Connect nodes at same level - GeeksforGeeks

Source: https://www.geeksforgeeks.org/connect-nodes-at-same-level/

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 Connect nodes at same level Last Updated : 23 Jul, 2025 Given a binary tree , the task is to connect the nodes that are at the same level. Given an addition nextRight pointer for the same.Initially, all the next right pointers point to garbage values, set these pointers to the point next right for each node. Examples: Input : Output: Explanation: The above tree represents the nextRight pointer connected the nodes that are at the same level. Table of Content [Expected Approach - 1] Using Level Order Traversal - O(n) Time and O(n) Space [Expected Approach - 2] Using Pre-Order Traversal - O(n) Time and O(n) Space [Expected Approach - 1] Using Level Order Traversal - O(n) Time and O(n) Space This idea is to use level order traversal to connect nodes at the same level. A NULL is pushed after each level to track the end of the level. As nodes are processed, each node's nextRight pointer is set to the next node in the queue. If a NULL is encountered and the queue isn't empty, another NULL is added to mark the end of the next level. This ensures that all nodes at the same level are linked. Please refre to Connect Nodes at same Level (Level Order Traversal) for implementation. [Expected Approach - 2] Using Pre-Order Traversal - O(n) Time and O(n) Space This approach works only for Complete Binary Trees. In this method we set nextRight in Pre Order manner to make sure that the nextRight of parent is set before its children. When we are at node p , we set the nextRight of its left and right children. Since the tree is complete tree, nextRight of p's left child (p->left->nextRight) will always be p's right child, and nextRight of p's right child (p->right->nextRight) will always be left child of p's nextRight (if p is not the rightmost node at its level). If p is the rightmost node, then nextRight of p's right child will be NULL . Follow the below steps to Implement the idea: Set root ->nextRight to NULL. Call for a recursive function of root. If root -> left is not NULL then set r oot -> left -> nextRight = root -> right If root -> right is not NULL then If root -> nextRight is not NULL set root -> right -> nextRight = root -> nextRight -> left . Else set root -> right -> nextRight to NULL . recursively call for left of root recursively call for right of root Below is the Implementation of the above approach: C++ // C++ Program to Connect nodes at same level #include <bits/stdc++.h> using namespace std ; class Node { public : int data ; Node * left ; Node * right ; Node * nextRight ; Node (int val) { data = val ; left = nullptr ; right = nullptr ; nextRight = nullptr ; } }; // Set next right of all descendants of root. // Assumption: root is a complete binary tree void connectRecur (Node * root) { if (! root) return ; // Set the nextRight pointer for root's left child if (root -> left) root -> left -> nextRight = root -> right ; // Set the nextRight pointer for root's right child // root->nextRight will be nullptr if root is the // rightmost child at its level if (root -> right) root -> right -> nextRight = (root -> nextRight) ? root -> nextRight -> left : nullptr ; // Set nextRight for other nodes // in pre-order fashion connectRecur (root -> left); connectRecur (root -> right); } // Sets the nextRight of root and calls connectRecur() // for other nodes void connect (Node * root) { // Set the nextRight for root root -> nextRight = nullptr ; // Set the next right for rest of the // nodes (other than root) connectRecur (root); } // Function to store the nextRight pointers in // level-order format and return as a vector of strings vector < string > getNextRightArray (Node * root) { vector < string > result ; if (! root) return result ; queue < Node *> q ; q . push (root); q . push (nullptr); while (! q . empty ()) { Node * node = q . front (); q . pop (); if (node != nullptr) { // Add the current node's data result . push_back (to_string (node -> data)); // If nextRight is nullptr, add '#' if (node -> nextRight == nullptr) { result . push_back ("#"); } // Push the left and right children to // the queue (next level nodes) if (node -> left) q . push (node -> left); if (node -> right) q . push (node -> right); } else if (! q . empty ()) { // Add level delimiter for the next level q . push (nullptr); } } return result ; } int main () { // Constructed binary tree is // 10 // / \ // 8 2 // / // 3 Node * root = new Node (10); root -> left = new Node (8); root -> right = new Node (2); root -> left -> left = new Node (3); connect (root); vector < string > output = getNextRightArray (root); for (const string & s : output) { cout << s << ' ' ; } cout << endl ; return 0 ; } Java // Java Program to Connect Nodes // at same Level class Node { int data ; Node left ; Node right ; Node nextRight ; Node (int val) { data = val ; left = null ; right = null ; nextRight = null ; } } class GfG { // Sets the nextRight of root and calls // connectRecur() for other nodes static void connect (Node root) { // Set the nextRight for root root . nextRight = null ; // Set the next right for rest of the // nodes (other than root) connectRecur (root); } // Set next right of all descendants of root. // Assumption: root is a complete binary tree static void connectRecur (Node root) { if (root == null) return ; // Set the nextRight pointer for root's left child if (root . left != null) root . left . nextRight = root . right ; // Set the nextRight pointer for root's right child // root.nextRight will be null if root is the // rightmost child at its level if (root . right != null) root . right . nextRight = (root . nextRight != null) ? root . nextRight . left : null ; // Set nextRight for other nodes in pre-order fashion connectRecur (root . left); connectRecur (root . right); } // Function to store the nextRight pointers in level-order format // and return as a list of strings static java . util . List < String > getNextRightArray (Node root) { java . util . List < String > result = new java . util . ArrayList <> (); if (root == null) return result ; java . util . Queue < Node > queue = new java . util . LinkedList <> (); queue . offer (root); queue . offer (null); while (! queue . isEmpty ()) { Node node = queue . poll (); if (node != null) { // Add the current node's data result . add (Integer . toString (node . data)); // If nextRight is null, add '#' if (node . nextRight == null) { result . add ("#"); } // Push the left and right children to the // queue (next level nodes) if (node . left != null) queue . offer (node . left); if (node . right != null) queue . offer (node . right); } else if (! queue . isEmpty ()) { // Add level delimiter for the next level queue . offer (null); } } return result ; } public static void main (String [] args) { // Constructed binary tree is // 10 // / \ // 8 2 // / // 3 Node root = new Node (10); root . left = new Node (8); root . right = new Node (2); root . left . left = new Node (3); connect (root); java . util . List < String > output = getNextRightArray (root); for (String s : output) { System . out . print (s + " "); } System . out . println (); } } Python # Python Program to Connect Nodes at same Level class Node : def __init__ (self , val): self . data = val self . left = None self . right = None self . nextRight = None # Forward declaration of connectRecur def connectRecur (root): if not root : return # Set the nextRight pointer for root's left child if root . left : root . left . nextRight = root . right # Set the nextRight pointer for root's right child # root.nextRight will be None if root is the # rightmost child at its level if root . right : root . right . nextRight = root . nextRight . left \ if root . nextRight else None # Set nextRight for other nodes in pre-order fashion connectRecur (root . left) connectRecur (root . right) # Sets the nextRight of root and calls # connectRecur() for other nodes def connect (root): # Set the nextRight for root root . nextRight = None # Set the next right for rest of the # nodes (other than root) connectRecur (root) # Function to store the nextRight pointers # in level-order format and return as a list # of strings def getNextRightArray (root): result = [] if not root : return result queue = [root , None] while queue : node = queue . pop (0) if node is not None : # Add the current node's data result . append (str (node . data)) # If nextRight is None, add '#' if node . nextRight is None : result . append ("#") # Push the left and right children to the # queue (next level nodes) if node . left : queue . append (node . left) if node . right : queue . append (node . right) elif queue : # Add level delimiter for the next level queue . append (None) return result if __name__ == "__main__" : # Constructed binary tree is # 10 # / \ # 8 2 # / # 3 root = Node (10) root . left = Node (8) root . right = Node (2) root . left . left = Node (3) connect (root) output = getNextRightArray (root) for s in output : print (s , end = ' ') print () C# // C# Program to Connect Nodes at same Level using System ; using System.Collections.Generic ; class Node { public int data ; public Node left ; public Node right ; public Node nextRight ; public Node (int val) { data = val ; left = null ; right = null ; nextRight = null ; } } class GfG { // Sets the nextRight of root and calls // connectRecur() for other nodes static void connect (Node root) { // Set the nextRight for root root . nextRight = null ; // Set the next right for rest of the nodes // (other than root) connectRecur (root); } // Set next right of all descendants of root. // Assumption: root is a complete binary tree static void connectRecur (Node root) { if (root == null) return ; // Set the nextRight pointer for root's left child if (root . left != null) root . left . nextRight = root . right ; // Set the nextRight pointer for root's right child // root.nextRight will be null if root is the // rightmost child at its level if (root . right != null) root . right . nextRight = (root . nextRight != null) ? root . nextRight . left : null ; // Set nextRight for other nodes in // pre-order fashion connectRecur (root . left); connectRecur (root . right); } // Function to store the nextRight pointers in // level-order format and return as a list of strings static List < string > getNextRightArray (Node root) { List < string > result = new List < string > (); if (root == null) return result ; Queue < Node > queue = new Queue < Node > (); queue . Enqueue (root); queue . Enqueue (null); while (queue . Count > 0) { Node node = queue .

Dequeue (); if (node != null) { // Add the current node's data result . Add (node . data . ToString ()); // If nextRight is null, add '#' if (node . nextRight == null) { result . Add ("#"); } // Push the left and right children to the // queue (next level nodes) if (node . left != null) queue . Enqueue (node . left); if (node . right != null) queue . Enqueue (node . right); } else if (queue . Count > 0) { // Add level delimiter for the next level queue . Enqueue (null); } } return result ; } static void Main (string [] args) { // Constructed binary tree is // 10 // / \ // 8 2 // / // 3 Node root = new Node (10); root . left = new Node (8); root . right = new Node (2); root . left . left = new Node (3); connect (root); List < string > output = getNextRightArray (root); foreach (string s in output) { Console . Write (s + " "); } Console . WriteLine (); } } JavaScript // JavaScript Program to Connect // Nodes at same Level class Node { constructor (val) { this . data = val ; this . left = null ; this . right = null ; this . nextRight = null ; } } // Forward declaration of connectRecur function connectRecur (root) { if (! root) return ; // Set the nextRight pointer for root's left child if (root . left) root . left . nextRight = root . right ; // Set the nextRight pointer for root's right child // root.nextRight will be null if root is the // rightmost child at its level if (root . right) root . right . nextRight = root . nextRight ? root . nextRight . left : null ; // Set nextRight for other nodes in pre-order fashion connectRecur (root . left); connectRecur (root . right); } // Sets the nextRight of root and calls // connectRecur() for other nodes function connect (root) { // Set the nextRight for root root . nextRight = null ; // Set the next right for rest of the // nodes (other than root) connectRecur (root); } // Function to store the nextRight pointers // in level-order format and return as an array // of strings function getNextRightArray (root) { const result = []; if (! root) return result ; const queue = [root , null]; while (queue . length > 0) { const node = queue . shift (); if (node !== null) { // Add the current node's data result . push (node . data . toString ()); // If nextRight is null, add '#' if (node . nextRight === null) { result . push ("#"); } // Push the left and right children to the // queue (next level nodes) if (node . left) queue . push (node . left); if (node . right) queue . push (node . right); } else if (queue . length > 0) { // Add level delimiter for the next level queue . push (null); } } return result ; } // Constructed binary tree is // 10 // / \ // 8 2 // / // 3 const root = new Node (10); root . left = new Node (8); root . right = new Node (2); root . left . left = new Node (3); connect (root); const output = getNextRightArray (root); console . log (output . join (' ')); Output 10 # 8 2 # 3 # Time Complexity: O(n) where n is the number of Node in Binary Tree. Auxiliary Space: O(n) Why this method doesn't work which are not Complete Binary Trees? Let us consider following tree as an example: In Method 2, we set the nextRight pointer in pre order fashion. When we are at node 4, we set the nextRight of its children which are 8 and 9 (the nextRight of 4 is already set as node 5). nextRight of 8 will simply be set as 9, but nextRight of 9 will be set as NULL which is incorrect . We can't set the correct nextRight , because when we set nextRight of 9, we only have nextRight of node 4 and ancestors of node 4, we don't have nextRight of nodes in right subtree of root. Related article: Connect nodes at same level using constant extra space Comment Article Tags: Article Tags: Tree DSA Microsoft Amazon Adobe Google Oracle Flipkart Accolite Ola Cabs Xome OYO Boomerang Commerce + 9 More