	Bansilal Ramnath Agarwal Charitable Trust's Vishwakarma Institute of Information Technology **Department of Artificial Intelligence and Data Science**

Name: Siddhesh Dilip Khairnar		
Class: SY	Division: B	Roll No: 272028
Semester: IV	Academic Year: 2022-2023	
Subject Name & Code: Advanced Data Structure, ADUA22202		
Title of Assignment: Write a program to implement binary search trees and perform operations.		

Aim:

Write a program to implement binary search tree and perform following.
operations:

a. Insert

b. Delete

c. Mirror image

d. Display level wise

Topic : Assignment no : 03 (AOS)

Page No.
Date : / /

Name : Siddhesh Bhavnar
ROLL NO : 272028
PRN NO : 22110398
Division : B(B2)

Aim : write a program to implement binary search tree perform following operation i) Insert ii) Delete iii) Mirror Image iv) Display level wise

Write up :-

i) Insertion in BST :

- To insert a new node in BST, we first compare the value of new node with root node.
- If new node is smaller that root node, we insert it in left subtree.
- If newnode is greater than root node, we insert it in right subtree.
- If newnode is greater than root node, we insert it in right subtree.

ii) Deletion in BST :-

- To Delete a node from BST, we first search for a node to be deleted.
- If node has no children, we simply remove it from tree.
- If node has one child, we replace the node with it child.
- If node has two children, we replace node with smallest node in it right subtree or largest node in its left subtree.

iii) mirroring a BST :-

- Mirroring a BST involves swapping left and right subtree of each node in a tree. To mirror a BST, we start at root node and recursively swap the left and right subtree of each node until we reach the leaf nodes

Algorithm:→

i BST creation:

```
create bst (root, val)
    if root is None:
        root = Node(val)
        return root
    else
        if value < root.value:
            root.left = create.bst(root.right, value)
    return root.
```

ii Insertion:

```
insert (root, value):
    if root is None
        root = Node(value)
    else
        if value < root.value:
            root.left = insert(root.left, value)
        else
            root.right = insert (root, right, value)
    return root.
```

iii Mirror Image.

```
mirror (root):
    if root is NULL
        return root
    else
        tmp = root.left
        root.left = mirror (root.right)
        root.right = mirror (root.left)
    return root.
```

IV Deletion

```
delete(root, value):
  if root is None:
    return root
  if value < root.value:
    root.left = delete(root.left, value)
  elif value > root.value:
    root.right = delete(root.right, value)
  else:
    if root.left is None:
      temp = root.right
      root = None
      return temp:
    elif root.right is None:
      temp = root.left
      root = None
      return temp.
```

Conclusion: Thus, I have performed all mentioned operation and created, inserted, deleted and displayed BST level wise.

8/9/22

Experiment:

```python
class Node:
    def __init__(self, val):
        self.left = None
        self.right = None
        self.val = val

class BST:
    def __init__(self):
        self.root = None

    def insert(self, val):
        if self.root is None:
            self.root = Node(val)
        else:
            self._insert(val, self.root)

    def _insert(self, val, curr_node):
        if val < curr_node.val:
            if curr_node.left is None:
                curr_node.left = Node(val)
            else:
                self._insert(val, curr_node.left)
        elif val > curr_node.val:
            if curr_node.right is None:
                curr_node.right = Node(val)
            else:
                self._insert(val, curr_node.right)
        else:
            print("Value already exists in tree.")

    def delete(self, val):
        if self.root is not None:
            self.root = self._delete(val, self.root)

    def _delete(self, val, curr_node):
        if curr_node is None:
            return curr_node
        elif val < curr_node.val:
            curr_node.left = self._delete(val, curr_node.left)
        elif val > curr_node.val:
            curr_node.right = self._delete(val, curr_node.right)
        else:
            if curr_node.left is None:
                return curr_node.right
            elif curr_node.right is None:
                return curr_node.left
            else:
                min_val = self._find_min_val(curr_node.right)
                curr_node.val = min_val
                curr_node.right = self._delete(min_val, curr_node.right)
        return curr_node

    def _find_min_val(self, curr_node):
        while curr_node.left is not None:
            curr_node = curr_node.left
        return curr_node.val

    def mirror(self):
        self._mirror(self.root)

    def _mirror(self, curr_node):
        if curr_node is not None:
            self._mirror(curr_node.left)
            self._mirror(curr_node.right)
            temp = curr_node.left
            curr_node.left = curr_node.right
            curr_node.right = temp

    def display_level_wise(self):
        if self.root is None:
            print("Tree is empty.")
        else:
            nodes = [self.root]
            while nodes:
                curr_node = nodes.pop(0)
                print(curr_node.val, end=' ')
                if curr_node.left is not None:
                    nodes.append(curr_node.left)
                if curr_node.right is not None:
                    nodes.append(curr_node.right)
            print()
tree = BST()

tree.insert(50)
```

```
tree.insert(30)
tree.insert(20)
tree.insert(40)
tree.insert(70)
tree.insert(60)
tree.insert(80)

# Display the tree level-wise
tree.display_level_wise()

# Delete a node and display the updated tree level-wise
tree.delete(20)
tree.display_level_wise()

# Mirror the tree and display the updated tree level-wise
tree.mirror()
tree.display_level_wise()
```

Output:

```
Binary Search Tree:
50 30 70 20 40 60 80
After deletion of 20:
50 30 70 40 60 80
Mirror BST Level Wise Display:
50 70 30 80 60 40
```

Conclusion: Thus, I've successfully completed and performed binary Search Tree operations. I've successfully inserted, deleted, and displayed level wise BST.