.left) **and**

        isSumTree(node.right)):

**return** 1

**return** 0

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** node(26)

    root.left**=** node(10)

    root.right **=** node(3)

    root.left.left **=** node(4)

    root.left.right **=** node(6)

    root.right.right **=** node(3)

**if**(isSumTree(root)):

**print**("The given tree is a SumTree ")

**else**:

**print**("The given tree is not a SumTree ")

# This code is contributed by Mohit Kumar 29

**Output**

The given tree is a SumTree

**Time Complexity:** O(n2) in the worst case. The worst-case occurs for a skewed tree.

**Auxiliary Space:** O(n) for stack space

**Method 2 (Tricky)**

Method 1 uses sum() to get the sum of nodes in left and right subtrees. Method 2 uses the following rules to get the sum directly.

1) If the node is a leaf node then the sum of the subtree rooted with this node is equal to the value of this node.

2) If the node is not a leaf node then the sum of the subtree rooted with this node is twice the value of this node (Assuming that the tree rooted with this node is SumTree).

1. C++
2. C
3. Java
4. Python3
5. C#
6. Javascript

# Python3 program to check if

# Binary tree is sum tree or not

# A binary tree node has data,

# left child and right child

**class** node:

**def** \_\_init\_\_(self, x):

        self.data **=** x

        self.left **=** None

        self.right **=** None

**def** isLeaf(node):

**if**(node **==** None):

**return** 0

**if**(node.left **==** None **and** node.right **==** None):

**return** 1

**return** 0

# A utility function to get the sum

# of values in tree with root as root

**def** sum(root):

**if**(root **==** None):

**return** 0

**return** (sum(root.left) **+**

            root.data **+**

            sum(root.right))

# returns 1 if SumTree property holds

# for the given tree

**def** isSumTree(node):

    # If node is None or

    # it's a leaf node then return true

**if**(node **==** None **or** isLeaf(node)):

**return** 1

**if**(isSumTree(node.left) **and** isSumTree(node.right)):

        # Get the sum of nodes in left subtree

**if**(node.left **==** None):

            ls **=** 0

**elif**(isLeaf(node.left)):

            ls **=** node.left.data

**else**:

            ls **=** 2 **\*** (node.left.data)

        # Get the sum of nodes in right subtree

**if**(node.right **==** None):

            rs **=** 0

**elif**(isLeaf(node.right)):

            rs **=** node.right.data

**else**:

            rs **=** 2 **\*** (node.right.data)

        # If root's data is equal to sum of nodes

        # in left and right subtrees then return 1

        # else return 0

**return**(node.data **==** ls **+** rs)

**return** 0

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** node(26)

    root.left **=** node(10)

    root.right **=** node(3)

    root.left.left **=** node(4)

    root.left.right **=** node(6)

    root.right.right **=** node(3)

**if**(isSumTree(root)):

        print("The given tree is a SumTree ")

**else**:

        print("The given tree is not a SumTree ")

# This code is contributed by kirtishsurangalikar

**Output:**

The given tree is a SumTree

**Time Complexity:** O(n)

**Auxiliary Space:**O(n)

**Method 3**

1. Similar to postorder traversal iteratively find the sum in each step
2. Return left + right + current data if left + right is equal to current node data
3. Else return -1

* C++
* C++14
* Java
* Python3
* C#
* Javascript

# Python3 program to check if

# Binary tree is sum tree or not

# A binary tree node has data,

# left child and right child

**class** node:

**def** \_\_init\_\_(self, x):

        self.data **=** x

        self.left **=** None

        self.right **=** None

**def** isLeaf(node):

**if**(node **==** None):

**return** 0

**if**(node.left **==** None **and** node.right **==** None):

**return** 1

**return** 0

# returns data if SumTree property holds for the given

#    tree else return -1

**def** isSumTree(node):

**if**(node **==** None):

**return** 0

    ls **=** isSumTree(node.left)

**if**(ls **== -**1):            #To stop for further traversal of tree if found not sumTree

**return -**1

    rs **=** isSumTree(node.right)

**if**(rs **== -**1):            #To stop for further traversal of tree if found not sumTree

**return -**1

**if**(isLeaf(node) **or** ls **+** rs **==** node.data):

**return** ls **+** rs **+** node.data

**else**:

**return -**1

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** node(26)

    root.left **=** node(10)

    root.right **=** node(3)

    root.left.left **=** node(4)

    root.left.right **=** node(6)

    root.right.right **=** node(3)

**if**(isSumTree(root)):

        print("The given tree is a SumTree ")

**else**:

**print**("The given tree is not a SumTree ")

# This code is contributed by Mugunthan

**Time Complexity:** O(n), since each element is traversed only once

**Auxiliary Space:**O(n), due to recursive stack space

*From <*[*https://www.geeksforgeeks.org/check-if-a-given-binary-tree-is-sumtree/*](https://www.geeksforgeeks.org/check-if-a-given-binary-tree-is-sumtree/)*>*

**Check if two nodes are cousins in a Binary Tree**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 23 Jun, 2022
* Read
* Discuss(130+)
* Courses
* Practice
* Video

Given the binary Tree and the two nodes say ‘a’ and ‘b’, determine whether the two nodes are cousins of each other or not.

Two nodes are cousins of each other if they are at same level and have different parents.

**Example:**

6  
 / \  
 3 5  
 / \ / \  
7 8 1 3  
Say two node be 7 and 1, result is TRUE.  
Say two nodes are 3 and 5, result is FALSE.  
Say two nodes are 7 and 5, result is FALSE.

Recommended Problem

Check if two Nodes are Cousins

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[D-E-Shaw](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=D-E-Shaw&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/check-if-two-nodes-are-cousins/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 30.3K

The idea is to find level of one of the nodes. Using the found level, check if ‘a’ and ‘b’ are at this level. If ‘a’ and ‘b’ are at given level, then finally check if they are not children of same parent.

Following is the implementation of the above approach.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to check if two nodes in a binary

# tree are cousins

# A Binary Tree Node

**class** Node:

    # Constructor to create a new Binary Tree

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

**def** isSibling(root, a , b):

    # Base Case

**if** root **is** None:

**return** 0

**return** ((root.left **==** a **and** root.right **==**b) **or**

            (root.left **==** b **and** root.right **==** a)**or**

            isSibling(root.left, a, b) **or**

            isSibling(root.right, a, b))

# Recursive function to find level of Node 'ptr' in

# a binary tree

**def** level(root, ptr, lev):

    # Base Case

**if** root **is** None :

**return** 0

**if** root **==** ptr:

**return** lev

    # Return level if Node is present in left subtree

    l **=** level(root.left, ptr, lev**+**1)

**if** l !**=** 0:

**return** l

    # Else search in right subtree

**return** level(root.right, ptr, lev**+**1)

# Returns 1 if a and b are cousins, otherwise 0

**def** isCousin(root,a, b):

    # 1. The two nodes should be on the same level in

    # the binary tree

    # The two nodes should not be siblings(means that

    # they should not have the same parent node

**if** ((level(root,a,1) **==** level(root, b, 1)) **and**

**not** (isSibling(root, a, b))):

**return** 1

**else**:

**return** 0

# Driver program to test above function

root **=** Node(1)

root.left **=** Node(2)

root.right **=** Node(3)

root.left.left **=** Node(4)

root.left.right **=** Node(5)

root.left.right.right **=** Node(15)

root.right.left **=** Node(6)

root.right.right **=** Node(7)

root.right.left.right **=** Node(8)

node1 **=** root.left.right

node2 **=** root.right.right

print ("Yes" **if** isCousin(root, node1, node2) **==** 1 **else** "No")

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Yes

**Time Complexity** of the above solution is O(n) as it does at most three traversals of binary tree.

**Space complexity**: O(n) for call stack since using recursion

*From <*[*https://www.geeksforgeeks.org/check-two-nodes-cousins-binary-tree/*](https://www.geeksforgeeks.org/check-two-nodes-cousins-binary-tree/)*>*

**Check if removing an edge can divide a Binary Tree in two halves**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 30 Jun, 2022
* Read
* Discuss(30)
* Courses
* Practice
* Video

Given a Binary Tree, find if there exists an edge whose removal creates two trees of equal size.

**Examples:**

Input : root of following tree  
 5  
 / \  
 1 6   
 / / \  
 3 7 4  
Output : true  
Removing edge 5-6 creates two trees of equal size

Input : root of following tree  
 5  
 / \  
 1 6   
 / \  
 7 4  
 / \ \  
 3 2 8  
Output : false  
There is no edge whose removal creates two trees  
of equal size.

Source- Kshitij IIT KGP

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Method 1 (Simple):**First count number of nodes in whole tree. Let count of all nodes be n. Now traverse tree and for every node, find size of subtree rooted with this node. Let the subtree size be s. If n-s is equal to s, then return true, else false.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to check if there

# exist an edge whose removal creates

# two trees of same size

# utility function to create a new node

**class** newNode:

**def** \_\_init\_\_(self, x):

        self.data **=** x

        self.left **=** self.right **=** None

# To calculate size of tree

# with given root

**def** count(root):

**if** (root **==** None):

**return** 0

**return** (count(root.left) **+**

            count(root.right) **+** 1)

# This function returns true if there

# is an edge whose removal can divide

# the tree in two halves n is size of tree

**def** checkRec(root, n):

    # Base cases

**if** (root **==** None):

**return** False

    # Check for root

**if** (count(root) **==** n **-** count(root)):

**return** True

    # Check for rest of the nodes

**return** (checkRec(root.left, n) **or**

            checkRec(root.right, n))

# This function mainly uses checkRec()

**def** check(root):

    # Count total nodes in given tree

    n **=** count(root)

    # Now recursively check all nodes

**return** checkRec(root, n)

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** newNode(5)

    root.left **=** newNode(1)

    root.right **=** newNode(6)

    root.left.left **=** newNode(3)

    root.right.left **=** newNode(7)

    root.right.right **=** newNode(4)

**if** check(root):

        print("YES")

**else**:

**print**("NO")

# This code is contributed by PranchalK

**Output**

YES

**Time complexity:** **O(n2)** where n is number of nodes in given Binary Tree.

**Auxiliary Space: O(n)** for call stack since using recursion, where n is no of nodes in binary tree

**Method 2 (Efficient):**We can find the solution in O(n) time. The idea is to traverse tree in bottom up manner and while traversing keep updating size and keep checking if there is a node that follows the required property.

Below is the implementation of above idea.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to check if there exist

# an edge whose removal creates two trees

# of same size

**class** Node:

**def** \_\_init\_\_(self, x):

        self.key **=** x

        self.left **=** None

        self.right **=** None

# To calculate size of tree with

# given root

**def** count(node):

**if** (node **==** None):

**return** 0

**return** (count(node.left) **+**

            count(node.right) **+** 1)

# This function returns size of tree rooted

# with given root. It also set "res" as true

# if there is an edge whose removal divides

# tree in two halves.n is size of tree

**def** checkRec(root, n):

**global** res

    # Base case

**if** (root **==** None):

**return** 0

    # Compute sizes of left and right children

    c **=** (checkRec(root.left, n) **+** 1 **+**

         checkRec(root.right, n))

    # If required property is true for

    # current node set "res" as true

**if** (c **==** n **-** c):

        res **=** True

    # Return size

**return** c

# This function mainly uses checkRec()

**def** check(root):

    # Count total nodes in given tree

    n **=** count(root)

    # Initialize result and recursively

    # check all nodes

    # bool res = false;

    checkRec(root, n)

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    res **=** False

    root **=** Node(5)

    root.left **=** Node(1)

    root.right **=** Node(6)

    root.left.left **=** Node(3)

    root.right.left **=** Node(7)

    root.right.right **=** Node(4)

    check(root)

**if** res:

**print**("YES")

**else**:

**print**("NO")

# This code is contributed by mohit kumar 29

**Output**

YES

**Time Complexity: O(n)**

**Auxiliary Space:  O(n)**

*From <*[*https://www.geeksforgeeks.org/check-if-removing-an-edge-can-divide-a-binary-tree-in-two-halves/*](https://www.geeksforgeeks.org/check-if-removing-an-edge-can-divide-a-binary-tree-in-two-halves/)*>*

**Check whether a given binary tree is perfect or not**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 12 Jul, 2022
* Read
* Discuss(20+)
* Courses
* Practice
* Video

Given a Binary Tree, write a function to check whether the given Binary Tree is a perfect Binary Tree or not.

A Binary tree is [Perfect Binary Tree](https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/) in which all internal nodes have two children and all leaves are at same level.

**Examples:**

The following tree is a perfect binary tree

10  
 / \   
 20 30   
 / \ / \  
 40 50 60 70

18  
 / \   
 15 30

The following tree is **not** a perfect binary tree

1  
 / \  
 2 3  
 \ / \   
 4 5 6

Recommended Problem

Perfect Binary Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[FactSet](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=FactSet&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/perfect-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 12.2K

A [Perfect Binary Tree](https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/)of height h (where height is number of nodes on path from root to leaf) has 2h – 1 nodes.

Below is an idea to check whether a given Binary Tree is perfect or not.

1. Find depth of any node (in below tree we find depth of leftmost node). Let this depth be d.
2. Now recursively traverse the tree and check for following two conditions.

* Every internal node should have both children non-empty
* All leaves are at depth ‘d’

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to check whether a

# given Binary Tree is Perfect or not

# Helper class that allocates a new

# node with the given key and None

# left and right pointer.

**class** newNode:

**def** \_\_init\_\_(self, k):

        self.key **=** k

        self.right **=** self.left **=** None

# Returns depth of leftmost leaf.

**def** findADepth(node):

    d **=** 0

**while** (node !**=** None):

        d **+=** 1

        node **=** node.left

**return** d

# This function tests if a binary tree

# is perfect or not. It basically checks

# for two things :

# 1) All leaves are at same level

# 2) All internal nodes have two children

**def** isPerfectRec(root, d, level **=** 0):

    # An empty tree is perfect

**if** (root **==** None):

**return** True

    # If leaf node, then its depth must

    # be same as depth of all other leaves.

**if** (root.left **==** None **and** root.right **==** None):

**return** (d **==** level **+** 1)

    # If internal node and one child is empty

**if** (root.left **==** None **or** root.right **==** None):

**return** False

    # Left and right subtrees must be perfect.

**return** (isPerfectRec(root.left, d, level **+** 1) **and**

            isPerfectRec(root.right, d, level **+** 1))

# Wrapper over isPerfectRec()

**def** isPerfect(root):

    d **=** findADepth(root)

**return** isPerfectRec(root, d)

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** None

    root **=** newNode(10)

    root.left **=** newNode(20)

    root.right **=** newNode(30)

    root.left.left **=** newNode(40)

    root.left.right **=** newNode(50)

    root.right.left **=** newNode(60)

    root.right.right **=** newNode(70)

**if** (isPerfect(root)):

**print**("Yes")

**else**:

**print**("No")

# This code is contributed by pranchalK

**Output**

Yes

**Complexity Analysis:**

* **Time complexity:** **O(n)**
* **Space Complexity: O(n)**

**Method 2: Using the length of the binary tree**

Since a full binary tree has 2^h – 1 nodes, we can count the number of nodes in the binary tree and determine whether it is a power of 2 or not.

To efficiently determine whether it is a power of 2 or not, we can use bitwise operation x & (x+1) == 0

**Implementation:**

* C++
* Java
* Python3
* Javascript

# Python3 program to check whether a

# given Binary Tree is Perfect or not

# Helper class that allocates a new

# node with the given key and None

# left and right pointer.

**class** newNode:

**def** \_\_init\_\_(self, k):

        self.key **=** k

        self.right **=** self.left **=** None

#This functions gets the size of binary tree

#Basically, the number of nodes this binary tree has

**def** getLength(root):

**if** root **==** None:

**return** 0

**return** 1 **+** getLength(root.left) **+** getLength(root.right)

#Returns True if length of binary tree is a power of 2 else False

**def** isPerfect(root):

  length **=** getLength(root)

**return** length & (length**+**1) **==** 0

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** None

    root **=** newNode(10)

    root.left **=** newNode(20)

    root.right **=** newNode(30)

    root.left.left **=** newNode(40)

    root.left.right **=** newNode(50)

    root.right.left **=** newNode(60)

    root.right.right **=** newNode(70)

**if** (isPerfect(root)):

**print**("Yes")

**else**:

        print("No")

# This code is contributed by beardedowl

**Output**

No

**Complexity Analysis:**

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

*From <*[*https://www.geeksforgeeks.org/check-weather-given-binary-tree-perfect-not/*](https://www.geeksforgeeks.org/check-weather-given-binary-tree-perfect-not/)*>*

**Check if a Binary Tree contains duplicate subtrees of size 2 or more**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 23 Dec, 2022
* Read
* Discuss(50+)
* Courses
* Practice
* Video

Given a Binary Tree, check whether the Binary tree contains a duplicate sub-tree of size 2 or more.

Note : Two same leaf nodes are not considered as subtree size of a leaf node is one.

***Input :****Binary Tree*

*A*

*/    \*

*B        C*

*/   \       \*

*D     E       B*

*/  \*

*D    E*

***Output :****Yes*

Asked in : Google Interview

Recommended Problem

Duplicate subtree in Binary Tree

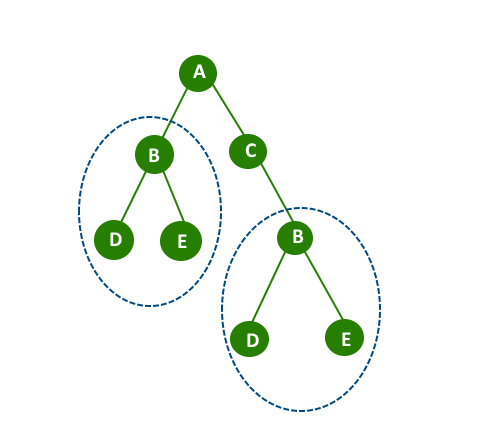
[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Google](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Google&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/duplicate-subtree-in-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 44.7K



**Tree with duplicate Sub-Tree [ highlight by blue color ellipse ]**

**[ Method 1]**

A simple solution is that, we pick every node of tree and try to find is any sub-tree of given tree is present in tree which is identical with that sub-tree. Here we can use below post to find if a subtree is present anywhere else in tree.

[Check if a binary tree is subtree of another binary tree](https://www.geeksforgeeks.org/check-if-a-binary-tree-is-subtree-of-another-binary-tree/)

**[Method 2 ]( Efficient solution )**

An Efficient solution based on [tree serialization](https://www.geeksforgeeks.org/serialize-deserialize-binary-tree/) and [hashing](https://www.geeksforgeeks.org/hashing-set-1-introduction/). The idea is to serialize subtrees as strings and store the strings in hash table. Once we find a serialized tree (which is not a leaf) already existing in hash-table, we return true.

Below The implementation of above idea.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find if there is

# a duplicate sub-tree of size 2 or more

# Separator node

MARKER **=** '$'

# Structure for a binary tree node

**class** Node:

**def** \_\_init\_\_(self, x):

        self.key **=** x

        self.left **=** None

        self.right **=** None

subtrees **=** {}

# This function returns empty if tree

# contains a duplicate subtree of size

# 2 or more.

**def** dupSubUtil(root):

**global** subtrees

    s **=** ""

    # If current node is None, return marker

**if** (root **==** None):

**return** s **+** MARKER

    # If left subtree has a duplicate subtree.

    lStr **=** dupSubUtil(root.left)

**if** (s **in** lStr):

**return** s

    # Do same for right subtree

    rStr **=** dupSubUtil(root.right)

**if** (s **in** rStr):

**return** s

    # Serialize current subtree

    s **=** s **+** root.key **+** lStr **+** rStr

    # If current subtree already exists in hash

    # table. [Note that size of a serialized tree

    # with single node is 3 as it has two marker

    # nodes.

**if** (len(s) > 3 **and** s **in** subtrees):

**return** ""

    subtrees[s] **=** 1

**return** s

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** Node('A')

    root.left **=** Node('B')

    root.right **=** Node('C')

    root.left.left **=** Node('D')

    root.left.right **=** Node('E')

    root.right.right **=** Node('B')

    root.right.right.right **=** Node('E')

    root.right.right.left**=** Node('D')

    str **=** dupSubUtil(root)

**if** "" **in** str:

**print**(" Yes ")

**else**:

**print**(" No ")

# This code is contributed by mohit kumar 29

**Output**

Yes

**Time Complexity: O(n)**

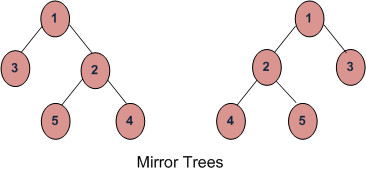
**Space Complexity: O(n)**

*From <*[*https://www.geeksforgeeks.org/check-binary-tree-contains-duplicate-subtrees-size-2/*](https://www.geeksforgeeks.org/check-binary-tree-contains-duplicate-subtrees-size-2/)*>*

**Check if two trees are Mirror**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 01 Jul, 2022
* Read
* Discuss(20+)
* Courses
* Practice
* Video

Given two Binary Trees, write a function that returns true if two trees are mirror of each other, else false. For example, the function should return true for following input trees.



[Recommended: Please solve it on “***PRACTICE*** ” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/check-mirror-in-n-ary-tree/0)

This problem is different from the problem discussed [here](https://www.geeksforgeeks.org/write-an-efficient-c-function-to-convert-a-tree-into-its-mirror-tree/).

For two trees ‘a’ and ‘b’ to be mirror images, the following three conditions must be true:

1. Their root node’s key must be same
2. Left subtree of root of ‘a’ and right subtree root of ‘b’ are mirror.
3. Right subtree of ‘a’ and left subtree of ‘b’ are mirror.

Below is implementation of above idea.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to check if two

# trees are mirror of each other

# A binary tree node

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Given two trees, return true

# if they are mirror of each other

**def** areMirror(a, b):

    # Base case : Both empty

**if** a **is** None **and** b **is** None:

**return** True

    # If only one is empty

**if** a **is** None **or** b **is** None:

**return** False

    # Both non-empty, compare them

    # recursively. Note that in

    # recursive calls, we pass left

    # of one tree and right of other tree

**return** (a.data **==** b.data **and**

            areMirror(a.left, b.right) **and**

            areMirror(a.right , b.left))

# Driver code

root1 **=** Node(1)

root2 **=** Node(1)

root1.left **=** Node(2)

root1.right **=** Node(3)

root1.left.left **=** Node(4)

root1.left.right **=** Node(5)

root2.left **=** Node(3)

root2.right **=** Node(2)

root2.right.left **=** Node(5)

root2.right.right **=** Node(4)

**if** areMirror(root1, root2):

**print** ("Yes")

**else**:

**print** ("No")

# This code is contributed by AshishR

**Output**

Yes

**Time Complexity:** O(n)

**Auxiliary Space:**O(h) where h is height of binary tree

*From <*[*https://www.geeksforgeeks.org/check-if-two-trees-are-mirror/*](https://www.geeksforgeeks.org/check-if-two-trees-are-mirror/)*>*

**Foldable Binary Trees**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 13 Dec, 2022
* Read
* Discuss(100+)
* Courses
* Practice
* Video

Given a binary tree, find out if the tree can be folded or not. A tree can be folded if the left and right subtrees of the tree are structure-wise mirror images of each other. An empty tree is considered foldable.

**Examples:**

***Input:***

*10*

*/     \*

*7      15*

*\     /*

*9  11*

***Output:****Can be folded*

***Input:***

*10*

*/  \*

*7   15*

*/    /*

*5   11*

***Output:****Cannot be folded*

Recommended Problem

Foldable Binary Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Expedia](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Expedia&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/foldable-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 35K

**Foldable Binary Trees by Changing Left Subtree to its Mirror:**

*The idea is to change the left subtree to its mirror then check that left subtree with its right subtree.*

Follow the steps below to solve the problem:

1. If tree is empty, then return true.
2. Convert the left subtree to its mirror image
3. Check if the structure of left subtree and right subtree is same and store the result.

* res = isStructSame(root->left, root->right). **isStructSame()** recursively compares structures of two subtrees and returns true if structures are same

1. Revert the changes made in step (2) to get the original tree.
2. Return result res stored in step 3.

Below is the implementation of the above approach.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 program to check foldable binary tree

#  A binary tree node has data,

# pointer to left child and a

# pointer to right child

**class** newNode:

**def** \_\_init\_\_(self, d):

        self.data **=** d

        self.left **=** None

        self.right **=** None

# Returns true if the given

# tree is foldable

**def** isFoldable(node):

    # base case

**if** node **==** None:

**return** true

    # convert left subtree to its mirror

    mirror(node.left)

    # Compare the structures of the right subtree and mirrored

    # left subtree

    res **=** isStructSame(node.left, node.right)

    # Get the original tree back

    mirror(node.left)

**return** res

**def** isStructSame(a, b):

**if** a **==** None **and** b **==** None:

**return** True

**if** a !**=** None **and** b !**=** None **and** isStructSame(a.left, b.left) **and** isStructSame(a.right, b.right):

**return** True

**return** False

**def** mirror(node):

**if** node **==** None:

**return**

**else**:

        # do the subtrees

        mirror(node.left)

        mirror(node.right)

        # swap the pointers in this node

        temp **=** node.left

        node.left **=** node.right

        node.right **=** temp

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    '''

    The constructed binary tree is

             1

           /   \

          2     3

           \    /

            4  5

    '''

    root **=** newNode(1)

    root.left **=** newNode(2)

    root.right **=** newNode(3)

    root.right.left **=** newNode(4)

    root.left.right **=** newNode(5)

**if** isFoldable(root):

**print**("tree is foldable")

**else**:

**print**("Tree is not foldable")

**Output**

tree is foldable

**Time complexity:** O(N), Visiting all the nodes of the tree of size N.

**Auxiliary Space:** O(N), If stack space is considered else O(1)

Thanks to ajaym for suggesting this approach.

**Foldable Binary Trees by Checking if Left and Right subtrees are Mirror:**

*The idea is to check the left and right subtree whether they are mirror or not.*

Follow the steps below to solve the problem:

* If tree is empty then return true.
* Else check if left and right subtrees are structure wise mirrors of each other. Use utility function IsFoldableUtil(root->left, root->right) for this.
* Checks if n1 and n2 are mirror of each other.
* If both trees are empty then return true.
* If one of them is empty and other is not then return false.
* Return true if following conditions are met
* n1->left is mirror of n2->right
* n1->right is mirror of n2->left

Below is the implementation of the above approach.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 program to check

# foldable binary tree

# Utility function to create a new

# tree node

**class** newNode:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** self.right **=** None

# Returns true if the given tree can be folded

**def** IsFoldable(root):

**if** (root **==** None):

**return** True

**return** IsFoldableUtil(root.left, root.right)

# A utility function that checks

# if trees with roots as n1 and n2

# are mirror of each other

**def** IsFoldableUtil(n1, n2):

    # If both left and right subtrees are NULL,

    # then return true

**if** n1 **==** None **and** n2 **==** None:

**return** True

    # If one of the trees is NULL and other is not,

    # then return false

**if** n1 **==** None **or** n2 **==** None:

**return** False

    # Otherwise check if left and

    # right subtrees are mirrors of

    # their counterparts

    d1 **=** IsFoldableUtil(n1.left, n2.right)

    d2 **=** IsFoldableUtil(n1.right, n2.left)

**return** d1 **and** d2

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    """ The constructed binary tree is

    1

    / \

    2 3

    \ /

    4 5

"""

    root **=** newNode(1)

    root.left **=** newNode(2)

    root.right **=** newNode(3)

    root.left.right **=** newNode(4)

    root.right.left **=** newNode(5)

**if** IsFoldable(root):

**print**("tree is foldable")

**else**:

        print("tree is not foldable")

# This code is contributed by

# Anupam Baranwal(anupambaranwal)

**Output**

tree is foldable

**Time Complexity:** O(N), Visiting every node of the tree of size N.

**Auxiliary Space:** O(N), If stack space is considered

**Foldable Binary Trees using**[Breadth first Search](https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/)**:**

*The idea is to use*[*Queue*](https://www.geeksforgeeks.org/queue-data-structure/)*for traversing the tree and using the*[*BFS*](https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/)*approach.*

Follow the steps below to solve the problem:

* The left child of the left subtree = the right child of the right subtree. Both of them should be not null.
* The right child of the left subtree = left child of the right subtree. Both of them should be null or not null.

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# class to create a node with key, left child and right child.

**class** Node:

**def** \_\_init\_\_(self, key):

        self.key **=** key

        self.left **=** None

        self.right **=** None

# Function to find whether the tree is foldable

**def** isFoldable(root):

     # Queue to store visited nodes

    q **=** []

    # Initially add the left and right nodes of root

**if** root !**=** None:

        q.append(root.left)

        q.append(root.right)

**while** (len(q) !**=** 0):

        # Remove the front 2 nodes to

        # check for None condition

        p **=** q.pop(0)

        r **=** q.pop(0)

        # If both are None, continue and check

        # the further elements

**if** (p **==** None **and** r **==** None):

**continue**

        # If one of them is not None, then return False

**if** ((p **==** None **and** r !**=** None) **or** (p !**=** None **and** r **==** None)):

**return** False

        ''' Insert in the same order:

            1. left of left subtree

            2. right of right subtree

            3. right of left subtree

            4. left of right subtree

        '''

        q.append(p.left)

        q.append(r.right)

        q.append(p.right)

        q.append(r.left)

    # Only if the tree is foldable

**return** True

# Driver code

# Insert data into the tree

root **=** Node(1)

root.left **=** Node(2)

root.right **=** Node(3)

root.right.left **=** Node(4)

root.left.right **=** Node(5)

# Function call

**if** isFoldable(root):

    print("tree is foldable")

**else**:

**print**("tree is not foldable")

    # This code is contributed by mariuscristiancapatina

**Output**

tree is foldable

**Time complexity:** O(N), Visiting all the nodes of the tree of size N.

**Auxiliary Space:** O(N), Using queue for storing nodes

*From <*[*https://www.geeksforgeeks.org/foldable-binary-trees/*](https://www.geeksforgeeks.org/foldable-binary-trees/)*>*

**Symmetric Tree (Mirror Image of itself)**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 28 Jun, 2022
* Read
* Discuss(60+)
* Courses
* Practice
* Video

Given a binary tree, check whether it is a mirror of itself.

**For example,** this binary tree is symmetric:

1  
 / \  
 2 2  
 / \ / \  
3 4 4 3

But the following is not:  
 1  
 / \  
 2 2  
 \ \  
 3 3

Recommended Problem

Symmetric Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Microsoft](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Microsoft&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/symmetric-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 88.5K

The idea is to write a recursive function isMirror() that takes two trees as an argument and returns true if trees are the mirror and false if trees are not mirrored. The isMirror() function recursively checks two roots and subtrees under the root.

Below is the implementation of the above algorithm.

* C++14
* C
* Java
* Python3
* C#
* Javascript

# Python program to check if a

# given Binary Tree is symmetric or not

# Node structure

**class** Node:

    # Utility function to create new node

**def** \_\_init\_\_(self, key):

        self.key **=** key

        self.left **=** None

        self.right **=** None

# Returns True if trees

#with roots as root1 and root 2  are mirror

**def** isMirror(root1, root2):

    # If both trees are empty, then they are mirror images

**if** root1 **is** None **and** root2 **is** None:

**return** True

    """ For two trees to be mirror images,

        the following three conditions must be true

        1 - Their root node's key must be same

        2 - left subtree of left tree and right subtree

          of the right tree have to be mirror images

        3 - right subtree of left tree and left subtree

           of right tree have to be mirror images

    """

**if** (root1 **is not** None **and** root2 **is not** None):

**if** root1.key **==** root2.key:

**return** (isMirror(root1.left, root2.right)**and**

                    isMirror(root1.right, root2.left))

    # If none of the above conditions is true then root1

    # and root2 are not mirror images

**return** False

**def** isSymmetric(root):

    # Check if tree is mirror of itself

**return** isMirror(root, root)

# Driver Code

# Let's construct the tree show in the above figure

root **=** Node(1)

root.left **=** Node(2)

root.right **=** Node(2)

root.left.left **=** Node(3)

root.left.right **=** Node(4)

root.right.left **=** Node(4)

root.right.right **=** Node(3)

**print** ("Symmetric" **if** isSymmetric(root) **==** True **else** "Not symmetric")

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Symmetric

**Time Complexity:**O(N)

**Auxiliary Space:**O(h) where h is the maximum height of the tree

*From <*[*https://www.geeksforgeeks.org/symmetric-tree-tree-which-is-mirror-image-of-itself/*](https://www.geeksforgeeks.org/symmetric-tree-tree-which-is-mirror-image-of-itself/)*>*

**Symmetric Tree (Mirror Image of itself)**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 28 Jun, 2022
* Read
* Discuss(60+)
* Courses
* Practice
* Video

Given a binary tree, check whether it is a mirror of itself.

**For example,** this binary tree is symmetric:

1  
 / \  
 2 2  
 / \ / \  
3 4 4 3

But the following is not:  
 1  
 / \  
 2 2  
 \ \  
 3 3

Recommended Problem

Symmetric Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Microsoft](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Microsoft&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/symmetric-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 88.5K

The idea is to write a recursive function isMirror() that takes two trees as an argument and returns true if trees are the mirror and false if trees are not mirrored. The isMirror() function recursively checks two roots and subtrees under the root.

Below is the implementation of the above algorithm.

* C++14
* C
* Java
* Python3
* C#
* Javascript

# Python program to check if a

# given Binary Tree is symmetric or not

# Node structure

**class** Node:

    # Utility function to create new node

**def** \_\_init\_\_(self, key):

        self.key **=** key

        self.left **=** None

        self.right **=** None

# Returns True if trees

#with roots as root1 and root 2  are mirror

**def** isMirror(root1, root2):

    # If both trees are empty, then they are mirror images

**if** root1 **is** None **and** root2 **is** None:

**return** True

    """ For two trees to be mirror images,

        the following three conditions must be true

        1 - Their root node's key must be same

        2 - left subtree of left tree and right subtree

          of the right tree have to be mirror images

        3 - right subtree of left tree and left subtree

           of right tree have to be mirror images

    """

**if** (root1 **is not** None **and** root2 **is not** None):

**if** root1.key **==** root2.key:

**return** (isMirror(root1.left, root2.right)**and**

                    isMirror(root1.right, root2.left))

    # If none of the above conditions is true then root1

    # and root2 are not mirror images

**return** False

**def** isSymmetric(root):

    # Check if tree is mirror of itself

**return** isMirror(root, root)

# Driver Code

# Let's construct the tree show in the above figure

root **=** Node(1)

root.left **=** Node(2)

root.right **=** Node(2)

root.left.left **=** Node(3)

root.left.right **=** Node(4)

root.right.left **=** Node(4)

root.right.right **=** Node(3)

**print** ("Symmetric" **if** isSymmetric(root) **==** True **else** "Not symmetric")

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Symmetric

**Time Complexity:**O(N)

**Auxiliary Space:**O(h) where h is the maximum height of the tree

*From <*[*https://www.geeksforgeeks.org/symmetric-tree-tree-which-is-mirror-image-of-itself/*](https://www.geeksforgeeks.org/symmetric-tree-tree-which-is-mirror-image-of-itself/)*>*

**Subtree with given sum in a Binary Tree**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 13 Jul, 2022
* Read
* Discuss
* Courses
* Practice
* Video

You are given a binary tree and a given sum. The task is to check if there exists a subtree whose sum of all nodes is equal to the given sum.

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image007.png

**Examples :**

*// For above tree*

***Input :****sum = 17*

***Output:****“Yes”*

*// sum of all nodes of subtree {3, 5, 9} = 17*

***Input :****sum = 11*

***Output:****“No”*

*// no subtree with given sum exist*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The idea is to traverse the tree in a Postorder fashion because here we have to think bottom-up. First, calculate the sum of the left subtree then the right subtree, and check if **sum\_left + sum\_right + cur\_node = sum**is satisfying the condition that means any subtree with a given sum exists. Below is the recursive implementation of the algorithm.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find if there is a

# subtree with given sum

# Binary Tree Node

""" utility that allocates a newNode

with the given key """

**class** newnode:

    # Construct to create a newNode

**def** \_\_init\_\_(self, key):

        self.data **=** key

        self.left **=** None

        self.right **=** None

# function to check if there exist any

# subtree with given sum

# cur\_sum -. sum of current subtree

#            from ptr as root

# sum\_left -. sum of left subtree from

#             ptr as root

# sum\_right -. sum of right subtree

#              from ptr as root

**def** sumSubtreeUtil(ptr,cur\_sum,sum):

    # base condition

**if** (ptr **==** None):

        cur\_sum[0] **=** 0

**return** False

    # Here first we go to left sub-tree,

    # then right subtree then first we

    # calculate sum of all nodes of subtree

    # having ptr as root and assign it as cur\_sum

    # cur\_sum = sum\_left + sum\_right + ptr.data

    # after that we check if cur\_sum == sum

    sum\_left, sum\_right **=** [0], [0]

    x**=**sumSubtreeUtil(ptr.left, sum\_left, sum)

    y**=**sumSubtreeUtil(ptr.right, sum\_right, sum)

    cur\_sum[0] **=** (sum\_left[0] **+**

                  sum\_right[0] **+** ptr.data)

**return** ((x **or** y)**or** (cur\_sum[0] **==** sum))

# Wrapper over sumSubtreeUtil()

**def** sumSubtree(root, sum):

    # Initialize sum of subtree with root

    cur\_sum **=** [0]

**return** sumSubtreeUtil(root, cur\_sum, sum)

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** newnode(8)

    root.left **=** newnode(5)

    root.right **=** newnode(4)

    root.left.left **=** newnode(9)

    root.left.right **=** newnode(7)

    root.left.right.left **=** newnode(1)

    root.left.right.right **=** newnode(12)

    root.left.right.right.right **=** newnode(2)

    root.right.right **=** newnode(11)

    root.right.right.left **=** newnode(3)

    sum **=** 22

**if** (sumSubtree(root, sum)) :

**print**("Yes" )

**else**:

**print**("No")

# This code is contributed by

# Shubham Singh(SHUBHAMSINGH10)

**Output**

Yes

**Time Complexity:** O(N), As we are visiting every node once.

**Auxiliary space:** O(h), Here h is the height of the tree and the extra space is used due to the recursion call stack.

*From <*[*https://www.geeksforgeeks.org/subtree-given-sum-binary-tree/*](https://www.geeksforgeeks.org/subtree-given-sum-binary-tree/)*>*

**Succinct Encoding of Binary Tree**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 22 Dec, 2022
* Read
* Discuss
* Courses
* Practice
* Video

A succinct encoding of Binary Tree takes close to minimum possible space. The number of structurally different binary trees on n nodes is [n’th Catalan number](https://www.geeksforgeeks.org/program-nth-catalan-number/). For large n, this is about 4n; thus we need at least about log2 4 n = 2n bits to encode it. A succinct binary tree therefore would occupy 2n+o(n) bits.

One simple representation which meets this bound is to visit the nodes of the tree in preorder, outputting “1” for an internal node and “0” for a leaf. If the tree contains data, we can simply simultaneously store it in a consecutive array in preorder.

Below is algorithm for encoding:

function EncodeSuccinct(node n, bitstring structure, array data) {  
 if n = nil then  
 append 0 to structure;  
 else  
 append 1 to structure;  
 append n.data to data;  
 EncodeSuccinct(n.left, structure, data);  
 EncodeSuccinct(n.right, structure, data);  
}

And below is algorithm for decoding

function DecodeSuccinct(bitstring structure, array data) {  
 remove first bit of structure and put it in b  
 if b = 1 then  
 create a new node n  
 remove first element of data and put it in n.data  
 n.left = DecodeSuccinct(structure, data)  
 n.right = DecodeSuccinct(structure, data)  
 return n  
 else  
 return nil  
}

**Example:**

Input:   
 10  
 / \  
 20 30  
 / \ \  
 40 50 70

Data Array (Contains preorder traversal)  
10 20 40 50 30 70

Structure Array  
1 1 1 0 0 1 0 0 1 0 1 0 0   
1 indicates data and 0 indicates NULL

Below is the implementation of above algorithms.

* C++
* Java
* Python3
* C#
* Javascript

# Python program to demonstrate Succinct Tree Encoding and Decoding

# Node structure

**class** Node:

    # Utility function to create new Node

**def** \_\_init\_\_(self , key):

        self.key **=** key

        self.left **=** None

        self.right **=** None

**def** EncodeSuccinct(root , struc , data):

    # If root is None , put 0 in structure array and return

**if** root **is** None :

        struc.append(0)

**return**

    # Else place 1 in structure array, key in 'data' array

    # and recur for left and right children

    struc.append(1)

    data.append(root.key)

    EncodeSuccinct(root.left , struc , data)

    EncodeSuccinct(root.right , struc ,data)

# Constructs tree from 'struc' and 'data'

**def** DecodeSuccinct(struc , data):

**if**(len(struc) <**=** 0):

**return** None

    # Remove one item from structure list

    b **=** struc[0]

    struc.pop(0)

    # If removed bit is 1

**if** b **==** 1:

        key **=** data[0]

        data.pop(0)

        #Create a tree node with removed data

        root **=** Node(key)

        #And recur to create left and right subtrees

        root.left **=** DecodeSuccinct(struc , data);

        root.right **=** DecodeSuccinct(struc , data);

**return** root

**return** None

**def** preorder(root):

**if** root **is not** None:

        print ("key: %d" **%**(root.key),end**=**" ")

**if** root.left **is not** None:

**print** ("| left child: %d" **%**(root.left.key),end**=**" ")

**if** root.right **is not** None:

**print** ("| right child %d" **%**(root.right.key),end**=**" ")

        print ()

        preorder(root.left)

        preorder(root.right)

# Driver Program

root **=** Node(10)

root.left **=** Node(20)

root.right **=** Node(30)

root.left.left **=** Node(40)

root.left.right **=** Node(50)

root.right.right **=** Node(70)

print ("Given Tree")

preorder(root)

struc **=** []

data **=** []

EncodeSuccinct(root , struc , data)

print ("\nEncoded Tree")

print ("Structure List")

**for** i **in** struc:

**print** (i ,end**=**" ")

**print** ("\nDataList")

**for** value **in** data:

    print (value,end**=**" ")

newroot **=** DecodeSuccinct(struc , data)

**print** ("\n\nPreorder Traversal of decoded tree")

preorder(newroot)

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Given Tree  
key: 10 | left child: 20 | right child: 30  
key: 20 | left child: 40 | right child: 50  
key: 40  
key: 50  
key: 30 | right child: 70  
key: 70

Encoded Tree  
Structure List  
1 1 1 0 0 1 0 0 1 0 1 0 0   
Data List  
10 20 40 50 30 70

Preorder traversal of decoded tree  
key: 10 | left child: 20 | right child: 30  
key: 20 | left child: 40 | right child: 50  
key: 40  
key: 50  
key: 30 | right child: 70  
key: 70

**Time complexity: O(n)**

**Auxiliary** **space: O(n)**.

*From <*[*https://www.geeksforgeeks.org/succinct-encoding-of-binary-tree/*](https://www.geeksforgeeks.org/succinct-encoding-of-binary-tree/)*>*

**Write a program to Calculate Size of a tree | Recursion**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 14 Sep, 2022
* Read
* Discuss(100)
* Courses
* Practice
* Video

Size of a tree is the number of elements present in the tree. Size of the below tree is 5.



Size() function recursively calculates the size of a tree. It works as follows:

Size of a tree = Size of left subtree + 1 + Size of right subtree.

Recommended Problem

Size of Binary Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/size-of-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 44.4K

**Algorithm:**

size(tree)  
1. If tree is empty then return 0  
2. Else  
 (a) Get the size of left subtree recursively i.e., call   
 size( tree->left-subtree)  
 (a) Get the size of right subtree recursively i.e., call   
 size( tree->right-subtree)  
 (c) Calculate size of the tree as following:  
 tree\_size = size(left-subtree) + size(right-  
 subtree) + 1  
 (d) Return tree\_size

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python Program to find the size of binary tree

# A binary tree node

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Computes the number of nodes in tree

**def** size(node):

**if** node **is** None:

**return** 0

**else**:

**return** (size(node.left)**+** 1 **+** size(node.right))

# Driver program to test above function

root **=** Node(1)

root.left **=** Node(2)

root.right **=** Node(3)

root.left.left  **=** Node(4)

root.left.right **=** Node(5)

print("Size of the tree is %d" **%**(size(root)))

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output:**

Size of the tree is 5

**Time Complexity:** O(N)

As every node is visited once.

**Auxiliary Space:**O(N)

*From <*[*https://www.geeksforgeeks.org/write-a-c-program-to-calculate-size-of-a-tree/*](https://www.geeksforgeeks.org/write-a-c-program-to-calculate-size-of-a-tree/)*>*

**Diameter of a Binary Tree**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 15 Sep, 2022
* Read
* Discuss(540+)
* Courses
* Practice
* Video

*The****diameter/width of a tree****is defined as the number of nodes on the longest path between two end nodes.*

*The diagram below shows two trees each with a diameter of nine, the leaves that form the ends of the longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).*



Recommended Problem

Diameter of a Binary Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Microsoft](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Microsoft&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/diameter-of-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 2L

**Approach:** The diameter of a tree T is the largest of the following quantities:

* *the diameter of T’s left subtree.*
* *the diameter of T’s right subtree.*
* *the longest path between leaves that goes through the root of T (this can be computed from the heights of the subtrees of T)*

Below is the implementation of the above approach

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 program to find the diameter of binary tree

# A binary tree node

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# The function Compute the "height" of a tree. Height is the

# number of nodes along the longest path from the root node

# down to the farthest leaf node.

**def** height(node):

    # Base Case : Tree is empty

**if** node **is** None:

**return** 0

    # If tree is not empty then height = 1 + max of left

    # height and right heights

**return** 1 **+** max(height(node.left), height(node.right))

# Function to get the diameter of a binary tree

**def** diameter(root):

    # Base Case when tree is empty

**if** root **is** None:

**return** 0

    # Get the height of left and right sub-trees

    lheight **=** height(root.left)

    rheight **=** height(root.right)

    # Get the diameter of left and right sub-trees

    ldiameter **=** diameter(root.left)

    rdiameter **=** diameter(root.right)

    # Return max of the following tree:

    # 1) Diameter of left subtree

    # 2) Diameter of right subtree

    # 3) Height of left subtree + height of right subtree +1

**return** max(lheight **+** rheight **+** 1, max(ldiameter, rdiameter))

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    """

    Constructed binary tree is

                1

              /   \

            2      3

          /  \

        4     5

    """

    root **=** Node(1)

    root.left **=** Node(2)

    root.right **=** Node(3)

    root.left.left **=** Node(4)

    root.left.right **=** Node(5)

    # Function Call

**print**(diameter(root))

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Diameter of the given binary tree is 4

**Time Complexity:** O(N2)

**Auxiliary Space**: O(N) for call stack

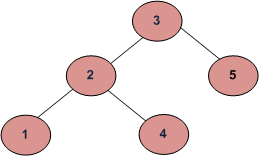
*From <*[*https://www.geeksforgeeks.org/diameter-of-a-binary-tree/*](https://www.geeksforgeeks.org/diameter-of-a-binary-tree/)*>*

**Get Level of a node in a Binary Tree**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 21 Nov, 2022
* Read
* Discuss(130+)
* Courses
* Practice
* Video

Given a Binary Tree and a key, write a function that returns level of the key.

For example, consider the following tree. If the input key is 3, then your function should return 1. If the input key is 4, then your function should return 3. And for key which is not present in key, then your function should return 0.



Recommended Problem

Level of a Node in Binary Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/level-of-a-node-in-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 20.1K

The idea is to start from the root and level as 1. If the key matches with root’s data, return level. Else recursively call for left and right subtrees with level as level + 1.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 program to Get Level of a

# node in a Binary Tree

# Helper function that allocates a

# new node with the given data and

# None left and right pairs.

**class** newNode:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Helper function for getLevel(). It

# returns level of the data if data is

# present in tree, otherwise returns 0

**def** getLevelUtil(node, data, level):

**if** (node **==** None):

**return** 0

**if** (node.data **==** data):

**return** level

    downlevel **=** getLevelUtil(node.left,

                             data, level **+** 1)

**if** (downlevel !**=** 0):

**return** downlevel

    downlevel **=** getLevelUtil(node.right,

                             data, level **+** 1)

**return** downlevel

# Returns level of given data value

**def** getLevel(node, data):

**return** getLevelUtil(node, data, 1)

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Let us construct the Tree shown

    # in the above figure

    root **=** newNode(3)

    root.left **=** newNode(2)

    root.right **=** newNode(5)

    root.left.left **=** newNode(1)

    root.left.right **=** newNode(4)

**for** x **in** range(1, 6):

        level **=** getLevel(root, x)

**if** (level):

            print("Level of", x,

                  "is", getLevel(root, x))

**else**:

            print(x, "is not present in tree")

# This code is contributed by

# Shubham Singh(SHUBHAMSINGH10)

**Output**

Level of 1 is 3  
Level of 2 is 2  
Level of 3 is 1  
Level of 4 is 3  
Level of 5 is 2

**Time Complexity:** O(n) where n is the number of nodes in the given Binary Tree.

**Auxiliary Space:**O(n)

**Alternative Approach:**The given problem can be solved with the help of level order traversal of given binary tree.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to print level in which X is present in

# binary tree

# A node structure

**class** Node:

    # A utility function to create a new node

**def** \_\_init\_\_(self, key):

        self.data **=** key

        self.left **=** None

        self.right **=** None

**def** printLevel(root, X):

    # Base Case

**if** root **is** None:

**return** 0

    # Create an empty queue

    # for level order traversal

    q **=** []

    #Create a var represent current level of tree

    currLevel **=** 1

    # Enqueue Root

    q.append(root)

**while**(len(q) > 0):

        size **=** len(q)

**for** i **in** range(size):

            node **=** q.pop(0)

**if**(node.data **==** X):

**return** currLevel

            # Enqueue left child

**if** node.left **is not** None:

                q.append(node.left)

            # Enqueue right child

**if** node.right **is not** None:

                q.append(node.right)

        currLevel **+=** 1

**return** 0

# Driver Program to test above function

root **=** Node(1)

root.left **=** Node(2)

root.right **=** Node(3)

root.left.left **=** Node(4)

root.left.right **=** Node(5)

root.right.left **=** Node(7)

root.right.right **=** Node(6)

**print**(printLevel(root, 6))

# This code is contributed by Abhijeet Kumar(abhijeet19403)

**Output**

3

**Time Complexity: O(n)** where n is the number of nodes in the binary tree.

**Auxiliary Space: O(n)** where n is the number of nodes in the binary tree.