Rajalakshmi Engineering College

Name: Sherwin G M
Email: 240701496@rajalakshmi.edu.in
Roll no: 240701496
Phone: 7708605966
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

You are given a series of magic levels (integers) and need to construct a Binary Search Tree (BST) from them. After constructing the BST, your task is to perform a range search, which involves finding and printing all the magic levels within a specified range [L, R].

Input Format

The first line of input consists of an integer N, the number of magic levels to insert into the BST.

The second line consists of N space-separated integers, representing the magic levels to insert.

The third line consists of two integers, L and R, which define the range for the search.

Output Format

The output prints all the magic levels within the range [L, R] in ascending order, separated by spaces.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 3 7
2 20
Output: 3 5 7 10 15

Answer

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


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

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

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

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
```

```
    return root;
}

void rangeSearch(struct Node* root, int L, int R) {
    if (root == NULL) return;

    if (root->data > L)
        rangeSearch(root->left, L, R);

    if (root->data >= L && root->data <= R)
        printf("%d ", root->data);

    if (root->data < R)
        rangeSearch(root->right, L, R);
}

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

    struct Node* root = NULL;
    int value;

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

    int L, R;
    scanf("%d %d", &L, &R);

    rangeSearch(root, L, R);
    printf("\n");

    return 0;
}
```

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

2. Problem Statement

Emily is studying binary search trees (BST). She wants to write a program that inserts characters into a BST and then finds and prints the minimum and maximum values.

Guide her with the program.

Input Format

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

The second line consists of N space-separated characters.

Output Format

The first line of output prints "Minimum value: " followed by the minimum value of the given inputs.

The second line prints "Maximum value: " followed by the maximum value of the given inputs.

Refer to the sample outputs for formatting specifications.

Sample Test Case

Input: 5
Z E W T Y
Output: Minimum value: E
Maximum value: Z

Answer

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

struct TreeNode {
    char data;
    struct TreeNode* left;
    struct TreeNode* right;
};
```

```
struct TreeNode* createNode(char data) {
    struct TreeNode* newNode = (struct TreeNode*) malloc(sizeof(struct
TreeNode));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}


struct TreeNode* insert(struct TreeNode* root, char data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
    return root;
}


char findMin(struct TreeNode* root) {
    while (root->left != NULL)
        root = root->left;
    return root->data;
}

char findMax(struct TreeNode* root) {
    while (root->right != NULL)
        root = root->right;
    return root->data;
}

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

    struct TreeNode* root = NULL;
    char val;
```

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

    printf("Minimum value: %c\n", findMin(root));
    printf("Maximum value: %c\n", findMax(root));

    return 0;
}
```

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

3. 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 TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

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


struct TreeNode* insert(struct TreeNode* root, int value) {
    if (root == NULL) return createNode(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

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

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

    (*count)++;
    if (*count == k) {
```

```
        *result = root->data;
        return;
    }

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


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

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

    struct TreeNode* root = NULL;
    int value;
    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;
        kthLargest(root, k, &count, &result);
        printf("%d\n", result);
    }

    return 0;
}
```

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