Rajalakshmi Engineering College

Name: vinnarasan j
Email: 241501245@rajalakshmi.edu.in
Roll no: 241501245
Phone: 7305999373
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are required to implement basic operations on a Binary Search Tree (BST), like insertion and searching.

Insertion: Given a list of integers, construct a Binary Search Tree by repeatedly inserting each integer into the tree according to the rules of a BST.

Searching: Given an integer, search for its presence in the constructed Binary Search Tree. Print whether the integer is found or not.

Write a program to calculate this efficiently.

Input Format

The first line of input consists of an integer n, representing the number of nodes

in the binary search tree.

The second line consists of the values of the nodes, separated by space as integers.

The third line consists of an integer representing, the value that is to be searched.

Output Format

The output prints, "Value <value> is found in the tree." if the given value is present, otherwise it prints: "Value <value> is not found in the tree."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 7
8 3 10 1 6 14 23
6
Output: Value 6 is found in the tree.

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Structure for a node in the binary search tree
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

// Function to create a new node
struct TreeNode* createNode(int val) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    if (newNode == NULL) {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }
```

```
    newNode->val = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a value into the BST
struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL) {
        return createNode(val);
    }
    if (val < root->val) {
        root->left = insert(root->left, val);
    } else if (val > root->val) {
        root->right = insert(root->right, val);
    } else {
        // Handle duplicate values (optional: for this problem, we assume no
duplicates)
        return root;
    }
    return root;
}

// Function to search for a value in the BST
struct TreeNode* search(struct TreeNode* root, int val) {
    if (root == NULL) {
        return NULL; // Value not found
    }
    if (root->val == val) {
        return root; // Value found
    } else if (val < root->val) {
        return search(root->left, val); // Search in the left subtree
    } else {
        return search(root->right, val); // Search in the right subtree
    }
}

// Function to free the memory allocated for the BST
void freeTree(struct TreeNode* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
```

```
        free(root);
    }
}

int main() {
    int n, val, searchValue;
    struct TreeNode* root = NULL;

    // Read the number of nodes
    if (scanf("%d", &n) != 1) {
        fprintf(stderr, "Error reading number of nodes.\n");
        return EXIT_FAILURE;
    }

    // Read the values of the nodes and insert them into the BST
    for (int i = 0; i < n; i++) {
        if (scanf("%d", &val) != 1) {
            fprintf(stderr, "Error reading node value.\n");
            //  Free any allocated memory before exiting.  A more robust solution
would
            //  free the tree as it is being built.
            freeTree(root);
            return EXIT_FAILURE;
        }
        root = insert(root, val);
    }

    // Read the value to be searched
    if (scanf("%d", &searchValue) != 1) {
        fprintf(stderr, "Error reading search value.\n");
        freeTree(root);
        return EXIT_FAILURE;
    }
    // Search for the value in the BST
    struct TreeNode* result = search(root, searchValue);

    // Print the result
    if (result != NULL) {
        printf("Value %d is found in the tree.\n", searchValue);
    } else {
        printf("Value %d is not found in the tree.\n", searchValue);
    }
```

```
    freeTree(root); // Free the allocated memory
    return 0;
}
```

Status : Correct ***Marks : 10/10***