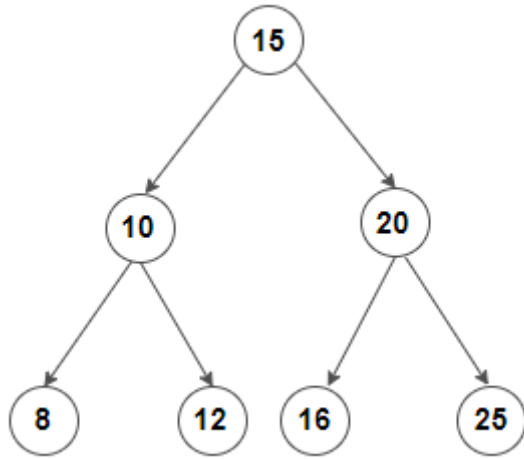




Find inorder successor for the given key in a BST

Given a BST, find the inorder successor of a given key in it. If the given key does not lie in the BST, then return the next greater node (if any) present in the BST.

An inorder successor of a node in the BST is the next node in the [inorder sequence](#). For example, consider the following BST:



- The inorder successor of 8 is 10
- The inorder successor of 12 is 15
- The inorder successor of 25 does not exist.

Practice this problem

A node's inorder successor is the node with the least value in its right subtree, i.e., its right subtree's leftmost child. If the right subtree of the node doesn't exist, then the inorder successor is one of its ancestors. To find which ancestors are the successor, we can move up the tree towards the root until we encounter a node that is left child of its parent. If any such node is found, then inorder successor is its parent; otherwise, inorder successor does not exist for the node.

Recursive Version

We can recursively check the above conditions. The idea is to search for the given node in the tree and update the successor to the current node before visiting its left subtree. If the node is found in the BST, return the least value node in its right subtree, and if the right subtree of the node doesn't exist, then inorder successor is one of the ancestors of it, which has already been updated while searching for the given key.

Following is the C++, Java, and Python implementation of the idea:

C++

Java

Python

```
1 // A class to store a BST node
2 class Node
3 {
4     int data;
5     Node left = null, right = null;
6
7     Node(int data) {
8         this.data = data;
9     }
10 }
11
12 class Main
13 {
14     // Recursive function to insert a key into a BST
15     public static Node insert(Node root, int key)
16     {
17         // if the root is null, create a new node and return it
18         if (root == null) {
19             return new Node(key);
20         }
21
22         // if the given key is less than the root node, recur for the left subtree
23         if (key < root.data) {
```

```
24     root.left = insert(root.left, key);
25 }
26
27 // if the given key is more than the root node, recur for the right subtree
28 else {
29     root.right = insert(root.right, key);
30 }
31
32 return root;
33 }
34
35 // Helper function to find minimum value node in a given BST
36 public static Node findMinimum(Node root)
37 {
38     while (root.left != null) {
39         root = root.left;
40     }
41
42     return root;
43 }
44
45 // Recursive function to find an inorder successor for the given key in the BST
46 public static Node findSuccessor(Node root, Node succ, int key)
47 {
48     // base case
49     if (root == null) {
50         return succ;
51     }
52
53     // if a node with the desired value is found, the successor is the minimum
54     // value node in its right subtree (if any)
55     if (root.data == key)
56     {
57         if (root.right != null) {
58             return findMinimum(root.right);
59         }
60     }
61
62     // if the given key is less than the root node, recur for the left subtree
63     else if (key < root.data)
64     {
65         // update successor to the current node before recursing in the
```

```

66         // left subtree
67         succ = root;
68         return findSuccessor(root.left, succ, key);
69     }
70
71     // if the given key is more than the root node, recur for the right subtree
72     else {
73         return findSuccessor(root.right, succ, key);
74     }
75
76     return succ;
77 }
78
79 public static void main(String[] args)
80 {
81     int[] keys = { 15, 10, 20, 8, 12, 16, 25 };
82
83     /* Construct the following tree
84           15
85          /  \
86         /    \
87        10     20
88       /  \   /  \
89      8   12 16   25
90     */
91
92     Node root = null;
93     for (int key: keys) {
94         root = insert(root, key);
95     }
96
97     // find inorder successor for each key
98     for (int key: keys)
99     {
100         Node succ = findSuccessor(root, null, key);
101
102         if (succ != null)
103         {
104             System.out.println("The successor of node " + key + " is "
105                               + succ.data);
106         }
107     }

```

```
108         else {  
109             System.out.println("No Successor exists for node " + key);  
110         }  
111     }  
112 }  
113 }
```

[Download](#) [Run Code](#)

Output:

```
The successor of node 15 is 16  
The successor of node 10 is 12  
The successor of node 20 is 25  
The successor of node 8 is 10  
The successor of node 12 is 15  
The successor of node 16 is 20  
No Successor exists for node 25
```

The time complexity of the above solution is $O(n)$, where n is the size of the BST, and requires space proportional to the tree's height for the call stack.

Iterative Version

The same algorithm can be easily implemented iteratively. Following is the C++, Java, and Python implementation of the idea:

C++

Java

Python

```
1 // A class to store a BST node
2 class Node
3 {
4     int data;
5     Node left = null, right = null;
6
7     Node(int data) {
8         this.data = data;
9     }
10 }
11
12 class Main
13 {
14     // Recursive function to insert a key into a BST
15     public static Node insert(Node root, int key)
16     {
17         // if the root is null, create a new node and return it
18         if (root == null) {
19             return new Node(key);
20         }
21
22         // if the given key is less than the root node, recur for the left subtree
23         if (key < root.data) {
24             root.left = insert(root.left, key);
25         }
26
27         // if the given key is more than the root node, recur for the right subtree
28         else {
29             root.right = insert(root.right, key);
30         }
31
32         return root;
33     }
34
35     // Helper function to find minimum value node in a given BST
36     public static Node findMinimum(Node root)
```

```
38     {
39         while (root.left != null) {
40             root = root.left;
41         }
42
43         return root;
44     }
45
46     // Iterative function to find an inorder successor for the given key in the BST
47     public static Node findSuccessor(Node root, int key)
48     {
49         // base case
50         if (root == null) {
51             return null;
52         }
53
54         Node succ = null;
55
56         while (true)
57         {
58             // if the given key is less than the root node, visit the left subtree
59             if (key < root.data)
60             {
61                 // update successor to the current node before visiting
62                 // left subtree
63                 succ = root;
64                 root = root.left;
65             }
66
67             // if the given key is more than the root node, visit the right subtree
68             else if (key > root.data) {
69                 root = root.right;
70             }
71
72             // if a node with the desired value is found, the successor is the minimum
73             // value node in its right subtree (if any)
74             else {
75                 if (root.right != null) {
76                     succ = findMinimum(root.right);
77                 }
78                 break;
79             }
80         }
81     }
82 }
```



```

80
81     // if the key doesn't exist in the binary tree, return next greater node
82     if (root == null) {
83         return succ;
84     }
85 }
86
87 // return successor, if any
88 return succ;
89 }
90
91 public static void main(String[] args)
92 {
93     int[] keys = { 15, 10, 20, 8, 12, 16, 25 };
94
95     /* Construct the following tree
96           15
97          /  \
98         /    \
99        10     20
100       /  \   /  \
101      8   12 16   25
102
103     */
104
105     Node root = null;
106     for (int key: keys) {
107         root = insert(root, key);
108     }
109
110     // find inorder successor for each key
111     for (int key: keys)
112     {
113         Node succ = findSuccessor(root, key);
114
115         if (succ != null)
116         {
117             System.out.println("The successor of node " + key + " is "
118                               + succ.data);
119         }
120         else {
121             System.out.println("No Successor exists for node " + key);

```

```
122 | }  
123 | }  
124 | }  
125 | }
```

[Download](#) [Run Code](#)

Output:

```
The successor of node 15 is 16  
The successor of node 10 is 12  
The successor of node 20 is 25  
The successor of node 8 is 10  
The successor of node 12 is 15  
The successor of node 16 is 20  
No Successor exists for node 25
```

The time complexity of the above solution is $O(n)$, where n is the size of the BST. The auxiliary space required by the program is $O(1)$.

📁 [Binary Tree, BST](#)

🔧 [Medium, Recursive](#)