Module 6 Critical Thinking: Binary Search Tree

Donghwan Kim

Colorado State University Global

CSC 506: Design and Analysis of Algorithms

Dr. Lori Farr

June 1, 2022

Module 6 Critical Thinking: Binary Search Tree

This Critical Assignment assignment builds a simple binary search tree. It includes the source code and screenshots of compilation and results.

Source Code

The Python source code is as follows:

```
# Module 6
# build a binary search tree

# sort function
def insertion_sort(A):
    #input: a list
    for i in range(1, len(A)):
        for j in range(i, 0, -1):
            if A[j] < A[j-1]:
                A[j], A[j-1] = A[j-1], A[j]
            else:
                break

# node class
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# tree class
class Tree:
    def __init__(self, X):
```

```
    self.X = X

def Build_Tree(self):
    """ find the root which used in the Tree_Struc() method """
    return self.X[len(self.X)//2]

def Tree_Struc(self):
    """ Construct a tree with the root of the result of Build_Tree()"""
    root = Node(self.Build_Tree())
    for x in self.X:
        if x != self.Build_Tree():
            self.Insert(root, Node(x))
    print("THE TREE IS BUILT")
    print("IN-ORDER PRINT IS")
    self.InOrderPrint(root)

def Insert(self, root, node):
    """ Insert node, used in Tree_Struc() """
    if root.data > node.data:
        if root.left is None:
            root.left = node
        else:
            self.Insert(root.left, node)
    else:
        if root.right is None:
            root.right = node
        else:
            self.Insert(root.right, node)
```

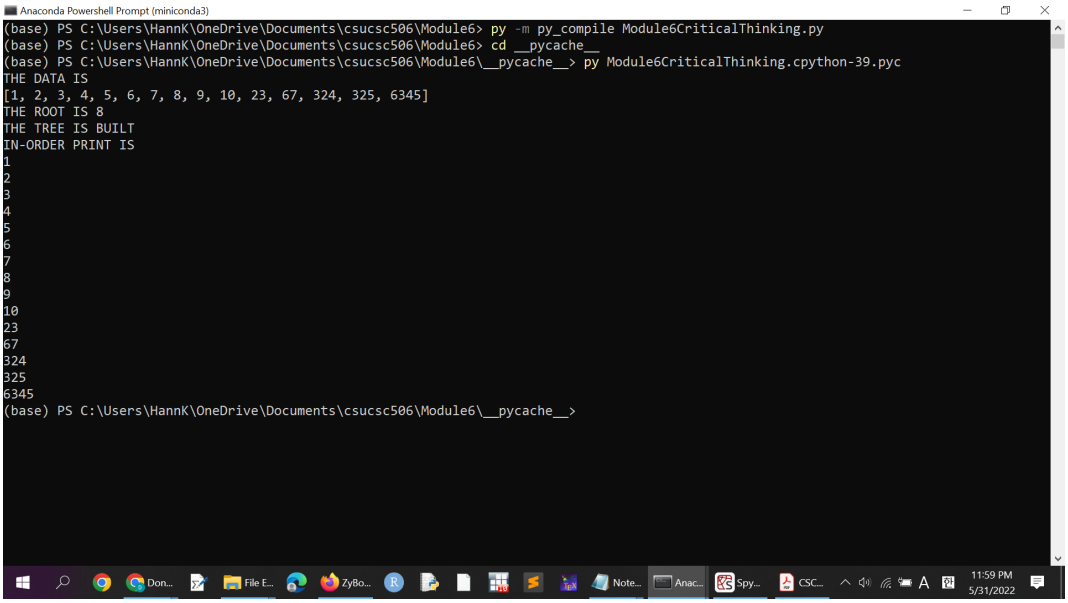
```
def InOrderPrint(self, root):
    if not root:
        return
    self.InOrderPrint(root.left)
    print(root.data)
    self.InOrderPrint(root.right)

# slightly revised one from the textbook
def Delete(self, x):
    parent = None
    root = Node(self.Build_Tree())
    current = root

    while current is not None:

        if current.data == x:
            if current.left is None and current.right is None:
                if parent is None:
                    root = None
                elif parent.left is current:
                    parent.left = None
                else:
                    parent.right = None
                return

            elif current.left is not None and current.right is None:
                if parent is None:
                    root = current.left
                elif parent.left is current:
```

```
                parent.left = current.left
            else:
                parent.right = current.left
            return

        elif current.left is None and current.right is not None:
            if parent is None:
                root = current.right
            elif parent.left is current:
                parent.left = current.right
            else:
                parent.right = current.right
            return

        else:
            successor = current.right
            while successor.left is not None:
                successor = successor.left
            current.data = successor.data
            parent = current
            current = current.right
            x = parent.data
    elif current.data < x:
        parent = current
        current = current.right

    else:
        parent = current
        current = current.left
```

```
X = [1, 7, 4, 10, 23, 6, 325, 8, 9, 2, 4, 3, 5, 7, 9, 67, 6345, 324]
X=list(set(X))
insertion_sort(X)

print("THE DATA IS")
print(X)
t = Tree(X)
print("THE ROOT IS", t.Build_Tree())
t.Tree_Struc()
t.Delete(6345)
```

Results

It uses the **py_compile** module to compile and execute the source code on Windows, as shown below.



Conclusions

This assignment builds a binary search tree using Python classes. It includes the source code and results.