Exercise:9

class StudentEnrollment:

def \_\_init\_\_(self, id, name, course):

self.id = id # Unique student ID (integer)

self.name = name # Student name (string)

self.course = course # Enrolled course (string)

def \_\_str\_\_(self):

return f"ID: {self.id}, Name: {self.name}, Course: {self.course}"

# Node class for AVL Tree

class Node:

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

self.data = data # StudentEnrollment object

self.left = None

self.right = None

self.height = 1

# AVL Tree class

class AVLTree:

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

self.root = None

# Helper function to get height of a node

def \_get\_height(self, node):

if not node:

return 0

return node.height

# Helper function to get balance factor of a node

def \_get\_balance(self, node):

if not node:

return 0

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

# Right rotate subtree rooted with y

def \_right\_rotate(self, y):

x = y.left

T2 = x.right

x.right = y

y.left = T2

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

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

return x

# Left rotate subtree rooted with x

def \_left\_rotate(self, x):

y = x.right

T2 = y.left

y.left = x

x.right = T2

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

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

return y

# Insert a new enrollment record (keyed by ID)

def insert(self, data):

def \_insert(node):

if not node:

return Node(data)

elif data.id < node.data.id:

node.left = \_insert(node.left)

elif data.id > node.data.id:

node.right = \_insert(node.right)

else:

# Duplicate ID, update the record (or raise error, here we update)

node.data = data

return node

# Update height

node.height = 1 + max(self.\_get\_height(node.left), self.\_get\_height(node.right))

# Get balance factor

balance = self.\_get\_balance(node)

# Balance the tree

if balance > 1:

if data.id < node.left.data.id:

return self.\_right\_rotate(node)

else:

node.left = self.\_left\_rotate(node.left)

return self.\_right\_rotate(node)

if balance < -1:

if data.id > node.right.data.id:

return self.\_left\_rotate(node)

else:

node.right = self.\_right\_rotate(node.right)

return self.\_left\_rotate(node)

return node

self.root = \_insert(self.root)

# Delete a record by ID

def delete(self, id):

def \_delete(node, id):

if not node:

return node

elif id < node.data.id:

node.left = \_delete(node.left, id)

elif id > node.data.id:

node.right = \_delete(node.right, id)

else:

# Node with only one child or no child

if node.left is None:

return node.right

elif node.right is None:

return node.left

if not node:

return node

# Update height

node.height = 1 + max(self.\_get\_height(node.left), self.\_get\_height(node.right))

# Get balance factor

balance = self.\_get\_balance(node)

# Balance the tree

if balance > 1:

if self.\_get\_balance(node.left) >= 0:

return self.\_right\_rotate(node)

else:

node.left = self.\_left\_rotate(node.left)

return self.\_right\_rotate(node)

if balance < -1:

if self.\_get\_balance(node.right) <= 0:

return self.\_left\_rotate(node)

else:

node.right = self.\_right\_rotate(node.right)

return self.\_left\_rotate(node)

return node

self.root = \_delete(self.root, id)

# Search for a student enrollment by ID

def search(self, id):

current = self.root

while current:

if id == current.data.id:

return current.data

elif id < current.data.id:

current = current.left

else:

current = current.right

return None # Not found

# Inorder traversal helper

def \_inorder(self, node, result):

if node:

self.\_inorder(node.left, result)

result.append(node.data)

self.\_inorder(node.right, result)

# Traverse all enrollment records (inorder traversal, sorted by ID)

def inorder\_traversal(self):

result = []

self.\_inorder(self.root, result)

return result

# Count total enrollments (number of nodes)

def count\_enrollments(self):

def \_count(node):

if not node:

return 0

return 1 + \_count(node.left) + \_count(node.right)

return \_count(self.root)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

tree = AVLTree()

# Insert records

tree.insert(StudentEnrollment(101, "Alice", "Math"))

tree.insert(StudentEnrollment(103, "Bob", "Science"))

tree.insert(StudentEnrollment(102, "Charlie", "History"))

tree.insert(StudentEnrollment(104, "David", "Art"))

# Inorder traversal

print("Inorder Traversal:")

for record in tree.inorder\_traversal():

print(record)

# Search

found = tree.search(102)

print("\nSearch for ID 102:", found if found else "Not found")

# Count

print("Total Enrollments:", tree.count\_enrollments())

# Delete

tree.delete(103)

print("\nAfter deleting ID 103:")

for record in tree.inorder\_traversal():

print(record)

print("Total Enrollments:", tree.count\_enrollments())

output:

Inorder Traversal:

ID: 101, Name: Alice, Course: Math

ID: 102, Name: Charlie, Course: History

ID: 103, Name: Bob, Course: Science

ID: 104, Name: David, Course: Art

Search for ID 102: ID: 102, Name: Charlie, Course: History

Total Enrollments: 4

After deleting ID 103:

ID: 101, Name: Alice, Course: Math

ID: 102, Name: Charlie, Course: History

ID: 104, Name: David, Course: Art

Total Enrollments: 3