**Second largest element in BST**

## Question -

Given a Binary Search Tree(BST), find the second largest element.

**Examples:**

Input: Root of below BST

    10

   /

  5

Output:  5

Input: Root of below BST

        10

      /   \

    5      20

             \

              30

Output:  2

## Solutions:

The second largest element is second last element in in-order traversal and second element in reverse in-order traversal. We traverse given Binary Search Tree in reverse in-order and keep track of counts of nodes visited. Once the count becomes 2, we print the node.  
Below is the implementation of above idea.

|  |
| --- |
| // Java code to find second largest element in BST  // A binary tree node  class Node {  int data;  Node left, right;  Node(int d)  {  data = d;  left = right = null;  }  }  class BinarySearchTree {  // Root of BST  Node root;  // Constructor  BinarySearchTree()  {  root = null;  }  // function to insert new nodes  public void insert(int data)  {  this.root = this.insertRec(this.root, data);  }    /\* A utility function to insert a new node with given  key in BST \*/  Node insertRec(Node node, int data)  {  /\* If the tree is empty, return a new node \*/  if (node == null) {  this.root = new Node(data);  return this.root;  }  /\* Otherwise, recur down the tree \*/  if (data < node.data) {  node.left = this.insertRec(node.left, data);  } else {  node.right = this.insertRec(node.right, data);  }  return node;  }  // class that stores the value of count  public class count {  int c = 0;  }  // Function to find 2nd largest element  void secondLargestUtil(Node node, count C)  {  // Base cases, the second condition is important to  // avoid unnecessary recursive calls  if (node == null || C.c >= 2)  return;    // Follow reverse inorder traversal so that the  // largest element is visited first  this.secondLargestUtil(node.right, C);    // Increment count of visited nodes  C.c++;    // If c becomes k now, then this is the 2nd largest  if (C.c == 2) {  System.out.print("2nd largest element is "+  node.data);  return;  }    // Recur for left subtree  this.secondLargestUtil(node.left, C);  }  // Function to find 2nd largest element  void secondLargest(Node node)  {  // object of class count  count C = new count();  this.secondLargestUtil(this.root, C);  }  // Driver function  public static void main(String[] args)  {  BinarySearchTree tree = new BinarySearchTree();    /\* Let us create following BST  50  / \  30 70  / \ / \  20 40 60 80 \*/    tree.insert(50);  tree.insert(30);  tree.insert(20);  tree.insert(40);  tree.insert(70);  tree.insert(60);  tree.insert(80);  tree.secondLargest(tree.root);  }  } |

## Output:

2nd largest element is 70

Time complexity of the above solution is O(h) where his height of BST.