Rajalakshmi Engineering College

Name: Vikashini M
Email: 241801314@rajalakshmi.edu.in
Roll no: 241801314
Phone: 9345747519
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_CY_Updated

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

You are given a series of magic levels (integers) and need to construct a Binary Search Tree (BST) from them. After constructing the BST, your task is to perform a range search, which involves finding and printing all the magic levels within a specified range [L, R].

Input Format

The first line of input consists of an integer N, the number of magic levels to insert into the BST.

The second line consists of N space-separated integers, representing the magic levels to insert.

The third line consists of two integers, L and R, which define the range for the search.

Output Format

The output prints all the magic levels within the range [L, R] in ascending order, separated by spaces.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 3 7
2 20
Output: 3 5 7 10 15

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 rangeSearch(struct Node* root, int L, int R) {
   if (root == NULL)
      return;
   if (root->data > L)
      rangeSearch(root->left, L, R);
   if (root->data >= L && root->data <= R)
      printf("%d ", root->data);
   if (root->data < R)
      rangeSearch(root->right, L, R);
}

int main() {
   int N;
   scanf("%d", &N);
   struct Node* root = NULL;
   for (int i = 0; i < N; i++) {
      int val;
      scanf("%d", &val);
      root = insert(root, val);
   }
   int L, R;
   scanf("%d %d", &L, &R);

   rangeSearch(root, L, R);
   printf("\n");
   return 0;
}
```

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

2. Problem Statement

Edward has a Binary Search Tree (BST) and needs to find the k-th largest element in it.

Given the root of the BST and an integer k, help Edward determine the k-th largest element in the tree. If k exceeds the number of nodes in the BST, return an appropriate message.

Input Format

The first line of input consists of integer n, the number of nodes in the BST.

The second line consists of the n elements, separated by space.

The third line consists of the value of k.

Output Format

The output prints the kth largest element in the binary search tree.

For invalid inputs, print "Invalid value of k".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 7
8 4 12 2 6 10 14
1

Output: 14

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 findKthLargest(struct Node* root, int k, int* count, int* result) {
   if (root == NULL || *count >= k)
       return;

   findKthLargest(root->right, k, count, result);

   (*count)++;
   if (*count == k) {
       *result = root->data;
       return;
   }

   findKthLargest(root->left, k, count, result);
}

int countNodes(struct Node* root) {
   if (root == NULL) return 0;
   return 1 + countNodes(root->left) + countNodes(root->right);
}

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

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

   int k;
   scanf("%d", &k);
   int totalNodes = countNodes(root);
   if (k > totalNodes || k <= 0) {
       printf("Invalid value of k\n");
   } else {
       int count = 0, result = -1;
       findKthLargest(root, k, &count, &result);
       printf("%d\n", result);
   }

   return 0;
}
```

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

3. Problem Statement

Dhruv is working on a project where he needs to implement a Binary Search Tree (BST) data structure and perform various operations on it.

He wants to create a program that allows him to build a BST, traverse it in different orders (inorder, preorder, postorder), and exit the program when needed.

Help Dhruv by designing a program that fulfils his requirements.

Input Format

The first input consists of the choice.

If the choice is 1, enter the number of elements N and the elements inserted into the tree, separated by a space in a new line.

If the choice is 2, print the in-order traversal.

If the choice is 3, print the pre-order traversal.

If the choice is 4, print the post-order traversal.

If the choice is 5, exit.

Output Format

The output prints the results based on the choice.

For choice 1, print "BST with N nodes is ready to use" where N is the number of nodes inserted.

For choice 2, print the in-order traversal of the BST.

For choice 3, print the pre-order traversal of the BST.

For choice 4, print the post-order traversal of the BST.

For choice 5, the program exits.

If the choice is greater than 5, print "Wrong choice".

Refer to the sample output for the formatting specifications.

Sample Test Case

```
Input: 1
5
12 78 96 34 55
2
3
4
5
Output: BST with 5 nodes is ready to use
BST Traversal in INORDER
12 34 55 78 96
BST Traversal in PREORDER
12 78 34 55 96
BST Traversal in POSTORDER
55 34 96 78 12
```

Answer

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


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

void postorder(struct Node* root) {
   if (root == NULL) return;
   postorder(root->left);
   postorder(root->right);
```

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

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

    while (scanf("%d", &choice) != EOF) {
        if (choice == 1) {
            int N, val;
            scanf("%d", &N);
            root = NULL;
            for (int i = 0; i < N; i++) {
                scanf("%d", &val);
                root = insert(root, val);
            }
            printf("BST with %d nodes is ready to use ", N);
        }
        else if (choice == 2) {
            printf("BST Traversal in INORDER ");
            inorder(root);
        }
        else if (choice == 3) {
            printf("BST Traversal in PREORDER ");
            preorder(root);
        }
        else if (choice == 4) {
            printf("BST Traversal in POSTORDER ");
            postorder(root);
        }
        else if (choice == 5) {
            break;
        }
        else {
            printf("Wrong choice ");
        }
    }

    return 0;
}
```

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