Rajalakshmi Engineering College

Name: SONASREE RP
Email: 240701521@rajalakshmi.edu.in
Roll no: 240701521
Phone: 7305340666
Branch: REC
Department: I CSE FE
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_CY_Updated

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

Arun is working on a Binary Search Tree (BST) data structure. His goal is to implement a program that reads a series of integers and inserts them into a BST. Once the integers are inserted, he needs to add a given integer value to each node in the tree and find the maximum value in the BST.

Your task is to help Arun implement this program.

Input Format

The first line of input consists of an integer N, representing the number of elements to be inserted into the BST.

The second line consists of N space-separated integers, each representing an element to be inserted into the BST.

The third line consists of an integer add, representing the value to be added to each node in the BST.

Output Format

The output prints the maximum value in the BST after adding the add value.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 20 25
5
Output: 30

Answer

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

struct Node {
    int data;
    struct Node *left, *right;
};

struct Node* newNode(int value) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = value;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return newNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
```

```
}
void addToAllNodes(struct Node* root, int addVal) {
    if (root == NULL)
        return;
    root->data += addVal;
    addToAllNodes(root->left, addVal);
    addToAllNodes(root->right, addVal);
}

int findMax(struct Node* root) {
    if (root == NULL)
        return -1;
    while (root->right != NULL)
        root = root->right;
    return root->data;
}

int main() {
    int N, value, addVal;
    scanf("%d", &N);

    struct Node* root = NULL;

    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    scanf("%d", &addVal);

    addToAllNodes(root, addVal);

    int maxVal = findMax(root);
    printf("%d\n", maxVal);

    return 0;
}
```

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

2. Problem Statement

Edward has a Binary Search Tree (BST) and needs to find the k-th largest element in it.

Given the root of the BST and an integer k, help Edward determine the k-th largest element in the tree. If k exceeds the number of nodes in the BST, return an appropriate message.

Input Format

The first line of input consists of integer n, the number of nodes in the BST.

The second line consists of the n elements, separated by space.

The third line consists of the value of k.

Output Format

The output prints the kth largest element in the binary search tree.

For invalid inputs, print "Invalid value of k".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 7
8 4 12 2 6 10 14
1
Output: 14

Answer

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

struct Node {
    int data;
    struct Node *left, *right;
```

```
};

struct Node* newNode(int value) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = value;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return newNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}

void kthLargestUtil(struct Node* root, int k, int* count, int* result) {
    if (root == NULL || *count >= k)
        return;

    kthLargestUtil(root->right, k, count, result);

    (*count)++;
    if (*count == k) {
        *result = root->data;
        return;
    }

    kthLargestUtil(root->left, k, count, result);
}

int countNodes(struct Node* root) {
    if (root == NULL)
        return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
}

int main() {
    int n, k, value;
```

```
    scanf("%d", &n);

    struct Node* root = NULL;

    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    scanf("%d", &k);

    int totalNodes = countNodes(root);

    if (k > totalNodes || k <= 0) {
        printf("Invalid value of k\n");
    } else {
        int count = 0, result = -1;
        kthLargestUtil(root, k, &count, &result);
        printf("%d\n", result);
    }

    return 0;
}
```

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

3. Problem Statement

John is building a system to store and manage integers using a binary search tree (BST). He needs to add a feature that allows users to search for a specific integer key in the BST using recursion.

Implement functions to create the BST and perform a recursive search for an integer.

Input Format

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

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

The third line consists of an integer representing, the key to be searched.

Output Format

The output prints whether the given key is present in the binary search tree or not.

Refer to the sample output for the exact format.

Sample Test Case

Input: 7
10 5 15 3 7 12 20
12
Output: The key 12 is found in the binary search tree

Answer

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

struct Node {
    int data;
    struct Node *left, *right;
};


struct Node* newNode(int value) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = value;
    node->left = node->right = NULL;
    return node;
}


struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return newNode(value);

    if (value < root->data)
```

```
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}


int searchBST(struct Node* root, int key) {
    if (root == NULL)
        return 0;
    if (root->data == key)
        return 1;
    if (key < root->data)
        return searchBST(root->left, key);
    else
        return searchBST(root->right, key);
}

int main() {
    int n, key, value;
    scanf("%d", &n);

    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    scanf("%d", &key);

    if (searchBST(root, key))
        printf("The key %d is found in the binary search tree\n", key);
    else
        printf("The key %d is not found in the binary search tree\n", key);

    return 0;
}
```

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