Rajalakshmi Engineering College

Name: Ravi Sankar
Email: 240701424@rajalakshmi.edu.in
Roll no: 240701424
Phone: 8122932671
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_CY_Updated

Attempt : 2
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 TreeNode {
    int value;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* insert_into_bst(struct TreeNode* root, int value) {
    if (root == NULL) {
        struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
        newNode->value = value;
        newNode->left = NULL;
        newNode->right = NULL;
        return newNode;
    }
    if (value < root->value) {
        root->left = insert_into_bst(root->left, value);
    } else {
        root->right = insert_into_bst(root->right, value);
    }
    return root;
}
```

```
void inorder_range_search(struct TreeNode* root, int L, int R) {
    if (root == NULL) {
        return;
    }
    inorder_range_search(root->left, L, R);
    if (root->value >= L && root->value <= R) {
        printf("%d ", root->value);
    }
    inorder_range_search(root->right, L, R);
}

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

    struct TreeNode* root = NULL;
    for (int i = 0; i < N; ++i) {
        int value;
        scanf("%d", &value);
        root = insert_into_bst(root, value);
    }

    int L, R;
    scanf("%d %d", &L, &R);

    inorder_range_search(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

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

struct TreeNode {
   int value;
   struct TreeNode* left;
   struct TreeNode* right;
};

struct TreeNode* insert_into_bst(struct TreeNode* root, int value) {
   if (root == NULL) {
      struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
```

```
        newNode->value = value;
        newNode->left = NULL;
        newNode->right = NULL;
        return newNode;
    }
    if (value < root->value) {
        root->left = insert_into_bst(root->left, value);
    } else {
        root->right = insert_into_bst(root->right, value);
    }
    return root;
}

void kth_largest_helper(struct TreeNode* root, int k, int* count, int* result) {
    if (root == NULL || *count >= k) {
        return;
    }
    kth_largest_helper(root->right, k, count, result);
    (*count)++;
    if (*count == k) {
        *result = root->value;
        return;
    }
    kth_largest_helper(root->left, k, count, result);
}

int kth_largest(struct TreeNode* root, int k) {
    int count = 0;
    int result = -1;
    kth_largest_helper(root, k, &count, &result);
    return result;
}

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

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

    int k;
    scanf("%d", &k);

    if (k <= 0 || k > n) {
        printf("Invalid value of k\n");
    } else {
        int result = kth_largest(root, k);
        printf("%d\n", result);
    }

    return 0;
}
```

Status : Correct **_Marks : 10/10_**

3. Problem Statement

Jake is learning about binary search trees(BST) and their operations. He wants to implement a program that can delete a node from a BST based on the given key value and print the remaining nodes in an in-order traversal.

Assist Jake in the program.

Input Format

The first line of input consists of an integer n, representing the number of elements in BST.

The second line consists of n space-separated integers, representing the elements of the tree.

The third line consists of an integer x, representing the key value of the node to be deleted.

Output Format

The first line of output prints "Before deletion: " followed by the in-order traversal of the initial BST.

The second line prints "After deletion: " followed by the in-order traversal after the deletion of the key value.

If the key value is not present in the BST, print the original tree as it is.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
8 6 4 3 1
4
Output: Before deletion: 1 3 4 6 8
After deletion: 1 3 6 8

Answer

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

struct TreeNode {
   int value;
   struct TreeNode* left;
   struct TreeNode* right;
};

struct TreeNode* insert_into_bst(struct TreeNode* root, int value) {
   if (root == NULL) {
      struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
      newNode->value = value;
      newNode->left = NULL;
      newNode->right = NULL;
      return newNode;
   }
   if (value < root->value) {
      root->left = insert_into_bst(root->left, value);
   } else {
      root->right = insert_into_bst(root->right, value);
   }
```

```
    return root;
}

void inorder_traversal(struct TreeNode* root) {
    if (root == NULL) {
        return;
    }
    inorder_traversal(root->left);
    printf("%d ", root->value);
    inorder_traversal(root->right);
}

struct TreeNode* find_min(struct TreeNode* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}

struct TreeNode* delete_node(struct TreeNode* root, int key) {
    if (root == NULL) {
        return root;
    }
    if (key < root->value) {
        root->left = delete_node(root->left, key);
    } else if (key > root->value) {
        root->right = delete_node(root->right, key);
    } else {
        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;
        }
        struct TreeNode* temp = find_min(root->right);
        root->value = temp->value;
        root->right = delete_node(root->right, temp->value);
    }
    return root;
```

```
}

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

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

   printf("Before deletion: ");
   inorder_traversal(root);
   printf("\n");

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

   struct TreeNode* newRoot = delete_node(root, x);

   printf("After deletion: ");
   inorder_traversal(newRoot);
   printf("\n");

   return 0;
}
```

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