PROGRAMMING USING C++ - CIPHERSCHOOLS

Binary Search Tree (BST) Mini Project

Table of Contents

1. Introduction

This document provides an overview and detailed explanation of a Binary Search Tree (BST) project implemented using C++. The project includes fundamental operations like insertion, deletion, searching, various tree traversals, and tree inversion.

2. Project Overview

In this project, a Binary Search Tree (BST) is constructed with the following functionalities:

- Insert
- Delete
- Search
- Preorder Traversal
- Postorder Traversal
- Inorder Traversal
- Inverting the Tree

3. Environment Setup

To run this project, the following environment is required:

- C++ compiler (e.g., g++)
- Integrated Development Environment (IDE) or text editor (e.g., Visual Studio Code, Code::Blocks)

4. Implementation Details

Insert Operation

The insert operation adds a new node to the BST while maintaining the BST property.

Delete Operation

The delete operation removes a node from the BST and reconfigures the tree to maintain the BST property.

Search Operation

The search operation checks whether a given value exists in the BST.

Traversals

- **Preorder Traversal**: Visit the root node first, then the left subtree, and finally the right subtree.
- **Postorder Traversal**: Visit the left subtree first, then the right subtree, and finally the root node.
- **Inorder Traversal**: Visit the left subtree first, then the root node, and finally the right subtree.

Inverting the BST

The invert operation transforms the BST into its mirror image.

5. Code Explanation

Insert Operation

```
TreeNode* insert(TreeNode* root, int key) {
    if (root == NULL) {
        return new TreeNode(key);
    }
    if (key < root->key) {
        root->left = insert(root->left, key);
    } else if (key > root->key) {
        root->right = insert(root->right, key);
    }
    return root;
}
```

Delete Operation

```
TreeNode* deleteNode(TreeNode* root, int key) {
    if (root == NULL) {
        return root;
    }
    if (key < root->key) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->key) {
```

```
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            TreeNode* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == NULL) {
            TreeNode* temp = root->left;
            delete root;
            return temp;
        }
```

Search Operation

```
TreeNode* search(TreeNode* root, int key) {
    if (root == NULL || root->key == key) {
        return root;
    }
    if (key < root->key) {
        return search(root->left, key);
    }
    return search(root->right, key);
}
```

Inorder Traversal

```
void inorder(TreeNode* root) {
    if (root != NULL) {
        inorder(root->left);
        cout << root->key << " ";
        inorder(root->right);
    }
}
```

Preorder Traversal

```
void preorder(TreeNode* root) {
    if (root != NULL) {
        cout << root->key << " ";
```

```
        preorder(root->left);
        preorder(root->right);
    }
}
```

Postorder Traversal

```
void postorder(TreeNode* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        cout << root->key << " ";
    }
}
```

Inverting a BST

```
TreeNode* invertTree(TreeNode* root) {
    if (root == NULL) {
        return NULL;
    }
    TreeNode* temp = root->left;
    root->left = invertTree(root->right);
    root->right = invertTree(temp);

    return root;
}
```

6. Conclusion

I have learnt to construct a binary tree and implement operations like insert ,delete,search,traversals,etc. I have learnt programming using C++ and I have also learnt the basics of DSA.

7. References

https://www.youtube.com/watch?v=ScdwdSCnXDU

https://www.geeksforgeeks.org/introduction-to-binary-search-tree/

ANUSREE R

ar2174ar@gmail.com