COMPETITIVE PROGRAMMING

Assignment-9

2303A51463

B-07

Exploring Applications of Advanced Tree Data Structures

Database Indexing Using B+ Tree

Algorithm:

1. B+ Tree Initialization

1. Create an empty B+ Tree.
2. Set the root as a **leaf node**.
3. Maximum keys in any node = `m - 1`.

2. Algorithm to Insert a Key

1. Start from the root node.
2. If the current node is **not a leaf**:
    a. Find the correct child pointer where the key should go.
    b. Move to that child.
3. If the current node is a **leaf**:
    a. Insert the key in sorted order.
4. If the number of keys exceeds `m - 1`:
    a. Split the node into two.
    b. Promote the middle key to the parent.
5. If the parent overflows:
    a. Repeat the split process upward.
6. If the root splits:
    a. Create a new root.

3. Algorithm to Search a Key

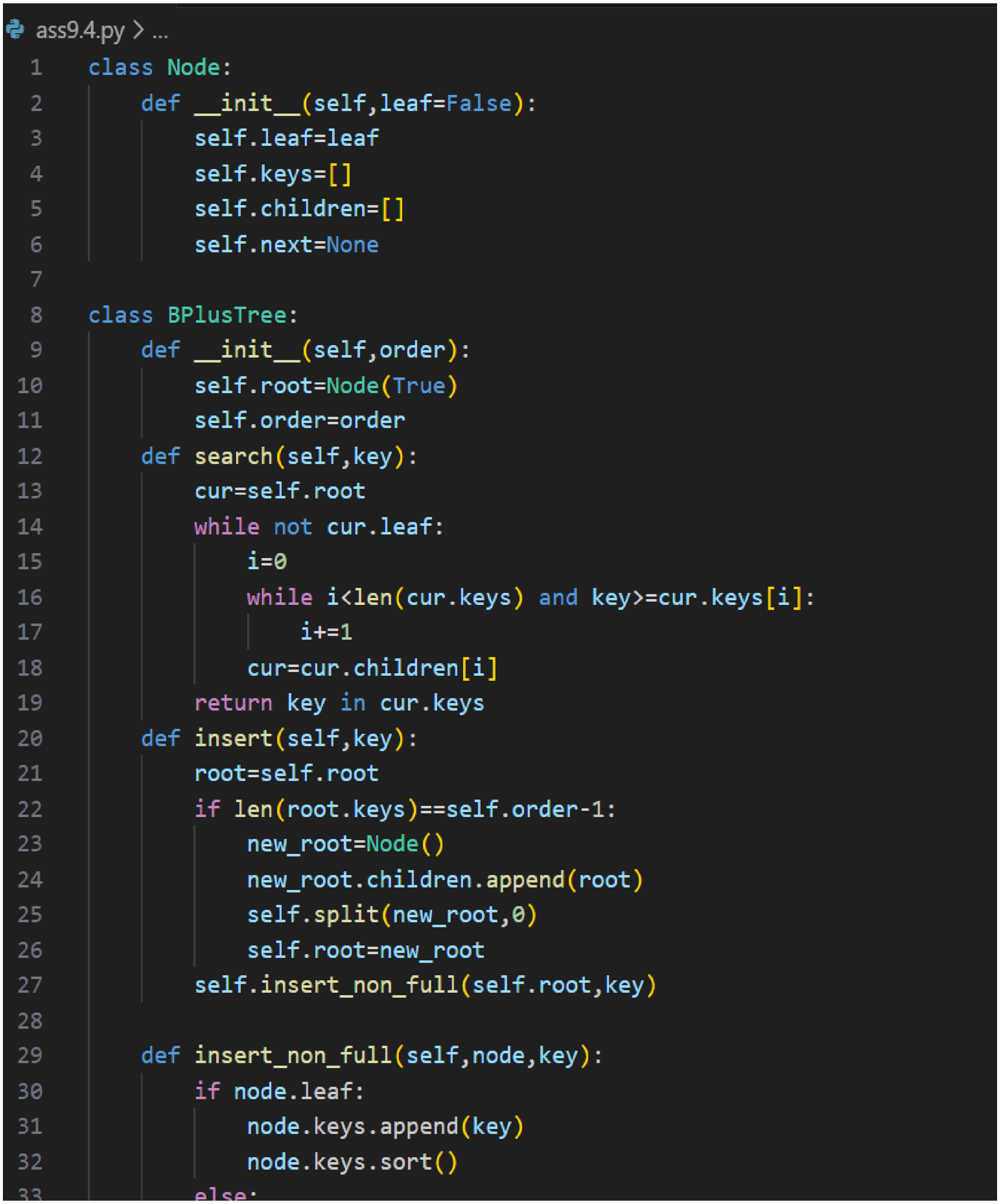
1. Start at the root.
2. While the current node is **not a leaf**:
    a. Compare the key with index keys.
    b. Follow the appropriate child pointer.
3. At the leaf node:
    a. If the key exists → **FOUND**
    b. Else → **NOTFOUND**

4. Algorithm to Delete a Key

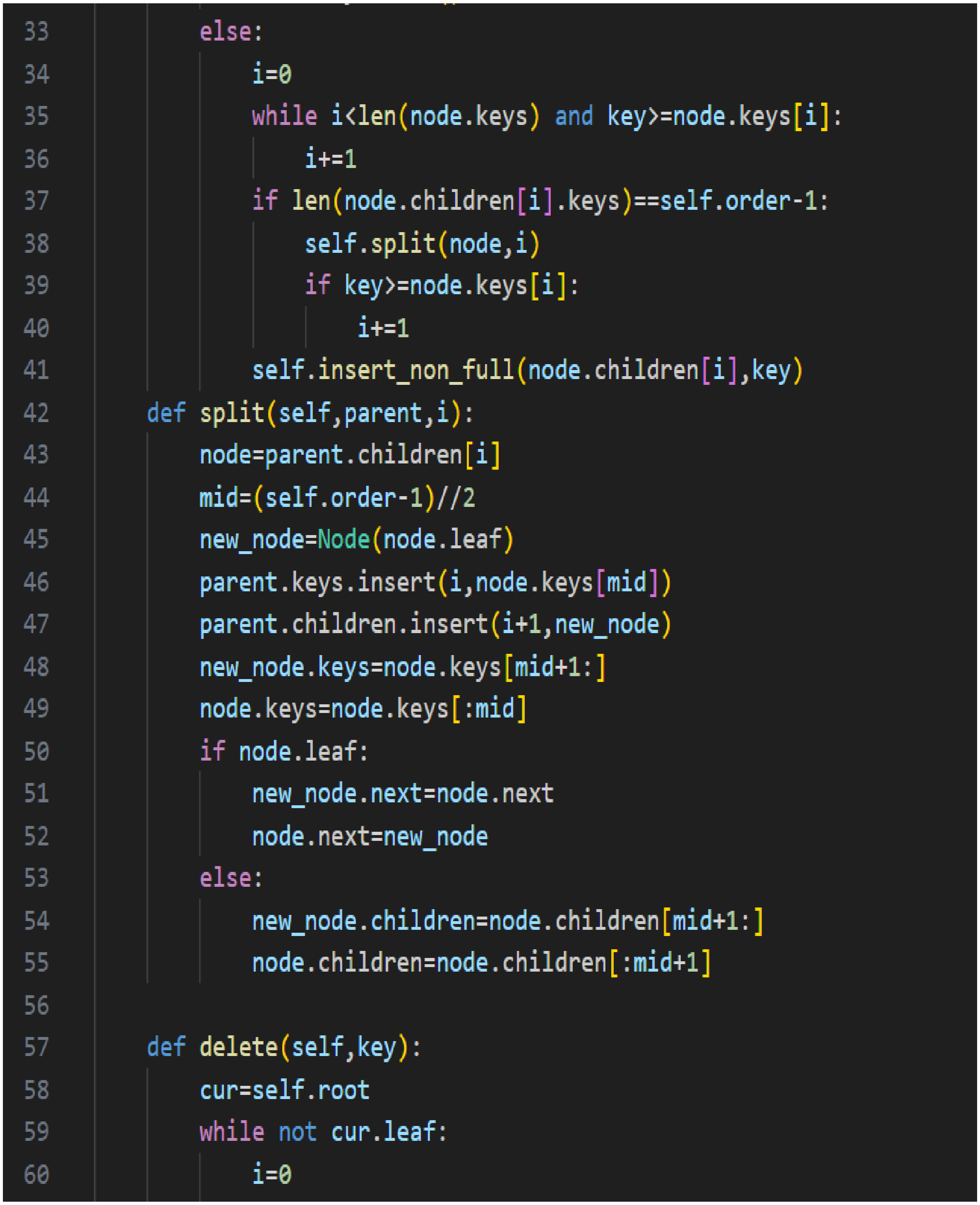
1. Start at the root.
2. Traverse to the appropriate **leaf node**.
3. Remove the key from the leaf.
4. If the node has fewer than the minimum required keys:
    a. Try to **borrow** a key from a sibling.
    b. If borrowing is not possible, **merge** with a sibling.
5. Update parent index keys if required.
6. If the root becomes empty:
    a. Replace root with its child.

5. Algorithm for Leaf-Level Traversal

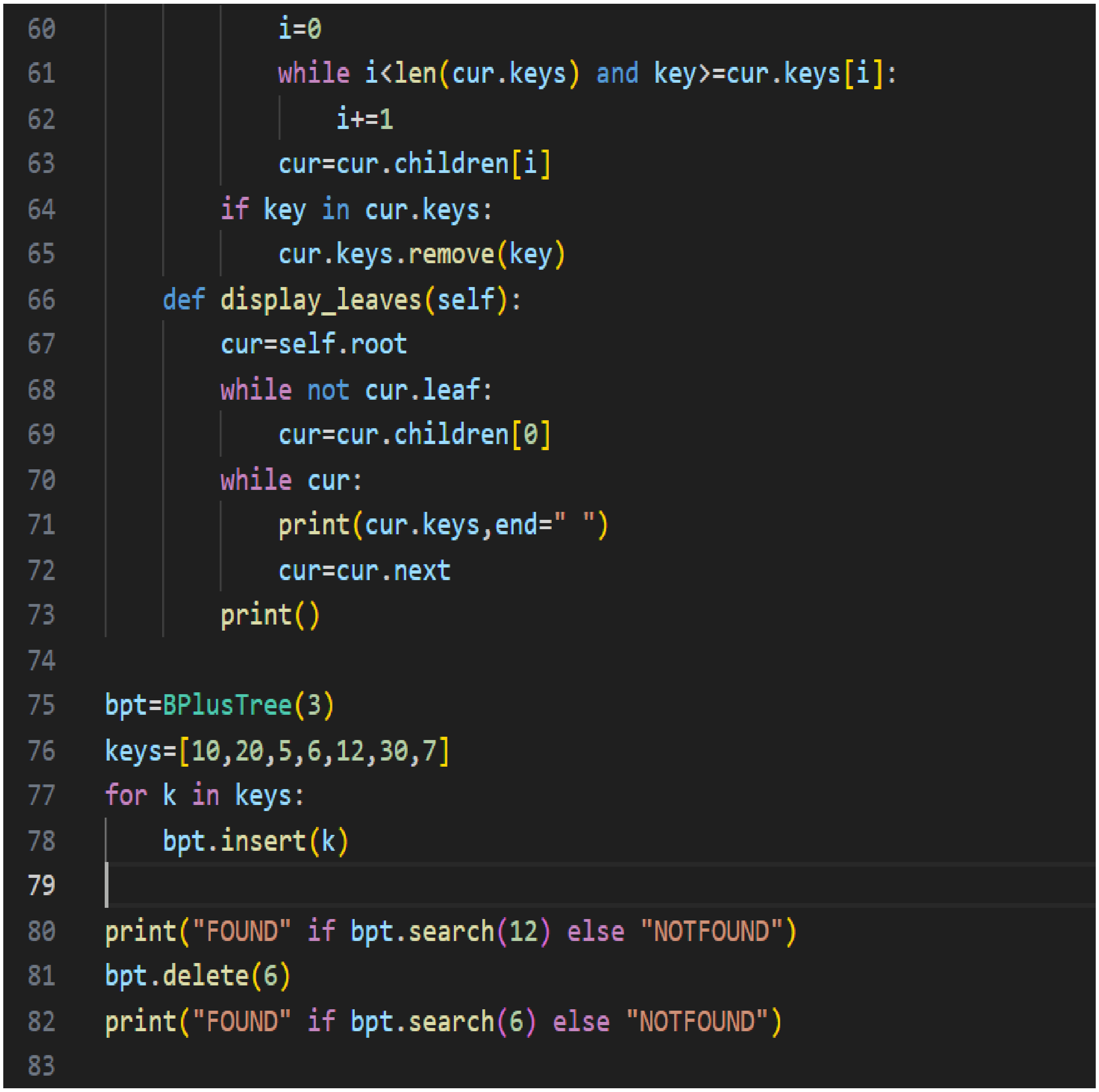
1. Move to the **leftmost leaf node**.
2. Print all keys in the current leaf.
3. Follow the leaf's `next` pointer.
4. Repeat until the last leaf node is reached.



```python
class Node:
    def __init__(self,leaf=False):
        self.leaf=leaf
        self.keys=[]
        self.children=[]
        self.next=None

class BPlusTree:
    def __init__(self,order):
        self.root=Node(True)
        self.order=order
    def search(self,key):
        cur=self.root
        while not cur.leaf:
            i=0
            while i<len(cur.keys) and key>=cur.keys[i]:
                i+=1
            cur=cur.children[i]
        return key in cur.keys
    def insert(self,key):
        root=self.root
        if len(root.keys)==self.order-1:
            new_root=Node()
            new_root.children.append(root)
            self.split(new_root,0)
            self.root=new_root
        self.insert_non_full(self.root,key)

    def insert_non_full(self,node,key):
        if node.leaf:
            node.keys.append(key)
            node.keys.sort()
        else:
```

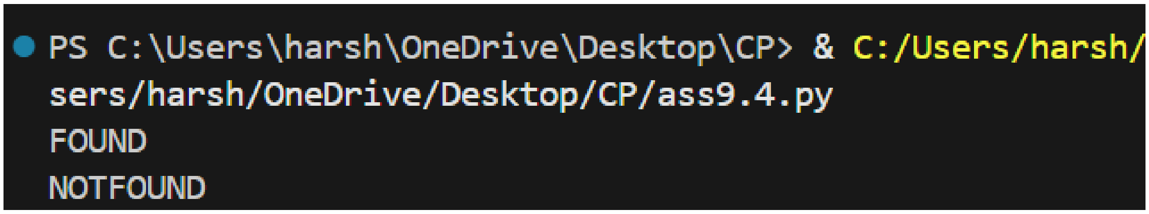


```
        else:
            i=0
            while i<len(node.keys) and key>=node.keys[i]:
                i+=1
            if len(node.children[i].keys)==self.order-1:
                self.split(node,i)
                if key>=node.keys[i]:
                    i+=1
            self.insert_non_full(node.children[i],key)
    def split(self,parent,i):
        node=parent.children[i]
        mid=(self.order-1)//2
        new_node=Node(node.leaf)
        parent.keys.insert(i,node.keys[mid])
        parent.children.insert(i+1,new_node)
        new_node.keys=node.keys[mid+1:]
        node.keys=node.keys[:mid]
        if node.leaf:
            new_node.next=node.next
            node.next=new_node
        else:
            new_node.children=node.children[mid+1:]
            node.children=node.children[:mid+1]

    def delete(self,key):
        cur=self.root
        while not cur.leaf:
            i=0
```



```
            i=0
            while i<len(cur.keys) and key>=cur.keys[i]:
                i+=1
            cur=cur.children[i]
        if key in cur.keys:
            cur.keys.remove(key)
    def display_leaves(self):
        cur=self.root
        while not cur.leaf:
            cur=cur.children[0]
        while cur:
            print(cur.keys,end=" ")
            cur=cur.next
        print()

bpt=BPlusTree(3)
keys=[10,20,5,6,12,30,7]
for k in keys:
    bpt.insert(k)

print("FOUND" if bpt.search(12) else "NOTFOUND")
bpt.delete(6)
print("FOUND" if bpt.search(6) else "NOTFOUND")

```

Output:



```
PS C:\Users\harsh\OneDrive\Desktop\CP> & C:/Users/harsh/
sers/harsh/OneDrive/Desktop/CP/ass9.4.py
FOUND
NOTFOUND
```

Recently Accessed File Optimization Using Splay Tree

Algorithm:

1. Insertion Algorithm

1. If the tree is empty, create a new node as root.
2. Otherwise, insert the file ID like a Binary Search Tree.
3. After insertion, **splay** the inserted node to the root.

2. Search Algorithm

1. Search the file ID as in a BST.
2. If found:
    a. Perform splaying to move the accessed node to the root.
    b. Print **FOUND**.
3. If not found:
    a. Print **NOTFOUND**.

3. Splaying Operation

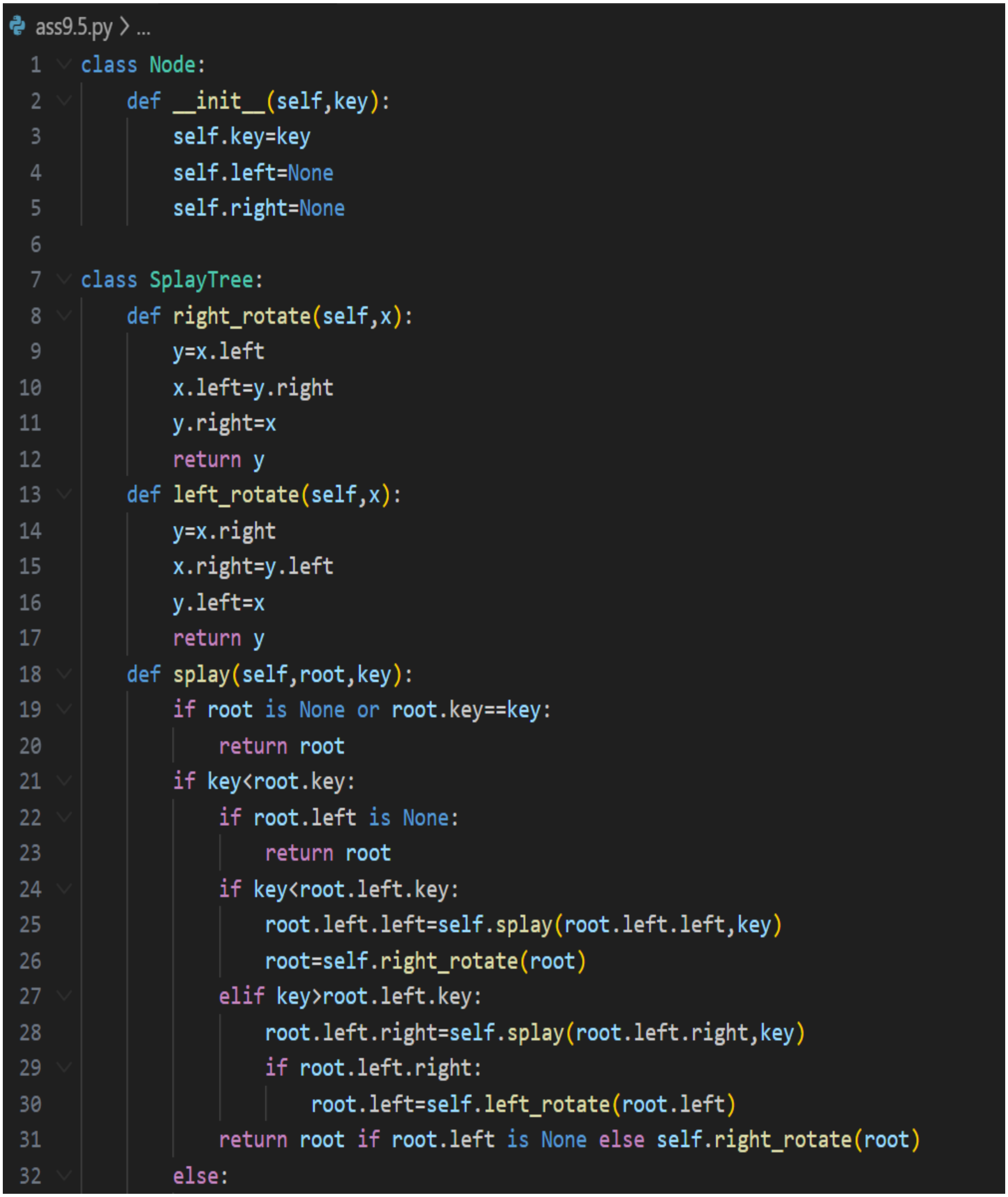
1. **Zig Rotation**: Node is child of root.
2. **Zig-Zig Rotation**: Node and parent are both left or both right children.
3. **Zig-Zag Rotation**: Node is left child and parent is right child (or vice versa).
4. Continue rotations until the node becomes the root.

4. Deletion Algorithm

1. Search and splay the node to be deleted to the root.
2. Remove the root.
3. Join left and right subtrees by splaying the maximum node of the left subtree.
4. Attach the right subtree.

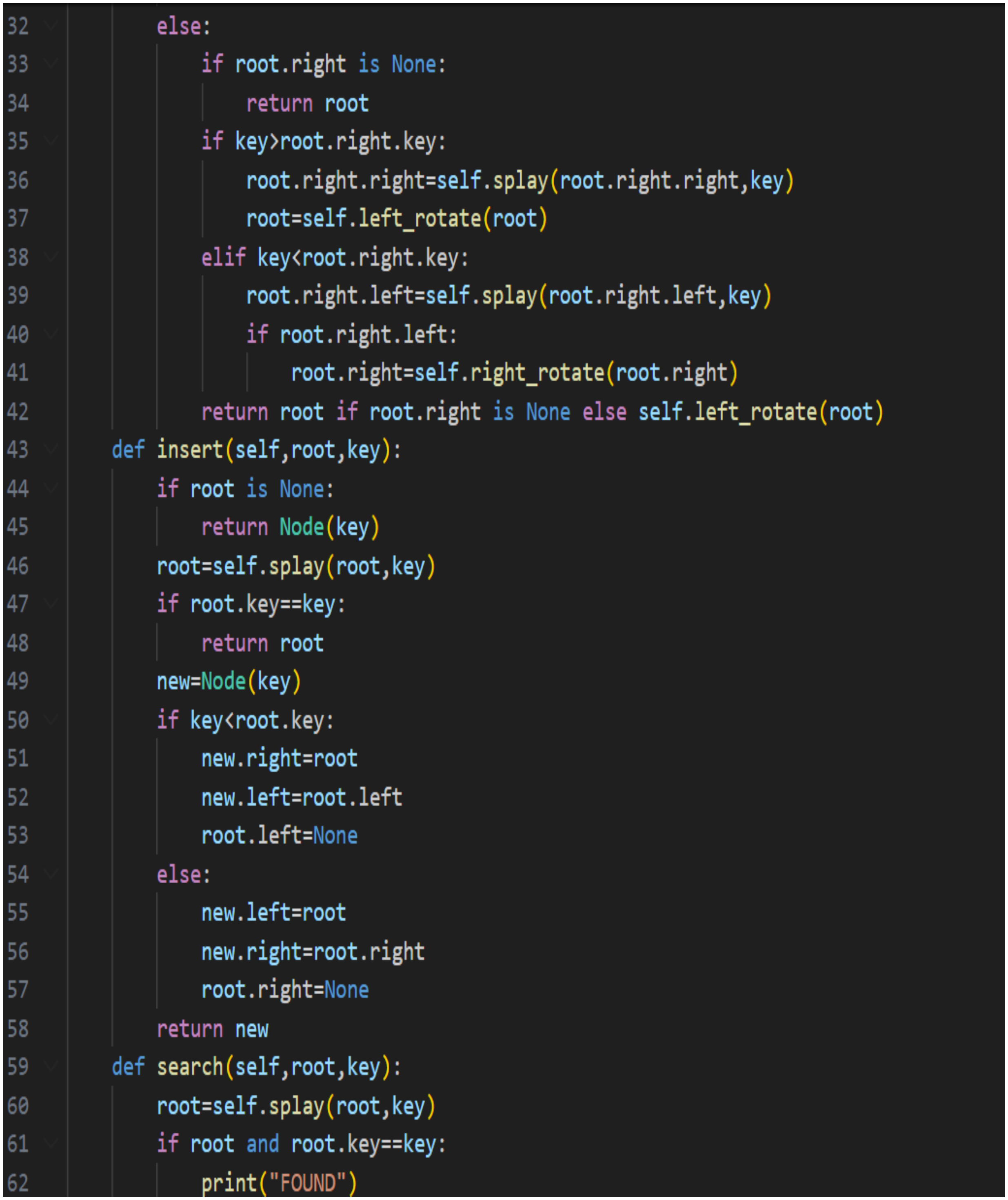
5. Display Algorithm

1. Perform **inorder traversal** to display the tree structure.

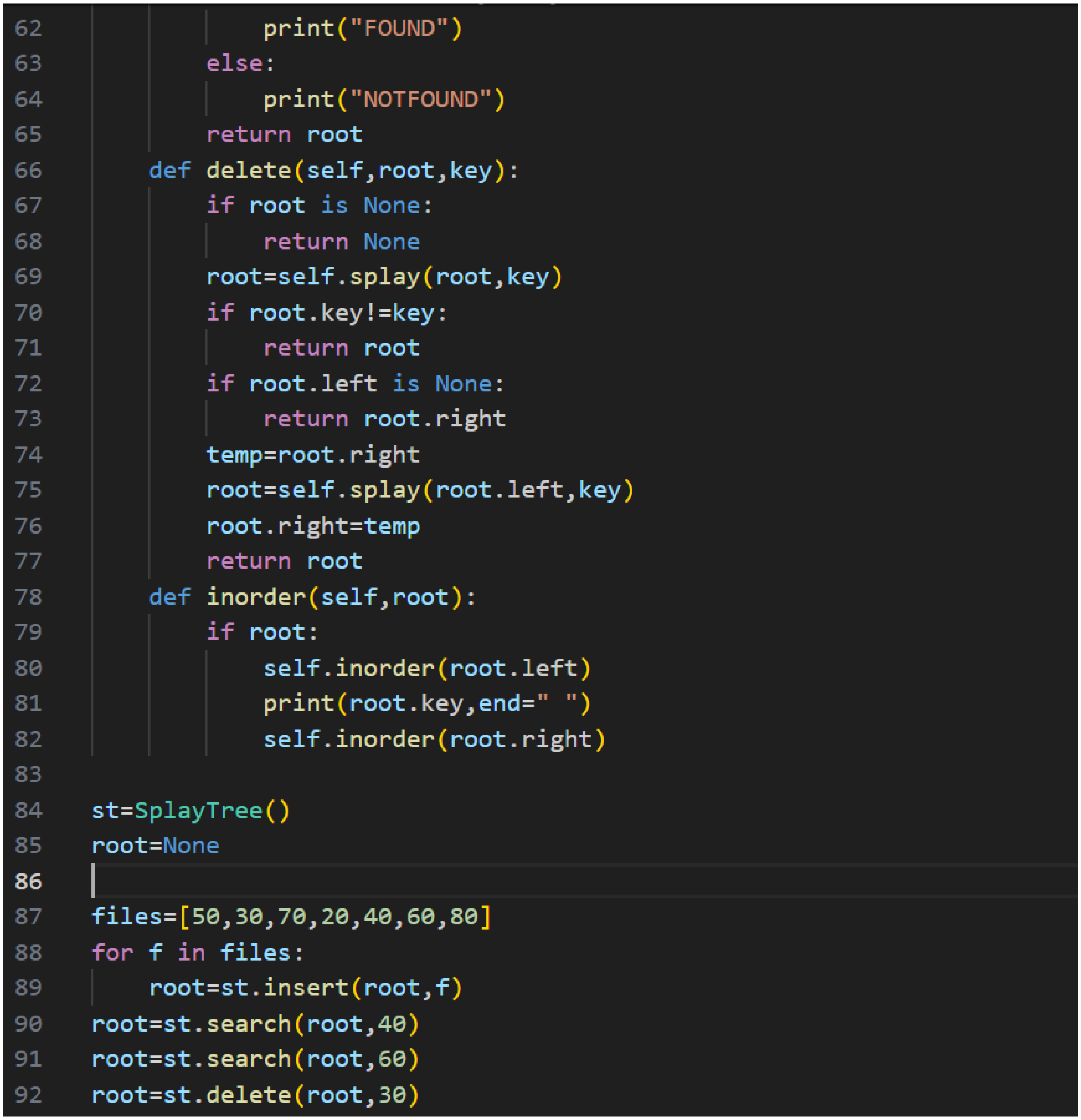


ass9.5.py > ...

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

class SplayTree:
    def right_rotate(self,x):
        y=x.left
        x.left=y.right
        y.right=x
        return y
    def left_rotate(self,x):
        y=x.right
        x.right=y.left
        y.left=x
        return y
    def splay(self,root,key):
        if root is None or root.key==key:
            return root
        if key<root.key:
            if root.left is None:
                return root
            if key<root.left.key:
                root.left.left=self.splay(root.left.left,key)
                root=self.right_rotate(root)
            elif key>root.left.key:
                root.left.right=self.splay(root.left.right,key)
                if root.left.right:
                    root.left=self.left_rotate(root.left)
            return root if root.left is None else self.right_rotate(root)
        else:
```

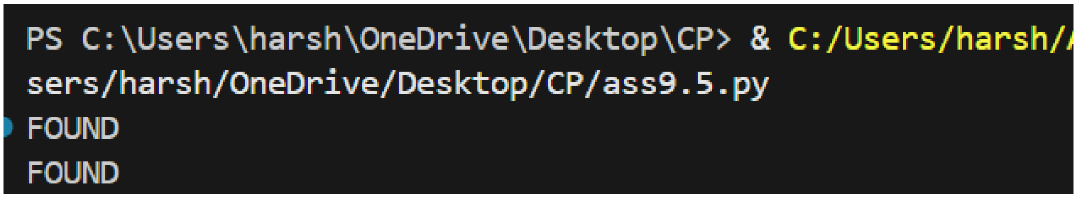


```python
        else:
            if root.right is None:
                return root
            if key>root.right.key:
                root.right.right=self.splay(root.right.right,key)
                root=self.left_rotate(root)
            elif key<root.right.key:
                root.right.left=self.splay(root.right.left,key)
                if root.right.left:
                    root.right=self.right_rotate(root.right)
            return root if root.right is None else self.left_rotate(root)
    def insert(self,root,key):
        if root is None:
            return Node(key)
        root=self.splay(root,key)
        if root.key==key:
            return root
        new=Node(key)
        if key<root.key:
            new.right=root
            new.left=root.left
            root.left=None
        else:
            new.left=root
            new.right=root.right
            root.right=None
        return new
    def search(self,root,key):
        root=self.splay(root,key)
        if root and root.key==key:
            print("FOUND")
```



```python
            print("FOUND")
        else:
            print("NOTFOUND")
        return root
    def delete(self,root,key):
        if root is None:
            return None
        root=self.splay(root,key)
        if root.key!=key:
            return root
        if root.left is None:
            return root.right
        temp=root.right
        root=self.splay(root.left,key)
        root.right=temp
        return root
    def inorder(self,root):
        if root:
            self.inorder(root.left)
            print(root.key,end=" ")
            self.inorder(root.right)

st=SplayTree()
root=None

files=[50,30,70,20,40,60,80]
for f in files:
    root=st.insert(root,f)
root=st.search(root,40)
root=st.search(root,60)
root=st.delete(root,30)
```

Output:



```
PS C:\Users\harsh\OneDrive\Desktop\CP> & C:/Users/harsh/
sers/harsh/OneDrive/Desktop/CP/ass9.5.py
FOUND
FOUND
```