Análisis de Complejidad de Métodos
Clase Temperaturas_DB

A continuación se presenta una tabla con el análisis del orden de complejidad temporal para cada uno de los métodos implementados en la clase Temperaturas_DB. El análisis está basado en el uso de un árbol AVL como estructura subyacente, lo cual garantiza operaciones balanceadas con complejidad logarítmica en el peor caso.

Método	**Complejidad (Big-O)**	**Explicación**
guardar_temperatura	O(log n)	Inserta un nodo en el árbol AVL. El árbol se balancea automáticamente.
devolver_temperatura	O(log n)	Busca una clave (fecha) en el árbol AVL. Búsqueda logarítmica.
borrar_temperatura	O(log n)	Elimina un nodo del árbol AVL. Operación logarítmica con balanceo.
cantidad_muestras	O(n)	Cuenta recursivamente los nodos del árbol. Podría optimizarse almacenando el total.
devolver_temperaturas	O(k + log n) a O(n)	Recorre los nodos entre dos fechas. En el peor caso puede recorrer todo el árbol.
max_temp_rango	O(k)	Busca la máxima temperatura en el rango ya recorrido.
min_temp_rango	O(k + log n)	Busca la mínima temperatura en el rango ya recorrido.
temp_extremos_rango	O(k + log n)	Calcula min y max a partir del rango obtenido.
_obtener_temperaturas_en_rango	O(k + log n) a O(n)	Recorre parcialmente el árbol entre dos fechas. Depende del tamaño del rango.

En todos los casos, **n** es la cantidad de muestras registradas.

Análisis General

La clase Temperaturas_DB utiliza internamente un árbol AVL, lo cual garantiza que la altura del árbol se mantenga acotada por O(log n). Gracias a esta propiedad de balanceo automático, las operaciones de **inserción**, **búsqueda** y **eliminación** se realizan con eficiencia logarítmica, aún cuando la base de datos crece significativamente en tamaño.

Por otro lado, los métodos que trabajan con **rangos de fechas** —como devolver_temperaturas, max_temp_rango, min_temp_rango y temp_extremos_rango— presentan una complejidad de O(k + log n), donde **k** representa la cantidad de nodos dentro del rango especificado, y log n el tiempo necesario para alcanzar el punto de inicio del recorrido dentro del árbol. En el **peor caso**, estos métodos pueden alcanzar una complejidad lineal O(n), si el rango abarca todos los elementos del árbol.

El método cantidad_muestras, tal como está implementado, realiza un recorrido completo del árbol para contar la cantidad total de nodos, por lo cual su complejidad es **O(n)**. Esta operación podría optimizarse fácilmente manteniendo un contador interno actualizado con cada operación de inserción o eliminación.

En síntesis, el uso del árbol AVL en esta implementación asegura un buen rendimiento en las operaciones más comunes, manteniendo la eficiencia y estabilidad que se espera de una base de datos en memoria que debe gestionar un volumen variable de registros cronológicos.