Find distance between two nodes of a Binary Tree - GeeksforGeeks

Source: https://www.geeksforgeeks.org/find-distance-between-two-nodes-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 Find distance between two nodes of a Binary Tree Last Updated : 23 Jul, 2025 Given a Binary tree , the task is to find the distance between two keys in a binary tree, no parent pointers are given. The distance between two nodes is the minimum number of edges to be traversed to reach one node from another. The given two nodes are guaranteed to be in the binary tree and all node values are unique . Table of Content Using LCA and Path Length - O(n) Time and O(h) Space Using LCA - O(n) Time and O(h) Space Using LCA (one pass) - O(n) Time and O(h) Space Using LCA and Path Length - O(n) Time and O(h) Space The idea to first find the Lowest Common Ancestor (LCA) of two given nodes in a binary tree. Once the LCA is found, we calculate the distance between the two target nodes by finding the path length f rom the root to each node and then subtracting twice the path length from the root to the LCA. Dist(n1, n2) = Dist(root, n1) + Dist(root, n2) - 2*Dist(root, lca) n1' and 'n2' are the two given keys 'root' is root of given Binary Tree. 'lca' is lowest common ancestor of n1 and n2 Dist(n1, n2) is the distance between n1 and n2. Step-By-Step Implementation : Start by traversing the tree to find the level of the first target node. If the node is found, return its level; otherwise, continue searching recursively in the left and right subtrees. Recursively check each node to see if it matches either of the target nodes. If a match is found, store the current level. If both nodes are located in different subtrees, calculate the distance based on their levels. If only one of the target nodes is found, determine the distance to the other node using the level obtained from the LCA. This involves further traversals to find the level of the remaining target node. Using the levels obtained, compute the total distance between the two nodes by adding their levels and subtracting twice the level of the LCA. This gives the direct distance between the two target nodes. Finally, return the calculated distance. Below is the implementation of the above approach. C++ // C++ Program to Find distance between // two nodes of a Binary Tree #include <bits/stdc++.h> using namespace std ; class Node { public : int data ; Node * left , * right ; Node (int val) { data = val ; left = nullptr ; right = nullptr ; } }; // Function to find the level of a node int findLevel (Node * root , int k , int level) { if (root == nullptr) return -1 ; if (root -> data == k) return level ; // Recursively call function on left child int leftLevel = findLevel (root -> left , k , level + 1); // If node is found on left, return level // Else continue searching on the right child if (leftLevel != -1) { return leftLevel ; } else { return findLevel (root -> right , k , level + 1); } } // Function to find the lowest common ancestor // and calculate distance between two nodes Node * findLcaAndDistance (Node * root , int a , int b , int & d1 , int & d2 , int & dist , int lvl) { if (root == nullptr) return nullptr ; if (root -> data == a) { // If first node found, store level and // return the node d1 = lvl ; return root ; } if (root -> data == b) { // If second node found, store level and // return the node d2 = lvl ; return root ; } // Recursively call function on left child Node * left = findLcaAndDistance (root -> left , a , b , d1 , d2 , dist , lvl + 1); // Recursively call function on right child Node * right = findLcaAndDistance (root -> right , a , b , d1 , d2 , dist , lvl + 1); if (left != nullptr && right != nullptr) { // If both nodes are found in different // subtrees, calculate the distance dist = d1 + d2 - 2 * lvl ; } // Return node found or nullptr if not found if (left != nullptr) { return left ; } else { return right ; } } // Function to find distance between two nodes int findDist (Node * root , int a , int b) { int d1 = -1 , d2 = -1 , dist ; // Find lowest common ancestor and calculate distance Node * lca = findLcaAndDistance (root , a , b , d1 , d2 , dist , 1); if (d1 != -1 && d2 != -1) { // Return the distance if both // nodes are found return dist ; } if (d1 != -1) { // If only first node is found, find // distance to

second node dist = findLevel (lca , b , 0); return dist ; } if (d2 != -1) { // If only second node is found, find // distance to first node dist = findLevel (lca , a , 0); return dist ; } // Return -1 if both nodes not found return -1 ; } int main () { // Hardcoded binary tree // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node * root = new Node (1); root -> left = new Node (2); root -> right = new Node (3); root -> left -> left = new Node (4); root -> left -> right = new Node (5); root -> right -> left = new Node (6); root -> right -> right = new Node (7); int a = 4 , b = 7 ; cout << findDist (root , a , b) << endl ; return 0 ; } C // C Program to Find distance between two // nodes of a Binary Tree #include <stdio.h> #include <stdlib.h> struct Node { int data ; struct Node * left , * right ; }; // Function to find the level of a node int findLevel (struct Node * root , int k , int level) { if (root == NULL) return -1 ; if (root -> data == k) return level ; // Recursively call function on left child int leftLevel = findLevel (root -> left , k , level + 1); // If node is found on left, return level // Else continue searching on the right child if (leftLevel != -1) { return leftLevel ; } else { return findLevel (root -> right , k , level + 1); } } // Function to find the lowest common ancestor // and calculate distance between two nodes struct Node * findLcaAndDistance (struct Node * root , int a , int b , int * d1 , int * d2 , int * dist , int lvl) { if (root == NULL) return NULL ; if (root -> data == a) { // If first node found, store level and // return the node * d1 = lvl ; return root ; } if (root -> data == b) { // If second node found, store level and // return the node * d2 = lvl ; return root ; } // Recursively call function on left child struct Node * left = findLcaAndDistance (root -> left , a , b , d1 , d2 , dist , lvl + 1); // Recursively call function on right child struct Node * right = findLcaAndDistance (root -> right , a , b , d1 , d2 , dist , lvl + 1); if (left != NULL && right != NULL) { // If both nodes are found in different // subtrees, calculate the distance * dist = * d1 + * d2 - 2 * lvl ; } // Return node found or NULL if not found if (left != NULL) { return left ; } else { return right ; } } // Function to find distance between two nodes int findDist (struct Node * root , int a , int b) { int d1 = -1 , d2 = -1 , dist ; // Find lowest common ancestor and calculate distance struct Node * lca = findLcaAndDistance (root , a , b , & d1 , & d2 , & dist , 1); if (d1 != -1 && d2 != -1) { // Return the distance if both nodes // are found return dist ; } if (d1 != -1) { // If only first node is found, find // distance to second node dist = findLevel (lca , b , 0); return dist ; } if (d2 != -1) { // If only second node is found, find // distance to first node dist = findLevel (lca , a , 0); return dist ; } // Return -1 if both nodes not found return -1 ; } struct Node * createNode (int value) { struct Node * node = (struct Node *) malloc (sizeof (struct Node)); node -> data = value ; node -> left = node -> right = NULL ; return node ; } int main () { // Hardcoded binary tree // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 struct Node * root = createNode (1); root -> left = createNode (2); root -> right = createNode (3); root -> left -> left = createNode (4); root -> left -> right = createNode (5); root -> right -> left = createNode (6); root -> right -> right = createNode (7); int a = 4 , b = 7 ; printf ("%d \n " , findDist (root , a , b)); return 0 ; } Java // Java Program to Find distance between two // nodes of a Binary Tree class Node { public int data ; public Node left , right ; Node (int val) { data = val ; left = null ; right = null ; } } class GfG { // Function to find the level of a node static int findLevel (Node root , int k , int level) { if (root == null) return - 1 ; if (root . data == k) return level ; // Recursively call function on left child int leftLevel = findLevel (root . left , k , level + 1); // If node is found on left, return level // Else continue searching on the right child if (leftLevel != - 1) { return leftLevel ; } else { return findLevel (root . right , k , level + 1); } } // Function to find the lowest common ancestor // and calculate distance between two nodes static Node findLcaAndDistance (Node root , int a , int b , int [] d1 , int [] d2 , int [] dist , int lvl) { if (root == null) return null ; if (root . data == a) { // If first node found, store level and // return the node d1 [0] = lvl ; return root ; } if (root . data == b) { // If second node found, store level and // return the node d2 [0] = lvl ; return root ; } // Recursively call function on left child Node left = findLcaAndDistance (root . left , a , b , d1 , d2 , dist , lvl + 1); // Recursively call function on right child Node right = findLcaAndDistance (root . right , a , b , d1 , d2 , dist , lvl + 1); if (left != null && right != null) { // If both nodes are found in different // subtrees, calculate the distance dist [0] = d1 [0] + d2 [0] - 2 * lvl ; } // Return node found or null if not found if (left != null) { return left ; } else { return right ; } } // Function to find distance between two nodes static int findDist (Node root , int a , int b) { int [] d1 = { - 1 }, d2 = { - 1 }, dist = { 0 }; // Find lowest common ancestor and calculate distance Node lca = findLcaAndDistance (root , a , b , d1 , d2 , dist , 1); if (d1 [0] != - 1 && d2 [0] != - 1) { // Return the distance if both nodes are found return dist [0] ; } if (d1 [0] != - 1) { // If only first node is found, find // distance to second node dist [0] = findLevel (lca , b , 0); return dist [0] ; } if (d2 [0] != - 1) { // If only second node is found, find // distance to first node dist [0] = findLevel (lca , a , 0); return dist [0] ; } // Return -1 if both nodes not found return - 1 ; } public static void main (String [] args) { // Hardcoded binary tree // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); int a = 4 , b = 7 ; System . out . println (findDist (root , a , b)); } }

Python # Python Program to Find distance between # two nodes of a Binary Tree class Node : def __init__ (self , val): self . data = val self . left = None self . right = None # Function to find the level of a node def findLevel (root , k , level): if root is None : return - 1 if root . data == k : return level # Recursively call function on left child leftLevel = findLevel (root . left , k , level + 1) # If node is found on left, return level # Else continue searching on the right child if leftLevel != - 1 : return leftLevel else : return findLevel (root . right , k , level + 1) # Function to find the lowest common ancestor # and calculate distance between two nodes def findLcaAndDistance (root , a , b , d1 , d2 , dist , lvl): if root is None : return None if root . data == a : # If first node found, store level and # return the node d1 [0] = lvl return root if root . data == b : # If second node found, store level and # return the node d2 [0] = lvl return root # Recursively call function on left child left = findLcaAndDistance (root . left , a , b , d1 , d2 , dist , lvl + 1) # Recursively call function on right child right = findLcaAndDistance (root . right , a , b , d1 , d2 , dist , lvl + 1) if left is not None and right is not None : # If both nodes are found in different # subtrees, calculate the distance dist [0] = d1 [0] + d2 [0] - 2 * lvl # Return node found or None if not found if left is not None : return left else : return right # Function to find distance between two nodes def findDist (root , a , b): d1 = [- 1] d2 = [- 1] dist = [0] # Find lowest common ancestor and calculate distance lca = findLcaAndDistance (root , a , b , d1 , d2 , dist , 1) if d1 [0] != - 1 and d2 [0] != - 1 : # Return the distance if both nodes are found return dist [0] if d1 [0] != - 1 : # If only first node is found, find # distance to second node dist [0] = findLevel (lca , b , 0) return dist [0] if d2 [0] != - 1 : # If only second node is found, find # distance to first node dist [0] = findLevel (lca , a , 0) return dist [0] # Return -1 if both nodes not found return - 1 if __name__ == "__main__" : # Hardcoded binary tree # 1 # / \ # 2 3 # / \ / \ # 4 5 6 7 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 (6) root . right . right = Node (7) a = 4 b = 7 print (findDist (root , a , b)) C# // C# Program to Find distance between // two nodes of a Binary Tree using System ; class Node { public int data ; public Node left , right ; public Node (int val) { data = val ; left = null ; right = null ; } } class GfG { // Function to find the level of a node static int FindLevel (Node root , int k , int level) { if (root == null) return - 1 ; if (root . data == k) return level ; // Recursively call function on left child int leftLevel = FindLevel (root . left , k , level + 1); // If node is found on left, return level // Else continue searching on the right child if (leftLevel != - 1) { return leftLevel ; } else { return FindLevel (root . right , k , level + 1); } } // Function to find the lowest common ancestor // and calculate distance between two nodes static Node FindLcaAndDistance (Node root , int a , int b , ref int d1 , ref int d2 , ref int dist , int lvl) { if (root == null) return null ; if (root . data == a) { // If first node found, store level and // return the node d1 = lvl ; return root ; } if (root . data == b) { // If second node found, store level and // return the node d2 = lvl ; return root ; } // Recursively call function on left child Node left = FindLcaAndDistance (root . left , a , b , ref d1 , ref d2 , ref dist , lvl + 1); // Recursively call function on right child Node right = FindLcaAndDistance (root . right , a , b , ref d1 , ref d2 , ref dist , lvl + 1); if (left != null && right != null) { // If both nodes are found in different // subtrees, calculate the distance dist = d1 + d2 - 2 * lvl ; } // Return node found or null if not found if (left != null) { return left ; } else { return right ; } } // Function to find distance between two nodes static int FindDist (Node root , int a , int b) { int d1 = - 1 , d2 = - 1 , dist = 0 ; // Find lowest common ancestor and calculate distance Node lca = FindLcaAndDistance (root , a , b , ref d1 , ref d2 , ref dist , 1); if (d1 != - 1 && d2 != - 1) { // Return the distance if both nodes // are found return dist ; } if (d1 != - 1) { // If only first node is found, find // distance to second node dist = FindLevel (lca , b , 0); return dist ; } if (d2 != - 1) { // If only second node is found, find // distance to first node dist = FindLevel (lca , a , 0); return dist ; } // Return -1 if both nodes not found return - 1 ; } static void Main (string [] args) { // Hardcoded binary tree // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); int a = 4 , b = 7 ; Console . WriteLine (FindDist (root , a , b)); } } JavaScript // JavaScript Program to Find distance between // two nodes of a Binary Tree class Node { constructor (val) { this . data = val ; this . left = null ; this . right = null ; } } // Function to find the level of a node function findLevel (root , k , level) { if (root === null) return - 1 ; if (root . data === k) return level ; // Recursively call function on left child const leftLevel = findLevel (root . left , k , level + 1); // If node is found on left, return level // Else continue searching on the right child if (leftLevel !== - 1) { return leftLevel ; } else { return findLevel (root . right , k , level + 1); } } // Function to find the lowest common ancestor // and calculate distance between two nodes function findLcaAndDistance (root , a , b , d1 , d2 , dist , lvl) { if (root === null) return null ; if (root . data === a) { // If first node found, store level and // return the node d1 [0] = lvl ; return root ; } if (root . data === b) { // If second node found, store level and // return the node d2 [0] = lvl ; return root ; } // Recursively call function on left child

const left = findLcaAndDistance (root . left , a , b , d1 , d2 , dist , lvl + 1); // Recursively call function on right child const right = findLcaAndDistance (root . right , a , b , d1 , d2 , dist , lvl + 1); if (left !== null && right !== null) { // If both nodes are found in different // subtrees, calculate the distance dist [0] = d1 [0] + d2 [0] - 2 * lvl ; } // Return node found or null if not found if (left !== null) { return left ; } else { return right ; } } // Function to find distance between two nodes function findDist (root , a , b) { const d1 = [- 1]; const d2 = [- 1]; const dist = [0]; // Find lowest common ancestor and calculate distance const lca = findLcaAndDistance (root , a , b , d1 , d2 , dist , 1); if (d1 [0] !== - 1 && d2 [0] !== - 1) { // Return the distance if both nodes are found return dist [0]; } if (d1 [0] !== - 1) { // If only first node is found, find // distance to second node dist [0] = findLevel (lca , b , 0); return dist [0]; } if (d2 [0] !== - 1) { // If only second node is found, find // distance to first node dist [0] = findLevel (lca , a , 0); return dist [0]; } // Return -1 if both nodes not found return - 1 ; } // Hardcoded binary tree // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 const root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); const a = 4 ; const b = 7 ; console . log (findDist (root , a , b)); Output 4 Using LCA - O(n) Time and O(h) Space The idea is to first identify their Lowest Common Ancestor (LCA). Once the LCA is determined, the distance between each node and the LCA is calculated. The sum of these distances gives the total distance between the two nodes Step-By-Step Implementation : Start at the root of the tree and recursively traverse the left and right subtrees. If the current node is null, return null. If the current node matches either of the two nodes, return the current node. If both left and right subtree calls return non-null values, the current node is the LCA. From the LCA node, find the level of the first node. If the current node matches the target, return the current depth. Recursively check both the left and right children, increasing the depth by one until the node is found. Perform the same level calculation for the second node. Collect the depth values for both nodes, which represent the distance from the LCA to each of the two nodes. Sum the distances obtained from the previous steps. This total represents the distance between the two nodes in the binary tree, giving the final result. Below is the implementation of the above approach. C++ // C++ Program to Find distance between // two nodes of a Binary Tree #include <bits/stdc++.h> using namespace std ; class Node { public : int data ; Node * left ; Node * right ; Node (int key) { data = key ; left = nullptr ; right = nullptr ; } }; // Function to find the Lowest Common Ancestor // (LCA) of two nodes Node * LCA (Node * root , int n1 , int n2) { if (root == nullptr) return root ; if (root -> data == n1 || root -> data == n2) return root ; Node * left = LCA (root -> left , n1 , n2); Node * right = LCA (root -> right , n1 , n2); if (left != nullptr && right != nullptr) return root ; if (left == nullptr && right == nullptr) return nullptr ; return (left != nullptr) ? LCA (root -> left , n1 , n2) : LCA (root -> right , n1 , n2); } // Returns level of key k if it is present in tree, // otherwise returns -1 int findLevel (Node * root , int k , int level) { if (root == nullptr) return -1 ; if (root -> data == k) return level ; int left = findLevel (root -> left , k , level + 1); if (left == -1) return findLevel (root -> right , k , level + 1); return left ; } // Function to find distance between two // nodes in a binary tree int findDistance (Node * root , int a , int b) { Node * lca = LCA (root , a , b); int d1 = findLevel (lca , a , 0); int d2 = findLevel (lca , b , 0); return d1 + d2 ; } int main () { // Create a sample tree: // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node * root = new Node (1); root -> left = new Node (2); root -> right = new Node (3); root -> left -> left = new Node (4); root -> left -> right = new Node (5); root -> right -> left = new Node (6); root -> right -> right = new Node (7); cout << findDistance (root , 4 , 7) << endl ; return 0 ; } C // C Program to Find distance between // two nodes of a Binary Tree #include <stdio.h> #include <stdlib.h> struct Node { int data ; struct Node * left ; struct Node * right ; }; // Function to find the Lowest Common Ancestor // (LCA) of two nodes struct Node * LCA (struct Node * root , int n1 , int n2) { if (root == NULL) return root ; if (root -> data == n1 || root -> data == n2) return root ; struct Node * left = LCA (root -> left , n1 , n2); struct Node * right = LCA (root -> right , n1 , n2); if (left != NULL && right != NULL) return root ; if (left == NULL && right == NULL) return NULL ; return (left != NULL) ? LCA (root -> left , n1 , n2) : LCA (root -> right , n1 , n2); } // Returns level of key k if it is present in // tree, otherwise returns -1 int findLevel (struct Node * root , int k , int level) { if (root == NULL) return -1 ; if (root -> data == k) return level ; int left = findLevel (root -> left , k , level + 1); if (left == -1) return findLevel (root -> right , k , level + 1); return left ; } // Function to find distance between two nodes // in a binary tree int findDistance (struct Node * root , int a , int b) { struct Node * lca = LCA (root , a , b); int d1 = findLevel (lca , a , 0); int d2 = findLevel (lca , b , 0); return d1 + d2 ; } struct Node * createNode (int key) { struct Node * newNode = (struct Node *) malloc (sizeof (struct Node)); newNode -> data = key ; newNode -> left = NULL ; newNode -> right = NULL ; return newNode ; } int main () { // Create a sample tree: // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 struct Node * root = createNode (1); root -> left = createNode (2); root -> right = createNode (3); root -> left -> left = createNode (4); root

-> left -> right = createNode (5); root -> right -> left = createNode (6); root -> right -> right = createNode (7); printf ("%d \n " , findDistance (root , 4 , 7)); return 0 ; } Java // Java Program to Find distance between two // nodes of a Binary Tree class Node { int data ; Node left ; Node right ; Node (int key) { data = key ; left = null ; right = null ; } } class GfG { // Function to find the Lowest Common // Ancestor (LCA) of two nodes static Node lca (Node root , int n1 , int n2) { if (root == null) { return root ; } if (root . data == n1 || root . data == n2) { return root ; } Node left = lca (root . left , n1 , n2); Node right = lca (root . right , n1 , n2); if (left != null && right != null) { return root ; } if (left == null && right == null) { return null ; } return (left != null) ? left : right ; } // Returns level of key k if it is present in // tree, otherwise returns -1 static int findLevel (Node root , int k , int level) { if (root == null) { return - 1 ; } if (root . data == k) { return level ; } int left = findLevel (root . left , k , level + 1); if (left == - 1) { return findLevel (root . right , k , level + 1); } return left ; } // Function to find distance between two // nodes in a binary tree static int findDistance (Node root , int a , int b) { Node lcaNode = lca (root , a , b); int d1 = findLevel (lcaNode , a , 0); int d2 = findLevel (lcaNode , b , 0); return d1 + d2 ; } public static void main (String [] args) { // Create a sample tree: // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); System . out . println (findDistance (root , 4 , 7)); } } Python # Python Program to Find distance between two # nodes of a Binary Tree class Node : def __init__ (self , key): self . data = key self . left = None self . right = None # Function to find the Lowest Common Ancestor # (LCA) of two nodes def lca (root , n1 , n2): if root is None : return root if root . data == n1 or root . data == n2 : return root left = lca (root . left , n1 , n2) right = lca (root . right , n1 , n2) if left is not None and right is not None : return root if left is None and right is None : return None return left if left is not None else right # Returns level of key k if it is present in tree, # otherwise returns -1 def findLevel (root , k , level): if root is None : return - 1 if root . data == k : return level left = findLevel (root . left , k , level + 1) if left == - 1 : return findLevel (root . right , k , level + 1) return left # Function to find distance between two # nodes in a binary tree def findDistance (root , a , b): lcaNode = lca (root , a , b) d1 = findLevel (lcaNode , a , 0) d2 = findLevel (lcaNode , b , 0) return d1 + d2 # Create a sample tree: # 1 # / \ # 2 3 # / \ / \ # 4 5 6 7 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 (6) root . right . right = Node (7) print (findDistance (root , 4 , 7)) C# // C# Program to Find distance between two // nodes of a Binary Tree using System ; class Node { public int data ; public Node left ; public Node right ; public Node (int key) { data = key ; left = null ; right = null ; } } class GfG { // Function to find the Lowest Common Ancestor // (LCA) of two nodes static Node Lca (Node root , int n1 , int n2) { if (root == null) { return root ; } if (root . data == n1 || root . data == n2) { return root ; } Node left = Lca (root . left , n1 , n2); Node right = Lca (root . right , n1 , n2); if (left != null && right != null) { return root ; } if (left == null && right == null) { return null ; } return (left != null) ? left : right ; } // Returns level of key k if it is present in tree, // otherwise returns -1 static int FindLevel (Node root , int k , int level) { if (root == null) { return - 1 ; } if (root . data == k) { return level ; } int left = FindLevel (root . left , k , level + 1); if (left == - 1) { return FindLevel (root . right , k , level + 1); } return left ; } // Function to find distance between two nodes // in a binary tree static int FindDistance (Node root , int a , int b) { Node lcaNode = Lca (root , a , b); int d1 = FindLevel (lcaNode , a , 0); int d2 = FindLevel (lcaNode , b , 0); return d1 + d2 ; } static void Main (string [] args) { // Create a sample tree: // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); Console . WriteLine (FindDistance (root , 4 , 7)); } } JavaScript // JavaScript Program to Find distance between two // nodes of a Binary Tree class Node { constructor (key) { this . data = key ; this . left = null ; this . right = null ; } } // Function to find the Lowest Common Ancestor // (LCA) of two nodes function lca (root , n1 , n2) { if (root === null) { return root ; } if (root . data === n1 || root . data === n2) { return root ; } let left = lca (root . left , n1 , n2); let right = lca (root . right , n1 , n2); if (left !== null && right !== null) { return root ; } if (left === null && right === null) { return null ; } return left !== null ? left : right ; } // Returns level of key k if it is present in tree, // otherwise returns -1 function findLevel (root , k , level) { if (root === null) { return - 1 ; } if (root . data === k) { return level ; } let left = findLevel (root . left , k , level + 1); if (left === - 1) { return findLevel (root . right , k , level + 1); } return left ; } // Function to find distance between two // nodes in a binary tree function findDistance (root , a , b) { let lcaNode = lca (root , a , b); let d1 = findLevel (lcaNode , a , 0); let d2 = findLevel (lcaNode , b , 0); return d1 + d2 ; } // Create a sample tree: // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 let root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right .

right = new Node (7); console . log (findDistance (root , 4 , 7)); Output 4 Using LCA (one pass) - O(n) Time and O(h) Space We first find the LCA of two nodes. Then we find the distance from LCA to two nodes. We know that distance between two node(let suppose n1 and n2) = distance between LCA and n1 + distance between LCA and n2. Step-By-Step Implementation : Recursively traverses the tree to find the distance between the two target nodes. During traversal, check if the current node is one of the target nodes. While traversing, if one of the target nodes is found, check if there are any descendants of the other target node. This will help in determining whether to continue counting the distance or to reset it. If both target nodes are found in the left and right subtrees of the current node, that node is their LCA . At this point, calculate the distance from the LCA to each target node. After calculating the distances from the LCA to both target nodes, return their sum as the final distance between the two nodes in the binary tree. Below is the implementation of the above approach. C++ // C++ Program to Find distance between two nodes // of a Binary Tree #include <bits/stdc++.h> using namespace std ; class Node { public : Node * left , * right ; int data ; Node (int val) { data = val ; left = nullptr ; right = nullptr ; } }; // Function that calculates distance between two nodes. // It returns a pair where the first element indicates // whether n1 or n2 is found and the second element // is the distance from the current node. pair < bool , int > calculateDistance (Node * root , int n1 , int n2 , int & distance) { if (! root) return { false , 0 }; // Recursively calculate the distance in // the left and right subtrees pair < bool , int > left = calculateDistance (root -> left , n1 , n2 , distance); pair < bool , int > right = calculateDistance (root -> right , n1 , n2 , distance); // Check if the current node is either n1 or n2 bool current = (root -> data == n1 || root -> data == n2); // If current node is one of n1 or n2 and // we found the other in a subtree, update distance if (current && (left . first || right . first)) { distance = max (left . second , right . second); return { false , 0 }; } // If left and right both returned true, // root is the LCA and we update the distance if (left . first && right . first) { distance = left . second + right . second ; return { false , 0 }; } // If either left or right subtree contains n1 or n2, // return the updated distance if (left . first || right . first || current) { return { true , max (left . second , right . second) + 1 }; } // If neither n1 nor n2 exist in the subtree return { false , 0 }; } // The function that returns distance between n1 and n2. int findDistance (Node * root , int n1 , int n2) { int distance = 0 ; calculateDistance (root , n1 , n2 , distance); return distance ; } int main () { // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node * root = new Node (1); root -> left = new Node (2); root -> right = new Node (3); root -> left -> left = new Node (4); root -> left -> right = new Node (5); root -> right -> left = new Node (6); root -> right -> right = new Node (7); cout << findDistance (root , 4 , 7); return 0 ; } C // C Program to Find distance between two // nodes of a Binary Tree #include <stdio.h> #include <stdlib.h> #include <stdbool.h> struct Node { int data ; struct Node * left ; struct Node * right ; }; // Function that calculates distance between two nodes. // It returns a pair where the first element indicates // whether n1 or n2 is found and the second element // is the distance from the current node. struct Pair { bool found ; int distance ; }; struct Pair calculateDistance (struct Node * root , int n1 , int n2 , int * distance) { if (! root) return (struct Pair){ false , 0 }; struct Pair left = calculateDistance (root -> left , n1 , n2 , distance); struct Pair right = calculateDistance (root -> right , n1 , n2 , distance); bool current = (root -> data == n1 || root -> data == n2); if (current && (left . found || right . found)) { * distance = (left . distance > right . distance) ? left . distance : right . distance ; return (struct Pair){ false , 0 }; } if (left . found && right . found) { * distance = left . distance + right . distance ; return (struct Pair){ false , 0 }; } if (left . found || right . found || current) { return (struct Pair){ true , (left . distance > right . distance ? left . distance : right . distance) + 1 }; } return (struct Pair){ false , 0 }; } // The function that returns distance between n1 and n2. int findDistance (struct Node * root , int n1 , int n2) { int distance = 0 ; calculateDistance (root , n1 , n2 , & distance); return distance ; } struct Node * createNode (int val) { struct Node * newNode = (struct Node *) malloc (sizeof (struct Node)); newNode -> data = val ; newNode -> left = NULL ; newNode -> right = NULL ; return newNode ; } int main () { // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 struct Node * root = createNode (1); root -> left = createNode (2); root -> right = createNode (3); root -> left -> left = createNode (4); root -> left -> right = createNode (5); root -> right -> left = createNode (6); root -> right -> right = createNode (7); printf ("%d" , findDistance (root , 4 , 7)); return 0 ; } Java // Java Program to Find distance between two // nodes of a Binary Tree class Node { int data ; Node left , right ; Node (int val) { data = val ; left = null ; right = null ; } } class GfG { // Function that calculates distance between two nodes. // It returns an array where the first element indicates // whether n1 or n2 is found and the second element // is the distance from the current node. static int [] calculateDistance (Node root , int n1 , int n2 , int [] distance) { if (root == null) return new int [] { 0 , 0 }; // Recursively calculate the distance in the // left and right subtrees int [] left = calculateDistance (root . left , n1 , n2 , distance); int [] right = calculateDistance (root . right , n1 , n2 , distance); // Check if the current node is either n1 or n2 boolean current = (root .

data == n1 || root . data == n2); // If current node is one of n1 or n2 and we // found the other in a subtree, update distance if (current && (left [0] == 1 || right [0] == 1)) { distance [0] = Math . max (left [1] , right [1]); return new int [] { 0 , 0 }; } // If left and right both returned true, // root is the LCA and we update the distance if (left [0] == 1 && right [0] == 1) { distance [0] = left [1] + right [1] ; return new int [] { 0 , 0 }; } // If either left or right subtree contains // n1 or n2, return the updated distance if (left [0] == 1 || right [0] == 1 || current) { return new int [] { 1 , Math . max (left [1] , right [1]) + 1 }; } // If neither n1 nor n2 exist in the subtree return new int [] { 0 , 0 }; } // The function that returns distance between n1 and n2. static int findDistance (Node root , int n1 , int n2) { int [] distance = { 0 }; calculateDistance (root , n1 , n2 , distance); return distance [0] ; } public static void main (String [] args) { // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); System . out . println (findDistance (root , 4 , 7)); } } Python # Python Program to Find distance between two # nodes of a Binary Tree class Node : def __init__ (self , val): self . data = val self . left = None self . right = None # Function that calculates distance between two nodes. # It returns a tuple where the first element indicates # whether n1 or n2 is found and the second element # is the distance from the current node. def calculateDistance (root , n1 , n2 , distance): if not root : return (False , 0) left = calculateDistance (root . left , n1 , n2 , distance) right = calculateDistance (root . right , n1 , n2 , distance) current = (root . data == n1 or root . data == n2) if current and (left [0] or right [0]): distance [0] = max (left [1], right [1]) return (False , 0) if left [0] and right [0]: distance [0] = left [1] + right [1] return (False , 0) if left [0] or right [0] or current : return (True , max (left [1], right [1]) + 1) return (False , 0) # The function that returns distance between n1 and n2. def findDistance (root , n1 , n2): distance = [0] calculateDistance (root , n1 , n2 , distance) return distance [0] if __name__ == "__main__" : # 1 # / \ # 2 3 # / \ / \ # 4 5 6 7 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 (6) root . right . right = Node (7) print (findDistance (root , 4 , 7)) C# // C# Program to Find distance between two nodes // of a Binary Tree using System ; class Node { public int data ; public Node left , right ; public Node (int val) { data = val ; left = null ; right = null ; } } class GfG { // Function that calculates distance between two nodes. // It returns an array where the first element indicates // whether n1 or n2 is found and the second element // is the distance from the current node. static int [] calculateDistance (Node root , int n1 , int n2 , ref int distance) { if (root == null) return new int []{ 0 , 0 }; // Recursively calculate the distance in the left // and right subtrees int [] left = calculateDistance (root . left , n1 , n2 , ref distance); int [] right = calculateDistance (root . right , n1 , n2 , ref distance); // Check if the current node is either n1 or n2 bool current = (root . data == n1 || root . data == n2); // If current node is one of n1 or n2 and we // found the other in a subtree, update distance if (current && (left [0] == 1 || right [0] == 1)) { distance = Math . Max (left [1], right [1]); return new int []{ 0 , 0 }; } // If left and right both returned true, // root is the LCA and we update the distance if (left [0] == 1 && right [0] == 1) { distance = left [1] + right [1]; return new int []{ 0 , 0 }; } // If either left or right subtree contains n1 or n2, // return the updated distance if (left [0] == 1 || right [0] == 1 || current) { return new int []{ 1 , Math . Max (left [1], right [1]) + 1 }; } // If neither n1 nor n2 exist in the subtree return new int []{ 0 , 0 }; } // The function that returns distance between n1 and n2. static int findDistance (Node root , int n1 , int n2) { int distance = 0 ; calculateDistance (root , n1 , n2 , ref distance); return distance ; } static void Main () { // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 Node root = new Node (1); root . left = new Node (2); root . right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); Console . WriteLine (findDistance (root , 4 , 7)); } } JavaScript // JavaScript Program to Find distance between // two nodes of a Binary Tree class Node { constructor (val) { this . data = val ; this . left = null ; this . right = null ; } } // Function that calculates distance between two nodes. // It returns an object where the first property indicates // whether n1 or n2 is found and the second property // is the distance from the current node. function calculateDistance (root , n1 , n2 , distance) { if (! root) return { found : false , dist : 0 }; let left = calculateDistance (root . left , n1 , n2 , distance); let right = calculateDistance (root . right , n1 , n2 , distance); let current = (root . data === n1 || root . data === n2); if (current && (left . found || right . found)) { distance . value = Math . max (left . dist , right . dist); return { found : false , dist : 0 }; } if (left . found && right . found) { distance . value = left . dist + right . dist ; return { found : false , dist : 0 }; } if (left . found || right . found || current) { return { found : true , dist : Math . max (left . dist , right . dist) + 1 }; } return { found : false , dist : 0 }; } // The function that returns distance // between n1 and n2. function findDistance (root , n1 , n2) { let distance = { value : 0 }; calculateDistance (root , n1 , n2 , distance); return distance . value ; } let root = new Node (1); root . left = new Node (2); root .

right = new Node (3); root . left . left = new Node (4); root . left . right = new Node (5); root . right . left = new Node (6); root . right . right = new Node (7); // 1 // / \ // 2 3 // / \ / \ // 4 5 6 7 console . log (findDistance (root , 4 , 7)); Output 4 Comment Article Tags: Article Tags: Tree DSA Amazon Samsung Linkedin MakeMyTrip LCA + 3 More