Project Of Data Sructure And Algorithm

Topic : Find Kth Smallest Element In BST

Submitted To : Mam Arshi Pervaiz

Submitted By : Group Members

1.)Muneeb ur Rehman(CU-4178-2023)

2.)Hashir Mehmood(CU-4220-2023)

Section : 'C'

Department : BSCS

Date : 14-Feb-2025

Introduction:

This project implements an algorithm to find the **k-th smallest element** in a **Binary Search Tree (BST)** using **in-order traversal**. In-order traversal of a BST visits the nodes in ascending order, which makes it an efficient approach to solve the problem of identifying the k-th smallest element in the tree.

A **Binary Search Tree (BST)** is a type of binary tree where the value of each node follows a specific rule:

- The left subtree contains only nodes with values **less than** the parent node.
- The right subtree contains only nodes with values **greater than** the parent node.

Purpose Of The Project:

The primary objective of this project is to showcase how to leverage the **in-order traversal** property of a BST to efficiently find the k-th smallest element. By traversing the tree in ascending order, the elements are visited one-by-one, and the k-th smallest element can be identified easily once we have traversed the first k elements.

This project demonstrates:

- How a Binary Search Tree (BST) is structured and how elements are inserted into it.
- How **in-order traversal** works and how it provides a natural ordering of elements in a BST.
- How to find the **k-th smallest element** in the tree by performing an in-order traversal.

Project Explanation:

1. The TreeNode class represents each node in the BST.
2. The Solution class contains the logic to find the k-th smallest element using in-order traversal.
3. The main program creates a sample BST, calls the kthSmallest method to find the desired element, and outputs the result.

Implementation:

```
#include <iostream>

using namespace std;

struct TreeNode {

   int val;

   TreeNode *left, *right;

   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

   int kthSmallest(TreeNode* root, int k) {

      int count = 0;

      int result = -1;

      inorderTraversal(root, k, count, result);

      return result;

   }

private:

   void inorderTraversal(TreeNode* root, int k, int &count, int &result) {

      if (!root) return;

        inorderTraversal(root->left, k, count, result);

         count++;

       if (count == k) {

          result = root->val;

          return;

      }

  inorderTraversal(root->right, k, count, result);
```

```
    }
};
TreeNode* insert(TreeNode* root, int val) {
    if (!root) {
        return new TreeNode(val);
    }
    if (val < root->val) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}
int main() {
  Solution solution;
 TreeNode* root = nullptr;
     root = insert(root, 5);
    root = insert(root, 3);
    root = insert(root, 6);
    root = insert(root, 2);
    root = insert(root, 4);
  int k = 4;
      int result = solution.kthSmallest(root, k);

    if (result != -1) {
        cout << "The " << k << "-th smallest element in the BST is: " << result << endl;
    } else {
        cout << "The tree doesn't have " << k << " elements." << endl;
```

```
    }
    return 0;
}
```

Explanation Of Code:

1. **TreeNode Structure**:
    - This struct defines a node in the BST, with an integer value val and two pointers left and right pointing to the left and right children, respectively. The constructor initializes the node with a value and sets the left and right pointers to nullptr.

2. **Solution Class**:

This class contains the main logic to solve the problem.

kthSmallest: This is the public method that will be called to find the k-th smallest element. It initializes the count and result and calls the helper function inorderTraversal.

inorderTraversal: This is a recursive function that performs in-order traversal. During the traversal:

- It first visits the left child of the node.
- Then, it processes the current node by incrementing a counter (count). If count equals k, it sets the result as the current node's value and exits early.
- Lastly, it visits the right child.

3. **Insert Function:**
    - This helper function inserts values into the BST. If the root is nullptr, it creates a new node. For values smaller than the current node's value, it moves to the left; for values larger, it moves to the right, ensuring that the tree maintains its binary search properties.

4. **Main Function:**
    - The main function demonstrates how to use the Solution class.
    - A sample BST is created by calling insert repeatedly. The nodes inserted are 5, 3, 6, 2, and 4, forming a valid BST.
    - It then calls kthSmallest to find the 4th smallest element in the BST and prints the result.

Out Put:

```
The 4-th smallest element in the BST is: 5


--------------------------------
Process exited after 1.443 seconds with return value 0
Press any key to continue . . .
```

Explanation Of The Output:

- The BST created is as follows:

```
    5
   / \
  3   6
 / \
2   4
```

- In-order traversal of the BST gives: 2, 3, 4, 5, 6.
- The 4th smallest element is 5, so the program outputs: The 4-th smallest element in the BST is: 5.

Conclusion:

This project demonstrates how to find the k-th smallest element in a Binary Search Tree using in-order traversal in C++. The code is simple and efficient, taking advantage of the properties of in-order traversal, where the nodes are visited in ascending order.