Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Name: Jhaveri Varun Nimitt

UID: 2023800042

Batch: CSE A Batch C

Experiment No.:6

Aim: Binary Search Tree OPerations

Problem:

1- Creation of BST- insertion

output: all test cases covering (like the skewed tree, full tree, and a normal BST)

2- Search

output: all test cases covering found and not found

3- Find Min and Max

4- Predecessor/ Successor

output: all test cases covering like corner cases of finding the predecessor of MIN and finding the successor of MAX

5- Deletion operation (leaf node, node with single child, node with 2 children)

Solution:

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

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

Node *createnode(int data)
{
   Node *newNode = (Node *)malloc(sizeof(Node));
   if (!newNode)
   {
       printf("out of bounds ahh\n");
       exit(1);
   }
   newNode->data = data;
   newNode->left = newNode->right = NULL;
   return newNode;
}

Node *insert(Node *root, int data)
{
   if (root == NULL)
       return createnode(data);

   if (data < root->data){
       root->left = insert(root->left, data);
       printf("Inserted %d successfully!\n", data);
   }
```

```
    else if (data > root->data){
        root->right = insert(root->right, data);
        printf("Inserted %d successfully!\n", data);
    }
    else {
        printf("Value %d already exists in the tree. Duplicate not inserted.\n", data);
        return root;
    }
    return root;
}

Node *search(Node *root, int key)
{
    if (root == NULL || root->data == key)
        return root;

    if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}

Node *minfinder(Node *root)
{
    if (!root)
        return NULL;
    while (root->left)
        root = root->left;
    return root;
}

Node *maxfinder(Node *root)
{
    if (!root)
        return NULL;
    while (root->right)
        root = root->right;
    return root;
}

Node *predeccsor(Node *root, Node *pred, int key)
{
    if (!root)
        return pred;

    if (root->data == key)
    {
```

```
        if (root->left)
            return maxfinder(root->left);
    }
    else if (key < root->data)
    {
        return predeccsor(root->left, pred, key);
    }
    else
    {
        pred = root;
        return predeccsor(root->right, pred, key);
    }
    return pred;
}

Node *succesor(Node *root, Node *succ, int key)
{
    if (!root)
        return succ;

    if (root->data == key)
    {
        if (root->right)
            return minfinder(root->right);
    }
    else if (key < root->data)
    {
        succ = root;
        return succesor(root->left, succ, key);
    }
    else
    {
        return succesor(root->right, succ, key);
    }
    return succ;
}

Node *deltenode(Node *root, int key)
{
    if (!root)
        return root;

    if (key < root->data)
        root->left = deltenode(root->left, key);
    else if (key > root->data)
        root->right = deltenode(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 = minfinder(root->right);
        root->data = temp->data;
        root->right = deltenode(root->right, temp->data);
    }
    return root;
}

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

void freee(Node *root)
{
    if (root)
    {
        freee(root->left);
        freee(root->right);
        free(root);
    }
}

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

    while (1)
    {
```

```
printf("================================\n");
printf("|      Operations Available      |\n");
printf("================================\n");
printf(" 1. Insert a value\n");
printf(" 2. Search for a value\n");
printf(" 3. Find the Minimum value\n");
printf(" 4. Find the Maximum value\n");
printf(" 5. Find Predecessor of a value\n");
printf(" 6. Find Successor of a value\n");
printf(" 7. Delete a value\n");
printf(" 8. Display In-order Traversal\n");
printf(" 9. Exit\n");
printf("================================\n");

printf("choose an option 1-9 : ");
scanf("%d", &choice);

switch (choice)
{
case 1:
    printf(">> Insert value: ");
    scanf("%d", &value);
    root = insert(root, value);
    break;

case 2:
    printf(">> Search for value: ");
    scanf("%d", &value);
    if (search(root, value) != NULL)
        printf("%d is found in the tree.\n", value);
    else
        printf("%d is not found in the tree.\n", value);
    break;

case 3:
    if (root)
    {
        Node *minNode = minfinder(root);
        printf("Minimum value in the tree is: %d\n", minNode->data);
    }
    else
    {
        printf("Tree is empty.\n");
    }
    break;

case 4:
```

```
        if (root)
        {
            Node *maxNode = maxfinder(root);
            printf("Maximum value in the tree is: %d\n", maxNode->data);
        }
        else
        {
            printf("Tree is empty.\n");
        }
        break;

    case 5:
        printf(">> Enter value to find predecessor: ");
        scanf("%d", &value);
        {
            Node *pred = predeccsor(root, NULL, value);
            if (pred != NULL)
                printf("Predecessor of %d is: %d\n", value, pred->data);
            else
                printf("No predecessor found for %d\n", value);
        }
        break;

    case 6:
        printf(">> Enter value to find successor: ");
        scanf("%d", &value);
        {
            Node *succ = succesor(root, NULL, value);
            if (succ != NULL)
                printf("Successor of %d is: %d\n", value, succ->data);
            else
                printf("No successor found for %d\n", value);
        }
        break;

    case 7:
        printf(">> Enter value to delete: ");
        scanf("%d", &value);
        root = deltenode(root, value);
        printf("Deleted %d successfully!\n", value);
        break;

    case 8:
        printf("In-order traversal of the tree: ");
        inorder(root);
        printf("\n");
        break;
```

```
        case 9:
            freee(root);
            printf("goodbye world\n");
            exit(0);

        default:
            printf("misinput in choice.\n");
        }

        if (choice != 9)
        {
            printf("Current in-order traversal of the tree: ");
            inorder(root);
            printf("\n");
        }
    }
}
```

OUTPUT:

https://imgur.com/a/eAkKqyF