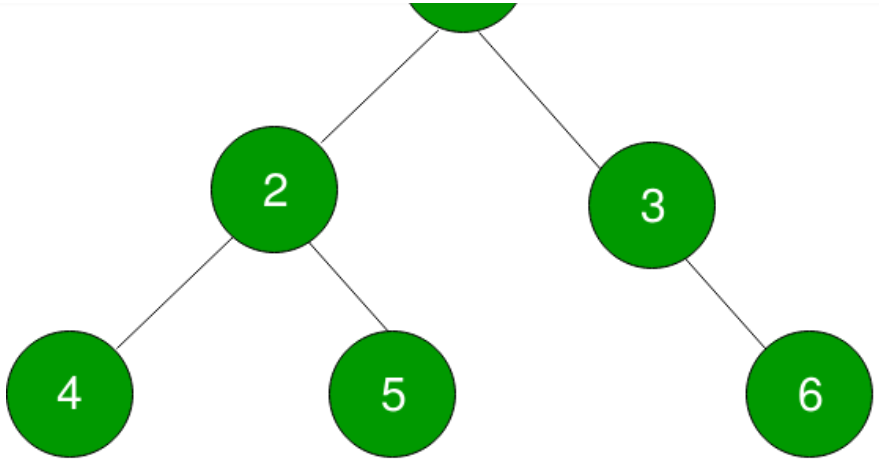
Inorder Successor of a node in Binary Tree

Difficulty Level : Hard • Last Updated : 21 Jun, 2022

Given a binary tree and a node, we need to write a program to find inorder successor of this node.
Inorder Successor of a node in binary tree is the next node in Inorder traversal of the Binary Tree. Inorder Successor is NULL for the last node in Inorder traversal.



In the above diagram, inorder successor of node **4** is **2** and node **5** is **1**.

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

We have already discussed how to find the inorder successor of a node in Binary Search Tree. We can not use the same approach to find the inorder successor in general Binary trees.

We need to take care of 3 cases for any node to find its inorder successor as described below:

1. Right child of node is not NULL. If the right child of the node is not NULL then the inorder successor of this node will be the leftmost node in it's right subtree.
2. Right Child of the node is NULL. If the right child of node is NULL. Then we keep finding the parent of the given node x, say p such that p->left = x. For example in the above given tree, inorder successor of node **5** will be **1**. First parent of 5 is 2 but 2->left != 5. So next parent o 1, now 1->left = 2. Therefore, inorder successor of 5 is

- Suppose the given node is **x**. Start traversing the tree from **root** node to find **x** recursively.
- If **root** == **x**, stop recursion otherwise find x recursively for left and right subtrees.
- Now after finding the node **x**, recursion will backtrack to the **root**. Every recursive call will return the node itself to the calling function, we will store this in a temporary node say **temp**.Now, when it backtracked to its parent which will be root now, check whether root.left = temp, if not , keep going up

3. If node is the rightmost node. If the node is the rightmost node in the given tree. For example, in the above tree node 6 is the right most node. In this case, there will be no inorder successor of this node. i.e. Inorder Successor of the rightmost node in a tree is NULL.

Below is the implementation of above approach:

C++

```
// CPP program to find inorder successor of a node
#include<bits/stdc++.h>
using namespace std;

// A Binary Tree Node
struct Node
{
    int data;
    struct Node *left, *right;
};

// Temporary node for case 2
Node* temp = new Node;
```

```
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// function to find left most node in a tree
Node* leftMostNode(Node* node)
{
    while (node != NULL && node->left != NULL)
        node = node->left;
    return node;
}

// function to find right most node in a tree
Node* rightMostNode(Node* node)
{
    while (node != NULL && node->right != NULL)
        node = node->right;
    return node;
}

// recursive function to find the Inorder Successor
// when the right child of node x is NULL
Node* findInorderRecursive(Node* root, Node* x )
{
    if (!root)
        return NULL;

    if (root==x || (temp = findInorderRecursive(root->left,x)) ||
                   (temp = findInorderRecursive(root->right,x)))
    {
        if (temp)
```

```
                cout << "Inorder Successor of " << x->data;
                cout << " is "<< root->data << "\n";
                return NULL;
            }
        }

        return root;
    }

    return NULL;
}

// function to find inorder successor of
// a node
void inorderSuccessor(Node* root, Node* x)
{
    // Case1: If right child is not NULL
    if (x->right != NULL)
    {
        Node* inorderSucc = leftMostNode(x->right);
        cout<<"Inorder Successor of "<<x->data<<" is ";
        cout<<inorderSucc->data<<"\n";
    }

    // Case2: If right child is NULL
    if (x->right == NULL)
    {
        Node* rightMost = rightMostNode(root);

        // case3: If x is the right most node
        if (rightMost == x)
            cout << "No inorder successor! Right most node.\n";
        else
```

```
// Driver program to test above functions
int main()
{
    // Let's construct the binary tree
    //          1
    //    /    \
    //   2      3
    //  / \    / \
    // 4   5  6   7
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
      root->right->left = newNode(6);
    root->right->right = newNode(7);

    // Case 1 : When there is a right child
    // eg: Node(1) has a right child ergo the inorder successor would be leftmost
      // node of the right subtree ie 6.
    inorderSuccessor(root, root);

    // case 2: When the right child is NULL
      // eg: From the above figure Node(5) satisfies this case
    inorderSuccessor(root, root->left->left);

    // case 3: When the node is the rightmost node of the binary tree
    inorderSuccessor(root, root->right->right);

    return 0;
}
```

```java
// Java program to find inorder successor of a node
class Solution
{
// A Binary Tree Node
 static class Node
{
    int data;
     Node left, right;
}

// Temporary node for case 2
static Node temp = new Node();

// Utility function to create a new tree node
static Node newNode(int data)
 {
    Node temp = new Node();
    temp.data = data;
    temp.left = temp.right = null;
    return temp;
}

// function to find left most node in a tree
static Node leftMostNode(Node node)
 {
    while (node != null && node.left != null)
        node = node.left;
    return node;
}

// function to find right most node in a tree
static Node rightMostNode(Node node)
 {
```

```
}

// recursive function to find the Inorder Successor
// when the right child of node x is null
static Node findInorderRecursive(Node root, Node x )
 {
    if (root==null)
        return null;

    if (root==x || (temp = findInorderRecursive(root.left,x))!=null ||
                (temp = findInorderRecursive(root.right,x))!=null)
    {
        if (temp!=null)
        {
            if (root.left == temp)
            {
                System.out.print( "Inorder Successor of "+x.data);
                System.out.print( " is "+ root.data + "\n");
                return null;
            }
        }

        return root;
    }

    return null;
}

// function to find inorder successor of
// a node
static void inorderSuccessor(Node root, Node x)
 {
    // Case1: If right child is not null
```

```
            System.out.print("Inorder Successor of "+x.data+" is ");
            System.out.print(inorderSucc.data+"\n");
        }

        // Case2: If right child is null
        if (x.right == null)
        {
            int f = 0;

            Node rightMost = rightMostNode(root);

            // case3: If x is the right most node
            if (rightMost == x)
                System.out.print("No inorder successor! Right most node.\n");
            else
                findInorderRecursive(root, x);
        }
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        // Let's construct the binary tree
        // as shown in above diagram

        Node root = newNode(1);
        root.left = newNode(2);
        root.right = newNode(3);
        root.left.left = newNode(4);
        root.left.right = newNode(5);
        root.right.right = newNode(6);

        // Case 1
```

```
        inorderSuccessor(root, root.left.left);

        // case 3
        inorderSuccessor(root, root.right.right);

    }
    }
    //contributed by Arnab Kundu
```

Python3

```
""" Python3 code for inorder successor
and predecessor of tree """

# A Binary Tree Node
# Utility function to create a new tree node
class newNode:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# function to find left most node in a tree
def leftMostNode(node):

    while (node != None and node.left != None):
        node = node.left
    return node
```

```
            node = node.right
        return node

    # recursive function to find the Inorder Successor
    # when the right child of node x is None
    def findInorderRecursive(root, x ):

        if (not root):
            return None
        if (root == x or (findInorderRecursive(root.left, x)) or
                          (findInorderRecursive(root.right, x))):
            if findInorderRecursive(root.right, x):
                temp=findInorderRecursive(root.right, x)
            else:
                temp=findInorderRecursive(root.left, x)
            if (temp):

                if (root.left == temp):

                    print("Inorder Successor of",
                                  x.data, end = "")
                    print(" is", root.data)
                    return None
            return root
        return None

    # function to find inorder successor
    # of a node
    def inorderSuccessor(root, x):

        # Case1: If right child is not None
        if (x.right != None) :
            inorderSucc = leftMostNode(x.right)
```

```
        # Case2: If right child is None
        if (x.right == None):
            f = 0
            rightMost = rightMostNode(root)

            # case3: If x is the right most node
            if (rightMost == x):
                print("No inorder successor!",
                            "Right most node.")
            else:
                findInorderRecursive(root, x)

    # Driver Code
    if __name__ == '__main__':

        root = newNode(1)
        root.left = newNode(2)
        root.right = newNode(3)
        root.left.left = newNode(4)
        root.left.right = newNode(5)
        root.right.right = newNode(6)

        # Case 1
        inorderSuccessor(root, root.right)

        # case 2
        inorderSuccessor(root, root.left.left)

        # case 3
        inorderSuccessor(root, root.right.right)

    # This code is contributed
```

C#

```
// C# program to find inorder
// successor of a node
using System;

class GFG
{

// A Binary Tree Node
public class Node
{
    public int data;
    public Node left, right;
}

// Temporary node for case 2
public static Node temp = new Node();

// Utility function to create
// a new tree node
public static Node newNode(int data)
{
    Node temp = new Node();
    temp.data = data;
    temp.left = temp.right = null;
    return temp;
}

// function to find left most
// node in a tree
public static Node leftMostNode(Node node)
```

```
        {
            node = node.left;
        }
        return node;
    }

    // function to find right most
    // node in a tree
    public static Node rightMostNode(Node node)
    {
        while (node != null &&
            node.right != null)
        {
            node = node.right;
        }
        return node;
    }

    // recursive function to find the
    // Inorder Successor when the right
    // child of node x is null
    public static Node findInorderRecursive(Node root,
                                            Node x)
    {
        if (root == null)
        {
            return null;
        }

        if (root == x ||
        (temp = findInorderRecursive(root.left, x)) != null ||
        (temp = findInorderRecursive(root.right, x)) != null)
        {
```

```
                {
                    Console.Write("Inorder Successor of " + x.data);
                    Console.Write(" is " + root.data + "\n");
                    return null;
                }
            }

            return root;
        }

        return null;
    }

    // function to find inorder successor
    // of a node
    public static void inorderSuccessor(Node root, Node x)
    {
        // Case1: If right child is not null
        if (x.right != null)
        {
            Node inorderSucc = leftMostNode(x.right);
            Console.Write("Inorder Successor of " +
                                x.data + " is ");
            Console.Write(inorderSucc.data + "\n");
        }

        // Case2: If right child is null
        if (x.right == null)
        {
            int f = 0;

            Node rightMost = rightMostNode(root);
```

```
                Console.Write("No inorder successor! " +
                              "Right most node.\n");
            }
            else
            {
                findInorderRecursive(root, x);
            }
        }
    }

    // Driver Code
    public static void Main(string[] args)
    {
        // Let's construct the binary tree
        // as shown in above diagram
        Node root = newNode(1);
        root.left = newNode(2);
        root.right = newNode(3);
        root.left.left = newNode(4);
        root.left.right = newNode(5);
        root.right.right = newNode(6);

        // Case 1
        inorderSuccessor(root, root.right);

        // case 2
        inorderSuccessor(root, root.left.left);

        // case 3
        inorderSuccessor(root, root.right.right);
    }
    }
```

Javascript

```
<script>
// Javascript program to find inorder successor of a node

// A Binary Tree Node
class Node
{
    constructor(data)
    {
        this.data=data;
        this.left=this.right=null;
    }
}

// Temporary node for case 2
let temp = new Node();

// function to find left most node in a tree
function leftMostNode(node)
{
     while (node != null && node.left != null)
        node = node.left;
    return node;
}

// function to find right most node in a tree
function rightMostNode(node)
{
    while (node != null && node.right != null)
        node = node.right;
    return node;
```

```
// when the right child of node x is null
function findInorderRecursive(root,x)
{
    if (root==null)
        return null;

    if (root==x || (temp = findInorderRecursive(root.left,x))!=null ||
                (temp = findInorderRecursive(root.right,x))!=null)
    {
        if (temp!=null)
        {
            if (root.left == temp)
            {
                document.write( "Inorder Successor of "+x.data);
                document.write( " is "+ root.data + "<br>");
                return null;
            }
        }

        return root;
    }

    return null;
}

// function to find inorder successor of
// a node
function inorderSuccessor(root,x)
{
    // Case1: If right child is not null
    if (x.right != null)
    {
        let inorderSucc = leftMostNode(x.right
```

```
    // Case2: If right child is null
    if (x.right == null)
    {
        let f = 0;

        let rightMost = rightMostNode(root);

        // case3: If x is the right most node
        if (rightMost == x)
            document.write("No inorder successor! Right most node.\n");
        else
            findInorderRecursive(root, x);
    }
}


// Driver program to test above functions
// Let's construct the binary tree
// as shown in above diagram

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.right = new Node(6);

// Case 1
inorderSuccessor(root, root.right);

// case 2
inorderSuccessor(root, root.left.left);
```

```
// This code is contributed by avanitrachhadiya2155
</script>
```

Output:

```
Inorder Successor of 1 is 6
Inorder Successor of 4 is 2
No inorder successor! Right most node.
```

Another approach:

We will do a reverse inorder traversal and keep the track of current visited node. Once we found the element, last tracked element would be our answer.

Below is the implementation of above approach:

C++

```
// C++ Program to find inorder successor.
#include<bits/stdc++.h>
using namespace std;

// structure of a Binary Node.
struct Node
{
    int data;
```

```
// Function to create a new Node.
Node* newNode(int val)
{
    Node* temp = new Node;
    temp->data = val;
    temp->left = NULL;
    temp->right = NULL;

    return temp;
}

// function that prints the inorder successor
// of a target node. next will point the last
// tracked node, which will be the answer.
void inorderSuccessor(Node* root,
                      Node* target_node,
                      Node* &next)
{
    // if root is null then return
    if(!root)
        return;

    inorderSuccessor(root->right, target_node, next);

    // if target node found then enter this condition
    if(root->data == target_node->data)
    {
        // this will be true to the last node
        // in inorder traversal i.e., rightmost node.
        if(next == NULL)
            cout << "inorder successor of "
                 << root->data << " is: null\n
```

```
                    << next->data << "\n";
    }
    next = root;
    inorderSuccessor(root->left, target_node, next);
}

// Driver Code
int main()
{

    // Let's construct the binary tree
    //          1
    //     /    \
    //   2       3
    //  / \     / \
    // 4   5  6   7
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
      root->right->left = newNode(6);
    root->right->right = newNode(7);

    // Case 1
    Node* next = NULL;
    inorderSuccessor(root, root, next);

    // case 2
    next = NULL;
    inorderSuccessor(root, root->left->left, next);

    // case 3
```

```
        return 0;
    }

    // This code is contributed by AASTHA VARMA
```

Java

```
// Java program to find inorder successor of a node.

class Node {
    int data;
    Node left, right;

    Node(int data) {
        this.data = data;
        left = null; right = null;
    }
}

// class to find inorder successor of
// a node
class InorderSuccessor {
    Node root;

    // to change previous node
    static class PreviousNode {
        Node pNode;
        PreviousNode() {
            pNode = null;
        }
    }
```

```
private void inOrderSuccessorOfBinaryTree(Node root,
                PreviousNode pre, int searchNode)
{
    // Case1: If right child is not NULL
    if(root.right != null)
    inOrderSuccessorOfBinaryTree(root.right, pre, searchNode);

    // Case2: If root data is equal to search node
    if(root.data == searchNode)
    System.out.println("inorder successor of " + searchNode + " is: "
                        + (pre.pNode != null ? pre.pNode.data : "null"));
        pre.pNode = root;

    if(root.left != null)
    inOrderSuccessorOfBinaryTree(root.left, pre, searchNode);
}

// Driver program to test above functions
public static void main(String[] args)
{
    InorderSuccessor tree = new InorderSuccessor();

    // Let's construct the binary tree
    // as shown in above diagram
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);

    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);
    tree.root.right.right = new Node(6);

    // Case 1
```

```
        // Case 2
        tree.inOrderSuccessorOfBinaryTree(tree.root,
                                          new PreviousNode(), 4);

        // Case 3
        tree.inOrderSuccessorOfBinaryTree(tree.root,
                                          new PreviousNode(), 6);
    }
}
// This code is contributed by Ashish Goyal.
```

Python3

```
# Python3 program to find inorder successor.

# A Binary Tree Node
# Utility function to create a new tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Function to create a new Node.
def newNode(val):

    temp = Node(0)
    temp.data = val
    temp.left = None
```

```
# function that prints the inorder successor
# of a target node. next will point the last
# tracked node, which will be the answer.
def inorderSuccessor(root, target_node):

    global next

    # if root is None then return
    if(root == None):
        return

    inorderSuccessor(root.right, target_node)

    # if target node found, then
    # enter this condition
    if(root.data == target_node.data):

        # this will be true to the last node
        # in inorder traversal i.e., rightmost node.
        if(next == None):
            print ("inorder successor of",
                   root.data , " is: None")
        else:
            print ( "inorder successor of",
                    root.data , "is:", next.data)

    next = root
    inorderSuccessor(root.left, target_node)

# global variable
next = None
```

```
    # Let's construct the binary tree
    # as shown in above diagram.
    root = newNode(1)
    root.left = newNode(2)
    root.right = newNode(3)
    root.left.left = newNode(4)
    root.left.right = newNode(5)
    root.right.right = newNode(6)

    # Case 1
    next = None
    inorderSuccessor(root, root.right)

    # case 2
    next = None
    inorderSuccessor(root, root.left.left)

    # case 3
    next = None
    inorderSuccessor(root, root.right.right)

# This code is contributed by Arnab Kundu
```

C#

```
// C# program to find inorder successor of a node.
using System;

class Node
{
    public int data;
```

```
        {
            this.data = data;
            left = null; right = null;
        }
    }

    // class to find inorder successor of
    // a node
    public class InorderSuccessor
    {
        Node root;

        // to change previous node
        class PreviousNode
        {
            public Node pNode;
            public PreviousNode()
            {
                pNode = null;
            }
        }

        // function to find inorder successor of
        // a node
        private void inOrderSuccessorOfBinaryTree(Node root,
                        PreviousNode pre, int searchNode)
        {
            // Case1: If right child is not NULL
            if(root.right != null)
            inOrderSuccessorOfBinaryTree(root.right,
                                    pre, searchNode);

            // Case2: If root data is equal to sea ▲ ode
```

```
                        searchNode + " is: ");
            if(pre.pNode != null)
                Console.WriteLine(pre.pNode.data);
            else
                Console.WriteLine("null");
        }
            pre.pNode = root;

        if(root.left != null)
            inOrderSuccessorOfBinaryTree(root.left,
                            pre, searchNode);
    }

    // Driver code
    public static void Main(String[] args)
    {
        InorderSuccessor tree = new InorderSuccessor();

        // Let's construct the binary tree
        // as shown in above diagram
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.right = new Node(6);

        // Case 1
        tree.inOrderSuccessorOfBinaryTree(tree.root,
                                    new PreviousNode(), 3);

        // Case 2
```

```
        // Case 3
        tree.inOrderSuccessorOfBinaryTree(tree.root,
                                          new PreviousNode(), 6);
    }
}

// This code is contributed by PrinciRaj1992
```

Javascript

```
<script>

// JavaScript program to find inorder successor of a node.

class Node
{
    constructor(data)
    {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

// class to find inorder successor of
// a node
var root = null;

// to change previous node
class PreviousNode
{
```

```
    }
}

// function to find inorder successor of
// a node
function inOrderSuccessorOfBinaryTree(root, pre, searchNode)
{
    // Case1: If right child is not NULL
    if(root.right != null)
    inOrderSuccessorOfBinaryTree(root.right,
                            pre, searchNode);

    // Case2: If root data is equal to search node
    if(root.data == searchNode)
    {
        document.write("inorder successor of " +
                        searchNode + " is: ");
        if(pre.pNode != null)
            document.write(pre.pNode.data+"<br>");
        else
            document.write("null<br>");
    }
        pre.pNode = root;

    if(root.left != null)
        inOrderSuccessorOfBinaryTree(root.left,
                            pre, searchNode);
}

// Driver code

// Let's construct the binary tree
// as shown in above diagram
```

```
root.left.left = new Node(4);
root.left.right = new Node(5);
root.right.right = new Node(6);

// Case 1
inOrderSuccessorOfBinaryTree(root,
                    new PreviousNode(), 3);

// Case 2
inOrderSuccessorOfBinaryTree(root,
                    new PreviousNode(), 4);

// Case 3
inOrderSuccessorOfBinaryTree(root,
                    new PreviousNode(), 6);


</script>
```

Output:

```
inorder successor of 1 is: 6
inorder successor of 4 is: 2
inorder successor of 7 is: null
```

Time Complexity: O(n), where n is the number of nodes in the tree.

Space complexity: O(n) for call stack

article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Previous
BFS vs DFS for Binary Tree

Next
Diagonal Traversal of Binary Tree

01 Replace each node in binary tree with the sum of its inorder predecessor and successor
07, Jul 17

02 Populate Inorder Successor for all nodes
26, Jan 12

03 Inorder predecessor and successor for a given key in BST
25, Jul 14

04 Modify Binary Tree by replacing each node with the sum of its Preorder Predecessor and Successor
25, Mar 21

05 Postorder successor of a Node in Binary Tree
03, Jun 18

06 Preorder Successor of a Node in Binary Tree
03, Jun 18

07 Level Order Successor of a node in Binary Tree
30, Nov 18

08 Check if given inorder and preorder traversals are valid for any Binary Tree without building the tree
03, Sep 21

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Improved By : ashishik, andrew1234, shrikanth13, pritiraj1992, SHUBHAMSINGH10, AASTHA VARMA, ritik, pranav7kaushik, sagartomar9927, avanitrachhadiya2155, simmytarika5, noviced3vq6

Article Tags : Binary Tree, Tree

Practice Tags : Tree

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org