Pranav Modgil

102115009

3NC1

Write a program for binary search tree (BST) having functions for the following operations:

Insert an element (no duplicates are allowed), Delete an existing element,

Traverse the BST (in-order, pre-order, and post-order), Maximum depth, and Minimum depth.

NOTE: Maximum depth: Number of nodes along the longest path from the root node down to the

Minimum depth: Number of nodes along the shortest path from the root node down to the nearest leaf

node.

Algorithms:

1. **Node Class:**
   * Declare a class **Node** with integer **key**, and pointers to left and right children.
   * Constructor initializes key and sets left and right pointers to **NULL**.
2. **Insert Function:**
   * If the root is NULL, create a new node with the given key and return it.
   * If the key is less than the root's key, recursively call insert on the left subtree.
   * If the key is greater than the root's key, recursively call insert on the right subtree.
   * Return the root of the modified subtree.
3. **Search Function:**
   * If the root is NULL, return false.
   * If the key matches the root's key, return true.
   * If the key is less than the root's key, recursively call search on the left subtree.
   * Otherwise, recursively call search on the right subtree.
4. **Print In-Order Function:**
   * If the root is NULL, return.
   * Recursively print the left subtree.
   * Print the root's key.
   * Recursively print the right subtree.
5. **Print Pre-Order Function:**
   * If the root is NULL, return.
   * Print the root's key.
   * Recursively print the left subtree.
   * Recursively print the right subtree.
6. **Print Post-Order Function:**
   * If the root is NULL, return.
   * Recursively print the left subtree.
   * Recursively print the right subtree.
   * Print the root's key.
7. **Find Min Function:**
   * Iterate to the leftmost leaf node starting from the given root.
   * Return the leftmost leaf node.
8. **Remove Function:**
   * If the root is NULL, return NULL.
   * If the key is less than the root's key, recursively call remove on the left subtree.
   * If the key is greater than the root's key, recursively call remove on the right subtree.
   * If the key matches the root's key:
     + If the node has no children, delete it and set the root to NULL.
     + If the node has one child, replace the node with its child.
     + If the node has two children, find the minimum key in the right subtree, replace the node's key with it, and recursively call remove on the right subtree.
   * Return the modified root.
9. **Max Depth Function:**
   * If the root is NULL, return 0.
   * Recursively find the maximum depth of the left and right subtrees.
   * Return 1 plus the maximum depth of the left and right subtrees.
10. **Min Depth Function:**
    * If the root is NULL, return 0.
    * Recursively find the minimum depth of the left and right subtrees.
    * Return 1 plus the minimum depth of the left and right subtrees.
11. **Main Function:**
    * Create an empty BST and insert elements from the array {8,3,10,1,6,14,4,7,13}.
    * Print the tree in in-order, pre-order, and post-order.
    * Remove the node with key 10.
    * Print the tree again in in-order, pre-order, and post-order.
    * Calculate and print the max and min depths of the tree.

Code:

#include <iostream>

using namespace std;

class Node

{

  public:

   int key;

   Node \*left;

   Node \*right;

   Node(int key){

       this->key = key;

       left = right  = NULL;

   }

};

Node\* insert(Node \* root, int key){

    if(root==NULL){

        return new Node(key);

    }

    //rec case

    if(key < root->key){

        root->left = insert(root->left,key);

    }

    else if(key > root->key){

        root->right = insert(root->right,key);

    }

    return root;

}

//O(H)

bool search(Node \* root, int key){

        if(root==NULL){

            return false;

        }

        if(root->key==key){

            return true;

        }

        if(key < root->key){

            return search(root->left,key);

        }

        return search(root->right,key);

}

void printInOrder(Node \*root){

    if(root==NULL){

        return;

    }

    //left, root, right

    printInOrder(root->left);

    cout << root-> key <<", ";

    printInOrder(root->right);

}

void printPreOrder(Node \*root){

    if(root==NULL){

        return;

    }

    //root, left, right

    cout << root-> key <<", ";

    printPreOrder(root->left);

    printPreOrder(root->right);

}

void printPostOrder(Node \*root){

    if(root==NULL){

        return;

    }

    //left, right, root

    printPostOrder(root->left);

    printPostOrder(root->right);

    cout << root-> key <<", ";

}

Node\* findMin(Node\* root){

        while(root->left!=NULL){

            root = root->left;

        }

        return root;

}

//BST Deletion

Node\* remove(Node\* root,int key){

        if(root==NULL){

            return NULL;

        }

        else if(key < root->key){

            root->left = remove(root->left,key);

        }

        else if(key > root->key){

            root->right = remove(root->right,key);

        }

        else{

            //when the current node matches with the key

            // No children

            if(root->left==NULL && root->right==NULL){

                delete root;

                root = NULL;

            }

            // Single Child

            else if(root->left==NULL){

                Node\* temp = root;

                root = root->right;

                delete temp;

            }

            else if(root->right==NULL){

                Node\*temp = root;

                root = root->left;

                delete temp;

            }

            //2 Children

            else{

                Node\* temp = findMin(root->right);

                root->key = temp->key;

                root->right = remove(root->right,temp->key);

            }

        }

        return root;

}

int maxDepth(Node\* root) {

    if (root == NULL) {

        return 0;

    }

    int leftDepth = maxDepth(root->left);

    int rightDepth = maxDepth(root->right);

    return 1 + max(leftDepth, rightDepth);

}

int minDepth(Node\* root) {

    if (root == NULL) {

        return 0;

    }

    int leftDepth = minDepth(root->left);

    int rightDepth = minDepth(root->right);

    return 1 + min(leftDepth, rightDepth);

}

int main(){

    cout<<"Pranav Modgil"<<endl;

    cout<<"102115009, 3NC1"<<endl;

    Node \* root = NULL;

    int arr[] = {8,3,10,1,6,14,4,7,13};

    for(int x : arr){

        root = insert(root,x);

    }

    cout<<"Inorder Traversal :";

    printInOrder(root);

    cout<<endl;

    cout<<"Inorder Traversal :";

    printPreOrder(root);

    cout<<endl;

    cout<<"Inorder Traversal :";

    printPostOrder(root);

    cout<<endl;

    cout<<"After Removing 10"<<endl;

    remove(root,10);

    cout<<"Inorder Traversal :";

    printInOrder(root);

    cout<<endl;

    cout<<"Inorder Traversal :";

    printPreOrder(root);

    cout<<endl;

    cout<<"Inorder Traversal :";

    printPostOrder(root);

    cout<<endl;

    int maxd = maxDepth(root);

    int mind = minDepth(root);

    cout<<endl;

    cout<<"Max Depth: "<<maxd<<endl;

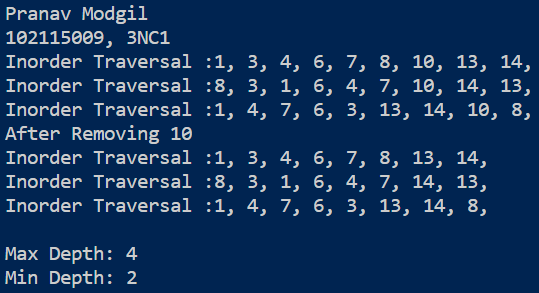
    cout<<"Min Depth: "<<mind<<endl;

    int key;

    return 0;

}

Output:



Top of Form