

# ADVANCE DATA STRUCTURES AND ALGORITHMS (ADSA)

Prabhnoor Singh

102115059

3NC3

## ASSIGNMENT 1

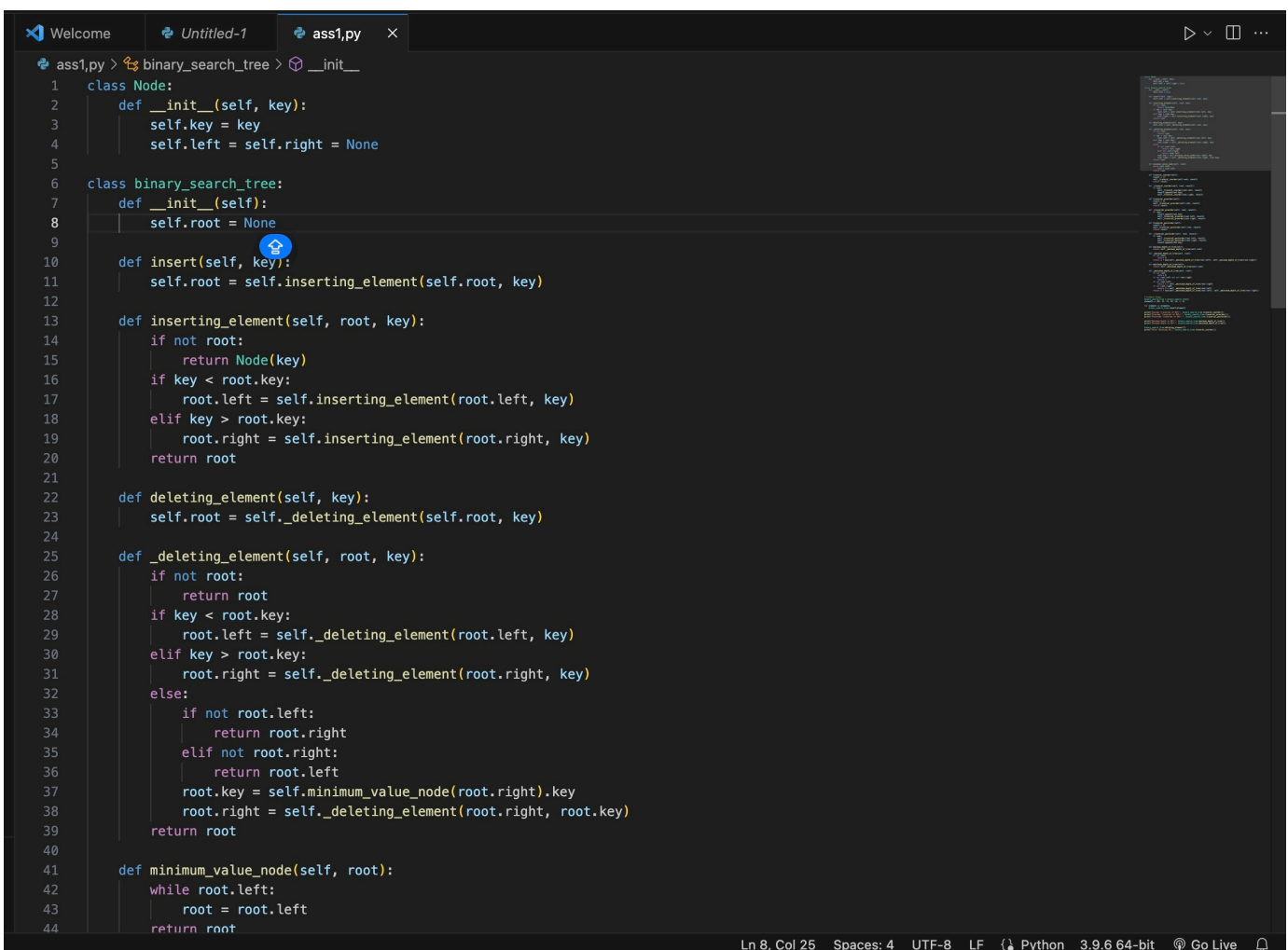
Write a program for a Binary Search Tree (BST) having functions for the following operations:

Insert an element (no duplicates are allowed),

Delete an existing element,

Traverse the BST (in-order, pre-order, and post-order), Maximum depth, and Minimum depth.

CODE WRITTEN VSCODE (PYTHON) WITH OUTPUT :



```
1 class Node:
2     def __init__(self, key):
3         self.key = key
4         self.left = self.right = None
5
6 class binary_search_tree:
7     def __init__(self):
8         self.root = None
9
10    def insert(self, key):
11        self.root = self.inserting_element(self.root, key)
12
13    def inserting_element(self, root, key):
14        if not root:
15            return Node(key)
16        if key < root.key:
17            root.left = self.inserting_element(root.left, key)
18        elif key > root.key:
19            root.right = self.inserting_element(root.right, key)
20        return root
21
22    def deleting_element(self, key):
23        self.root = self._deleting_element(self.root, key)
24
25    def _deleting_element(self, root, key):
26        if not root:
27            return root
28        if key < root.key:
29            root.left = self._deleting_element(root.left, key)
30        elif key > root.key:
31            root.right = self._deleting_element(root.right, key)
32        else:
33            if not root.left:
34                return root.right
35            elif not root.right:
36                return root.left
37            root.key = self.minimum_value_node(root.right).key
38            root.right = self._deleting_element(root.right, root.key)
39        return root
40
41    def minimum_value_node(self, root):
42        while root.left:
43            root = root.left
44        return root
```

```

Welcome  Untitled-1  ass1.py  x
ass1.py > binary_search_tree > inserting_element
75         self._traversal_postorder(root.left, result)
76         self._traversal_postorder(root.right, result)
77         result.append(root.key)
78
79     def maximum_depth_of_tree(self):
80         return self._maximum_depth_of_tree(self.root)
81
82     def _maximum_depth_of_tree(self, root):
83         if not root:
84             return 0
85         return 1 + max(self._maximum_depth_of_tree(root.left), self._maximum_depth_of_tree(root.right))
86
87     def mminimum_depth_of_tree(self):
88         return self._mminimum_depth_of_tree(self.root)
89
90     def _mminimum_depth_of_tree(self, root):
91         if not root:
92             return 0
93         if not root.left and not root.right:
94             return 1
95         if not root.left:
96             return 1 + self._mminimum_depth_of_tree(root.right)
97         if not root.right:
98             return 1 + self._mminimum_depth_of_tree(root.left)
99         return 1 + min(self._mminimum_depth_of_tree(root.left), self._mminimum_depth_of_tree(root.right))
100
101
102 # Example Usage:
103 binary_search_tree = binary_search_tree()
104 elements = [56, 23, 74, 52, 18, 7, 8]
105
106 for element in elements:
107     binary_search_tree.insert(element)
108
109 print("Inorder Traversal of BST:", binary_search_tree.traversal_inorder())
110 print("Preorder Traversal of BST::", binary_search_tree.traversal_preorder())
111 print("Postorder Traversal of BST::", binary_search_tree.traversal_postorder())
112
113 print("Maximum Depth in BST:", binary_search_tree.maximum_depth_of_tree())
114 print("Minimum Depth in BST:", binary_search_tree.mminimum_depth_of_tree())
115
116 binary_search_tree.deleting_element(7)
117 print("After deleting 30:", binary_search_tree.traversal_inorder())
118
Ln 20, Col 20  Spaces: 4  UTF-8  LF  Python  3.9.6 64-bit  Go Live
```

```

Welcome  Untitled-1  ass1.py  x
ass1.py > binary_search_tree > inserting_element
42     while root.left:
43         root = root.left
44     return root
45
46     def traversal_inorder(self):
47         result = []
48         self._traversal_inorder(self.root, result)
49         return result
50
51     def _traversal_inorder(self, root, result):
52         if root:
53             self._traversal_inorder(root.left, result)
54             result.append(root.key)
55             self._traversal_inorder(root.right, result)
56
57     def traversal_preorder(self):
58         result = []
59         self._traversal_preorder(self.root, result)
60         return result
61
62     def _traversal_preorder(self, root, result):
63         if root:
64             result.append(root.key)
65             self._traversal_preorder(root.left, result)
66             self._traversal_preorder(root.right, result)
67
68     def traversal_postorder(self):
69         result = []
70         self._traversal_postorder(self.root, result)
71         return result
72
73     def _traversal_postorder(self, root, result):
74         if root:
75             self._traversal_postorder(root.left, result)
76             self._traversal_postorder(root.right, result)
77             result.append(root.key)
78
79     def maximum_depth_of_tree(self):
80         return self._maximum_depth_of_tree(self.root)
81
82     def _maximum_depth_of_tree(self, root):
83         if not root:
84             return 0
85         return 1 + max(self._maximum_depth_of_tree(root.left), self._maximum_depth_of_tree(root.right))
86
Ln 20, Col 20  Spaces: 4  UTF-8  LF  Python  3.9.6 64-bit  Go Live
```

## OUTPUT GENERATED :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE
shailymandal@Shailys-MacBook-Air adsa % cd /Users/shailymandal/Desktop/daa/adsa ; /usr/bin/env /usr/bin/python3 /Users/shailymandal/.vscode/extensions/ms-python.python-2023.20.0/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 52552 -- /Users/shailymandal/Desktop/daa/adsa/ass1.py
Inorder Traversal of BST: [7, 8, 18, 23, 52, 56, 74]
Preorder Traversal of BST:: [56, 23, 18, 7, 8, 52, 74]
Postorder Traversal of BST:: [8, 7, 18, 52, 23, 74, 56]
Maximum Depth in BST: 5
Minimum Depth in BST: 2
After deleting element: [8, 18, 23, 52, 56, 74]
shailymandal@Shailys-MacBook-Air adsa %
```

## CODE :

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

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

    def insert(self, key):
        self.root = self.inserting_element(self.root, key)

    def inserting_element(self, root, key):
        if not root:
            return Node(key)
        if key < root.key:
            root.left = self.inserting_element(root.left, key)
        elif key > root.key:
            root.right = self.inserting_element(root.right, key)
        return root

    def deleting_element(self, key):
        self.root = self._deleting_element(self.root, key)

    def _deleting_element(self, root, key):
        if not root:
            return root
        if key < root.key:
            root.left = self._deleting_element(root.left, key)
        elif key > root.key:
            root.right = self._deleting_element(root.right, key)
        else:
            if not root.left:
                return root.right
            elif not root.right:
                return root.left
            root.key = self.minimum_value_node(root.right).key
            root.right = self._deleting_element(root.right,
root.key)
```

```
    return root
```

```
def minimum_value_node(self, root):  
    while root.left:  
        root = root.left  
    return root
```

```
def traversal_inorder(self):  
    result = []  
    self._traversal_inorder(self.root, result)  
    return result
```

```
def _traversal_inorder(self, root, result):  
    if root:  
        self._traversal_inorder(root.left, result)  
        result.append(root.key)  
        self._traversal_inorder(root.right, result)
```

```
def traversal_preorder(self):  
    result = []  
    self._traversal_preorder(self.root, result)  
    return result
```

```
def _traversal_preorder(self, root, result):  
    if root:  
        result.append(root.key)  
        self._traversal_preorder(root.left, result)  
        self._traversal_preorder(root.right, result)
```

```
def traversal_postorder(self):  
    result = []  
    self._traversal_postorder(self.root, result)  
    return result
```

```
def _traversal_postorder(self, root, result):  
    if root:  
        self._traversal_postorder(root.left, result)  
        self._traversal_postorder(root.right, result)  
        result.append(root.key)
```

```
def maximum_depth_of_tree(self):  
    return self._maximum_depth_of_tree(self.root)
```

```
def _maximum_depth_of_tree(self, root):  
    if not root:  
        return 0  
    return 1 + max(self._maximum_depth_of_tree(root.left),  
self._maximum_depth_of_tree(root.right))
```

```
def mminimum_depth_of_tree(self):  
    return self._mminimum_depth_of_tree(self.root)
```

```

def _mminimum_depth_of_tree(self, root):
    if not root:
        return 0
    if not root.left and not root.right:
        return 1
    if not root.left:
        return 1 + self._mminimum_depth_of_tree(root.right)
    if not root.right:
        return 1 + self._mminimum_depth_of_tree(root.left)
    return 1 + min(self._mminimum_depth_of_tree(root.left),
self._mminimum_depth_of_tree(root.right))

```

# Example Usage:

```

binary_search_tree = binary_search_tree()
elements = [56, 23, 74, 52, 18, 7, 8]

```

```

for element in elements:
    binary_search_tree.insert(element)

```

```

print("Inorder Traversal of BST:",
binary_search_tree.traversal_inorder())
print("Preorder Traversal of BST::",
binary_search_tree.traversal_preorder())
print("Postorder Traversal of BST::",
binary_search_tree.traversal_postorder())

```

```

print("Maximum Depth in BST:",
binary_search_tree.maximum_depth_of_tree())
print("Minimum Depth in BST:",
binary_search_tree.mminimum_depth_of_tree())

```

```

binary_search_tree.deleting_element(7)
print("After deleting 30:", binary_search_tree.traversal_inorder())

```