Rajalakshmi Engineering College

Name: SUBASH R
Email: 240701538@rajalakshmi.edu.in
Roll no: 240701538
Phone: 9150202710
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

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 key;
   struct Node *left, *right;
} Node;

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

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

```
Node* deleteMin(Node* root) {
    if (root == NULL) return NULL;
    if (root->left == NULL) {
        Node* rightChild = root->right;
        free(root);
        return rightChild;
    }
    root->left = deleteMin(root->left);
    return root;
}

void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->key);
    inorder(root->right);
}

int main() {
    int N;
    scanf("%d", &N);
    int val;
    Node* root = NULL;

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

    root = deleteMin(root);
    inorder(root);
    printf("\n");

    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>

typedef struct Node {
   int key;
   struct Node *left, *right;
} Node;

Node* newNode(int key) {
   Node* node = (Node*)malloc(sizeof(Node));
```

```
    node->key = key;
    node->left = node->right = NULL;
    return node;
}

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

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

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

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

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

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

int main() {
    int n, k;
    scanf("%d", &n);
    int val;
    Node* root = NULL;

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

    scanf("%d", &k);

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

    return 0;
}
```

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

3. 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>

typedef struct Node {
   int key;
   struct Node *left, *right;
} Node;

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

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

void rangeSearch(Node* root, int L, int R) {
   if (root == NULL) return;
   if (root->key > L)
      rangeSearch(root->left, L, R);
   if (root->key >= L && root->key <= R)
```

```
        printf("%d ", root->key);
    if (root->key < R)
        rangeSearch(root->right, L, R);
}

int main() {
    int N, val, L, R;
    scanf("%d", &N);
    Node* root = NULL;

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

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

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

    return 0;
}
```

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