**Experiment No 1**

**AIM:** Study of Binary search tree as a non-linear Data Structure

**PROBLEM STATEMENT:** Develop a system for an online bookstore to manage its catalog using binary search trees (BSTs). Allow operations like adding new books, searching for books by title or author efficiently, and updating book information.

**REQUIREMENT:**Turbo C/ GCC Compiler

**OPERATING SYSTEM:** Windows/Linux/Unix.

**THEORY:**

**A Binary Search Tree (BST)** is a type of binary tree in which the data is organized and stored in a sorted order. Unlike, a binary tree that doesn't follow a specific order for node placement, in a binary search tree all the elements on the left side of a node are smaller than the node itself, and elements on the right side of a node are greater.

**What is a Binary Search Tree (BST)?**

A Binary Search Tree (BST) is a binary tree in which every node contains only smaller values in its left subtree and only larger values in its right subtree. This property is called the BST property and every binary search tree follows this property as it allows efficient insertion, deletion, and search operations in a tree.

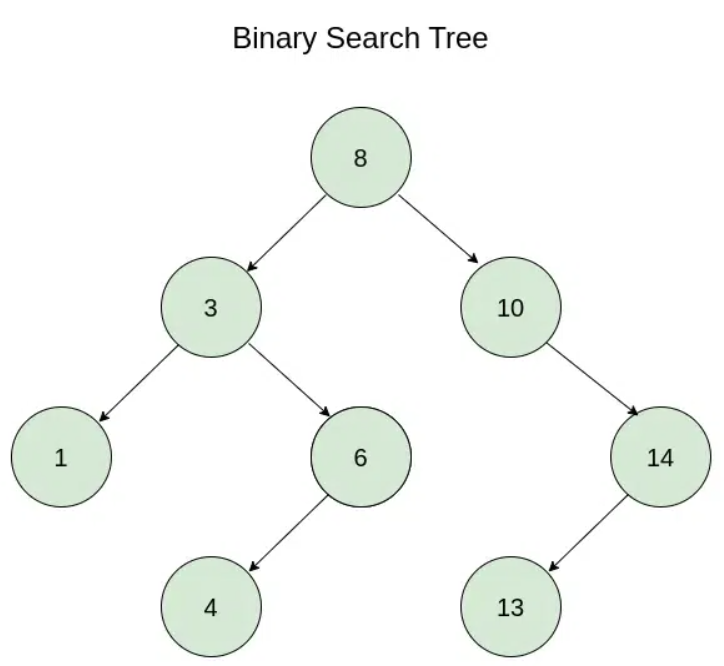
Conditions for a Tree to be a Binary Search Tree

For a tree to be called a binary search, it should fulfill the following conditions:

* All the nodes in the left subtree of any node contain smaller values and all the nodes in the right subtree of any node contain larger values.
* Both the left and right subtrees of any node in the tree should themselves be a BST. This means that they should follow the BST rule.
* A unique path exists from the root node to every other node.

Binary Search Tree Representation in C++

In BST, every value on the left subtree < parent node < right subtree value.



**Binary Search Tree**

Following are the basics terminologies used in BST:

* Children: The successor nodes of a node are called its children.
* Parent: The predecessor node of a node is called its parent.
* Root: The "beginning" node is called the root i.e. a node that has no parent.
* leaf: A node that has no children is called a leaf.

Basic Operations on Binary Search Tree

The following are the basics operations performed on a binary search tree:

* Creation
* Traversing (Pre-order, Post-order and In-order)
* Search
* Insertion
* Deletion

Here, we will discuss the basic three operation: search, insertion and deletion in a binary search tree.

| **Operation** | **Best Case Time Complexity** | **Average Case Time Complexity** | **Worse Case Time Complexity** |
| --- | --- | --- | --- |
| Search | O(log n) | O(log n) | O(n) |
| Insertion | O(log n) | O(log n) | O(n) |
| Deletion | O(log n) | O(log n) | O(n) |

The space complexity for all the above operations is O(n).

**CONCLUSION:** Program for book store using BST is implemented successfully.

// C++ Program to implement binary search tree

#include <iostream>

#include <cstring>

using namespace std;

// Node structure for a Binary Search Tree

struct Node {

int bookno;

char name[30];

char pub[30];

int pages;

Node\* left;

Node\* right;

};

// Function to create a new Node

Node\* createNode(int bookno, char \* const name, char \* const pub, int pages)

{

Node\* newNode = new Node();

newNode->bookno = bookno;

strcpy(newNode->name , name);

strcpy(newNode->pub , pub);

newNode->pages = pages;

newNode->left = newNode->right = nullptr;

return newNode;

}

// Function to insert a node in the BST

Node\* insertNode(Node\* root, int bookno, char \* const name, char \* const pub, int pages)

{

if (root == nullptr) { // If the tree is empty, return a

// new node

return createNode(bookno, name, pub, pages);

}

// Otherwise, recur down the tree

if (bookno< root->bookno) {

root->left = insertNode(root->left, bookno, name, pub, pages);

}

else if (bookno > root->bookno) {

root->right = insertNode(root->right, bookno, name, pub, pages);

}

// return the (unchanged) node pointer

return root;

}

// Function to do inorder traversal of BST

void inorderTraversal(Node\* root)

{

if (root != nullptr) {

inorderTraversal(root->left);

cout << "\n";

cout << root->bookno << " ";

cout << root->name << " ";

cout << root->pub << " ";

cout << root->pages << " ";

cout << "\n";

inorderTraversal(root->right);

}

}

void preorderTraversal(Node\* root)

{

if (root != nullptr) {

cout << "\n";

cout << root->bookno << " ";

cout << root->name << " ";

cout << root->pub << " ";

cout << root->pages << " ";

cout << "\n";

preorderTraversal(root->left);

preorderTraversal(root->right);

}

}

void postorderTraversal(Node\* root)

{

if (root != nullptr) {

postorderTraversal(root->left);

postorderTraversal(root->right);

cout << "\n";

cout << root->bookno << " ";

cout << root->name << " ";

cout << root->pub << " ";

cout << root->pages << " ";

cout << "\n";

}

}

// Function to search a given key in a given BST

Node\* searchNode(Node\* root, char \* key)

{

// Base Cases: root is null or key is present at root

if (root == nullptr || strcmp(root->name, key)==0) {

return root;

}

// Key is greater than root's key

if (strcmp(root->name, key)>0) {

return searchNode(root->right, key);

}

// Key is smaller than root's key

return searchNode(root->left, key);

}

Node\* updateNode(Node\* root, char \* key)

{

Node\* found = searchNode(root, (char \*)"bbb");

cout<<"Enter the title of book";

cin>>found->name;

cout<<"Enter the publication of book";

cin>>found->pub;

cout<<"Enter the number of pages of book";

cin>>found->pages;

return found;

}

// Function to find the inorder successor

Node\* minValueNode(Node\* node)

{

Node\* current = node;

// loop down to find the leftmost leaf

while (current && current->left != nullptr) {

current = current->left;

}

return current;

}

// Function to delete a node

Node\* deleteNode(Node\* root, int bookno)

{

if (root == nullptr)

return root;

// If the data to be deleted is smaller than the root's

// data, then it lies in the left subtree

if (bookno < root->bookno) {

root->left = deleteNode(root->left, bookno);

}

// If the data to be deleted is greater than the root's

// data, then it lies in the right subtree

else if (bookno > root->bookno) {

root->right = deleteNode(root->right, bookno);

}

// if data is same as root's data, then This is the node

// to be deleted

else {

// node with only one child or no child

if (root->left == nullptr) {

Node\* temp = root->right;

delete root;

return temp;

}

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

Node\* temp = root->left;

delete root;

return temp;

}

// node with two children: Get the inorder successor

// (smallest in the right subtree)

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

// Copy the inorder successor's content to this node

root->bookno = temp->bookno;

// Delete the inorder successor

root->right = deleteNode(root->right, temp->bookno);

}

return root;

}

// Main function to demonstrate the operations of BST

int main()

{

Node\* root = nullptr;

// create a BST

root = insertNode(root, 50,(char \*)"aaa", (char \*)"zzz", 100);

root = insertNode(root, 30,(char \*)"bbb", (char \*)"yyy", 100);

root = insertNode(root, 20,(char \*)"ccc", (char \*)"xxx", 100);

root = insertNode(root, 40,(char \*)"ddd", (char \*)"vvv", 100);

root = insertNode(root, 70,(char \*)"eee", (char \*)"www", 100);

root = insertNode(root, 60,(char \*)"fff", (char \*)"qqq", 100);

root = insertNode(root, 80,(char \*)"ggg", (char \*)"rrr", 100);

// Print the inorder traversal of a BST

cout << "Inorder traversal of the given Binary Search "

"Tree is: ";

inorderTraversal(root);

cout << endl;

// delete a node in BST

root = deleteNode(root, 20);

cout << "After deletion of 20: ";

inorderTraversal(root);

cout << endl;

// Insert a node in BST

root = insertNode(root, 25,(char \*)"iii", (char \*)"ttt", 100);

cout << "After insertion of 25: ";

preorderTraversal(root);

cout << endl;

inorderTraversal(root);

cout << endl;

postorderTraversal(root);

cout << endl;

// Search a key in BST

Node\* found = searchNode(root, (char \*)"bbb"); //by title

// check if the key is found or not

if (found != nullptr) {

cout << "Node bbb found in the BST." << endl;

}

else {

cout << "Node bbb not found in the BST." << endl;

}

found = updateNode(root, (char \*)"bbb");

cout << endl;

inorderTraversal(root);

cout << endl;

return 0;

}