In [10]:

```
class Node:
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None
class BST:
    def __init__(self):
        self.root=None
    def Insert(self,value):
        self.root=self.__Insert(self.root,value)
    def __Insert(self,root,value):
        if root==None:
            root=Node(value)
        else:
            if value<root.value:
                if root.left == None:
                    root.left=Node(value)
                else:
                    root.left=self.__Insert(root.left,value)
            else:
                if root.right == None:
                    root.right=Node(value)
                else:
                    root.right=self.__Insert(root.right,value)
        return root
    def InOrder(self):
        return self.__inOrder(self.root)
    def __inOrder(self,root):
        if root:
            self.__inOrder(root.left)
            print(root.value)
            self.__inOrder(root.right)
    def PostOrder(self):
        return self.__postOrder(self.root)
    def __postOrder(self,root):
        if root:
            self.__postOrder(root.left)
            self.__postOrder(root.right)
            print(root.value)
    def PreOrder(self):
        return self.__preOrder(self.root)
    def __preOrder(self,root):
        if root:
            print(root.value)
            self.__preOrder(root.left)
            self.__preOrder(root.right)
    def Height(self):
        return self.__Height(self.root)
    def __Height(self,node):
        if node==None:
            return 0
        else:
            leftH=self.__Height(node.left)
            rightH=self.__Height(node.right)
            if leftH>rightH:
                return leftH+1
            else:
                return rightH+1
    def FindMax(self):
        return self.__Max(self.root).value
    def __Max(self,node):
        while node is not None:
            if node.right is None:
                break
            node = node.right
        return node
    def FindMin(self):
        return self.__Min(self.root).value
    def __Min(self,node):
        while node is not None:
            if node.left is None:
                break
            node = node.left
        return node
    def Successor(self):
        if self.root.right is None:
            return None
        else:
            return self.__Min(self.root.right).value
    def Predecessor(self):
        if self.root.left is None:
            return None
        else:
            return self.__Max(self.root.left).value
    def Delete(self,value):
        return self.__Delete(self.root,value)
    def __Delete(self,root,value):
        if root is None:
            return root
```

```
        if value < root.value:
            root.left = self.__Delete(root.left,value)
        elif value > root.value:
            root.right = self.__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
            temp = self.__FindMin(root.right)
            root.value = temp.value
            root.right = self.__Delete(root.right,temp.value)
        return root
    def Search(self,value):
        return self.__Search(self.root,value)
    def __Search(self,root,value):
        if root is None or root.value == value:
            return root
        if root.value < value:
            return self.__Search(root.right,value)
        return self.__Search(root.left,value)
a=BST()
a.Insert(1)
a.Insert(3)
a.Insert(2)
a.Insert(40)
a.Insert(50)
a.Insert(0.5)
print("Height is ",a.Height())
print("Max is ",a.FindMax())
print("Min is ",a.FindMin())
print("Sucessor is",a.Successor())
print("Predecessor is",a.Predecessor())
print("----------InOrder----------")
a.InOrder()
print("----------PostOrder----------")
a.PostOrder()
print("----------PreOrder----------")
a.PreOrder()
a.Delete(50)
print("----------InOrder after deleting 50----------")
a.InOrder()
a.Search(40)
```

```
Height is  4
Max is  50
Min is  0.5
Sucessor is 2
Predecessor is 0.5
----------InOrder----------
0.5
1
2
3
40
50
----------PostOrder----------
0.5
2
50
40
3
1
----------PreOrder----------
1
0.5
3
2
40
50
----------InOrder after deleting 50----------
0.5
1
2
3
40
```

Out[10]: 40