**S.Tejeswar (AI&DS, 22AD104)**

**BST Operations**

1. **Insertion of Nodes in a BST by recursion:**

#include <iostream>

using namespace std;

struct Node{

int val;

Node\* left;

Node\* right;

Node(int key){

val = key;

left = NULL;

right = NULL;

}

};

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

if(root == NULL){

return new Node(key);

}

if(root->val == key){

return root;

}

if(root->val < key){

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

}

else if(root->val > key){

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

}

return root;

}

void inorder(Node\* root){

if(root != NULL){

inorder(root->left);

cout << root->val << ' ';

inorder(root->right);

}

}

int main()

{

Node\* root = new Node(10);

root = insert(root, 12);

root = insert(root, 8);

root = insert(root, 2);

root = insert(root, 11);

root = insert(root, 9);

root = insert(root, 19);

inorder(root);

return 0;

}

1. **Inserton of Nodes in BST by iteration:**

#include <iostream>

using namespace std;

struct Node{

int val;

Node\* left;

Node\* right;

Node(int key){

val = key;

left = NULL;

right = NULL;

}

};

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

if(root == NULL){

return new Node(key);

}

else if(root->val == key){

return root;

}

Node\* curr = root;

while(curr){

if(curr->val > key){

if(curr->left == NULL){

curr->left = new Node(key);

break;

}

else{

curr = curr->left;

}

}

else if(curr->val < key){

if(curr->right == NULL){

curr->right = new Node(key);

break;

}

else{

curr = curr->right;

}

}

}

return root;

}

int main()

{

Node\* root = new Node(10);

root = insertByIteration(root, 12);

root = insertByIteration(root, 8);

root = insertByIteration(root, 2);

root = insertByIteration(root, 11);

root = insertByIteration(root, 9);

root = insertByIteration(root, 19);

inorder(root);

return 0;

}

1. **BST Traversal**

#include <iostream>

using namespace std;

struct Node{

int val;

Node\* left;

Node\* right;

Node(int key){

val = key;

left = NULL;

right = NULL;

}

};

void inorder(Node\* root){

if(root != NULL){

inorder(root->left);

cout << root->val << ' ';

inorder(root->right);

}

}

void preorder(Node\* root){

if(root != NULL){

cout << root->val << ' ';

preorder(root->left);

preorder(root->right);

}

}

void postOrder(Node\* root){

if(root != NULL){

postOrder(root->right);

postOrder(root->left);

cout << root->val << ' ';

}

}

int main()

{

Node\* root = new Node(10);

root = insertByIteration(root, 12);

root = insertByIteration(root, 8);

root = insertByIteration(root, 2);

root = insertByIteration(root, 11);

root = insertByIteration(root, 9);

root = insertByIteration(root, 19);

cout << "\nPreorder Traversal: \n";

preorder(root);

cout << "\nIorder Traversal: \n";

inorder(root);

cout << "\nPost Order Traersal: \n";

postOrder(root);

return 0;

}

1. **Validate Binary Search Tree**

bool dfs(TreeNode\* root, long low, long high){

if(root != NULL){

if(low >= root->val || root->val >= high){

return false;

}

bool leftVal = dfs(root->left, low, root->val);

bool rightVal = dfs(root->right, root->val, high);

return ( leftVal && rightVal );

}

else{

return true;

}

}

bool isValidBST(TreeNode\* root) {

return dfs(root, LONG\_MIN, LONG\_MAX);

}

**Time Complexity : O(N)**

**Space Complexity: O(1)**

1. **Print Right View of Binary Tree**

void dfsRightMost(TreeNode\* root, vector<int>& answer, int level){

        if(root) {

            if(answer.size() > level){

                answer[level] = root->val;

            }

            else{

                answer.push\_back(root->val);

            }

            dfsRightMost(root->left, answer, level + 1);

            dfsRightMost(root->right, answer, level + 1);

        }

    }

vector<int> rightSideView(TreeNode\* root) {

        vector<int> answer;

        dfsRightMost(root, answer, 0);

        return answer;

**Time Complexity: O(N), N – no of nodes**

**Space Complexity: O(1)**