**🔥 อธิบายโค้ด AVL Tree ใน Python**

โค้ดนี้เป็น **AVL Tree** ที่รองรับฟังก์ชันพื้นฐาน ได้แก่

1. **Insertion (avl\_insert)** – เพิ่มข้อมูลและปรับสมดุล
2. **Deletion (avl\_delete)** – ลบข้อมูลและปรับสมดุล
3. **Rotation (rotate\_left, rotate\_right)** – ใช้ปรับสมดุลเมื่อต้นไม้เสียสมดุล
4. **Printing (print\_tree)** – แสดงโครงสร้างต้นไม้

**🏗️ 1. โครงสร้างของโหนด (Node)**

class Node:

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

self.key = key

self.left = None

self.right = None

**🔹 อธิบาย:**

* **Node** ใช้เก็บข้อมูล **key**
* มีตัวชี้ไปทางซ้าย (**left**) และขวา (**right**)

**📏 2. คำนวณความสูงของโหนด (avl\_height)**

def avl\_height(node):

return -1 if node is None else 1 + max(avl\_height(node.left), avl\_height(node.right))

**🔹 อธิบาย:**

* คำนวณ **ความสูงของต้นไม้** โดยนับจากโหนดไปทางซ้ายหรือขวาที่สูงที่สุด
* ถ้าโหนดเป็น **None** ให้คืนค่า -1

**🔄 3. ฟังก์ชันหมุนต้นไม้ (Rotation)**

**🔸 Left Rotation (rotate\_left)**

def rotate\_left(r):

u = r.right

r.right = u.left

u.left = r

return u

**🔹 อธิบาย:**

* ใช้เมื่อโหนดขวาสูงกว่าซ้ายมากเกินไป (Right-Right Case)

**🔹 Right Rotation (rotate\_right)**

def rotate\_right(r):

o = r.left

r.left = o.right

o.right = r

return o

**🔹 อธิบาย:**

* ใช้เมื่อโหนดซ้ายสูงกว่าขวามากเกินไป (Left-Left Case)

**🛠️ 4. ปรับสมดุลต้นไม้ (avl\_adjust)**

def avl\_adjust(r):

balance = avl\_height(r.right) - avl\_height(r.left)

if balance <= -2: # ซ้ายหนักเกินไป

bl = avl\_height(r.left.right) - avl\_height(r.left.left)

if bl <= 0:

return rotate\_right(r) # Left-Left Case

else:

r.left = rotate\_left(r.left)

return rotate\_right(r) # Left-Right Case

if balance >= 2: # ขวาหนักเกินไป

br = avl\_height(r.right.right) - avl\_height(r.right.left)

if br >= 0:

return rotate\_left(r) # Right-Right Case

else:

r.right = rotate\_right(r.right)

return rotate\_left(r) # Right-Left Case

return r

**🔹 อธิบาย:**

* ตรวจสอบ **Balance Factor**
* ถ้า **ซ้ายหนัก** → ใช้ **Right Rotation**
* ถ้า **ขวาหนัก** → ใช้ **Left Rotation**
* ถ้าเป็น **Left-Right หรือ Right-Left Case** → ใช้ **Double Rotation**

**📥 5. การเพิ่มข้อมูล (Insertion)**

def avl\_insert(r, key):

if r is None:

return Node(key)

if key < r.key:

r.left = avl\_insert(r.left, key)

elif key > r.key:

r.right = avl\_insert(r.right, key)

return avl\_adjust(r)

**🔹 อธิบาย:**

* แทรกค่าลงใน **BST ตามกฎปกติ**
* หลังจากเพิ่มแล้ว **เรียก avl\_adjust()** เพื่อปรับสมดุล

**🗑️ 6. การลบข้อมูล (Deletion)**

def avl\_minimum(r):

return float('inf') if r is None else r.key if r.left is None else avl\_minimum(r.left)

**🔹 อธิบาย:**

* หา **ค่าต่ำสุด (Minimum)** ในต้นไม้

def avl\_delete(r, key):

if r is None:

return None

if key < r.key:

r.left = avl\_delete(r.left, key)

elif key > r.key:

r.right = avl\_delete(r.right, key)

else:

if r.left is None:

return r.right

elif r.right is None:

return r.left

min\_key = avl\_minimum(r.right)

r.key = min\_key

r.right = avl\_delete(r.right, min\_key)

return avl\_adjust(r)

**🔹 อธิบาย:**

* ค้นหาโหนดที่ต้องการลบ
* ถ้าไม่มีลูก → **ลบได้เลย**
* ถ้ามี **ลูกเดียว** → **แทนที่ด้วยลูก**
* ถ้ามี **ลูกสองฝั่ง** → **ใช้ค่าที่น้อยสุดของฝั่งขวาแทน** แล้วลบค่านั้น

**🖨️ 7. แสดงผลต้นไม้ (print\_tree)**

def print\_tree(r):

if r is None:

print("( )", end="")

return

if r.left is None and r.right is None:

print(f"({r.key})", end="")

return

print("(", end="")

print\_tree(r.left)

print(f"({r.key})", end="")

print\_tree(r.right)

print(")", end="")

**🔹 อธิบาย:**

* แสดงโครงสร้าง **AVL Tree** โดยใช้วงเล็บเพื่อแสดงโครงสร้างต้นไม้

**🏁 8. ตัวอย่างการใช้งาน**

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

root = None

for i in range(7):

root = avl\_insert(root, i)

print\_tree(root)

print()

root = avl\_delete(root, 0)

print\_tree(root)

print()

root = avl\_delete(root, 1)

print\_tree(root)

print()

root = avl\_delete(root, 2)

print\_tree(root)

print()

**🔹 อธิบาย:**

* **เพิ่มค่า** ลงใน **AVL Tree** ทีละค่า
* **ลบค่าบางค่า** แล้วแสดงผล

**🎯 สรุป**

* **AVL Tree** คือ **Binary Search Tree** ที่ **รักษาสมดุล** เสมอ
* ใช้ **Rotation (LL, RR, LR, RL)** เพื่อปรับสมดุล
* มี **Insertion, Deletion และ Search ที่ทำงานใน O(log n)**
* เหมาะสำหรับการ **เก็บข้อมูลที่ต้องการเข้าถึงรวดเร็ว**

✅ **โค้ดนี้รองรับการเพิ่ม, ลบ และพิมพ์โครงสร้าง AVL Tree ได้อย่างสมบูรณ์!** 🚀