Binary Search Tree

Assignment Solutions

PW SKILLS

Q1. Write an iterative program to search for an element in BST. Also construct a sample BST and try to search for elements in the same.
The input for BST is : 15, 10, 20, 8, 12, 16, 25

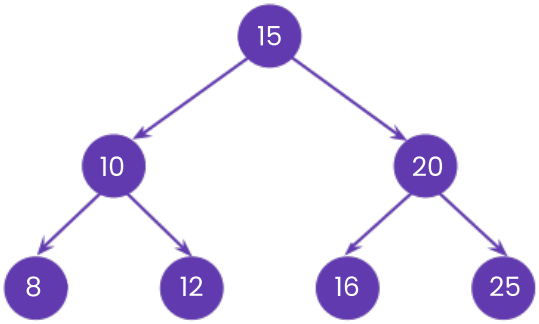
Search for 25 in it.

Expected output: The given key is the right node of the node with key 20

ASS_Code1.java

Q2. Given a BST and a positive number k, find the k'th largest node in the BST.

For example, consider the following binary search tree. If k = 2, the k'th largest node is 20.

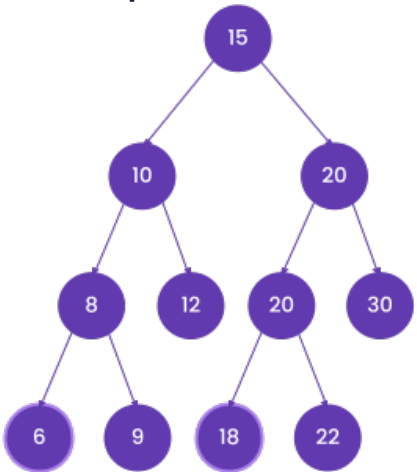


We know that an inorder traversal of a binary search tree returns the nodes in ascending order. To find the k'th smallest node, we can perform inorder traversal and store the inorder sequence in an array. Then the k'th largest node would be the (n-k)'th smallest node, where n is the total number of nodes present in the BST.
The problem with this approach is that it requires two traversals of the array. We can solve this problem in a single traversal of the array by using reverse inorder traversal (traverse the right subtree before the left subtree for every node). Then the reverse inorder traversal of a binary search tree will process the nodes in descending order.

ASS_Code2.java

Q3. Given a binary search tree, find a pair with a given sum present in it.

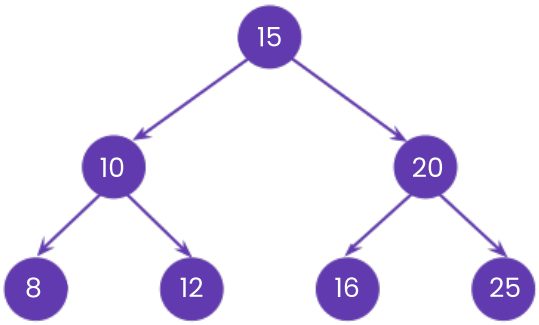
For example, consider the following BST. If the given sum is 14, the pair is (8, 6).



We can easily solve this problem by using hashing. The idea is to traverse the tree in an inorder fashion and insert every node's value into a set. Also check if, for any node, the difference between the given sum and node's value is found in the set, then the pair with the given sum exists in the tree.

ASS_Code3.java

Q4. Given a BST, find the inorder predecessor of a given key in it. If the key does not lie in the BST, return the previous greater node (if any) present in the BST.



The predecessor of node 15 is 12
The predecessor of node 10 is 8
The predecessor of node 20 is 16
The predecessor doesn't exist for node 8
The predecessor of node 12 is 10
The predecessor of node 16 is 15
The predecessor of node 25 is 20

A node's inorder predecessor is a node with maximum value in its left subtree, i.e., its left subtree's right-most child. If the left subtree of the node doesn't exist, then the inorder predecessor is one of its ancestors.

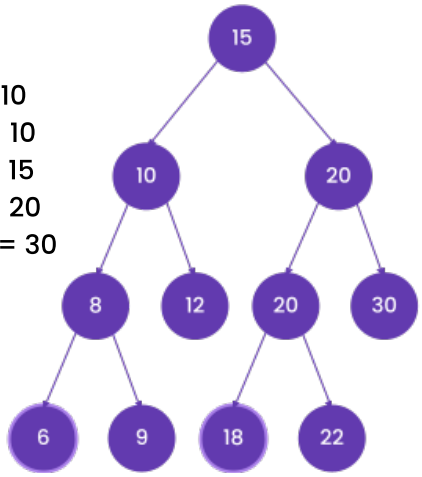
To find which ancestors are the predecessor, move up the tree towards the root until we encounter a node that is the right child of its parent. If any such node is found, then the inorder predecessor is its parent; otherwise, the inorder predecessor does not exist for the node.

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

ASS_Code4.java

Q5. Given a BST and two nodes x and y in it, find the lowest common ancestor (LCA) of x and y. The solution should return null if either x or y is not the actual node in the tree.

LCA (6, 12) = 10
LCA (10, 12) = 10
LCA (20, 6) = 15
LCA (18, 22)= 20
LCA (30, 30) = 30



We can recursively find the lowest common ancestor of nodes x and y present in the BST. The trick is to find the BST node, which has one key present in its left subtree and the other key present in the right subtree. If any such node is present in the tree, then it is LCA; if y lies in the subtree rooted at node x, then x is the LCA; otherwise, if x lies in the subtree rooted at node y, then y is the LCA.

ASS_Code5.java