EJERCICIO DE LABORATORIO: ÁRBOLES AVL

Alejandra Díaz Navarro. Estudiante – Ingeniería de Sistemas.

Si buscas resultados distintos, no hagas siempre lo mismo.
Albert Einstein

Resumen

En el presente informe se exponen los errores en el código desarrollado por la docente junto a las adiciones de las funciones al mismo que se encuentran indicadas en la definición del ejercicio del laboratorio.

INTRODUCCIÓN

En el presente informe se documenta el desarrollo y corrección de una implementación de un Árbol AVL, una estructura de datos auto-balanceada basada en árboles binarios de búsqueda.

El objetivo principal del trabajo es garantizar que las operaciones de inserción mantengan el equilibrio del árbol, lo cual permite asegurar una eficiencia óptima en las búsquedas, inserciones y eliminaciones. Durante el análisis del código proporcionado inicialmente, se detectaron errores relacionados con la lógica de rotaciones, los cuales impedían el correcto balanceo del árbol. Respecto a esto, se realiza una corrección del código, que es ejecutada por el estudiante.

Ahora bien, el código desarrollado por el estudiante cumple con los requisitos establecidos para la implementación de un árbol AVL. Por lo cual, este incluye la clase *AVLTree* con una clase auxiliar *Node*, funciones auxiliares para altura y balanceo, rotaciones simples, inserción con mantenimiento del balance AVL, recorrido in-order para visualizar los elementos ordenados y funciones necesarias para asegurar la integridad estructural del árbol.

Además, se ha agrega la funcionalidad de eliminación de nodos con rebalanceo posterior, así como una visualización en consola opcional que permite verificar la estructura del árbol mostrando los nodos, sus alturas y balance; lo cual termina facilitando la validación de que el árbol permanece correctamente balanceado. Estas modificaciones se presentan en el archivo denominado *Lab_AVL-error.py*, presente en el link de repositorio de GitHub indicado en los anexos.

METODOLOGÍA

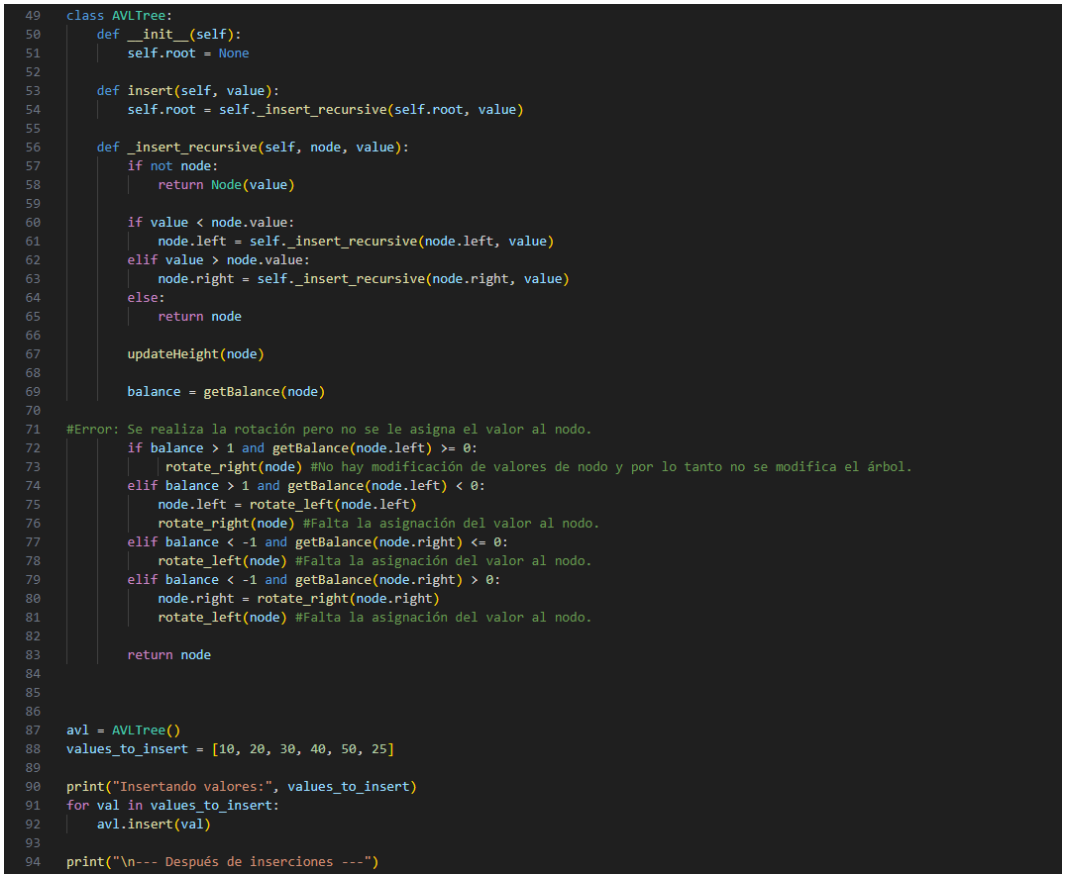
El proceso a seguir corresponde a analizar línea por línea el archivo de código base, para poder manejar las definiciones de las acciones a ejecutar el árbol AVL. En este, caso se presentan las funciones: *getHeight(node)* obtiene la altura de un nodo, mientras que *getBalance(node)* calcula el factor de balanceo para detectar desequilibrios. La función *updateHeight(node)* actualiza la altura de un nodo después de

insertar o rotar. Las funciones *rotate_left(node)* y *rotate_right(node)* realizan rotaciones necesarias para reequilibrar el árbol cuando se detecta un desbalance. Finalmente, dentro de la clase AVLTree, la función *insert(value)* y su auxiliar *_insert_recursive(node, value)* se encargan de insertar nuevos valores de forma recursiva, ajustando el árbol y aplicando las rotaciones necesarias para mantenerlo balanceado.

ANÁLISIS DE RESULTADOS.

Mediante el análisis de las funciones presentadas en el código base, sin las funcionalidades indicadas añadidas, se obtienen las siguientes incongruencias en el mismo.

Figura 1 Errores presentes en el código.



```
49  class AVLTree:
50      def __init__(self):
51          self.root = None
52
53      def insert(self, value):
54          self.root = self._insert_recursive(self.root, value)
55
56      def _insert_recursive(self, node, value):
57          if not node:
58              return Node(value)
59
60          if value < node.value:
61              node.left = self._insert_recursive(node.left, value)
62          elif value > node.value:
63              node.right = self._insert_recursive(node.right, value)
64          else:
65              return node
66
67          updateHeight(node)
68
69          balance = getBalance(node)
70
71  #Error: Se realiza la rotación pero no se le asigna el valor al nodo.
72          if balance > 1 and getBalance(node.left) >= 0:
73              rotate_right(node) #No hay modificación de valores de nodo y por lo tanto no se modifica el árbol.
74          elif balance > 1 and getBalance(node.left) < 0:
75              node.left = rotate_left(node.left)
76              rotate_right(node) #Falta la asignación del valor al nodo.
77          elif balance < -1 and getBalance(node.right) <= 0:
78              rotate_left(node) #Falta la asignación del valor al nodo.
79          elif balance < -1 and getBalance(node.right) > 0:
80              node.right = rotate_right(node.right)
81              rotate_left(node) #Falta la asignación del valor al nodo.
82
83          return node
84
85
86
87  avl = AVLTree()
88  values_to_insert = [10, 20, 30, 40, 50, 25]
89
90  print("Insertando valores:", values_to_insert)
91  for val in values_to_insert:
92      avl.insert(val)
93
94  print("\n--- Después de inserciones ---")
```

Fuente: Código LabError en Visual Studio Code.

Como se observa en la Figura 1, los errores encontrados se presentan a partir de la línea de código 71 hasta la línea 81; es posible visualizar cuatro errores que no permiten el correcto funcionamiento del código base. Los cuales consisten fundamentalmente en:

- **Apreciación error 1: No se asigna el resultado de la rotación al nodo.** En múltiples partes del código, se realiza correctamente la llamada a funciones de rotación, como *rotate_left(node)* o *rotate_right(node)*, pero no se asigna el resultado de esas funciones al nodo correspondiente. Esto implica que, aunque la rotación se ejecute, los cambios no se reflejan en

la estructura del árbol. Por ejemplo, en el caso *if balance > 1 and getBalance(node.left) > 0*:, se llama a *rotate_right(node)* pero no se hace *return rotate_right(node)*, por lo que el nodo sigue apuntando a la versión anterior y sin cambios. Lo cual conlleva a que el árbol no se reestructure y pierda las propiedades de balanceo del AVL.

- **Apreciación error 2: No se actualiza el subárbol después de una doble rotación** En los casos de rotaciones dobles, se realiza la rotación interna correctamente, pero falta nuevamente asignar el resultado de la rotación al subárbol correspondiente. Por ejemplo, se hace *node.left = rotate_left(node.left)* correctamente, pero luego *rotate_right(node)* se ejecuta sin guardar el resultado, lo que hace inútil la rotación principal.

Los errores mencionados describen de forma general el conflicto presentado en el código durante las distintas rotaciones que realiza el árbol. Por lo que es preciso recalcar que el error principal en el código es que, durante el proceso de rebalanceo del árbol AVL, no se tiene en cuenta la importancia de guardar el valor del nodo después de aplicar una rotación.

A pesar de que las condiciones para detectar desequilibrios están correctamente implementadas, las funciones de rotación se llaman sin asignar su resultado al nodo afectado; lo cual implica que el árbol no se actualiza con la nueva estructura resultante, lo que impide que se mantenga el balance esperado en un árbol AVL; así que, este descuido en la asignación anula por completo el efecto del rebalanceo.

Ahora bien, con los errores para el código base corregidos se procede con la segunda parte del ejercicio de laboratorio que consiste en agregar las funcionales indicadas en el PDF de instrucciones junto a pruebas de las mismas; respecto al análisis de las pruebas en el nuevo archivo de código se tiene lo siguiente:

Se hicieron cinco pruebas diferentes, y se observó la estructura del árbol usando una función que imprime el árbol.

- *Prueba 1: Inserción con rotación simple (Izquierda-Izquierda)*

En esta prueba se insertaron los valores 30, 20 y 10. Esto causó un desbalance del tipo Izquierda-Izquierda, por lo que se aplicó una rotación simple hacia la derecha.

Resultado:

D: (30, h=1)
Raiz: (20, h=2)
I: (10, h=1)

- *Prueba 2: Inserción con rotación doble (Izquierda-Derecha)*

Aquí se insertaron los valores 30, 10 y 20. Esto generó un desbalance del tipo Izquierda-Derecha. Primero se hizo una rotación izquierda sobre 10 y luego una rotación derecha sobre 30.

Resultado:

D: (30, h=1)
Raiz: (20, h=2)
I: (10, h=1)

- *Prueba 3: Inserción con rotación doble (Derecha-Izquierda)*

En esta prueba se insertaron los valores 10, 30 y 25. El desbalance fue del tipo Derecha-Izquierda. Se hizo una rotación derecha sobre 30 y luego una rotación izquierda sobre 10.

Resultado:

D: (30, h=1)
Raiz: (25, h=2)
I: (10, h=1)

- *Prueba 4: Eliminación de nodo con dos hijos*

En este caso se insertaron varios valores: 50, 30, 70, 20, 40, 60 y 80. Después se eliminó el nodo 50, que tenía dos hijos. El árbol reemplazó ese nodo con su sucesor 60 y se reestructuró.

Antes de eliminar 50:

D: (80, h=1)
D: (70, h=2)
I: (60, h=1)
Raiz: (50, h=3)
D: (40, h=1)
I: (30, h=2)
I: (20, h=1)

Después de eliminar 50:

D: (80, h=1)
D: (70, h=2)
Raiz: (60, h=3)
D: (40, h=1)
I: (30, h=2)
I: (20, h=1)

- *Prueba 5: Inserciones y eliminación mixta*

En esta prueba se insertaron los valores 10, 20, 30, 40, 50 y 25. Después se eliminó el nodo 30. El árbol se reestructuró automáticamente para mantenerse balanceado.

Antes de eliminar 30:

D: (50, h=1)
D: (40, h=2)
Raiz: (30, h=3)
D: (25, h=1)
I: (20, h=2)
I: (10, h=1)

Después de eliminar 30:

D: (50, h=1)
Raiz: (40, h=3)
D: (25, h=1)
I: (20, h=2)
I: (10, h=1)

CONCLUSIONES

Finalmente, mediante las secciones de este informe es posible realizar una breve descripción de la implementación del código en general.

Así que, es posible afirmar que se implementó un árbol AVL en Python utilizando clases donde cada nodo de este se representa con la clase *Node*, que contiene su valor, referencias a sus hijos izquierdo y derecho, y su altura. Así mismo, dentro del código a clase AVLTree maneja el árbol completo e incluye métodos para insertar, eliminar y balancear automáticamente los nodos.

El rebalanceo se realiza aplicando rotaciones simples y dobles (izquierda-izquierda, derecha-derecha, izquierda-derecha, y derecha-izquierda) según el balance de alturas de los subárboles. Además, se creó una función para imprimir el árbol en forma estructurada y funciones para recorrerlo en orden.

Durante las operaciones, se asegura que el árbol mantenga su propiedad de balanceo, lo que garantiza un rendimiento eficiente en búsquedas, inserciones y eliminaciones.

ANEXOS

En esta sección se adjunta el link del repositorio de GitHub que contiene el presente informe y el archivo de código final en la rama “VersionFinal”.

Link: *https://github.com/alejandradiazn/TALLERES-RBOLES-AVL.git*