# Binary Search Tree

March 16, 2021

Binary Search Tree

[ ]:
```
class Binary_Search_Tree:

    """
    Constructor with vaue we are going
    to insert in tree with assigning
    left and right child with default None
    """

    def __init__(self, data):
        self.data = data
        self.Left_child = None
        self.Right_child = None

    """
    If the data we are inserting already
    present in tree it will not add it
    to avoid the duplicate values
    """

    def Add_Node(self, data):
        if data == self.data:
            return # node already exist

        """
        If the data we are inserting is Less
        than the value of the current node, then
        data will insert in Left node
        """

        if data < self.data:
            if self.Left_child:
                self.Left_child.Add_Node(data)
            else:
                self.Left_child = Binary_Search_Tree(data)

            """
```

```
            If the data we are inserting is Greater
            than the value of the current node, then
            data will insert in Right node
            """

        else:
            if self.Right_child:
                self.Right_child.Add_Node(data)
            else:
                self.Right_child = Binary_Search_Tree(data)


    def Find_Node(self, val):

        """
        If current node is equal to
        data we are finding return true
        """

        if self.data == val:
            return True

        """
        If current node is lesser than
        data we are finding we have search
        in Left child node
        """

        if val < self.data:
            if self.Left_child:
                return self.Left_child.Find_Node(val)
            else:
                return False

        """
        If current node is Greater than
        data we are finding we have search
        in Right child node
        """

        if val > self.data:
            if self.Right_child:
                return self.Right_child.Find_Node(val)
            else:
                return False

    """
```

```
    First it will visit Left node then
    it will visit Root node and finally
    it will visit Right and display a
    list in specific order
    """

    def In_Order_Traversal(self):
        elements = []
        if self.Left_child:
            elements += self.Left_child.In_Order_Traversal()

        elements.append(self.data)

        if self.Right_child:
            elements += self.Right_child.In_Order_Traversal()

        return elements

    """
    First it will visit Left node then
    it will visit Right node and finally
    it will visit Root node  and display a
    list in specific order
    """

    def Post_Order_Traversal(self):
        elements = []
        if self.Left_child:
            elements += self.Left_child.Post_Order_Traversal()
        if self.Right_child:
            elements += self.Right_child.Post_Order_Traversal()

        elements.append(self.data)

        return elements

    """
    First it will visit Root node then
    it will visit Left node and finally
    it will visit Right node  and display a
    list in specific order
    """

    def Pre_Order_Traversal(self):
        elements = [self.data]
        if self.Left_child:
            elements += self.Left_child.Pre_Order_Traversal()
```

```
        if self.Right_child:
            elements += self.Right_child.Pre_Order_Traversal()

        return elements

    """
    This method will give
    the Max value of tree
    """

    def Find_Maximum_Node(self):
        if self.Right_child is None:
            return self.data
        return self.Right_child.Find_Maximum_Node()

    """
    This method will give
    the Min value of tree
    """

    def Find_Minimum_Node(self):
        if self.Left_child is None:
            return self.data
        return self.Left_child.Find_Minimum_Node()
```

[ ]:
```
"""
This method helps to build the tree
whith the element we inserted in it
"""
def Build_Tree(elements):
    root = Binary_Search_Tree(elements[0])

    for i in range(1,len(elements)):
        root.Add_Node(elements[i])

    return root
```

[ ]:
```
T=Build_Tree([10,99,1,55,26])
```

[ ]:
```
T.Find_Maximum_Node()
```

[ ]:
```
T.In_Order_Traversal()
```

[ ]:
```

```