Print nodes at k distance from root - GeeksforGeeks

Source: https://www.geeksforgeeks.org/print-nodes-at-k-distance-from-root/

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 Print nodes at k distance from root Last Updated : 23 Jul, 2025 Given a binary tree , and an integer k . The task is to return all the nodes which are at k distance from the root. Examples: Input: Output: 2 9 13 Explanation: In the above tree 2, 9 & 13 are at distance 2 from root. Input: Output: 5 11 Explanation: In the above tree 5 & 11 are at distance 1 from root. Try it on GfG Practice Table of Content [Expected Approach - 1] Using Recursion - O(n) time and O(h) Space [Expected Approach - 2] Using Queue - O(n) time and O(n) Space [Expected Approach - 3] Using Stack - O(n) time and O(n) Space [Expected Approach - 1] Using Recursion - O(n) time and O(h) Space The idea is to use a recursive approach to find all nodes at a specified distance k from the root of a binary tree. or nodes at greater distances, the function recursively explores both left and right children, decrementing k with each call . And add the c urrent node's data when k becomes 0. Follow the steps below to solve the problem: If the root is NULL or k < 0 , return an empty result. If k == 0 , add the current node's data to the result. For k > 0, recursively explores both left and right children, decrementing k with each call. Below is the implementation of the above approach: C++ // c++ of find all nodes that are at distance // k from the root of a binary tree usign recursion. #include <bits/stdc++.h> using namespace std ; class Node { public : int data ; Node * left ; Node * right ; Node (int x) { data = x ; left = nullptr ; right = nullptr ; } }; // Function to collect nodes at distance // k from the root in a vector void KdistanceUill (Node * root , int k , vector < int > & result) { // If root is null and k is not zero return it if (root == NULL || k < 0) return ; // if k is zero then store the data and return if (k == 0) { result . push_back (root -> data); return ; } // Make recursive call on left and right pointer KdistanceUill (root -> left , k - 1 , result) ; KdistanceUill (root -> right , k - 1 , result) ; } // Function to result all nodes at kth distance from root vector < int > Kdistance (struct Node * root , int k) { vector < int > result ; KdistanceUill (root , k , result); return result ; } int main () { // Constructed binary tree: // 1 // / \ // 2 3 // / \ / // 4 5 8 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 (8); vector < int > result = Kdistance (root , 2); for (int i : result) { cout << i << " " ; } return 0 ; } Java // Java program to find all nodes that are at distance // k from the root of a binary tree using recursion. import java.util.ArrayList ; class Node { int data ; Node left , right ; Node (int x) { data = x ; left = right = null ; } } class GfG { // Function to collect nodes at distance // k from the root in a list static void KdistanceUtil (Node root , int k , ArrayList < Integer > result) { // If root is null and k is not zero return it if (root == null || k < 0) return ; // if k is zero then store the data and return if (k == 0) { result . add (root . data); return ; } // Make recursive call on left and right pointer KdistanceUtil (root . left , k - 1 , result); KdistanceUtil (root . right , k - 1 , result); } // Function to get all nodes at kth distance from root static ArrayList < Integer > Kdistance (Node root , int k) { ArrayList < Integer > result = new ArrayList <> (); KdistanceUtil (root , k , result); return result ; } public static void main (String [] args) { // Constructed binary tree: // 1 // / \ // 2 3 // / \ / // 4 5 8 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 (8); ArrayList < Integer > result = Kdistance (root , 2); for (int i : result) { System . out . print (i + " "); } } } Python # Python program to find all nodes that are at distance # k from the root of a binary tree using recursion. class Node : def __init__ (self , x): self . data = x self . left = None self . right = None # Function to collect nodes at distance # k from the root in a list def KdistanceUtil (root , k , result): # If root is null and k is not zero return it if root is None or k < 0 : return # if k is zero then store the data and return if k == 0 : result . append (root . data) return # Make recursive call on left and right pointer KdistanceUtil (root . left , k - 1 , result) KdistanceUtil (root . right , k - 1 , result) # Function to get all nodes at kth distance from root def Kdistance (root , k): result = [] KdistanceUtil (root , k , result) return result if __name__ == "__main__" : # Constructed binary tree: # 1 # / \ # 2 3 # / \ / # 4 5 8 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 (8) result = Kdistance (root , 2) for i in result : print (i , end = " ") C# // C# program to find all nodes that are at distance // k from the root of a binary tree using recursion. using System ; using System.Collections.Generic ; class Node { public int data ; public Node left , right ; public Node (int x) { data = x ; left = right = null ; } } class GfG { // Function to collect nodes at distance // k from the root in a list static void KdistanceUtil (Node root , int k , List < int > result) { // If root is null and k is not zero return it if (root == null || k < 0) return ; // if k is zero then store the data and return if (k == 0) { result . Add (root . data); return ; } // Make recursive call on left and right pointer KdistanceUtil (root . left , k - 1 , result); KdistanceUtil (root . right , k - 1 , result); } // Function to get all nodes at kth distance from root static public List < int > Kdistance (Node root , int k) { List < int > result = new List < int > (); KdistanceUtil (root , k , result); return result ; } static void Main () { // Constructed binary tree: // 1 // / \ // 2 3 // / \ / // 4 5 8 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 (8); List < int > result = Kdistance (root , 2); foreach (int i in result) { Console . Write (i + " "); } } } JavaScript // JavaScript program to find all nodes that are at distance // k from the root of a binary tree using recursion. class Node { constructor (x) { this . data = x ; this . left = null ; this . right = null ; } } // Function to collect nodes at distance // k from the root in an array function KdistanceUtil (root , k , result) { // If root is null and k is not zero return it if (root == null || k < 0) return ; // if k is zero then store the data and return if (k === 0) { result . push (root . data); return ; } // Make recursive call on left and right pointer KdistanceUtil (root . left , k - 1 , result); KdistanceUtil (root . right , k - 1 , result); } // Function to get all nodes at kth distance from root function Kdistance (root , k) { let result = []; KdistanceUtil (root , k , result); return result ; } // Constructed binary tree: // 1 // / \ // 2 3 // / \ / // 4 5 8 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 (8); let result = Kdistance (root , 2); console . log (... result); Output 4 5 8 Time Complexity: O(n) where n is number of nodes in the given binary tree. Space Complexity : O(h) where h is the height of binary tree. [Expected Approach - 2] Using Queue - O(n) time and O(n) Space This idea is to use level order traversal to find all nodes at distance k from the root. Start by adding the root node to a queue and initialize the level to 0 . While queue is not empty , process all nodes at the current level by dequeuing them, and enqueue their children. Once we have processed all nodes at a given level, we increment the level. If you reach level k , return the values of the nodes at that level. If the queue becomes empty before reaching k , return an empty list, indicating no nodes exist at that distance. Please refer to Print nodes at k distance from root | Iterative for implementation. [Expected Approach - 3] Using Stack - O(n) time and O(n) Space The idea is to use stack data structure to find all nodes at distance k from the root. Start by pushing the root node along with its l evel (0) onto the stack. While the stack has elements, we pop the top node and process it. If the node's level matches k , we add its value to the result list. Otherwise, push its r ight child (with the incremented level) first, followed by the left child . This ensures the left child is processed first. Finally, return the result list with nodes at distance k . Please refer to Print nodes at k distance from root | Iterative for implementation. Comment Article Tags: Article Tags: Tree DSA Microsoft Amazon Ola Cabs + 1 More