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

Difficulty Level Medium

Frequently asked in Amazon Cisco Google UHG Optum

Tags Binary Search Tree Binary Tree Tree Views 25

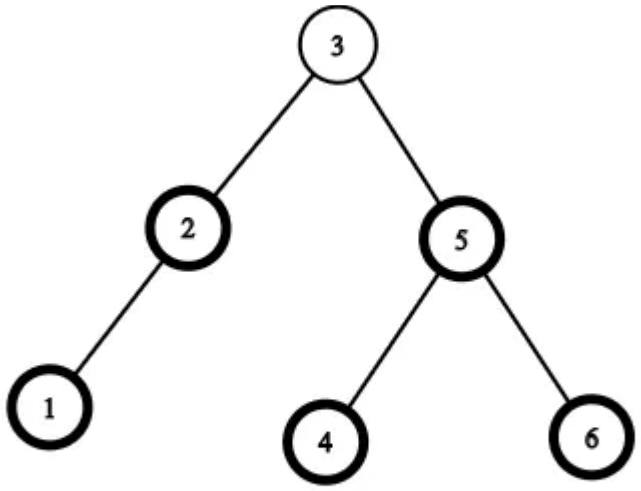
Table of Contents

Problem Statement

"K'th Largest Element in BST when modification to BST is not allowed" states that you are given a binary search tree and you need to find the kth largest element. This means that when all the elements of the binary search tree are arranged in descending order. Then we need to return the kth element from the sequence.

Example

Input



k=4

Output

3

Explanation: The image of the tree is shown above. And we need to find 4th largest element. So if we arrange them in the

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

We have already done a similar question, where we found the k-th smallest element in the BST. The problem is just a modification over that. In the previous post, we discussed solutions for that question. The first approach was to modify the tree data structure. While the other was to run an in-order traversal for the binary search tree and keep counting the nodes. Since the previous question was to return the kth smallest. We simply counted the number of nodes and once the count was equal to k we returned the value on that node.

Naive Approach

Efficient Approach

traversal of the binary search tree traverses the tree in descending order. So while doing reverse in-order, we will keep on counting the nodes. and when the count becomes equal to k, we return the value on that node.

Code

C++ code to find K'th Largest Element in BST when modification to BST is not allowed

Code

```
#include <bits/stdc++.h>
using namespace std;
struct node{
    int data;
    node *left;
    node *right;
} ;
node* create(int data){
    node * tmp = new node();
    tmp->data = data;
    tmp->left = tmp->right = NULL;
    return tmp;
}
// normally insert the node in BST
```



```
        root->left = insert(root->left, x);
    root->data)
    right = insert(root->right, x);
```

```
node *findKthLargest(node *root, int& k)
{
    // base case
    if(!root)
        return NULL;
    node *right = findKthLargest(root->right, k);
    if(right)
        return right;
    // if current element is kth largest
    if(k==1)
        return root;
    // if the kth largest is not found in the left subtree
    // search in left subtree
    k--;
    return findKthLargest(root->left, k);
}
int main()
{
    node *root = NULL;
    int n;cin>>n;
    for(int i=0;i<n;i++){
        int element;cin>>element;
        root = insert(root, element);
    }
    int k;cin>>k;
    node* res = findKthLargest(root, k);
    if(res == NULL)
```

Output

```
Kth largest element is 3
```

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

Code

```
import java.util.*;
import java.lang.*;
import java.io.*;

class node{
  int data;
  node left;
  node right;
}

class Tree{
  static node root;
  static int count;
  static node create(int data){
      node tmp = new node();
      tmp.data = data;
```



```
  static node insert(node root, int x)
```

```
        if(x<root.data)
            root.left = insert(root.left, x);
        else if(x>root.data)
            root.right = insert(root.right, x);
        return root;
    }

    static node findKthLargest(node root, int k)
    {
        // base case
        if(root == null)
            return null;
        node right = findKthLargest(root.right, k);
        if(right != null)
            return right;
        count++;
        // if current element is kth largest
        if(k==count)
            return root;
        // if the kth largest is not found in the left subtree
        // search in left subtree
        return findKthLargest(root.left, k);
    }

    public static void main(String[] args)
    {
      Scanner sc = new Scanner(System.in);
```



```
        int k = sc.nextInt();
        0;
        = findKthLargest(root, k);
```

```
            System.out.println("Kth largest element is "+res.data);
    }
}
```

Input

```
6
3 2 1 5 4 6
4
```

Output

```
Kth largest element is 3
```

Complexity Analysis

Time Complexity

O(N), because we have simply traversed through the BST, and since the BST had only N nodes. We have achieved a linear time complexity of O(N).

Space Complexity

O(1), we have used only constant extra space. Thus only this algorithm has constant space complexity but the program as a whole has linear space complexity. Because we are storing N binary tree nodes.

Tree Interview Questions
Amazon, Binary Search Tree, Binary Tree, Cisco, Google, Medium, Tree, UHG Optum
< Tracking current Maximum Element in a Stack
> Arrange given numbers to form the biggest number