Saturday, March 12, 2022 3:56 PM

k-th smallest element in BST

Medium Accuracy: 49.44% Submissions: 55030 Points: 4

Given a BST and an integer K. Find the Kth Smallest element in the BST.

Example 1:

```
Input:
     2
   /   \
  1     3
K = 2
Output: 2
```

Example 2:

```
Input:
       2
     /  \
    1    3
K = 5
Output: -1
```

Your Task:

You don't need to read input or print anything. Your task is to complete the function **KthSmallestElement()** which takes the root of the BST and integer K as inputs and return the Kth smallest element in the BST, if no such element exists return -1.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(1).

Constraints:

1<=Number of nodes<=100000

From <https://practice.geeksforgeeks.org/problems/find-k-th-smallest-element-in-bst/1#>

```
package BST;
public class Kth_element_in_BST {


class Solution {
    class Ans{
    int x=-1;



}
    // Return the Kth smallest element in the given BST

    // int ans=0;
 int index ;


   public int KthSmallestElement(Node root, int K) {

        Ans ele=new Ans () ;

        getElement(root,K , ele);

        return ele.x  ;

    }
    public void getElement(Node root,int key , Ans  ele ){
        if(root==null)
        {
            return ;
        }
        getElement(root.left , key ,ele  );
        if((index+1)==key && ele.x == -1){

            ele.x=root.data ;

            return ;

        }
        ++index;



        getElement(root.right , key , ele  );


    }
}
```

GIVEN SOLUTION

```
class Solution {
    public int KthSmallestElement(Node root, int k) {
        Node temp = MorrisInorderTraversal(root, k);
        if (temp != null)
            return temp.data;
        else
            return -1;
    }
    public Node MorrisInorderTraversal(Node root, int k) {
        if (root == null) return null;
        Node prev = null;
        Node curr = root;
        while (curr != null) {
            // check for presence of left subtree
            if (curr.left == null) {
                // If kth smallest is found
                if (k == 1) {
                    // Return the current node
                    return curr;
                }
                k--;
                // Traverse right subtree otherwise
                curr = curr.right;
            } else {
                // Find the inorder predecessor of current
                prev = curr.left;
                while (prev.right != null && prev.right != curr) {
                    prev = prev.right;
                }
                if (prev.right == null) {
                    // Make current as the right child of
                    // its inorder predecessor
                    prev.right = curr;
                    curr = curr.left;
                } else {
                    // Revert the changes to right child
                    // of predecessor
                    prev.right = null;
                    k--;
                    if (k == 0) return curr;
                    // Traverse right subtree
                    curr = curr.right;
                }
            }
        }
        return null;
    }
}
```