GeeksforGeeks

K’th Largest element in BST using constant extra space

Difficulty Level : Hard • Last Updated : 21 May, 2021

Given a binary search tree, task is to find Kth largest element in the binary search tree.

Example:

```
Input :  k = 3
         Root of following BST
            10
          /    \
         4      20
        /      /   \
       2     15     40
Output : 15
```

Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

The special thing about Morris traversal is that we can do Inorder traversal without using stack or recursion which saves us memory consumed by stack or recursion call stack.
Reverse Morris traversal is just the reverse of Morris traversal which is majorly used to do Reverse Inorder traversal with constant O(1) extra memory consumed as it does not uses any Stack or Recursion.
To find Kth largest element in a Binary search tree, the simplest logic is to do reverse inorder traversal and while doing reverse inorder traversal simply keep a count of number of Nodes visited. When the count becomes equal to k, we stop the traversal and print the data. It uses the fact that reverse inorder traversal will give us a list sorted in descending order.

Algorithm

```
1) Initialize Current as root.
2) Initialize a count variable to 0.
3) While current is not NULL :
   3.1) If current has no right child
   a) Increment count and check if count is equal to K.
      1) If count is equal to K, simply return current
         Node as it is the Kth largest Node.
```

```
     inorder successor is the left most Node
     in the right subtree or right child itself.
  b) If the left child of the inorder successor is NULL:
     1) Set current as the left child of its inorder
        successor.
     2) Move current Node to its right.
  c) Else, if the threaded link between the current Node
     and it's inorder successor already exists :
     1) Set left pointer of the inorder successor as NULL.
     2) Increment count and check if count is equal to K.
          a) If count is equal to K, simply return current
             Node as it is the Kth largest Node.

     3) Otherwise, Move current to it's left child.
```

C++

```cpp
// CPP code for finding K-th largest Node using O(1)
// extra memory and reverse Morris traversal.
#include <bits/stdc++.h>
using namespace std;

struct Node {
```

```
Node* newNode(int data)
{
    Node* temp = new Node;
    temp->data = data;
    temp->right = temp->left = NULL;
    return temp;
}

Node* KthLargestUsingMorrisTraversal(Node* root, int k)
{
    Node* curr = root;
    Node* Klargest = NULL;

    // count variable to keep count of visited Nodes
    int count = 0;

    while (curr != NULL) {
        // if right child is NULL
        if (curr->right == NULL) {

            // first increment count and check if count = k
            if (++count == k)
                Klargest = curr;

            // otherwise move to the left child
            curr = curr->left;
        }

        else {
            if (succ->left == NULL) {

                // set left child of successor to the
                // current Node
                succ->left = curr;

                // move current to its right
                curr = curr->right;
            }

            // restoring the tree back to original binary
            //  search tree removing threaded links
            else {

                succ->left = NULL;

                if (++count == k)
                    Klargest = curr;

                // move current to its left child
                curr = curr->left;
            }
        }
    }

    return Klargest;
}

int main()
```

```
     /  \   /  \
    1    3  6    10 */

    Node* root = newNode(4);
    root->left = newNode(2);
    root->right = newNode(7);
    root->left->left = newNode(1);
    root->left->right = newNode(3);
    root->right->left = newNode(6);
    root->right->right = newNode(10);

    cout << "Finding K-th largest Node in BST : "
         << KthLargestUsingMorrisTraversal(root, 2)->data;

    return 0;
}
```

Java

```
// Java Program for finding K-th largest Node using O(1)
// extra memory and reverse Morris traversal.
class GfG
{

static class Node
{
    int data;
{
    Node temp = new Node();
    temp.data = data;
    temp.right = null;
    temp.left = null;
    return temp;
}

static Node KthLargestUsingMorrisTraversal(Node root, int k)
{
    Node curr = root;
    Node Klargest = null;

    // count variable to keep count of visited Nodes
    int count = 0;

    while (curr != null)
    {
        // if right child is NULL
        if (curr.right == null)
        {

            // first increment count and check if count = k
            if (++count == k)
                Klargest = curr;

            // otherwise move to the left child
            curr = curr.left;
        }
        while (succ.left != null && succ.left != curr)
            succ = succ.left;

        if (succ.left == null)
        {

            // set left child of successor to the
            // current Node
            succ.left = curr;

            // move current to its right
            curr = curr.right;
        }

        // restoring the tree back to original binary
        // search tree removing threaded links
        else
        {

            succ.left = null;

            if (++count == k)
                Klargest = curr;

            // move current to its left child
            curr = curr.left;
        }
    }
}
```

```
    // Your Java Code
    /* Constructed binary tree is
        4
        / \
    2 7
    / \ / \
    1 3 6 10 */

    Node root = newNode(4);
    root.left = newNode(2);
    root.right = newNode(7);
    root.left.left = newNode(1);
    root.left.right = newNode(3);
    root.right.left = newNode(6);
    root.right.right = newNode(10);

    System.out.println("Finding K-th largest Node in BST : " +
                    KthLargestUsingMorrisTraversal(root, 2).data);
}
}
```

Python3

```
# Python3 code for finding K-th largest
# Node using O(1) extra memory and
# reverse Morris traversal.
```

```
def KthLargestUsingMorrisTraversal(root, k):
    curr = root
    Klargest = None

    # count variable to keep count
    # of visited Nodes
    count = 0

    while (curr != None):

        # if right child is None
        if (curr.right == None):

            # first increment count and
            # check if count = k
            count += 1
            if (count == k):
                Klargest = curr

            # otherwise move to the left child
            curr = curr.left

        else:

            # find inorder successor of
            # current Node
            succ = curr.right

            while (succ.left != None and
                # set left child of successor
                # to the current Node
                succ.left = curr

                # move current to its right
                curr = curr.right

            # restoring the tree back to
            # original binary search tree
            # removing threaded links
            else:

                succ.left = None
                count += 1
                if (count == k):
                    Klargest = curr

                # move current to its left child
                curr = curr.left

    return Klargest

# Driver Code
if __name__ == '__main__':

    # Constructed binary tree is
    #     4
    #     / \
    # 2     7
    # / \ / \
    root.left.right = newNode(3)
    root.right.left = newNode(6)
    root.right.right = newNode(10)

    print("Finding K-th largest Node in BST : ",
           KthLargestUsingMorrisTraversal(root, 2).data)

# This code is contributed by PranchalK
```

C#

```
// C# Program for finding K-th largest Node using O(1)
// extra memory and reverse Morris traversal.
using System;
using System.Collections.Generic;

class GfG
{

public class Node
{
    public int data;
    public Node left, right;
}

// helper function to create a new Node
static Node newNode(int data)
{
}

static Node KthLargestUsingMorrisTraversal(Node root, int k)
{
    Node curr = root;
    Node Klargest = null;

    // count variable to keep count of visited Nodes
    int count = 0;

    while (curr != null)
    {
        // if right child is NULL
        if (curr.right == null)
        {

            // first increment count and check if count = k
            if (++count == k)
                Klargest = curr;

            // otherwise move to the left child
            curr = curr.left;
        }

        else
        {

            // find inorder successor of current Node
            Node succ = curr.right;
```

```
                    // set left child of successor to the
                    // current Node
                    succ.left = curr;

                    // move current to its right
                    curr = curr.right;
                }

                // restoring the tree back to original binary
                // search tree removing threaded links
                else
                {

                    succ.left = null;

                    if (++count == k)
                        Klargest = curr;

                    // move current to its left child
                    curr = curr.left;
                }
            }
        }
        return Klargest;
    }

    // Driver code
    public static void Main(String[] args)
    {
```

```
    /  \  /  \
    1 3 6 10 */

    Node root = newNode(4);
    root.left = newNode(2);
    root.right = newNode(7);
    root.left.left = newNode(1);
    root.left.right = newNode(3);
    root.right.left = newNode(6);
    root.right.right = newNode(10);

    Console.Write("Finding K-th largest Node in BST : " +
                KthLargestUsingMorrisTraversal(root, 2).data);
}
}

// This code has been contributed by 29AjayKumar
```

Javascript

```
<script>

// JavaScript Program for finding
// K-th largest Node using O(1)
// extra memory and reverse
// Morris traversal.

 class Node
```

```
}

// helper function to create a new Node
function newNode(data)
{
    var temp = new Node();
    temp.data = data;
    temp.right = null;
    temp.left = null;
    return temp;
}

function KthLargestUsingMorrisTraversal(root , k)
{
    var curr = root;
    var Klargest = null;

    // count variable to keep count of visited Nodes
    var count = 0;

    while (curr != null)
    {
        // if right child is NULL
        if (curr.right == null)
        {

            // first increment count and
            // check if count = k
            if (++count == k)
```

```
else
{

    // find inorder successor of
    // current Node
    var succ = curr.right;

    while (succ.left != null && succ.left != curr)
        succ = succ.left;

    if (succ.left == null)
    {

        // set left child of successor to the
        // current Node
        succ.left = curr;

        // move current to its right
        curr = curr.right;
    }

    // restoring the tree back to original binary
    // search tree removing threaded links
    else
    {

        succ.left = null;

        if (++count == k)
```

```
            }
        }
        return Klargest;
    }

    // Driver code

        // Your JavaScript Code
        /* Constructed binary tree is
            4
            / \
           2 7
        / \ / \
        1 3 6 10 */

         root = newNode(4);
        root.left = newNode(2);
        root.right = newNode(7);
        root.left.left = newNode(1);
        root.left.right = newNode(3);
        root.right.left = newNode(6);
        root.right.right = newNode(10);

        document.write("Finding K-th largest Node in BST : " +
                   KthLargestUsingMorrisTraversal(root, 2).data);

    // This code contributed by aashish1995

    </script>
```

Time Complexity : O(n)

Auxiliary Space : O(1)

Like

Previous

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

Next

Second largest element in BST

RECOMMENDED ARTICLES

Page :

Article Contributed By :

AnishSinghWalia
@AnishSinghWalia

Vote for difficulty

Current difficulty : Hard

Easy Normal Medium Hard Expert

Improved By : prerna saini, PranchalKatiyar, 29AjayKumar, aashish1995

Article Tags : Order-Statistics, Binary Search Tree

Practice Tags : Binary Search Tree

GeeksforGeeks

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

feedback@geeksforgeeks.org

Top News

Technology

Work & Career

Business

Finance

Lifestyle

Java

CPP

Golang

C#

SQL

Web Development

Web Tutorials

Django Tutorial

HTML

CSS

JavaScript

Bootstrap

Contribute

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship