Rajdhani
DATE / /

Roll No. 23355

Title : Assignment 5 : Binary Search Tree

Aim : To implement a binary search tree

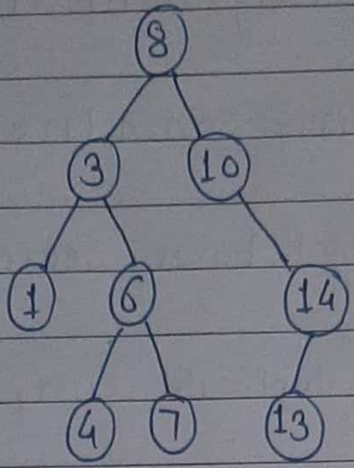
Problem statement : Implement binary search tree and perform following operations :

1a) Insert (Handle insertion of duplicate entry
2b) Delete
3) Search
4) Traversal
5) Display depth of tree
6) Display mirror image
7) Create a copy
8) Display all parent nodes with their child nodes
9) Display leaf nodes
10) Display tree level wise

Theory :

Binary search tree :
Binary search tree is a node based binary tree data structure which has following properties :
- Left subtree of node contains nodes with keys lesser than node's key
- Right subtree of node contains only nodes with keys greater than node's key
- The left & right subtree each must be also binary search tree.

Example:



Applications of BST:

- Used to efficiently store data in sorted form inorder to access and search stored elements quickly.
- BST is used in Unix kernels for managing a set of virtual memory areas.

- BST ADT

Node structure:

```
struct Node
{
    int data;
    struct node * left;
    struct node * right;
};
```

Operations:

```
bool search(int);
void insert(int);
int height();
```

```
Node * delete (int);
void preorder ();
void inorder ();
void postorder();
```

- Algorithm / Pseudocode:

1) BST creation Recursive:

```
Node* create (Node*p, int x)
    If p= NULL
        p=getNode ()
        return P
    Else
        if (x < p->data)
        {
            if p->lchild != NULL {
                p->lchild= create (p->lchild, x)
            Else
                p->lchild= getNode ()
        Else
            if p->rchild != NULL
                p->rchild = create(p->rchild, x)
            Else
                p->rchild = getNode ()
    Return P
```

Rajdhani
DATE / /

BST creation Non-recursive

```
Node* create (int num)
    Node * p = getNode (num)
    If root = NULL
        root = p
    Else
        Node * temp = root , *parent
        while (temp ≠ NULL)
            parent = temp
            If temp->data = num
                return NULL
            If temp->data < num
                temp = temp->right
            Else
                temp = temp->left
        End while
        If parent->data > num
            parent->left = p
        Else
            parent->right = p
        Return p
```

2) BST search Recursive

```
Node* Search ( key, Node * root )
    p = root
    If key < p->data
    p = Search ( key , p->left )
    else
```

Rajdhani
DATE / /

```
    if (key > p->data)
    p=search (key, p->right)
Return P
```

BST search Non recursive

```
Node * search (int num)
    Node * temp = root
    while temp ≠ NULL
        If temp->data > num
            temp = temp->left
        Else ~~If~~
        If temp->data < num
            temp = temp->right
        Else
            return temp
    End while
    Return NULL
```

3) BST delete Recursive

```
Node * deleteN(Node *T, int num)
    If T = NULL
        Return T
    If num < T->data
        T->left = deleteN (T->left, num)
    If num > T->right
        T->right = deleteN (T->right, num)
```

Rajdhani DATE / /

```
ELSE
        Node * temp = T
        If T->left = NULL
            T = T->right
            free (temp)
            return (T)
        Else if T->right = NULL
            T = T->left
            free (temp)
            return (T)

        temp = findmin (T->right)
        T->data = temp->data
        T->right = deleteN (T->right , temp->data)
    Return (T)
```

4) Level order traversal

```
DisplayLevelwise ()
    // create a queue
    enqueue (root)
    enqueue (NULL)
    while ( q.size > 1)
        Node* curr = dequeue ()
        if current = NULL
            equeue (NULL)
            print "\n"
        Else
            If current->left ≠ NULL
                enqueue (current->left)
```

Rajdhani
DATE / /

```
If current → right ≠ NULL
    enqueue (current → right)
print "current → data"

End If
End while
```

5) Depth of tree Recursive

```
int treeDepth(Node * T)
    If T = NULL
        Return 0
    Return 1+ max(treeDepth (T->left), treeDepth (T->right))
```

Non-recursive

```
int treeDepth (root)
    If root = NULL
        return 0
    // create an empty queue for level order traversal
    q. insert (root)
    height = 0
    while True
        nodecount = q. size ()
        If nodecount = 0
            return height
        height = height + 1
        while nodecount > 0
            temp = q. delete1()
            if temp → left ≠ NULL
```

Rajdhani
DATE / /

```
        q.insert (temp->left)
      If temp->right != NULL
          q.insert (temp->right)
      Nodecount --
    End while
  End while
  Return height
```

6) Mirror image
Recursive

```
  ~~mirrorImg (root)~~
  mirrorImg (T)
    If T= NULL
        return
    temp = T->left
    T-> left = T-> right
    T-> right = temp
    mirrorImg (T->left)
    mirror Img (T->right)
```

Non- recursive

```
  mirrorImg ( )
    //create an empty queue
    If root=NULL
        return
    q. insert (root)
    while ( ! q. isEmpty( )
      T = q. deque ( )
```

```
If T->left = NULL && T->right == NULL
    continue
Else If T->left ≠ NULL & T->right ≠ NULL.
    Temp = T->left
    T->left = T->right
    T->right = Temp
    q.insert (T->left)
    q.insert (T->right)
Else If T->left = NULL
    T->left = T->right
    T->right = NULL
    q.insert (T->left)
Else
    T->right = T->left
    T->left = NULL
    q.insert (T->right)
End If
End While
```

7) Copy of tree
Recursive

```
Node * createCopy (T)
  If T == NULL
    return NULL
  // create a new node
    newNode->left = createCopy (newNode->left)
    newNode->right = createCopy (newNode->right)
  Return newNode
```

8) Count number of leaf, non leaf nodes

a) leaf node :

```
int countleatNode (T)
  If ( T= NULL
    return 0
  If T->left = NULL and T->right == NULL
    return 1
  ElSE
    return countleafNode (T->left) + countleafNode (T->right)
```

b) Non leaf nodes

```
int countNonleafNode (T)
  If T = NULL OR T->left = NULL and T->right = NULL
    return 0
  return 1 + countNonleafNode (T->left) + countNonleafNode (T->righ
```

g) Traversal

a) Inorder :

```
inorder (T)
If T = NULL
  return
inorder (T->left)
print T->data
inorder (T->right
```

b) Preorder

```
preorder (T)
  If T = NULL
    return
  print "T->data"
  preorder (T->left)
  preorder (T->right)
```

c) Postorder

```
postorder (T)
  If T = NULL
    return
  postorder (T->left)
  postorder (T->right)
  print "T->data"
```

- Test cases / validations

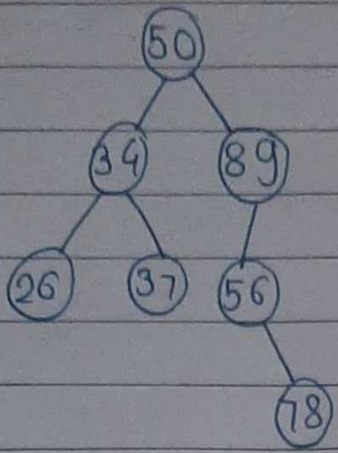
Validations :

valid key input for insertion , deletion , search operations.

Test cases :

1) Random input :
2) Sorted input
3) Input for skewed tree concept.

Example:



insert (11) ⇒
Number of comparison = 3

insert (66)
Number of comparison = 4

Conclusion:
Binary search tree is a sorted binary tree whose internal nodes each store a key greater than all keys in th left subtree and less than those in right subtree.
Using BST, we can perform various operations like ~~ies~~ searching, deleting, inserting effectively.