Rajalakshmi Engineering College

Name: Pranav Narayanan
Email: 240701393@rajalakshmi.edu.in
Roll no:
Phone: 9500579427
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 37.5

Section 1 : Coding

1.   Problem Statement

Aishu is participating in a coding challenge where she needs to reconstruct a Binary Search Tree (BST) from given preorder traversal data and then print the in-order traversal of the reconstructed BST.

Since Aishu is just learning about tree data structures, she needs your help to write a program that does this efficiently.

Input Format

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

The second line of input contains n integers separated by spaces, which represent the preorder traversal of the BST.

Output Format

The output displays n space-separated integers, representing the in-order traversal of the reconstructed BST.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 6
10 5 1 7 40 50

Output: 1 5 7 10 40 50

Answer

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

// Define a structure for a tree node
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function to create a new node
Node* newNode(int value) {
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = value;
    temp->left = temp->right = NULL;
    return temp;
}

// Function to construct BST from preorder
Node* buildBST(int preorder[], int* index, int min, int max, int n) {
    if (*index >= n) return NULL;

    int val = preorder[*index];
    if (val < min || val > max) return NULL;
```

```c
    Node* root = newNode(val);
    (*index)++;

    root->left = buildBST(preorder, index, min, val - 1, n);
    root->right = buildBST(preorder, index, val + 1, max, n);

    return root;
}

// Function to print inorder traversal
void inorderTraversal(Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

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

    int preorder[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &preorder[i]);
    }

    int index = 0;
    Node* root = buildBST(preorder, &index, INT_MIN, INT_MAX, n);

    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

Status : Correct   *Marks : 10/10*

2. Problem Statement

Arun is exploring operations on binary search trees (BST). He wants to

write a program with an unsorted distinct integer array that represents the BST keys and construct a height-balanced BST from it.

After constructing, he wants to perform the following operations that can alter the structure of the tree and traverse them using a level-order traversal:

InsertionDeletion

Your task is to assist Arun in completing the program without any errors.

Input Format

The first line of input consists of an integer N, representing the number of initial keys in the BST.

The second line consists of N space-separated integers, representing the initial keys.

The third line consists of an integer X, representing the new key to be inserted into the BST.

The fourth line consists of an integer Y, representing the key to be deleted from the BST.

Output Format

The first line of output prints "Initial BST: " followed by a space-separated list of keys in the initial BST after constructing it in level order traversal.

The second line prints "BST after inserting a new node X: " followed by a space-separated list of keys in the BST after inserting X n level order traversal.

The third line prints "BST after deleting node Y: " followed by a space-separated list of keys in the BST after deleting Y n level order traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
25 14 56 28 12

34
12
Output: Initial BST: 25 14 56 12 28
BST after inserting a new node 34: 25 14 56 12 28 34
BST after deleting node 12: 25 14 56 28 34

Answer

-

Status : Skipped   *Marks : 0/10*

3.   Problem Statement

Yogi is working on a program to manage a binary search tree (BST) containing integer values. He wants to implement a function that removes nodes from the tree that fall outside a specified range defined by a minimum and maximum value.

Help Yogi by writing a function that achieves this.

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, representing the elements to be inserted into the BST.

The third line consists of two space-separated integers min and max, representing the minimum value and the maximum value of the range.

Output Format

The output prints the remaining elements of the BST in an in-order traversal, after removing nodes that fall outside the specified range.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 20 12
5 15
Output: 5 10 12 15

Answer

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

// Define BST node structure
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function to create a new node
Node* newNode(int value) {
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = value;
    temp->left = temp->right = NULL;
    return temp;
}

// Insert a new node into the BST
Node* insert(Node* root, int value) {
    if (root == NULL)
        return newNode(value);

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

    return root;
}

// In-order traversal of the BST
void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
```

```c
        printf("%d ", root->data);
        inorder(root->right);
}

// Function to trim BST nodes outside the given range
Node* trimBST(Node* root, int min, int max) {
    if (root == NULL)
        return NULL;

    // Recursively trim left and right subtrees
    root->left = trimBST(root->left, min, max);
    root->right = trimBST(root->right, min, max);

    // Remove current node if it's out of range
    if (root->data < min) {
        Node* rightChild = root->right;
        free(root);
        return rightChild;
    }
    if (root->data > max) {
        Node* leftChild = root->left;
        free(root);
        return leftChild;
    }

    // Node is within range
    return root;
}

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

    int value;
    Node* root = NULL;

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

    int min, max;
```

```
    scanf("%d %d", &min, &max);

    // Trim the BST
    root = trimBST(root, min, max);

    // Print in-order traversal of the trimmed tree
    inorder(root);
    printf("\n");

    return 0;
}
```

Status : Partially correct   *Marks : 7.5/10*

4. Problem Statement

Joseph, a computer science student, is interested in understanding binary
search trees (BST) and their node arrangements. He wants to create a
program to explore BSTs by inserting elements into a tree and displaying
the nodes using post-order traversal of the tree.

Write a program to help Joseph implement the program.

Input Format

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

The second line consists of N space-separated integers data, which is the data
to be inserted into the BST.

Output Format

The output prints N space-separated integer values after the post-order
traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
10 15 5 3

Output: 3 5 15 10

Answer

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

// Define the structure of a BST node
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function to create a new node
Node* newNode(int value) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = value;
    node->left = node->right = NULL;
    return node;
}

// Function to insert a value into the BST
Node* insert(Node* root, int value) {
    if (root == NULL) return newNode(value);

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

    return root;
}

// Function to perform post-order traversal
void postOrder(Node* root) {
    if (root == NULL) return;

    postOrder(root->left);
    postOrder(root->right);
```

```
    printf("%d ", root->data);
}

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

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

    postOrder(root);
    printf("\n");

    return 0;
}
```

Status : Correct   *Marks : 10/10*

5. Problem Statement

Viha, a software developer, is working on a project to automate searching
for a target value in a Binary Search Tree (BST). She needs to create a
program that takes an integer target value as input and determines if that
value is present in the BST or not.

Write a program to assist Viha.

Input Format

The first line of input consists of integers separated by spaces, which represent
the elements to be inserted into the BST. The input is terminated by entering -1.

The second line consists of an integer target, which represents the target value
to be searched in the BST.

Output Format

If the target value is found in the BST, print "[target] is found in the BST".

Else, print "[target] is not found in the BST"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5 3 7 1 4 6 8 -1
4

Output: 4 is found in the BST

Answer

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

// Define the structure for a BST node
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function to create a new BST node
Node* newNode(int value) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = value;
    node->left = node->right = NULL;
    return node;
}

// Function to insert a value into the BST
Node* insert(Node* root, int value) {
    if (root == NULL) return newNode(value);

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

```c
    return root;
}

// Function to search for a target value in the BST
int search(Node* root, int target) {
    if (root == NULL) return 0;
    if (root->data == target) return 1;

    if (target < root->data)
        return search(root->left, target);
    else
        return search(root->right, target);
}

int main() {
    Node* root = NULL;
    int val;

    // Read BST elements until -1 is encountered
    while (1) {
        scanf("%d", &val);
        if (val == -1) break;
        root = insert(root, val);
    }

    // Read target value to search
    int target;
    scanf("%d", &target);

    // Search in BST and output result
    if (search(root, target))
        printf("%d is found in the BST\n", target);
    else
        printf("%d is not found in the BST\n", target);

    return 0;
}
```

Status : Correct   *Marks : 10/10*