Diameter of a Binary Tree - GeeksforGeeks

Source: https://www.geeksforgeeks.org/diameter-of-a-binary-tree/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Diameter of a Binary Tree Last Updated : 31 Jan, 2026 Given the root of a binary tree, find its diameter. The diameter of a tree is defined as the number of edges in the longest path between any two nodes. Examples: Input: The binary tree for this example is shown in the image below. Output: 2 Explanation: The longest path has 2 edges (the path 2 -> 1 -> 3). Input: The binary tree for this example is shown in the image below. Output: 4 Explanation: The longest path has 4 edges (the path 5 -> 3 -> 2 -> 4 -> 6). Try it on GfG Practice Table of Content [Naive Approach] By Calculating Height For Each Node - O(n^2) Time and O(h) Space [Expected Approach] Using Single Traversal - O(n) Time and O(h) Space [Naive Approach] By Calculating Height For Each Node - O(n 2) Time and O(h) Space The core idea involves recursively traversing the tree. At each node, we determine the heights of its left and right subtrees and update the maximum diameter by calculating the sum of these heights. C++ #include <iostream> #include <algorithm> using namespace std ; // Node Structure class Node { public : int data ; Node * left , * right ; Node (int x) { data = x ; left = nullptr ; right = nullptr ; } }; // Function to compute the height of a tree. int height (Node * root) { if (root == nullptr) return 0 ; // If tree is not empty then height = 1 + max of left height and right heights return 1 + max (height (root -> left), height (root -> right)); } // Function to get diameter of a binary tree int diameter (Node * root) { if (root == nullptr) return 0 ; // Get the height of left and right sub-trees int lheight = height (root -> left); int rheight = height (root -> right); // Get the diameter of left and right sub-trees int ldiameter = diameter (root -> left); int rdiameter = diameter (root -> right); return max ({ lheight + rheight , ldiameter , rdiameter }); } int main () { Node * root = new Node (1); root -> right = new Node (2); root -> right -> left = new Node (3); root -> right -> right = new Node (4); root -> right -> left -> left = new Node (5); root -> right -> right -> right = new Node (6); cout << diameter (root) << endl ; return 0 ; } C #include <stdio.h> #include <stdlib.h> // Node Structure struct Node { int data ; struct Node * left ; struct Node * right ; }; // Function to compute the height of a tree. int height (struct Node * root) { if (root == NULL) return 0 ; // If tree is not empty then height = 1 + max of left height and right heights int leftHeight = height (root -> left); int rightHeight = height (root -> right); return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight); } // Function to get diameter of a binary tree int diameter (struct Node * root) { if (root == NULL) return 0 ; // Get the height of left and right sub-trees int lheight = height (root -> left); int rheight = height (root -> right); // Get the diameter of left and right sub-trees int ldiameter = diameter (root -> left); int rdiameter = diameter (root -> right); // Diameter of current subtree int curr = lheight + rheight ; if (ldiameter > rdiameter && ldiameter > curr) return ldiameter ; else if (rdiameter > ldiameter && rdiameter > curr) return rdiameter ; return curr ; } struct Node * createNode (int x) { struct Node * newNode = (struct Node *) malloc (sizeof (struct Node)); newNode -> data = x ; newNode -> left = NULL ; newNode -> right = NULL ; return newNode ; } int main () { struct Node * root = createNode (1); root -> right = createNode (2); root -> right -> left = createNode (3); root -> right -> right = createNode (4); root -> right -> left -> left = createNode (5); root -> right -> right -> right = createNode (6); printf ("%d \n " , diameter (root)); return 0 ; } Java import java.util.ArrayList ; // Node Structure class Node { int data ; Node left , right ; Node (int x) { data = x ; left = null ; right = null ; } } class GFG { // Function to compute the height of a tree. static int height (Node root) { if (root == null) return 0 ; // If tree is not empty then height = 1 + max of left height and right heights return 1 + Math . max (height (root . left), height (root . right)); } // Function to get diameter of a binary tree static int diameter (Node root) { if (root == null) return 0 ; // Get the height of left and right sub-trees int lheight = height (root . left); int rheight = height

(root . right); // Get the diameter of left and right sub-trees int ldiameter = diameter (root . left); int rdiameter = diameter (root . right); return Math . max (lheight + rheight , Math . max (ldiameter , rdiameter)); } public static void main (String [] args) { Node root = new Node (1); root . right = new Node (2); root . right . left = new Node (3); root . right . right = new Node (4); root . right . left . left = new Node (5); root . right . right . right = new Node (6); System . out . println (diameter (root)); } } Python # Node Structure class Node : def __init__ (self , x): self . data = x self . left = None self . right = None # Function to compute the height of a tree def height (root): if root is None : return 0 # If tree is not empty then height = 1 + max of left height and right heights return 1 + max (height (root . left), height (root . right)) # Function to get diameter of a binary tree def diameter (root): 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 (lheight + rheight , ldiameter , rdiameter) if __name__ == "__main__" : root = Node (1) root . right = Node (2) root . right . left = Node (3) root . right . right = Node (4) root . right . left . left = Node (5) root . right . right . right = Node (6) print (diameter (root)) C# using System ; using System.Collections.Generic ; // Node Structure class Node { public int data ; public Node left , right ; public Node (int x) { data = x ; left = null ; right = null ; } } class GFG { // Function to compute the height of a tree static int height (Node root) { if (root == null) return 0 ; // If tree is not empty then height = 1 + max of left height and right heights return 1 + Math . Max (height (root . left), height (root . right)); } // Function to get diameter of a binary tree static int diameter (Node root) { if (root == null) return 0 ; // Get the height of left and right sub-trees int lheight = height (root . left); int rheight = height (root . right); // Get the diameter of left and right sub-trees int ldiameter = diameter (root . left); int rdiameter = diameter (root . right); return Math . Max (lheight + rheight , Math . Max (ldiameter , rdiameter)); } static void Main (string [] args) { Node root = new Node (1); root . right = new Node (2); root . right . left = new Node (3); root . right . right = new Node (4); root . right . left . left = new Node (5); root . right . right . right = new Node (6); Console . WriteLine (diameter (root)); } } JavaScript // Node Structure class Node { constructor (x) { this . data = x ; this . left = null ; this . right = null ; } } // Function to compute the height of a tree. function height (root) { if (root === null) return 0 ; // If tree is not empty then height = 1 + max of left height and right heights return 1 + Math . max (height (root . left), height (root . right)); } // Function to get diameter of a binary tree function diameter (root) { if (root === null) return 0 ; // Get the height of left and right sub-trees const lheight = height (root . left); const rheight = height (root . right); // Get the diameter of left and right sub-trees const ldiameter = diameter (root . left); const rdiameter = diameter (root . right); return Math . max (lheight + rheight , ldiameter , rdiameter); } let root = new Node (1); root . right = new Node (2); root . right . left = new Node (3); root . right . right = new Node (4); root . right . left . left = new Node (5); root . right . right . right = new Node (6); console . log (diameter (root)); Output 4 [Expected Approach] Using Single Traversal - O(n) Time and O(h) Space The core idea is to efficiently calculate the diameter, avoiding redundant height calculations. We use recursion to find both height and diameter in a single traversal. For each node, the longest path through it is the sum of its left and right subtree heights. By tracking the maximum diameter, we find the longest path. C++ #include <iostream> using namespace std ; // Node Structure class Node { public : int data ; Node * left ; Node * right ; Node (int x) { data = x ; left = nullptr ; right = nullptr ; } }; // Global variable to store the maximum diameter int maxDiameter = 0 ; int diameterRecur (Node * root) { if (! root) return 0 ; // Find the height of left and right subtree int lHeight = diameterRecur (root -> left); int rHeight = diameterRecur (root -> right); // Update the global max diameter if this node gives a longer path if (lHeight + rHeight > maxDiameter) maxDiameter = lHeight + rHeight ; // Return height of current subtree return 1 + max (lHeight , rHeight); } // Function to get diameter of a binary tree int diameter (Node * root) { maxDiameter = 0 ; diameterRecur (root); return maxDiameter ; } int main () { Node * root = new Node (1); root -> right = new Node (2); root -> right -> left = new Node (3); root -> right -> right = new Node (4); root -> right -> left -> left = new Node (5); root -> right -> right -> right = new Node (6); cout << diameter (root) << endl ; return 0 ; } C #include <stdio.h> #include <stdlib.h> // Node Structure struct Node { int data ; struct Node * left ; struct Node * right ; }; // Function to create a new Node struct Node * createNode (int x) { struct Node * node = (struct Node *) malloc (sizeof (struct Node)); node -> data = x ; node -> left = NULL ; node -> right = NULL ; return node ; } int max (int a , int b) { return a > b ? a : b ; } // Global variable to store the maximum diameter int maxDiameter = 0 ; int diameterRecur (struct Node * root) { if (root == NULL) return 0 ; // Find the height of left and right subtree int lHeight = diameterRecur (root -> left); int rHeight = diameterRecur (root -> right); // Update the global max diameter if this node gives a longer path if (lHeight + rHeight > maxDiameter) maxDiameter = lHeight + rHeight ; // Return height of current subtree return 1 + max (lHeight , rHeight); } // Function to get diameter of a binary tree int diameter (struct Node * root) { maxDiameter = 0 ; diameterRecur (root); return maxDiameter ; } int main () { struct Node * root = createNode (1); root -> right = createNode (2); root -> right -> left = createNode (3); root -> right -> right = createNode (4); root -> right -> left -> left = createNode (5); root -> right -> right -> right = createNode (6); printf ("%d \n " , diameter (root)); return 0 ; } Java // Node Structure class Node { int data ; Node left , right ; Node (int x) { data = x ; left = null ; right = null ; } } class GFG { // Static variable to store maximum diameter static int maxDiameter = 0 ; // Recursive function to calculate height and update diameter static int diameterRecur (Node root) { if (root == null) return 0 ; // Find the height of left and right subtree int lHeight = diameterRecur (root . left); int rHeight = diameterRecur (root . right); // Update the global max diameter if this node gives a longer path if (lHeight + rHeight > maxDiameter) maxDiameter = lHeight + rHeight ; // Return height of current subtree return 1 + Math . max (lHeight , rHeight); } // Function to get diameter of a binary tree static int diameter (Node root) { maxDiameter = 0 ; diameterRecur (root); return maxDiameter ; } public static void main (String [] args) { Node root = new Node (1); root . right = new Node (2); root . right . left = new Node (3); root . right . right = new Node (4); root . right . left . left = new Node (5); root . right . right . right = new Node (6); System . out . println (diameter (root)); } } Python # Node Structure class Node : def __init__ (self , x): self . data = x self . left = None self . right = None # Global variable to store the maximum diameter maxDiameter = 0 # Recursive function to calculate height and update diameter def diameterRecur (root): global maxDiameter if root is None : return 0 # Find the height of left and right subtree lHeight = diameterRecur (root . left) rHeight = diameterRecur (root . right) # Update the global max diameter if this node gives a longer path maxDiameter = max (maxDiameter , lHeight + rHeight) # Return height of current subtree return 1 + max (lHeight , rHeight) # Function to get diameter of a binary tree def diameter (root): global maxDiameter maxDiameter = 0 diameterRecur (root) return maxDiameter if __name__ == "__main__" : root = Node (1) root . right = Node (2) root . right . left = Node (3) root . right . right = Node (4) root . right . left . left = Node (5) root . right . right . right = Node (6) print (diameter (root)) C# using System ; // Node Structure class Node { public int data ; public Node left , right ; public Node (int x) { data = x ; left = null ; right = null ; } } class GFG { // global variable to store maximum diameter static int maxDiameter = 0 ; // Recursive function which finds the diameter of the tree. static int diameterRecur (Node root) { if (root == null) return 0 ; // find the height of left and right subtree int lHeight = diameterRecur (root . left); int rHeight = diameterRecur (root . right); // Check if diameter of root is greater than maxDiameter. maxDiameter = Math . Max (maxDiameter , lHeight + rHeight); // return the height of current subtree. return 1 + Math . Max (lHeight , rHeight); } // Function to get diameter of a binary tree static int diameter (Node root) { maxDiameter = 0 ; diameterRecur (root); return maxDiameter ; } static void Main (string [] args) { Node root = new Node (1); root . right = new Node (2); root . right . left = new Node (3); root . right . right = new Node (4); root . right . left . left = new Node (5); root . right . right . right = new Node (6); Console . WriteLine (diameter (root)); } } JavaScript // Node Structure class Node { constructor (x) { this . data = x ; this . left = null ; this . right = null ; } } // global variable to store the maximum diameter let maxDiameter = 0 ; // Recursive function which finds the diameter of the tree. function diameterRecur (root) { if (root === null) return 0 ; // find the height of left and right subtree let lHeight = diameterRecur (root . left); let rHeight = diameterRecur (root . right); // Check if diameter of root is greater than maxDiameter. maxDiameter = Math . max (maxDiameter , lHeight + rHeight); // return the height of current subtree. return 1 + Math . max (lHeight , rHeight); } // Function to get diameter of a binary tree function diameter (root) { maxDiameter = 0 ; diameterRecur (root); return maxDiameter ; } let root = new Node (1); root . right = new Node (2); root . right . left = new Node (3); root . right . right = new Node (4); root . right . left . left = new Node (5); root . right . right . right = new Node (6); console . log (diameter (root)); Output 4 Related article: Diameter of an N-ary tree Comment Article Tags: Article Tags: Tree DSA Microsoft Amazon Oracle Directi VMWare Cadence India Snapdeal MakeMyTrip Salesforce OYO Philips + 9 More