NAMAL UNIVERSITY MIANWALI

DEPARTMENT OF ELECTRICAL ENGINEERING

DATA STRUCTURE AND ALGORITHM

LAB # 09

REPORT

Title : BST in Python

Name	*Fahim-Ur-Rehman Shah*
Roll No	*NIM-BSEE-2021-24*
Intructor	*Ms. Naureen Shaukat*
Lab Engineer	*Mr .Ali Hasnain*
Date	*29-May-2023*
Marks	

Instructions:

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a) Code (copy and pasted, NOT a screenshot)
    b) Output figure (as instructed in 3)
    c) Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

Task 1: Using above example1 code, insert 10 data elements and make a method (Inorder) for in-order traversal of data in BST. Also make a method (print_bst) to print the BST.

Python Code :

```
class BST:
def __init__(self, data):
    self.left = None
    self.right = None
    self.data = data

# Insert method to create nodes
def insert(self, data):
    if self.data:
        if data < self.data:
            if self.left is None:
                self.left = BST(data)
            else:
                self.left.insert(data)
        elif data > self.data:
            if self.right is None:
                self.right = BST(data)
            else:
                self.right.insert(data)
    else:
        self.data = data
```

```
    # In-order traversal
    def inorder(self):
        if self.left:
            self.left.inorder()
        print(self.data)
        if self.right:
            self.right.inorder()

    # Print the BST
    def print_bst(self):
        self._print_bst(self, 0)

    def _print_bst(self, node, level):
        if node is not None:
            self._print_bst(node.right, level + 1)
            print('    ' * level + str(node.data))
            self._print_bst(node.left, level + 1)

# Create the BST and insert elements
root = BST(54)
root.insert(34)
root.insert(46)
root.insert(12)
root.insert(23)
root.insert(5)
root.insert(18)
root.insert(80)
root.insert(62)
root.insert(45)

print("In-order Traversal of Binary Search Tree:")
root.inorder()

print("BST Structure:")
root.print_bst()
```

OUTPUT:

```
lgorithm\Lab\Lab 09\Task_01"
In-order Traversal of Binary Search Tree:
5
12
18
23
34
45
46
54
62
80
BST Structure:
    80
        62
54
        46
            45
    34
            23
                18
        12
            5
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 09>
```

Explanation :

In this updated code, the BST class now includes two additional methods: inorder and print_bst.

The inorder method performs an in-order traversal of the BST, which visits the nodes in ascending order. It recursively calls itself to traverse the left subtree, then prints the current node's data, and finally traverses the right subtree.

The print_bst method prints the structure of the BST in a tree-like format. It uses an auxiliary method _print_bst which performs a right-to-left in-order traversal to print the nodes in a reverse order, allowing the tree to be displayed correctly. It uses indentation to represent the levels of the tree.

After inserting the elements into the BST, the code demonstrates the in-order traversal by calling root.inorder() and prints the structure of the BST by calling root.print_bst().

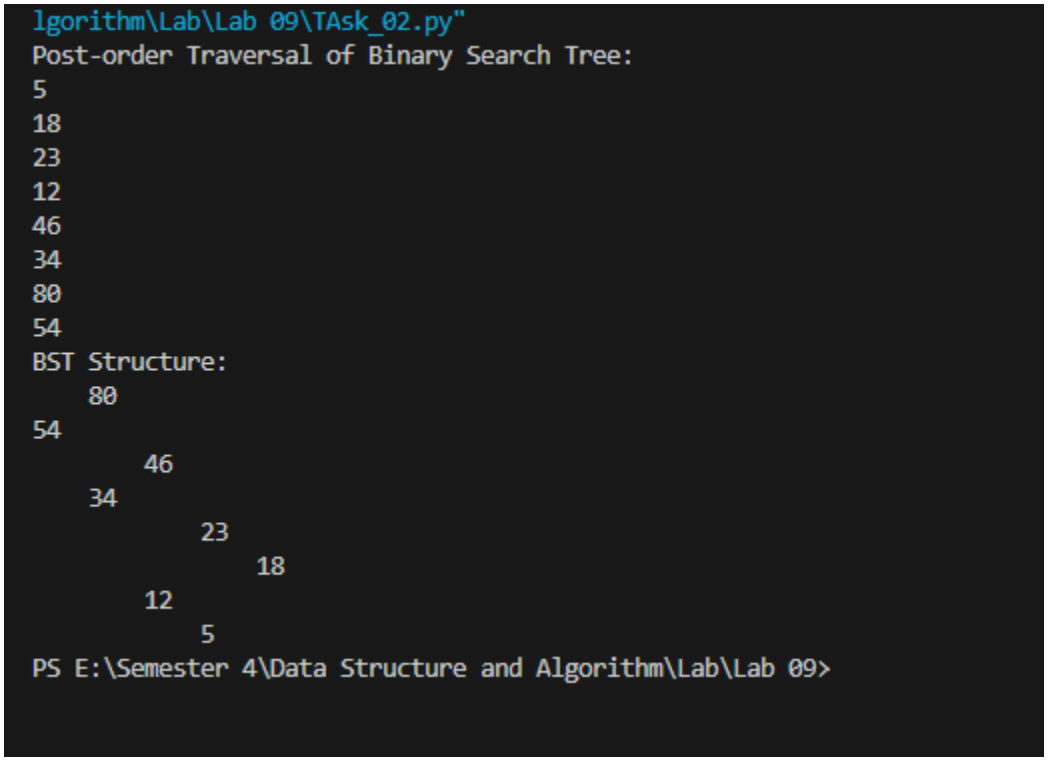
Task 2: Using above example2 code, insert 8 data elements and make a method (Postorder) for post-order traversal of data in BST. Also make a method (print_bst) to print the BST.

Python Code :

```
   class BST:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

    # Insert method to create nodes
    def insert(self, data):
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = BST(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = BST(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data

    # Post-order traversal
    def postorder(self):
        if self.left:
            self.left.postorder()
        if self.right:
            self.right.postorder()
        print(self.data)

    # Print the BST
    def print_bst(self):
        self._print_bst(self, 0)

    def _print_bst(self, node, level):
        if node is not None:
            self._print_bst(node.right, level + 1)
            print('    ' * level + str(node.data))
            self._print_bst(node.left, level + 1)

# Create the BST and insert elements
root = BST(54)
root.insert(34)
root.insert(46)
root.insert(12)
```

```
root.insert(23)
root.insert(5)
root.insert(18)
root.insert(80)

print("Post-order Traversal of Binary Search Tree:")
root.postorder()

print("BST Structure:")
root.print_bst()
```

OUTPUT:



Explanation :

In this updated code, the BST class now includes two additional methods: postorder and print_bst.

The postorder method performs a post-order traversal of the BST, which visits the nodes in the following order: left subtree, right subtree, and finally the current node. It recursively calls itself to traverse the left and right subtrees, and then prints the data of the current node.

The print_bst method remains the same as in the previous code and prints the structure of the BST in a tree-like format.

After inserting the elements into the BST, the code demonstrates the post-order traversal by calling root.postorder() and prints the structure of the BST by calling root.print_bst().

Task 3: Using above code, insert 15 data elements and make a method (delete) to delete the nodes from BST. Also make a method (print_bst) to print the BST before and after deletion of data.

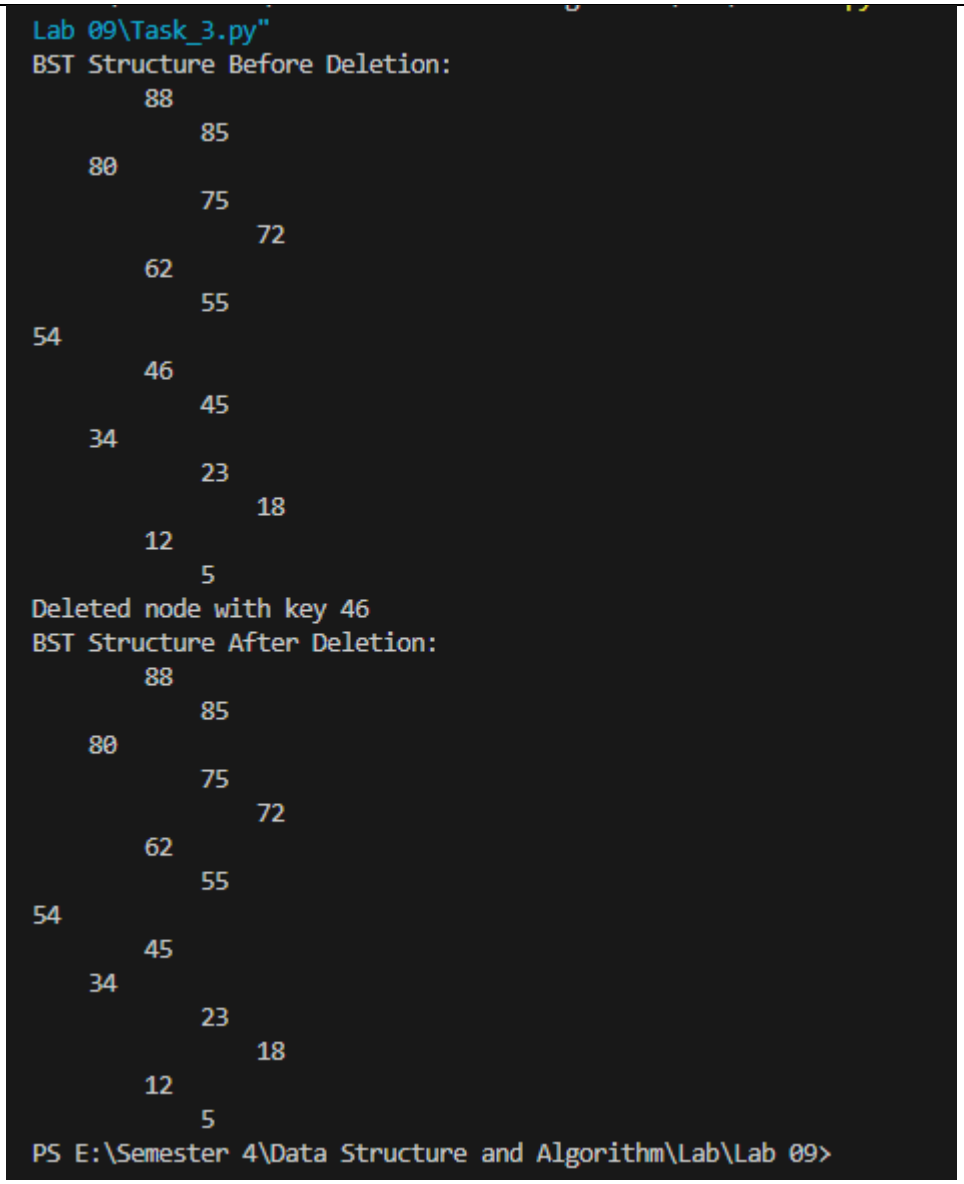
Python Code :

```
class BST:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

    # Insert method to create nodes
    def insert(self, data):
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = BST(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = BST(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data

    # Post-order traversal
    def postorder(self):
        if self.left:
            self.left.postorder()
        if self.right:
            self.right.postorder()
        print(self.data)

    # Print the BST
    def print_bst(self):
        self._print_bst(self, 0)

    def _print_bst(self, node, level):
        if node is not None:
```

```
            self._print_bst(node.right, level + 1)
            print('    ' * level + str(node.data))
            self._print_bst(node.left, level + 1)

    # Delete a node from the BST
    def delete(self, key):
        if self.data is None:
            return self

        # If the key is smaller than the root's data, go left
        if key < self.data:
            self.left = self.left.delete(key)
        # If the key is greater than the root's data, go right
        elif key > self.data:
            self.right = self.right.delete(key)
        # If the key is equal to the root's data, delete the node
        else:
            # Node with only one child or no child
            if self.left is None:
                temp = self.right
                self = None
                return temp
            elif self.right is None:
                temp = self.left
                self = None
                return temp
            # Node with two children: Get the inorder successor (smallest in the right
subtree)
            temp = self.right
            while temp.left is not None:
                temp = temp.left
            # Copy the inorder successor's content to the current node
            self.data = temp.data
            # Delete the inorder successor
            self.right = self.right.delete(temp.data)

        return self


# Create the BST and insert elements
root = BST(54)
root.insert(34)
root.insert(46)
root.insert(12)
root.insert(23)
root.insert(5)
```

```
root.insert(18)
root.insert(80)
root.insert(62)
root.insert(45)
root.insert(55)
root.insert(75)
root.insert(88)
root.insert(72)
root.insert(85)

print("BST Structure Before Deletion:")
root.print_bst()

# Delete a node from the BST
key_to_delete = 46
root = root.delete(key_to_delete)
print(f"Deleted node with key {key_to_delete}")

print("BST Structure After Deletion:")
root.print_bst()
```

Output:



Explanation :

In this updated code, the BST class now includes a delete method and an example of deleting a node from the BST.

The delete method is used to delete a node from the BST. It follows the rules of BST deletion:

- If the key to be deleted is smaller than the root's data, go to the left subtree.
- If the key to be deleted is greater than the root's data, go to the right subtree.
- If the key to be deleted is equal to the root's data,

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-5**	**Marginal 5-10**	**Satisfactory 11-15**	**Exemplary 16-20**	**Allocated Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested by others. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend for the instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is not given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is not given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is not given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around.Chased others, ran, or played around. More than two incidents of talking non-lab related stuff in laband/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff inlab. Voice level exceeding theappropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceedingthe appropriate level, use of cell phones and involvementin any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Tookcare of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	