Kartheeka.R

192372289

CSE-AI

30/07/2024

Binary tree traversal:

Inorder:

Preorder:

Postorder:

Sourse code:

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

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

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

// Function to perform Inorder traversal
void inorderTraversal(Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

// Function to perform Preorder traversal
void preorderTraversal(Node* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

// Function to perform Postorder traversal
void postorderTraversal(Node* root) {
    if (root == NULL) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->data);
}
```

```
int main() {
    Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

    printf("Inorder traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Preorder traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Postorder traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

OUTPUT:

```
Inorder traversal: 4 2 5 1 6 3 7
Preorder traversal: 1 2 4 5 3 6 7
Postorder traversal: 4 5 2 6 7 3 1
```

2. Binary tree search,insert and delete:

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

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

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

// Function to insert a node into the BST
Node* insertNode(Node* root, int data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}

// Function to search for a node in the BST
Node* searchNode(Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    } else if (data < root->data) {
        return searchNode(root->left, data);
    } else {
        return searchNode(root->right, data);
    }
}

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

// Function to delete a node from the BST
Node* deleteNode(Node* root, int data) {
    if (root == NULL) {
        return root;
    } else if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        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;
        } else {
            Node* temp = findMinNode(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }
    return root;
}

// Function to perform Inorder traversal
void inorderTraversal(Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

int main() {
    Node* root = NULL;
    root = insertNode(root, 5);
    root = insertNode(root, 3);
    root = insertNode(root, 7);
    root = insertNode(root, 2);
    root = insertNode(root, 4);
    root = insertNode(root, 6);
    root = insertNode(root, 8);

    printf("Inorder traversal: ");
    inorderTraversal(root);
    printf("\n");

    Node* searchedNode = searchNode(root, 4);
    if (searchedNode != NULL) {
        printf("Node %d found\n", searchedNode->data);
    } else {
        printf("Node not found\n");
    }

    root = deleteNode(root, 3);
    printf("Inorder traversal after deletion: ");
    inorderTraversal(root);
    printf("\n");
    return 0;
}
```

OUTPUT:

```
Inorder after insertion traversal: 2 3 4 5 6 7 8
Node 4 found
Inorder traversal after deletion: 2 4 5 6 7 8
```