Rajalakshmi Engineering College

Name: KEERTHI PRIYA T
Email: 240701258@rajalakshmi.edu.in
Roll no: 2116240701258
Phone: 7397397221
Branch: REC
Department: I CSE FC
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_PAH_Updated

Attempt : 2
Total Mark : 50
Marks Obtained : 37.5

Section 1 : Coding

1. 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>

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

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

// Insert into BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);
```

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

    // Remove nodes outside the [min, max] range
    struct Node* trimBST(struct Node* root, int min, int max) {
        if (root == NULL)
            return NULL;

        // First fix the left and right subtrees
        root->left = trimBST(root->left, min, max);
        root->right = trimBST(root->right, min, max);

        // Now fix the current node
        if (root->data < min) {
            struct Node* rightChild = root->right;
            free(root);
            return rightChild;
        }
        if (root->data > max) {
            struct Node* leftChild = root->left;
            free(root);
            return leftChild;
        }

        return root;
    }

    int main() {
        int n;
```

```c
    scanf("%d", &n);

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

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

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

    // Print remaining BST in in-order traversal
    inorder(root);
    printf("\n");

    return 0;
}
```

Status : Partially correct   *Marks : 7.5/10*

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

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

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

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

// Insert a node into BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
```

```c
    else if (value > root->data)
        root->right = insert(root->right, value);
    return root;
}

// Search for a target value in BST
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 value;

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

    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*

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

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

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

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

// Function to insert a value into BST
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;
}

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

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

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

    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        root = insert(root, value);
    }
```

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

    return 0;
}
```

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

4. 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 structure for BST node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

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

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

    struct Node* root = newNode(key);
    (*index)++;

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

    return root;
}

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

```c
        inorder(root->left);
        printf("%d ", root->data);
        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 Node* root = buildBST(preorder, &index, n, INT_MIN, INT_MAX);

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

    return 0;
}
```

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

5. 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
```

```c
// BST Node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Queue for level-order traversal
struct Queue {
    struct Node* data[MAX];
    int front, rear;
};

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

int isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, struct Node* node) {
    if (q->rear == MAX - 1) return;
    if (isEmpty(q)) q->front = 0;
    q->data[++q->rear] = node;
}

struct Node* dequeue(struct Queue* q) {
    if (isEmpty(q)) return NULL;
    struct Node* temp = q->data[q->front];
    if (q->front == q->rear)
        q->front = q->rear = -1;
    else
        q->front++;
    return temp;
}

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

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

// Sort comparator for qsort
int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

// Build balanced BST from sorted array
struct Node* buildBalancedBST(int arr[], int start, int end) {
    if (start > end)
        return NULL;

    int mid = (start + end) / 2;
    struct Node* root = createNode(arr[mid]);

    root->left = buildBalancedBST(arr, start, mid - 1);
    root->right = buildBalancedBST(arr, mid + 1, end);

    return root;
}

// Insert into BST
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;
}

// Find min value node (for deletion)
struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
```

```c
// Delete a node in BST
struct Node* deleteNode(struct Node* 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 {
        // Node with one or no child
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        // Node with two children
        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

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

    struct Queue q;
    initQueue(&q);
    enqueue(&q, root);

    while (!isEmpty(&q)) {
        struct Node* temp = dequeue(&q);
        printf("%d ", temp->data);
        if (temp->left)
            enqueue(&q, temp->left);
        if (temp->right)
```

```c
            enqueue(&q, temp->right);
        }
    }

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

        int arr[20];
        for (int i = 0; i < n; i++)
            scanf("%d", &arr[i]);

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

        // Sort to build balanced BST
        qsort(arr, n, sizeof(int), compare);
        struct Node* root = buildBalancedBST(arr, 0, n - 1);

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

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

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

        return 0;
    }
```

Status : Wrong   *Marks : 0/10*