Ejercicio de Laboratorio:

Implementación de Árboles AVL

Curso: Estructuras de Datos y Análisis de Algoritmos

Introducción:

En el estudio de las estructuras de datos, los árboles binarios de búsqueda (BST) son fundamentales por su eficiencia en operaciones como búsqueda, inserción y eliminación. Sin embargo, la eficiencia de un BST depende críticamente de su balanceo. Un árbol desbalanceado puede degradar el rendimiento de O(logn) a O(n) en el peor de los casos.

Para superar esta limitación, existen estructuras como los Árboles AVL (Adelson-Velsky y Landis). Los Árboles AVL son BSTs auto-balanceados que garantizan que la diferencia de altura entre los subárboles izquierdo y derecho de cualquier nodo (el factor de balance) nunca sea mayor a 1. Esta propiedad asegura que la altura del árbol se mantenga siempre en O(logn), preservando así la eficiencia de O(logn) para las operaciones básicas.

El Problema:

Su tarea en este laboratorio es diseñar e implementar desde cero una estructura de datos para Árboles AVL en Python. Deberán desarrollar las clases y funciones necesarias para manejar esta estructura de manera eficiente y correcta, asegurando que la propiedad de balanceo AVL se mantenga después de cada operación de inserción y eliminación.

Requisitos de Implementación:

Deberán implementar una clase AVLTree que contenga al menos las siguientes funcionalidades:

1. **Clase Node:** Una clase interna o auxiliar para representar los nodos del árbol, almacenando el valor, referencias a los hijos izquierdo y derecho, y la altura del nodo.
2. **Funciones Auxiliares:** Implementar funciones para calcular la altura de un nodo y el factor de balance. También una función para actualizar la altura de un nodo después de operaciones que puedan modificar la subestructura.
3. **Rotaciones:** Implementar las operaciones de rotación simple a la izquierda y a la derecha, que son la base para rebalancear el árbol.

4. **Inserción:** Implementar una función que permita insertar un nuevo valor en el árbol, manteniendo la propiedad de BST y realizando las rotaciones necesarias para asegurar que el árbol permanezca balanceado (AVL) después de la inserción.
5. **Eliminación:** Implementar una función que permita eliminar un valor específico del árbol. Después de la eliminación, deberán realizar las rotaciones necesarias para rebalancear el árbol y mantener la propiedad AVL.
6. **Recorrido:** Implementar una función para realizar un recorrido in-order del árbol, retornando los elementos en orden ascendente.
7. **Visualización (Opcional pero Recomendado):** Implementar una función que les permita visualizar la estructura del árbol, mostrando los nodos, sus valores, alturas y factores de balance. Esto es extremadamente útil para verificar que el árbol se mantiene correctamente balanceado.

Entregable:

- Un archivo Python que contenga el código completo de su implementación del Árbol AVL. El código debe ser claro, legible y estar bien comentado.
- Un documento (en formato PDF) que incluya:
    - Una breve descripción de su implementación.
    - La descripción de los casos de prueba que utilizaron para verificar el correcto funcionamiento de la inserción, eliminación y balanceo. Incluyan ejemplos de secuencias de operaciones y muestren (usando su función de visualización o describiendo) cómo el árbol se comporta y se mantiene balanceado.