Rajalakshmi Engineering College

Name: Priyan S
Email: 240701402@rajalakshmi.edu.in
Roll no: 240701402
Phone: 9150170939
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 : 50

Section 1 : Coding

1. 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>

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

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

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 {
        root->right = insert(root->right, value);
    }
```

```c
    return root;
}

void postOrderTraversal(struct Node* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(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);
    }

    postOrderTraversal(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

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

```c
    }
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 {
        root->right = insert(root->right, value);
    }
    return root;
}

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

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

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

    return root;
}

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

```
}
int main() {
    int n, min, max;
    scanf("%d", &n);

    struct Node* root = NULL;

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

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

    root = trimBST(root, min, max);

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

    return 0;
}
```

Status : Correct   *Marks : 10/10*

3. 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 a BST Node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// 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 node into the 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 {
        root->right = insert(root->right, value);
    }
    return root;
}

// Function to search for a target in the BST
int searchBST(struct Node* root, int target) {
    if (root == NULL) {
        return 0;
    }
    if (root->data == target) {
        return 1;
    } else if (target < root->data) {
        return searchBST(root->left, target);
    } else {
        return searchBST(root->right, target);
    }
}

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

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

    // Read the target value
    scanf("%d", &target);

    // Search for the target in the BST and print the result
    if (searchBST(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*

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>

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

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

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 {
        root->right = insert(root->right, value);
    }
    return root;
}

struct Node* constructBST(int preorder[], int n) {
    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        root = insert(root, preorder[i]);
    }
    return root;
}

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

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

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

      struct Node* root = constructBST(preorder, n);

      inorderTraversal(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

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

typedef struct TreeNode {
    int val;
    struct TreeNode *left, *right;
} TreeNode;

TreeNode* createNode(int key) {
    TreeNode* node = (TreeNode*)malloc(sizeof(TreeNode));
```

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

    TreeNode* insertIntoBST(TreeNode* root, int key) {
        if (!root) return createNode(key);
        if (key < root->val) root->left = insertIntoBST(root->left, key);
        else root->right = insertIntoBST(root->right, key);
        return root;
    }

    TreeNode* findMin(TreeNode* root) {
        while (root->left) root = root->left;
        return root;
    }

    TreeNode* deleteFromBST(TreeNode* root, int key) {
        if (!root) return NULL;
        if (key < root->val) root->left = deleteFromBST(root->left, key);
        else if (key > root->val) root->right = deleteFromBST(root->right, key);
        else {
            if (!root->left) {
                TreeNode* temp = root->right;
                free(root);
                return temp;
            } else if (!root->right) {
                TreeNode* temp = root->left;
                free(root);
                return temp;
            }
            TreeNode* temp = findMin(root->right);
            root->val = temp->val;
            root->right = deleteFromBST(root->right, temp->val);
        }
        return root;
    }

    void levelOrderTraversal(TreeNode* root) {
        if (!root) return;
        TreeNode* queue[100];
        int front = 0, rear = 0;
```

```c
        queue[rear++] = root;
        while (front < rear) {
            TreeNode* node = queue[front++];
            printf("%d ", node->val);
            if (node->left) queue[rear++] = node->left;
            if (node->right) queue[rear++] = node->right;
        }
        printf("\n");
    }

    int main() {
        int N, X, Y;
        scanf("%d", &N);
        int keys[N];
        for (int i = 0; i < N; i++) scanf("%d", &keys[i]);
        scanf("%d %d", &X, &Y);

        TreeNode* root = NULL;
        for (int i = 0; i < N; i++) {
            root = insertIntoBST(root, keys[i]);
        }

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

        root = insertIntoBST(root, X);
        printf("BST after inserting a new node %d: ", X);
        levelOrderTraversal(root);

        root = deleteFromBST(root, Y);
        printf("BST after deleting node %d: ", Y);
        levelOrderTraversal(root);

        return 0;
    }
```

Status : Correct   *Marks : 10/10*