Program:

class Node:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

self.height = 1

class AVLTree:

def get\_height(self, node):

if not node:

return 0

return node.height

def get\_balance(self, node):

if not node:

return 0

return self.get\_height(node.left) - self.get\_height(node.right)

def right\_rotate(self, y):

x = y.left

T2 = x.right

x.right = y

y.left = T2

y.height = 1 + max(self.get\_height(y.left), self.get\_height(y.right))

x.height = 1 + max(self.get\_height(x.left), self.get\_height(x.right))

return x

def left\_rotate(self, x):

y = x.right

T2 = y.left

y.left = x

x.right = T2

x.height = 1 + max(self.get\_height(x.left), self.get\_height(x.right))

y.height = 1 + max(self.get\_height(y.left), self.get\_height(y.right))

return y

def insert(self, root, key):

if not root:

return Node(key)

elif key < root.key:

root.left = self.insert(root.left, key)

else:

root.right = self.insert(root.right, key)

root.height = 1 + max(self.get\_height(root.left), self.get\_height(root.right))

balance = self.get\_balance(root)

# Left Left

if balance > 1 and key < root.left.key:

return self.right\_rotate(root)

# Right Right

if balance < -1 and key > root.right.key:

return self.left\_rotate(root)

# Left Right

if balance > 1 and key > root.left.key:

root.left = self.left\_rotate(root.left)

return self.right\_rotate(root)

# Right Left

if balance < -1 and key < root.right.key:

root.right = self.right\_rotate(root.right)

return self.left\_rotate(root)

return root

def display( node, level=0, prefix="Root: "):

if node is not None:

print(" " \* level + prefix + str(node.key))

display(node.left, level + 1, "L---- ")

display(node.right, level + 1, "R---- ")

# Example usage:

avl = AVLTree()

root = None

for key in [10, 20, 30, 40, 50, 25]:

root = avl.insert(root, key)

display(root)

output:

Root: 30

L---- 20

L---- 10

R---- 25

R---- 40

R---- 50