Rajalakshmi Engineering College

Name: Pavithra J
Email: 240701381@rajalakshmi.edu.in
Roll no: 240701381
Phone: 9363364978
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_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

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

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

// Create a new node
struct TreeNode* createNode(int data) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insert a node into BST
struct TreeNode* insert(struct TreeNode* root, int data) {
    if (root == NULL)
        return createNode(data);

    if (data < root->data)
```

240701381

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

    return root;
}

// In-order traversal (Left, Root, Right)
void inorder(struct TreeNode* root) {
    if (root == NULL)
        return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

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

    int val;
    struct TreeNode* root = NULL;

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

    inorder(root);  // Print in-order traversal

    return 0;
}
```

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

2. 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

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

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

// Create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
```

240701381

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

// Insert into BST
struct Node* insert(struct Node* root, int 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;
}

// Search the BST for a target
int search(struct Node* root, int target) {
    if (root == NULL)
        return 0;

    if (target == root->data)
        return 1;
    else if (target < root->data)
        return search(root->left, target);
    else
        return search(root->right, target);
}

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

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

240701381

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

    if (search(root, target))
        printf("%d is found in the BST", target);
    else
        printf("%d is not found in the BST", target);

    return 0;
}
```

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

3. 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

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

#define MAX 100

// Define structure for tree node
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Create new node
struct TreeNode* createNode(int value) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = value;
    node->left = node->right = NULL;
```

```
    return node;
}

// Insert value into BST
struct TreeNode* insert(struct TreeNode* root, int key) {
    if (root == NULL)
        return createNode(key);
    if (key < root->data)
        root->left = insert(root->left, key);
    else if (key > root->data)
        root->right = insert(root->right, key);
    return root;
}

// Find minimum value node
struct TreeNode* findMin(struct TreeNode* root) {
    while (root && root->left)
        root = root->left;
    return root;
}

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

    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        // Found the node to delete
        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);
            return temp;
        } else {
            // Node with two children
```

```
            struct TreeNode* temp = findMin(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }
    return root;
}

// Define queue for level-order traversal
struct Queue {
    struct TreeNode* items[MAX];
    int front, rear;
};

// Initialize queue
void initQueue(struct Queue* q) {
    q->front = q->rear = -1;
}

// Enqueue
void enqueue(struct Queue* q, struct TreeNode* node) {
    if (q->rear < MAX - 1) {
        q->items[++q->rear] = node;
        if (q->front == -1)
            q->front = 0;
    }
}

// Dequeue
struct TreeNode* dequeue(struct Queue* q) {
    if (q->front == -1 || q->front > q->rear)
        return NULL;
    return q->items[q->front++];
}

// Check if queue is empty
int isEmpty(struct Queue* q) {
    return q->front == -1 || q->front > q->rear;
}

// Level order traversal
void levelOrder(struct TreeNode* root) {
```

```
    struct Queue q;
    initQueue(&q);

    if (root)
        enqueue(&q, root);

    while (!isEmpty(&q)) {
        struct TreeNode* current = dequeue(&q);
        printf("%d ", current->data);

        if (current->left)
            enqueue(&q, current->left);
        if (current->right)
            enqueue(&q, current->right);
    }
}

int main() {
    int N, i, val, insertVal, deleteVal;
    struct TreeNode* root = NULL;

    // Input number of nodes
    scanf("%d", &N);

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

    // Input values for insertion and deletion
    scanf("%d", &insertVal);
    scanf("%d", &deleteVal);

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

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

240701381

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

    return 0;
}
```

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

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

```
// 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* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = value;
    node->left = node->right = NULL;
    return node;
}

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 postOrder(struct Node* root) {
    if (root == NULL)
        return;

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

240701381

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

    scanf("%d", &N);

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

    postOrder(root);

    return 0;
}
```

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

5. 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

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


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


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


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;
}


struct Node* removeOutsideRange(struct Node* root, int min, int max) {
    if (root == NULL) {
        return NULL;
    }

    root->left = removeOutsideRange(root->left, min, max);
    root->right = removeOutsideRange(root->right, min, max);


    if (root->data < min) {
        struct Node* temp = root->right;
        free(root);
        return temp;
    } else if (root->data > max) {
        struct Node* temp = root->left;
        free(root);
        return temp;
    }

    return root;
}


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

240701381

```
int main() {
    int N, value, min, max;
    struct Node* root = NULL;


    scanf("%d", &N);


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


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

    root = removeOutsideRange(root, min, max);


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

    return 0;
}
```

Status : Partially correct **Marks : 7.5/10**

240701381