Rajalakshmi Engineering College

Name: SHIVANISREE K B
Email: 240701501@rajalakshmi.edu.in
Roll no: 240701501
Phone: 7358464804
Branch: REC
Department: I CSE FE
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 : 47.5

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>

// Definition for a binary tree node
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Function to create a new tree node
struct TreeNode* newNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
    return node;
}

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

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

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

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

    return root;
}

// Function for inorder traversal
void inorder(struct TreeNode* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->val);
    inorder(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;
    struct TreeNode* root = buildBST(preorder, &index, n, INT_MIN, INT_MAX);

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

    return 0;
}
```

Status : Correct   *Marks : 10/10*

2.   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

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

// Definition for a binary tree node
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
```

```c
};
// Function to create a new node
struct TreeNode* newNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
    return node;
}

// Function to insert a value in BST
struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL) return newNode(val);

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

    return root;
}

// Function to trim the BST
struct TreeNode* trimBST(struct TreeNode* root, int min, int max) {
    if (root == NULL) return NULL;

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

    // If current node is out of range, remove it
    if (root->val < min) {
        struct TreeNode* rightChild = root->right;
        free(root);
        return rightChild;
    }
    if (root->val > max) {
        struct TreeNode* leftChild = root->left;
        free(root);
        return leftChild;
    }
}
```

```c
        return root;
    }

    // Function for in-order traversal
    void inorder(struct TreeNode* root) {
        if (root == NULL) return;
        inorder(root->left);
        printf("%d ", root->val);
        inorder(root->right);
    }

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

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

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

        // Build BST
        struct TreeNode* root = NULL;
        for (int i = 0; i < n; i++) {
            root = insert(root, arr[i]);
        }

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

        // Print inorder traversal
        inorder(root);
        printf("\n");

        return 0;
    }
```

Status : Partially correct   *Marks : 7.5/10*

3. 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

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

// Definition for a binary tree node
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
```

```c
};
// Function to create a new node
struct TreeNode* newNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
    return node;
}

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

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

    return root;
}

// Function for post-order traversal
void postorder(struct TreeNode* root) {
    if (root == NULL)
        return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->val);
}

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

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

    // Build BST
```

```
    struct TreeNode* root = NULL;
    for (int i = 0; i < n; i++) {
        root = insert(root, arr[i]);
    }

    // Post-order traversal
    postorder(root);
    printf("\n");

    return 0;
}
```

Status : Correct   *Marks : 10/10*

4.   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.

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

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

// Define BST node structure
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Create a new BST node
struct TreeNode* newNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
```

```c
        return node;
    }

    // Insert into BST
    struct TreeNode* insert(struct TreeNode* root, int val) {
        if (root == NULL)
            return newNode(val);

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

        return root;
    }

    // Find minimum node in BST (used in deletion)
    struct TreeNode* findMin(struct TreeNode* root) {
        while (root->left != NULL)
            root = root->left;
        return root;
    }

    // Delete node from BST
    struct TreeNode* deleteNode(struct TreeNode* root, int val) {
        if (root == NULL)
            return NULL;

        if (val < root->val)
            root->left = deleteNode(root->left, val);
        else if (val > root->val)
            root->right = deleteNode(root->right, val);
        else {
            // Node found
            if (root->left == NULL) {
                struct TreeNode* temp = root->right;
                free(root);
                return temp;
            }
            else if (root->right == NULL) {
                struct TreeNode* temp = root->left;
                free(root);
```

```c
            return temp;
        }
        else {
            struct TreeNode* temp = findMin(root->right);
            root->val = temp->val;
            root->right = deleteNode(root->right, temp->val);
        }
    }
    return root;
}

// Level-order traversal
void levelOrder(struct TreeNode* root) {
    if (root == NULL)
        return;

    struct TreeNode* queue[100];
    int front = 0, rear = 0;
    queue[rear++] = root;

    while (front < rear) {
        struct TreeNode* current = queue[front++];
        printf("%d ", current->val);

        if (current->left != NULL)
            queue[rear++] = current->left;
        if (current->right != NULL)
            queue[rear++] = current->right;
    }
    printf("\n");
}

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

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

    int x, y;
    scanf("%d", &x);
```

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

    struct TreeNode* root = NULL;

    // Build initial BST
    for (int i = 0; i < n; i++) {
        root = insert(root, keys[i]);
    }

    printf("Initial BST: ");
    levelOrder(root);

    // Insert node x
    root = insert(root, x);
    printf("BST after inserting a new node %d: ", x);
    levelOrder(root);

    // Delete node y
    root = deleteNode(root, y);
    printf("BST after deleting node %d: ", y);
    levelOrder(root);

    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>

// Definition for a binary tree node
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};

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

// Function to insert a value into BST
struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL)
        return newNode(val);
```

```c
    if (val < root->val)
        root->left = insert(root->left, val);
    else if (val > root->val)
        root->right = insert(root->right, val);

    return root;
}

// Function to search target in BST
int search(struct TreeNode* root, int target) {
    if (root == NULL)
        return 0;
    if (root->val == target)
        return 1;
    else if (target < root->val)
        return search(root->left, target);
    else
        return search(root->right, target);
}

int main() {
    struct TreeNode* root = NULL;
    int val;

    // Read until -1
    while (scanf("%d", &val) && val != -1) {
        root = insert(root, val);
    }

    int target;
    scanf("%d", &target);

    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*