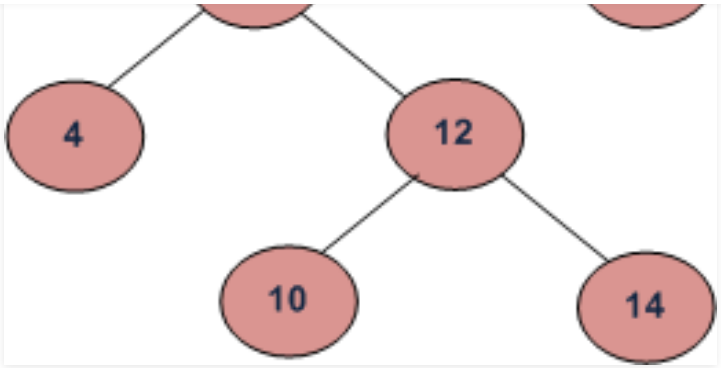
K’th Largest Element in BST when modification to BST is not allowed

Difficulty Level : Easy • Last Updated : 18 Mar, 2022

Given a Binary Search Tree (BST) and a positive integer k, find the k’th largest element in the Binary Search Tree.
For example, in the following BST, if k = 3, then output should be 14, and if k = 5, then output should be 10.



We have discussed two methods in this post. The method 1 requires O(n) time. The method 2 takes O(h) time where h is height of BST, but requires augmenting the BST (storing count of nodes in left subtree with every node).
Can we find k'th largest element in better than O(n) time and no augmentation?

Approach:

1. The idea is to do reverse inorder traversal of BST. Keep a count of nodes visited.

C++

```
// C++ program to find k'th largest element in BST
#include<bits/stdc++.h>
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 k'th largest element in a given tree.
void kthLargestUtil(Node *root, int k, int &c)
{
    // Base cases, the second condition is important to
```

```
    kthLargestUtil(root->right, k, c);

    // Increment count of visited nodes
    c++;

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

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

// Function to find k'th largest element
void kthLargest(Node *root, int k)
{
    // Initialize count of nodes visited as 0
    int c = 0;

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

/* A utility function to insert a new node with given key in BST */
Node* insert(Node* node, int 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);

    int c = 0;
    for (int k=1; k<=7; k++)
        kthLargest(root, k);
```

Java

```java
// Java code to find k'th 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 nodes
```

```
/* 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;
    }

    if (data == node.data) {
        return node;
    }

    /* 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;
}

// utility function to find kth largest no in
// a given tree
```

```
        return;

    // Follow reverse inorder traversal so that the
    // largest element is visited first
    this.kthLargestUtil(node.right, k, C);

    // Increment count of visited nodes
    C.c++;

    // If c becomes k now, then this is the k'th largest
    if (C.c == k) {
        System.out.println(k + "th largest element is " +
                                            node.data);
        return;
    }

    // Recur for left subtree
    this.kthLargestUtil(node.left, k, C);
}

// Method to find the kth largest no in given BST
void kthLargest(int k)
{
    count c = new count(); // object of class count
    this.kthLargestUtil(this.root, k, c);
}

// Driver function
public static void main(String[] args)
{
```

```
          /  \    /  \
        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);

         for (int i = 1; i <= 7; i++) {
             tree.kthLargest(i);
         }
    }
}

// This code is contributed by Kamal Rawal
```

Python3

```
# Python3 program to find k'th largest
# element in BST

class Node:

    # Constructor to create a new node
    def __init__(self, data):
```

```
# element in a given tree.
def kthLargestUtil(root, k, c):

    # Base cases, the second condition
    # is important to avoid unnecessary
    # recursive calls
    if root == None or c[0] >= k:
        return

    # Follow reverse inorder traversal
    # so that the largest element is
    # visited first
    kthLargestUtil(root.right, k, c)

    # Increment count of visited nodes
    c[0] += 1

    # If c becomes k now, then this is
    # the k'th largest
    if c[0] == k:
        print("K'th largest element is",
                               root.key)
        return

    # Recur for left subtree
    kthLargestUtil(root.left, k, c)

# Function to find k'th largest element
def kthLargest(root, k):
```

```
# A utility function to insert a new
# node with given key in BST */
def insert(node, key):

    # If the tree is empty,
    # return a new node
    if node == None:
        return Node(key)

    # Otherwise, recur down the tree
    if key < node.key:
        node.left = insert(node.left, key)
    elif key > node.key:
        node.right = insert(node.right, key)

    # return the (unchanged) node pointer
    return node

# Driver Code
if __name__ == '__main__':

    # Let us create following BST
    #          50
    #      /     \
    #     30      70
    # / \ / \
    # 20 40 60 80 */
    root = None
```

```
    insert(root, 80)

    for k in range(1,8):
        kthLargest(root, k)

# This code is contributed by PranchalK
```

C#

```
using System;

// C# code to find k'th largest element in BST

// A binary tree node
public class Node
{

    public int data;
    public Node left, right;

    public Node(int d)
    {
        data = d;
        left = right = null;
    }
}
```

```
// Constructor
public BinarySearchTree()
{
    root = null;
}

// function to insert nodes
public virtual void insert(int data)
{
    this.root = this.insertRec(this.root, data);
}

/* A utility function to insert a new node
with given key in BST */
public virtual 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;
    }

    if (data == node.data)
    {
        return node;
    }

    /* Otherwise, recur down the tree */
```

```
        {
            node.right = this.insertRec(node.right, data);
        }
        return node;
    }

    // class that stores the value of count
    public class count
    {
        private readonly BinarySearchTree outerInstance;

        public count(BinarySearchTree outerInstance)
        {
            this.outerInstance = outerInstance;
        }

        internal int c = 0;
    }

    // utility function to find kth largest no in
    // a given tree
    public virtual void kthLargestUtil(Node node, int k, count C)
    {
        // Base cases, the second condition is important to
        // avoid unnecessary recursive calls
        if (node == null || C.c >= k)
        {
            return;
        }
```

```
    // If c becomes k now, then this is the k'th largest
    if (C.c == k)
    {
        Console.WriteLine(k + "th largest element is " + node.data);
        return;
    }

    // Recur for left subtree
    this.kthLargestUtil(node.left, k, C);
}

// Method to find the kth largest no in given BST
public virtual void kthLargest(int k)
{
    count c = new count(this); // object of class count
    this.kthLargestUtil(this.root, k, c);
}

// Driver function
public static void Main(string[] args)
{
    BinarySearchTree tree = new BinarySearchTree();

    /* Let us create following BST
          50
       /     \
      30      70
     /  \    /  \
```

```
        tree.insert(60);
        tree.insert(80);

        for (int i = 1; i <= 7; i++)
        {
            tree.kthLargest(i);
        }
    }
}

  // This code is contributed by Shrikant13
```

Javascript

```
<script>
// javascript code to find k'th largest element in BST

// A binary tree node
class Node {

   constructor(d)
    {
        this.data = d;
        this.left = this.right = null;
    }
}
```

```
// function to insert nodes
function insert(data)
{
    this.root = this.insertRec(this.root, data);
}

/* A utility function to insert a new node
with given key in BST */
function insertRec( node , data)
{
    /* If the tree is empty, return a new node */
    if (node == null) {
        this.root = new Node(data);
        return this.root;
    }

    if (data == node.data) {
        return node;
    }

    /* 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;
```

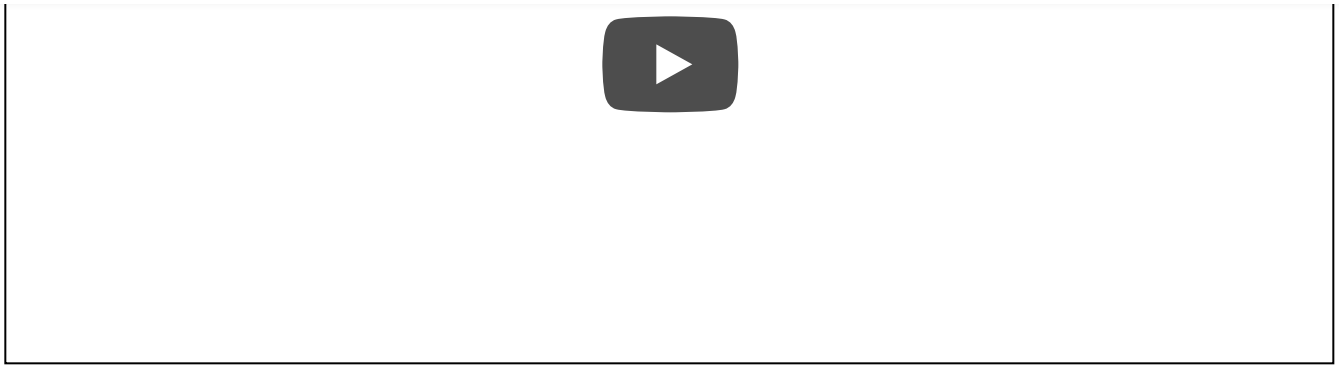
```
}

// utility function to find kth largest no in
// a given tree
function kthLargestUtil( node , k,  C)
{
    // Base cases, the second condition is important to
    // avoid unnecessary recursive calls
    if (node == null || C.c >= k)
        return;

    // Follow reverse inorder traversal so that the
    // largest element is visited first
    this.kthLargestUtil(node.right, k, C);

    // Increment count of visited nodes
    C.c++;

    // If c becomes k now, then this is the k'th largest
    if (C.c == k) {
        document.write(k + "th largest element is " +
                                            node.data+"<br/>");
        return;
    }

    // Recur for left subtree
    this.kthLargestUtil(node.left, k, C);
}
```

```
}

// Driver function


    /* Let us create following BST
          50
       /     \
      30      70
     /  \    /  \
    20   40  60   80 */
    insert(50);
    insert(30);
    insert(20);
    insert(40);
    insert(70);
    insert(60);
    insert(80);

    for (i = 1; i <= 7; i++) {
        kthLargest(i);
    }

// This code contributed by gauravrajput1
</script>
```

```
K'th largest element is 70
K'th largest element is 60
K'th largest element is 50
K'th largest element is 40
K'th largest element is 30
K'th largest element is 20
```

Complexity Analysis:

1. **Time Complexity:** O(n).
   In worst case the code can traverse each and every node of the tree if the k given is equal to n (total number of nodes in the tree). Therefore overall time complexity is O(n).
2. **Auxiliary Space:** O(h).
   Max recursion stack of height h at a given time.



?list=PLqM7alHXFySHCXD7r1J0ky9Zg_GBB1dbk

This article is contributed by **Chirag Sharma**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Find k-th smallest element in BST (Order Statistics in BST)

K'th Largest element in BST using constant extra space

RECOMMENDED ARTICLES

Page :

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Article Tags : Accolite, Amazon, Order-Statistics, Samsung, Binary Search Tree, Tree

Practice Tags : Accolite, Amazon, Samsung, Binary Search Tree, Tree

GeeksforGeeks
5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org