Ceil in a BST - GeeksforGeeks

Source: https://www.geeksforgeeks.org/floor-and-ceil-from-a-bst/

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 Ceil in a BST Last Updated : 31 Jan, 2026 Given a Binary Search Tree and a value x , find the ceil value of x .Ceil means the smallest node value greater than or equal to the x. Example: Input: x=11 Output: 12 Explanation: 12 is the smallest value greater than or equal to 11. Input: x=4 Output: 4 Try it on GfG Practice [Approach 1] Using Recursion - O(h) Time and O(h) Space The idea is to find the ceil value recursively: if the current node’s value is greater than or equal to x , consider it as a answer and move left for a closer value; otherwise, move right to search for a larger value. Steps to solve the problem: Set ans = -1 While node exists: If node->data == key , return key . If key < node->data , set ans = node->data and move left (maybe there’s a smaller valid value). Else (key > node->data), move right (need a bigger value). Return ans . C++ #include <iostream> using namespace std ; struct Node { int key ; Node * left ; Node * right ; Node (int value) { key = value ; left = right = nullptr ; } }; // Function to find the ceiling of a given // input in BST. If the input is more than // the max key in BST, return -1. int findCeil (Node * root , int x) { // Base case if (root == nullptr) return -1 ; // We found equal key if (root -> key == x) return root -> key ; // If root's key is smaller, // ceil must be in the right subtree if (root -> key < x) return findCeil (root -> right , x); // Else, either left subtree or // root has the ceil value int ceil = findCeil (root -> left , x); return (ceil >= x) ? ceil : root -> key ; } int main () { Node * root = new Node (8); root -> left = new Node (4); root -> right = new Node (12); root -> left -> left = new Node (2); root -> left -> right = new Node (6); root -> right -> left = new Node (10); root -> right -> right = new Node (14); int x = 11 ; cout << findCeil (root , x) << endl ; return 0 ; } C #include <stdio.h> #include <stdlib.h> // Structure of each Node in the tree struct Node { int key ; struct Node * left ; struct Node * right ; }; // Function to find the ceiling of a given // input in BST. If the input is more than // the max key in BST, return -1. int findCeil (struct Node * root , int x) { // Base case if (root == NULL) return -1 ; // We found equal key if (root -> key == x) return root -> key ; // If root's key is smaller, // ceil must be in the right subtree if (root -> key < x) return findCeil (root -> right , x); // Else, either left subtree or // root has the ceil value int ceil = findCeil (root -> left , x); return (ceil >= x) ? ceil : root -> key ; } // Function to create a new node struct Node * newNode (int key) { struct Node * node = (struct Node *) malloc (sizeof (struct Node)); node -> key = key ; node -> left = node -> right = NULL ; return node ; } int main () { struct Node * root = newNode (8); root -> left = newNode (4); root -> right = newNode (12); root -> left -> left = newNode (2); root -> left -> right = newNode (6); root -> right -> left = newNode (10); root -> right -> right = newNode (14); int x = 11 ; printf ("%d" , findCeil (root , x)); return 0 ; } Java class Node { int key ; Node left , right ; Node (int value) { key = value ; left = right = null ; } } // Function to find the ceiling of a given input in BST. // If the input is more than the max key in BST, return -1. public class GfG { static int findCeil (Node root , int x) { // Base case if (root == null) { return - 1 ; } // We found equal key if (root . key == x) { return root . key ; } // If root's key is smaller, // ceil must be in the right subtree if (root . key < x) { return findCeil (root . right , x); } // Else, either left subtree or root // has the ceil value int ceil = findCeil (root . left , x); return (ceil >= x) ? ceil : root . key ; } public static void main (String [] args) { Node root = new Node (8); root . left = new Node (4); root . right = new Node (12); root . left . left = new Node (2); root . left . right = new Node (6); root . right . left = new Node (10); root . right . right = new Node (14); int x = 11 ; System . out . println (findCeil (root , x)); } } Python class Node : def __init__ (self , value): self . key = value self . left = None self . right = None # Function to find the ceiling of a given input in BST. # If the input is more than the max key in BST, return -1. def findCeil (root , x): # Base case if root is None : return - 1 # We found equal key if root . key == x

: return root . key # If root's key is smaller, # ceil must be in the right subtree if root . key < x : return findCeil (root . right , x) # Else, either left subtree or root has the ceil value ceil = findCeil (root . left , x) return ceil if ceil >= x else root . key if __name__ == "__main__" : root = Node (8) root . left = Node (4) root . right = Node (12) root . left . left = Node (2) root . left . right = Node (6) root . right . left = Node (10) root . right . right = Node (14) x = 11 ; print (findCeil (root , x)); C# using System ; class Node { public int Key ; public Node Left , Right ; public Node (int value) { Key = value ; Left = Right = null ; } } class GfG { // Function to find the ceiling of a given input in BST. // If the input is more than the max key in BST, return -1. static int findCeil (Node root , int x) { // Base case if (root == null) { return -1 ; } // We found equal key if (root . Key == x) { return root . Key ; } // If root's key is smaller, // ceil must be in the right subtree if (root . Key < x) { return findCeil (root . Right , x); } // Else, either left subtree or root // has the ceil value int ceil = findCeil (root . Left , x); return (ceil >= x) ? ceil : root . Key ; } static void Main () { Node root = new Node (8); root . Left = new Node (4); root . Right = new Node (12); root . Left . Left = new Node (2); root . Left . Right = new Node (6); root . Right . Left = new Node (10); root . Right . Right = new Node (14); int x = 11 ; Console . WriteLine (findCeil (root , x)); } } JavaScript class Node { constructor (key) { this . key = key ; this . left = null ; this . right = null ; } } // Function to find the ceiling of a given input in BST. // If the input is more than the max key in BST, return -1. function findCeil (root , x) { // Base case if (root === null) { return - 1 ; } // We found equal key if (root . key === x) { return root . key ; } // If root's key is smaller, // ceil must be in the right subtree if (root . key < x) { return findCeil (root . right , x); } // Else, either left subtree or root has the ceil value const ceil = findCeil (root . left , x); return (ceil >= x) ? ceil : root . key ; } // Driver code const root = new Node (8); root . left = new Node (4); root . right = new Node (12); root . left . left = new Node (2); root . left . right = new Node (6); root . right . left = new Node (10); root . right . right = new Node (14); let x = 11 ; console . log (findCeil (root , x)); Output 12 Time complexity: O(h) where h is height of the given BST Auxiliary Space: O(h) [Approach 2] Using Iterative Way - O(h) Time and O(1) Space The idea is to traverse the BST: if the current node’s value is ≥ x , move left to try for a smaller valid value; otherwise, move right to look for a greater value. Below is the implementation of the above approach: C++ #include <iostream> using namespace std ; struct Node { int data ; Node * left , * right ; Node (int value) { data = value ; left = right = nullptr ; } }; // Helper function to find ceil of a given key in BST int findCeil (Node * root , int x) { int ceil = -1 ; while (root) { // If root itself is ceil if (root -> data == x) { return root -> data ; } // If root is smaller, the ceil // must be in the right subtree if (x > root -> data) { root = root -> right ; } // Else either root can be ceil // or a node in the left child else { ceil = root -> data ; root = root -> left ; } } return ceil ; } // Driver code int main () { Node * root = new Node (8); root -> left = new Node (4); root -> right = new Node (12); root -> left -> left = new Node (2); root -> left -> right = new Node (6); root -> right -> left = new Node (10); root -> right -> right = new Node (14); int x = 11 ; cout << findCeil (root , x); return 0 ; } C #include <stdio.h> #include <stdlib.h> // Structure for a tree node struct Node { int data ; struct Node * left ; struct Node * right ; }; // Helper function to find ceil of a given // key in BST int findCeil (struct Node * root , int x) { int ceil = -1 ; // -1 indicates no ceiling found yet while (root) { // If root itself is ceil if (root -> data == x) { return root -> data ; } // If root is smaller, the ceil // must be in the right subtree if (x > root -> data) { root = root -> right ; } // Else either root can be ceil // or a node in the left child else { ceil = root -> data ; root = root -> left ; } } return ceil ; } // Function to create a new node struct Node * newNode (int x) { struct Node * node = (struct Node *) malloc (sizeof (struct Node)); node -> data = x ; node -> left = node -> right = NULL ; return node ; } // Driver code int main () { struct Node * root = newNode (8); root -> left = newNode (4); root -> right = newNode (12); root -> left -> left = newNode (2); root -> left -> right = newNode (6); root -> right -> left = newNode (10); root -> right -> right = newNode (14); int x = 11 ; printf ("%d" , findCeil (root , x)); return 0 ; } Java class Node { int data ; Node left , right ; Node (int value) { data = value ; left = right = null ; } } // Function to find the ceiling of a given // input in BST. If the input is more than // the max key in BST, return -1. class GfG { static int findCeil (Node root , int x) { int ceil = - 1 ; // -1 indicates no ceiling found yet while (root != null) { // If root itself is ceil if (root . data == x) { return root . data ; } // If root's data is smaller, // ceil must be in the right subtree if (x > root . data) { root = root . right ; } // Else either root can be ceil // or a node in the left child else { ceil = root . data ; root = root . left ; } } return ceil ; } public static void main (String [] args) { Node root = new Node (8); root . left = new Node (4); root . right = new Node (12); root . left . left = new Node (2); root . left . right = new Node (6); root . right . left = new Node (10); root . right . right = new Node (14); int x = 11 ; System . out . println (findCeil (root , x)); } } Python class Node : def __init__ (self , value): self . data = value self . left = None self . right = None # Helper function to find the ceiling # of a given x in BST def findCeil (root , x): ceil = - 1 # -1 indicates no ceiling found yet while root : # If root itself is ceil if root . data == x : return root . data # If

root's data is smaller, # ceil must be in the right subtree if x > root . data : root = root . right else : # Else either root can be ceil # or a node in the left child ceil = root . data root = root . left return ceil # Driver code root = Node (8) root . left = Node (4) root . right = Node (12) root . left . left = Node (2) root . left . right = Node (6) root . right . left = Node (10) root . right . right = Node (14) x = 11 ; print (findCeil (root , x)); C# using System ; class Node { public int Data ; public Node Left , Right ; public Node (int value) { Data = value ; Left = Right = null ; } } class GfG { // Function to find the ceiling of a given input in BST. // If the input is more than the max key in BST, return -1. static int findCeil (Node root , int x) { int ceil = - 1 ; // -1 indicates no ceiling found yet while (root != null) { // If root itself is ceil if (root . Data == x) { return root . Data ; } // If root's data is smaller, // ceil must be in the right subtree if (x > root . Data) { root = root . Right ; } // Else either root can be ceil // or a node in the left child else { ceil = root . Data ; root = root . Left ; } } return ceil ; } static void Main () { Node root = new Node (8); root . Left = new Node (4); root . Right = new Node (12); root . Left . Left = new Node (2); root . Left . Right = new Node (6); root . Right . Left = new Node (10); root . Right . Right = new Node (14); int x = 11 ; Console . WriteLine (findCeil (root , x)); } } JavaScript class Node { constructor (value) { this . data = value ; this . left = null ; this . right = null ; } } // Helper function to find the ceil of a given // x in BST function findCeil (root , x) { let ceil = - 1 ; // -1 indicates no ceiling found yet while (root) { // If root itself is ceil if (root . data === x) { return root . data ; } // If root is smaller, the ceil // must be in the right subtree if (x > root . data) { root = root . right ; } // Else either root can be ceil // or a node in the left child else { ceil = root . data ; root = root . left ; } } return ceil ; } // Driver code const root = new Node (8); root . left = new Node (4); root . right = new Node (12); root . left . left = new Node (2); root . left . right = new Node (6); root . right . left = new Node (10); root . right . right = new Node (14); let x = 11 ; console . log (findCeil (root , x)); Output 12 Comment Article Tags: Article Tags: Binary Search Tree DSA