5. Binary Search Tree

Title: Implement binary search tree and perform following operations:

a. Insert (Handle insertion of duplicate entry)
b. Delete
c. Search
d. Display Tree (Traversal)
e. Display- Depth of tree
f. Display- Mirror Image
g. Create a Copy
h. Display all parent nodes with their child nodes
i. Display leaf nodes
j. Display tree level wise

(Note: Insertion, Deletion, Search and Traversal are compulsory, from rest of operations,

perform Any three)

OBJECTIVE:

1. To understand the concept of binary search tree as a data structure.
2. Applications of BST.

THEORY:

- **Definition of binary search tree**

A binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

- **Searching**

Searching a binary tree for a specific value can be a recursive or iterative process. This explanation covers a recursive method.We begin by examining the root node. If the tree is null, the value we are searching for does not exist in the tree. Otherwise, if the value equals the root, the search is successful. If the value is less than the root, search the left subtree. Similarly, if it is greater than the root, search the right subtree. This process is repeated until the value is found or the indicated subtree is null. If the searched value is not found before a null subtree is reached, then the item must not be present in the tree.

- **Insertion**

Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root.

- **Deletion**

There are three possible cases to consider:

- **Deleting a leaf (node with no children):** Deleting a leaf is easy, as we can simplyremove it from the tree.
- **Deleting a node with one child:** Remove the node and replace it with its child.

- **Deleting a node with two children:** Call the node to be deleted N. Do not deleteN. Instead, choose either its in-order successor node or its in-order predecessor node, R. Replace the value of N with the value of R, then delete R.

As with all binary trees, a node's in-order successor is the left-most child of its right subtree, and a node's in-order predecessor is the right-most child of its left subtree. In either case, this node will have zero or one children. Delete it according to one of the two simpler cases above.

- **ALGORITHM:**

Define structure for Binary Tree (Mnemonics, Left Pointer, Right Pointer)

Insert Node:

Insert (Root, Node)
Root is a variable of type structure, represent Root of the Tree. Node is a variable of type structure ,represent new Node to insert in a tree.
Step 1: Repeat Steps 2,3& 4 Until Node do not insert at appropriate position.
Step 2: If Node Data is less that Root Data & Root Left Tree is NULL
Then insert Node to Left.
Else Move Root to Left
Step 3: Else If Node Data is Greater that equal that Root Data & Root Right Tree is NULL
Then insert Node to Right.
Else Move Root to Right.
Step 4: Stop.

Search Node:

Search (Root, Mnemonics)
Root is a variable of type structure, represent Root of the Tree. Mnemonics is array of character. This function search Mnemonics in a Tree.
Step 1: Repeat Steps 2,3& 4 Until Mnemonics Not find && Root != NULL
Step 2: If Mnemonics Equal to Root Data
Then print message Mnemonics present.
Step 3: Else If Mnemonics Greater that Equal that Root Data Then
Move Root to Right.
Step 4: Else Move Root to Left.
Step 5: Stop.

Delete Node:

Dsearch(Root, Mnemonics)
Root is a variable of type structure ,represent Root of the Tree. Mnemonics is array of character. Stack is an pointer array of type structure. PTree(Parent of Searched Node),Tree(Node to be deleted), RTree(Pointg to Right Tree),Temp are pointer variable of type structure;
Step 1: Search Mnemonics in a Binary Tree
Step 2: If Root = = NULL Then Tree is NULL
Step 3: Else //Delete Leaf Node

If Tree->Left = = NULL && Tree->Right = = NULL
Then a) If Root = = Tree Then Root = NULL;
b) If Tree is a Right Child PTree->Right=NULL;
Else PTree->Left=NULL;
Step 4: Else // delete Node with Left and Right children If
Tree->Left != NULL && Tree->Right != NULL Then
a) RTree=Temp=Tree->Right;
b)Do steps i && ii while Temp->Left !=NULL
i) RTree=Temp;
ii) Temp=Temp->Left;
c)RTree->Left=Temp->Right;
d)If Root == Tree//Delete Root Node
Root=Temp;
e)If Tree is a Right Child PTree->Right=Temp;
Else PTree->Left=Temp;
f)Temp->Left=Tree->Left;
g)If RTree!=Temp
Then Temp->Right = Tree->Right;

b) If Tree is a Right Child PTree->Right=Tree->Right; Else PTree->Left=Tree->Left;

c) If Tree is a Right Child PTree->Right=Tree->Left; Else PTree->Left=Tree->Left;
Step 7: Stop.

Depth First Search:

Root is a variable of type structure ,represent Root of the Tree.
Stack is an pointer array of type structure. Top variable of type integer.
DFS(Root)
Step 1: Repeat Steps 2,3,4,5,6 Until Stack is Empty
Step 2: print Root Data
Step 3: If Root->Right != NULL//Root Has a Right SubTree
Then Stack[Top++] = Tree->Right;//Push Right Tree into Stack Step 4: Root = Root ->Left;//Move to Left Step 5: If Root == NULL

Step 6: Root = Stack[--Top];//Pop Node from Staacl
Step 7: Stop.

Breath First Search (Levelwise Display):

Root is a variable of type structure ,represent Root of the Tree.
Queue is an pointer array of type structure. Front & Rear variable of type integer.BFS(Root)
Step 1:If Root == NULL Then Empty Tree;

Step 2: Else Queue[0] = Root;// insert Root of the Tree in a Queue
Step 3: Repeat Steps 4,5,6& 7 Until Queue is Empty
Step 4: Tree=Queue[Front++]; //Remove Node From Queue
Step 5: print Root Data
Step 6: If Root->Left != NULL
Then Queue[++Rear] = Tree->Left;//insert Left Subtree in a Queue
Step 7: If Root->Right != NULL
Then Queue[++Rear] = Root->Right; //insert Left Subtree in a Queue
Else if Root->Right == NULL And Root->Left == NULL
Leaf++;//Number of Leaf Nodes
Step 8: Stop.

Mirror Image:

Root is a variable of type structure ,represent Root of the Tree.
Queue is an pointer array of type structure. Front & Rear variable of type integer.Mirror(Root)
Step 1: Queue[0]=Root;//Insert Root Node in a Queue
Step 2: Repeat Steps 3,4,5,6,7& 8 Until Queue is Empty
Step 3: Root = Queue[Front++];
Step 4: Temp1 = Root->Left;
Step 5: Root->Left = Root->Right;
Step 6: Root->Right = Temp1;
Step 7: If Root->Left != NULL
Then Queue[Rear++] = Tree->Left;//insert Left SubTree
Step 8: If Root->Right != NULL
Then Queue[Rear++] = Root->Right;//insert Right SubTree
Step 9: Stop.

INPUT:

Accept the nodes from the user like: add, mult, div, sub

OUTPUT:

Display result of each operation with error checking.

CONCLUSION: Thus, we have studied and implement Binary Search Tree (BST) and perform various operations.

FAQ:

1. What is Binary search tree?
2. What are rules to construct binary search tree?
3. How general tree is converted into binary tree?