Rajalakshmi Engineering College

Name: SUBASH R
Email: 240701538@rajalakshmi.edu.in
Roll no: 240701538
Phone: 9150202710
Branch: REC
Department: I CSE FE
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 : 50

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

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

struct node {
   int key;
   struct node* left;
   struct node* right;
};

struct node* newNode(int num) {
   struct node* temp = (struct node*)malloc(sizeof(struct node));
   temp->key = num;
   temp->left = temp->right = NULL;
   return temp;
}

struct node* insert(struct node* root, int key) {
   if (root == NULL)
      return newNode(key);
   if (root->key > key)
      root->left = insert(root->left, key);
   else
      root->right = insert(root->right, key);
```

```
    return root;
}

struct node* removeRange(struct node* root, int min, int max) {
    if (root == NULL)
        return NULL;
    root->left = removeRange(root->left, min, max);
    root->right = removeRange(root->right, min, max);
    if (root->key < min) {
        struct node* rChild = root->right;
        free(root);
        return rChild;
    }
    if (root->key > max) {
        struct node* lChild = root->left;
        free(root);
        return lChild;
    }
    return root;
}

void inorder(struct node* root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

int main() {
    struct node* root = NULL;
    int num, min, max;
    scanf("%d", &num);
    for (int i = 0; i < num; i++) {
        int key;
        scanf("%d", &key);
        root = insert(root, key);
    }
    scanf("%d", &min);
    scanf("%d", &max);
    root = removeRange(root, min, max);
    inorder(root);
```

```
    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

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

```
typedef struct node {
    int element;
    struct node* left;
    struct node* right;
} node;

node* createNode(int e) {
    node* newNode = (node*)malloc(sizeof(node));
    newNode->element = e;
    newNode->left = newNode->right = NULL;
    return newNode;
}

node* insert(node* tree, int e) {
    if (tree == NULL)
        return createNode(e);
    else if (e < tree->element)
        tree->left = insert(tree->left, e);
    else if (e > tree->element)
        tree->right = insert(tree->right, e);
    return tree;
}

int search(node* tree, int x) {
    if (tree == NULL)
        return 0;
    else if (x == tree->element)
        return 1;
    else if (x < tree->element)
        return search(tree->left, x);
    else
        return search(tree->right, x);
}

int main() {
    node* tree = NULL;
    int e;
    while (1) {
        scanf("%d", &e);
        if (e == -1)
            break;
```

```
    tree = insert(tree, e);
  }
  int x;
  scanf("%d", &x);
  if (search(tree, x))
    printf("%d is found in the BST\n", x);
  else
    printf("%d is not found in the BST\n", x);

  return 0;
}
```

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

3. 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

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

typedef struct node {
   int element;
   struct node* left;
   struct node* right;
} node;

node* createNode(int e) {
   node* newNode = (node*)malloc(sizeof(node));
   newNode->element = e;
   newNode->left = newNode->right = NULL;
   return newNode;
}

node* insert(node* tree, int e) {
   if (tree == NULL)
      return createNode(e);
   else if (e < tree->element)
      tree->left = insert(tree->left, e);
   else
      tree->right = insert(tree->right, e);
   return tree;
}

void inorder(node* tree) {
   if (tree != NULL) {
      inorder(tree->left);
      printf("%d ", tree->element);
      inorder(tree->right);
   }
}

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

```
    int e;
    node* tree = NULL;
    for (int i = 0; i < n; i++) {
        scanf("%d", &e);
        tree = insert(tree, e);
    }
    inorder(tree);
    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.

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

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

typedef struct Node {
   int key;
   struct Node *left, *right;
} Node;

typedef struct QueueNode {
   Node *treeNode;
   struct QueueNode *next;
} QueueNode;

typedef struct {
   QueueNode *front, *rear;
} Queue;

Node* newNode(int key) {
```

```
    Node* node = (Node*)malloc(sizeof(Node));
    node->key = key;
    node->left = node->right = NULL;
    return node;
}

Queue* createQueue() {
    Queue *q = (Queue*)malloc(sizeof(Queue));
    q->front = q->rear = NULL;
    return q;
}

void enqueue(Queue *q, Node *node) {
    QueueNode *temp = (QueueNode*)malloc(sizeof(QueueNode));
    temp->treeNode = node;
    temp->next = NULL;
    if (!q->rear) {
        q->front = q->rear = temp;
    } else {
        q->rear->next = temp;
        q->rear = temp;
    }
}

Node* dequeue(Queue *q) {
    if (!q->front) return NULL;
    QueueNode *temp = q->front;
    Node *node = temp->treeNode;
    q->front = q->front->next;
    if (!q->front) q->rear = NULL;
    free(temp);
    return node;
}

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

Node* insert(Node* root, int key) {
    if (!root) return newNode(key);
    if (key < root->key)
        root->left = insert(root->left, key);
```

```
    else if (key > root->key)
        root->right = insert(root->right, key);
    return root;
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current && current->left)
        current = current->left;
    return current;
}

Node* deleteNode(Node* root, int key) {
    if (!root) return root;
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else {
        if (!root->left) {
            Node* temp = root->right;
            free(root);
            return temp;
        } else if (!root->right) {
            Node* temp = root->left;
            free(root);
            return temp;
        }
        Node* temp = minValueNode(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}

void levelOrder(Node* root) {
    if (!root) return;
    Queue* q = createQueue();
    enqueue(q, root);
    while (!isEmpty(q)) {
        Node* node = dequeue(q);
        printf("%d ", node->key);
```

```
        if (node->left) enqueue(q, node->left);
        if (node->right) enqueue(q, node->right);
    }
    free(q);
}

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

    scanf("%d", &X);
    scanf("%d", &Y);

    Node* root = NULL;
    for (int i = 0; i < N; i++) {
        root = insert(root, arr[i]);
    }

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

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

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

    return 0;
}
```

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

5. 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

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

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

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

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

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;
}

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

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

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

```
        scanf("%d", &data);
        root = insert(root, data);
    }
    postOrderTraversal(root);
    printf("\n");
    freeTree(root);
    return 0;
}
```

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