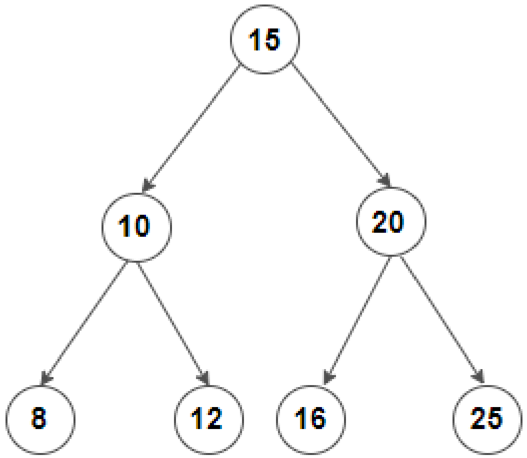
Find inorder successor for the given key in a BST

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

```java
// A class to store a BST node
class Node
{
    int data;
    Node left = null, right = null;

    Node(int data) {
        this.data = data;
    }
}

class Main
{
    // Recursive function to insert a key into a BST
    public static Node insert(Node root, int key)
    {
        // if the root is null, create a new node and return it
        if (root == null) {
            return new Node(key);
        }

        // if the given key is less than the root node, recur for the left subtree
        if (key < root.data) {
```

```
            root.left = insert(root.left, key);
        }

        // if the given key is more than the root node, recur for the right subtree
        else {
            root.right = insert(root.right, key);
        }

        return root;
    }

    // Helper function to find minimum value node in a given BST
    public static Node findMinimum(Node root)
    {
        while (root.left != null) {
            root = root.left;
        }

        return root;
    }

    // Recursive function to find an inorder successor for the given key in the BST
    public static Node findSuccessor(Node root, Node succ, int key)
    {
        // base case
        if (root == null) {
            return succ;
        }

        // if a node with the desired value is found, the successor is the minimum
        // value node in its right subtree (if any)
        if (root.data == key)
        {
            if (root.right != null) {
                return findMinimum(root.right);
            }
        }

        // if the given key is less than the root node, recur for the left subtree
        else if (key < root.data)
        {
            // update successor to the current node before recursing in the
```

```
            // left subtree
            succ = root;
            return findSuccessor(root.left, succ, key);
        }

        // if the given key is more than the root node, recur for the right subtree
        else {
            return findSuccessor(root.right, succ, key);
        }

        return succ;
    }

    public static void main(String[] args)
    {
        int[] keys = { 15, 10, 20, 8, 12, 16, 25 };

        /* Construct the following tree
                   15
                 /    \
                /      \
               10       20
              / \      /  \
             /   \    /    \
            8    12  16    25
        */

        Node root = null;
        for (int key: keys) {
            root = insert(root, key);
        }

        // find inorder successor for each key
        for (int key: keys)
        {
            Node succ = findSuccessor(root, null, key);

            if (succ != null)
            {
                System.out.println("The successor of node " + key + " is "
                                        + succ.data);
            }
```

```
            else {
                System.out.println("No Successor exists for node " + key);
            }
        }
    }
}
```

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

```
// A class to store a BST node
class Node
{
    int data;
    Node left = null, right = null;

    Node(int data) {
        this.data = data;
    }
}

class Main
{
    // Recursive function to insert a key into a BST
    public static Node insert(Node root, int key)
    {
        // if the root is null, create a new node and return it
        if (root == null) {
            return new Node(key);
        }

        // if the given key is less than the root node, recur for the left subtree
        if (key < root.data) {
            root.left = insert(root.left, key);
        }

        // if the given key is more than the root node, recur for the right subtree
        else {
            root.right = insert(root.right, key);
        }

        return root;
    }


    // Helper function to find minimum value node in a given BST
    public static Node findMinimum(Node root)
```

```
    {
        while (root.left != null) {
            root = root.left;
        }

        return root;
    }

    // Iterative function to find an inorder successor for the given key in the BST
    public static Node findSuccessor(Node root, int key)
    {
        // base case
        if (root == null) {
            return null;
        }

        Node succ = null;

        while (true)
        {
            // if the given key is less than the root node, visit the left subtree
            if (key < root.data)
            {
                // update successor to the current node before visiting
                // left subtree
                succ = root;
                root = root.left;
            }

            // if the given key is more than the root node, visit the right subtree
            else if (key > root.data) {
                root = root.right;
            }

            // if a node with the desired value is found, the successor is the minimum
            // value node in its right subtree (if any)
            else {
                if (root.right != null) {
                    succ = findMinimum(root.right);
                }
                break;
            }
```

```
            // if the key doesn't exist in the binary tree, return next greater node
            if (root == null) {
                return succ;
            }
        }

        // return successor, if any
        return succ;
    }

    public static void main(String[] args)
    {
        int[] keys = { 15, 10, 20, 8, 12, 16, 25 };

        /* Construct the following tree
                   15
                 /    \
                /      \
               10       20
              / \      /  \
             /   \    /    \
            8    12  16    25
        */

        Node root = null;
        for (int key: keys) {
            root = insert(root, key);
        }

        // find inorder successor for each key
        for (int key: keys)
        {
            Node succ = findSuccessor(root, key);

            if (succ != null)
            {
                System.out.println("The successor of node " + key + " is "
                                    + succ.data);
            }
            else {
                System.out.println("No Successor exists for node " + key);
```

```
            }
        }
    }
}
```

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