Second largest element in BST

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:  20
```

Source: Microsoft Interview

We strongly recommend you to minimize your browser and try this yourself first.

The idea is similar to below post.

K'th Largest Element in BST when modification to BST is not allowed

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

Below is C++ implementation of above idea.

```
// C++ program to find 2nd largest element in BST
#include<iostream>
using namespace std;

struct Node
{
    int key;
    Node *left, *right;
};

// A utility function to create a new BST node
Node *newNode(int item)
{
    Node *temp = new Node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// A function to find 2nd largest element in a given tree.
void secondLargestUtil(Node *root, int &c)
{
    // Base cases, the second condition is important to
    // avoid unnecessary recursive calls
    if (root == NULL || c >= 2)
        return;

    // Follow reverse inorder traversal so that the
    // largest element is visited first
    secondLargestUtil(root->right, c);

    // Increment count of visited nodes
    c++;
```

```
    // If c becomes k now, then this is the 2nd largest
    if (c == 2)
    {
        cout << "2nd largest element is "
             << root->key << endl;
        return;
    }

    // Recur for left subtree
    secondLargestUtil(root->left, c);
}

// Function to find 2nd largest element
void secondLargest(Node *root)
{
    // Initialize count of nodes visited as 0
    int c = 0;

    // Note that c is passed by reference
    secondLargestUtil(root, c);
}

/* A utility function to insert a new node with given key in BST */
Node* insert(Node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}

// Driver Program to test above functions
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         /  \    /  \
        20   40  60   80 */
    Node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    secondLargest(root);

    return 0;
}
```

Output:

```
2nd largest element is 70
```

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