Rajalakshmi Engineering College

Name: Pranav Narayanan
Email: 240701393@rajalakshmi.edu.in
Roll no:
Phone: 9500579427
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

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

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

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

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

// Function to insert a value into the BST
Node* insert(Node* root, char value) {
   if (root == NULL) {
      return createNode(value);
   }
   if (value < root->data) {
      root->left = insert(root->left, value);
```

```
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}

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

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

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

    Node* root = NULL;
    for (int i = 0; i < N; i++) {
        char ch;
        scanf(" %c", &ch); // Space before %c to skip whitespace
        root = insert(root, ch);
    }

    char minVal = findMin(root);
    char maxVal = findMax(root);

    printf("Minimum value: %c\n", minVal);
    printf("Maximum value: %c\n", maxVal);

    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>
```

```
// Define the structure for a node in the BST
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 value in BST
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 if (value > root->data)
      root->right = insert(root->right, value);
   return root;
}

// Reverse in-order traversal to find k-th largest
void kthLargestUtil(struct Node* root, int k, int* count, int* result) {
   if (root == NULL || *count >= k)
      return;

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

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

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

```
// Wrapper function
int findKthLargest(struct Node* root, int k, int totalNodes) {
    if (k > totalNodes || k <= 0)
        return -1;

    int count = 0, result = -1;
    kthLargestUtil(root, k, &count, &result);
    return result;
}

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

    struct Node* root = NULL;

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

    scanf("%d", &k);

    int result = findKthLargest(root, k, n);

    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>

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

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

Node* insert(Node* root, int key) {
    if (root == NULL)
        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* findMin(Node* root) {
    while (root->left)
        root = root->left;
    return root;
}

Node* deleteNode(Node* root, int key, int* deleted) {
    if (root == NULL) return NULL;
    if (key < root->key)
        root->left = deleteNode(root->left, key, deleted);
    else if (key > root->key)
        root->right = deleteNode(root->right, key, deleted);
    else {
        *deleted = 1;
        if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }
        Node* temp = findMin(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key, deleted);
    }
    return root;
}

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

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

    scanf("%d", &x);

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

    int deleted = 0;
    root = deleteNode(root, x, &deleted);

    printf("After deletion: ");
    inorder(root);
    printf("\n");

    return 0;
}
```

Status : Correct *Marks : 10/10*