Rajalakshmi Engineering College

Name: Ritesh Sivakumar
Email: 240701427@rajalakshmi.edu.in
Roll no: 240701427
Phone: 9342061449
Branch: REC
Department: I CSE FD
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

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int value;
    struct Node* left;
    struct Node* right;
} Node;

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

Node* insertNode(Node* root, int value) {
    if (!root)
    return createNode(value);
    if (value < root->value)
        root->left = insertNode(root->left, value);
    else
        root->right = insertNode(root->right, value);
    return root;
```

```
}
void addValue(Node* root, int add) {
    if (root) {
        root->value += add;
        addValue(root->left, add);
        addValue(root->right, add);
    }
}

int findMax(Node* root) {
    while (root->right)
        root = root->right;
    return root->value;
}

int main() {
    int N, add, value;
    Node* root = NULL;

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

    scanf("%d", &add);
    addValue(root, add);

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

    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

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int value;
    struct Node* left;
    struct Node* right;
} Node;

Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->value = value;
```

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

Node* insertNode(Node* root, int value) {
    if (!root)
    return createNode(value);
    if (value < root->value)
        root->left = insertNode(root->left, value);
    else
        root->right = insertNode(root->right, value);
    return root;
}

void reverseInorder(Node* root, int* k, int* result) {
    if (!root || *k <= 0) return;
    reverseInorder(root->right, k, result);
    if (--(*k) == 0) {
        *result = root->value;
        return;
    }
    reverseInorder(root->left, k, result);
}

int findKthLargest(Node* root, int k, int totalNodes) {
    if (k > totalNodes || k <= 0) {
        return -1;
    }
    int result = -1;
    reverseInorder(root, &k, &result);
    return result;
}

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

int main() {
    int n, k, value;
    Node* root = NULL;
```

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

    scanf("%d", &k);

    int totalNodes = countNodes(root);
    int result = findKthLargest(root, k, totalNodes);

    if (result == -1) {
        printf("Invalid value of k\n");
    } else {
        printf("%d\n", result);
    }

    return 0;
}
```

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

3. Problem Statement

Kishore is studying data structures, and he is currently working on implementing a binary search tree (BST) and exploring its basic operations. He wants to practice creating a BST, inserting elements into it, and performing a specific operation, which is deleting the minimum element from the tree.

Write a program to help him perform the delete operation.

Input Format

The first line of input consists of an integer N, representing the number of elements Kishore wants to insert into the BST.

The second line consists of N space-separated integers, where each integer

represents an element to be inserted into the BST.

Output Format

The output prints the remaining elements of the BST in ascending order (in-order traversal) after deleting the minimum element.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 6
5 3 8 2 4 6
Output: 3 4 5 6 8

Answer

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int value;
    struct Node* left;
    struct Node* right;
} Node;
Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->value = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
Node* insertNode(Node* root, int value) {
    if (!root) return createNode(value);
    if (value < root->value)
        root->left = insertNode(root->left, value);
    else
        root->right = insertNode(root->right, value);
    return root;
}
Node* findMin(Node* root) {
    while (root && root->left)
        root = root->left;
    return root;
```

```
}
Node* deleteMin(Node* root) {
    if (!root) return NULL;
    if (!root->left) {
        Node* temp = root->right;
        free(root);
        return temp;
    }
    root->left = deleteMin(root->left);
    return root;
}
void inorderTraversal(Node* root) {
    if (root) {
        inorderTraversal(root->left);
        printf("%d ", root->value);
        inorderTraversal(root->right);
    }
}
int main() {
    int N, value;
    Node* root = NULL;
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        root = insertNode(root, value);
    }
    root = deleteMin(root);
    inorderTraversal(root);
    printf("\n");
    return 0;
}
```

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