Rajalakshmi Engineering College

Name: SHIVANISREE K B
Email: 240701501@rajalakshmi.edu.in
Roll no: 240701501
Phone: 7358464804
Branch: REC
Department: I CSE FE
Batch: 2028
Degree: B.E - CSE

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

Emily is studying binary search trees (BST). She wants to write a program that inserts characters into a BST and then finds and prints the minimum and maximum values.

Guide her with the program.

Input Format

The first line of input consists of an integer N, representing the number of values to be inserted into the BST.

The second line consists of N space-separated characters.

Output Format

The first line of output prints "Minimum value: " followed by the minimum value of the given inputs.

The second line prints "Maximum value: " followed by the maximum value of the given inputs.

Refer to the sample outputs for formatting specifications.

Sample Test Case

Input: 5
Z E W T Y

Output: Minimum value: E
Maximum value: Z

Answer

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

// Define the Node structure for the BST
struct Node {
   char data;
   struct Node* left;
   struct Node* right;
};

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

// Insert function to insert a new node into the BST
struct Node* insert(struct Node* root, char data) {
   if (root == NULL)
      return newNode(data);

   // If data is smaller, insert into the left subtree
```

```
    if (data < root->data)
        root->left = insert(root->left, data);
    // If data is greater, insert into the right subtree
    else if (data > root->data)
        root->right = insert(root->right, data);

    return root;
}

// Function to find the minimum value in the BST
char findMin(struct Node* root) {
    while (root != NULL && root->left != NULL) {
        root = root->left;
    }
    return root->data;
}

// Function to find the maximum value in the BST
char findMax(struct Node* root) {
    while (root != NULL && root->right != NULL) {
        root = root->right;
    }
    return root->data;
}

int main() {
    int N;

    // Read the number of nodes to insert into the BST
    scanf("%d", &N);

    char values[N];

    // Read the characters to insert into the BST
    for (int i = 0; i < N; i++) {
        scanf(" %c", &values[i]);  // Read each character with a space before to
ignore any newline
    }

    // Construct the BST
    struct Node* root = NULL;
    for (int i = 0; i < N; i++) {
```

```
        root = insert(root, values[i]);
    }

    // Find and print the minimum and maximum values in the BST
    printf("Minimum value: %c\n", findMin(root));
    printf("Maximum value: %c\n", findMax(root));

    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>

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

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

// Insert function to insert a new node into the BST
struct Node* insert(struct Node* root, int data) {
   if (root == NULL)
      return newNode(data);

   // If data is smaller, insert into the left subtree
   if (data < root->data)
      root->left = insert(root->left, data);
   // If data is greater, insert into the right subtree
   else if (data > root->data)
      root->right = insert(root->right, data);

   return root;
}

// Reverse In-order Traversal to find the k-th largest element
int kthLargestUtil(struct Node* root, int* k) {
   if (root == NULL)
```

```
        return -1;

    // Recur on the right subtree first (reverse in-order)
    int right = kthLargestUtil(root->right, k);
    if (right != -1) // If we've already found the k-th largest element
        return right;

    // Decrease k and check if this is the k-th largest element
    (*k)--;
    if (*k == 0) {
        return root->data; // Found the k-th largest element
    }

    // Recur on the left subtree
    return kthLargestUtil(root->left, k);
}

int main() {
    int n, k;

    // Read number of nodes in the BST
    scanf("%d", &n);

    int values[n];

    // Read the elements to insert into the BST
    for (int i = 0; i < n; i++) {
        scanf("%d", &values[i]);
    }

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

    // Construct the BST
    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    // Find the k-th largest element
    int result = kthLargestUtil(root, &k);
    if (result == -1) {
```

```
        printf("Invalid value of k\n");
    } else {
        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>

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

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

// Insert function to insert a new node into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL)
        return newNode(data);

    // If data is smaller, insert into the left subtree
    if (data < root->data)
        root->left = insert(root->left, data);
    // If data is greater, insert into the right subtree
    else if (data > root->data)
        root->right = insert(root->right, data);
```

```
    return root;
}

// Function to perform an in-order traversal of the BST
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function to find the node with the minimum value in the BST
struct Node* findMin(struct Node* root) {
    while (root != NULL && root->left != NULL) {
        root = root->left;
    }
    return root;
}

// Function to delete a node from the BST
struct Node* deleteNode(struct Node* root, int key) {
    // Base case: If the tree is empty
    if (root == NULL)
        return root;

    // Recurse down the tree
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        // Node to be deleted found

        // Case 1: Node has no children (leaf node)
        if (root->left == NULL && root->right == NULL) {
            free(root);
            return NULL;
        }

        // Case 2: Node has only one child
        else 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;
    }

    // Case 3: Node has two children
    else {
        // Get the in-order successor (smallest in the right subtree)
        struct Node* temp = findMin(root->right);

        // Replace the node's data with the in-order successor's data
        root->data = temp->data;

        // Delete the in-order successor
        root->right = deleteNode(root->right, temp->data);
    }
  }

  return root;
}

int main() {
  int n, x;

  // Read the number of nodes in the BST
  scanf("%d", &n);

  int values[n];

  // Read the elements to insert into the BST
  for (int i = 0; i < n; i++) {
    scanf("%d", &values[i]);
  }

  // Read the key value to be deleted
  scanf("%d", &x);
```

```
    // Construct the BST
    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    // Before deletion: Print in-order traversal
    printf("Before deletion: ");
    inorderTraversal(root);
    printf("\n");

    // Delete the node with the given key value
    root = deleteNode(root, x);

    // After deletion: Print in-order traversal
    printf("After deletion: ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

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