Análisis del Orden de Complejidad Big-O

A continuación, se presenta una tabla con el análisis del orden de complejidad Big-O para cada uno de los métodos implementados en la clase `Temperaturas_DB`.

guardar_temperatura: complejidad O(logN). La inserción en un árbol AVL implica buscar la posición adecuada, insertar el nodo y, potencialmente, realizar rotaciones para mantener el balanceo. Todas estas operaciones se realizan en un tiempo proporcional a la altura del árbol, que en un AVL es O(logN), donde N es la cantidad de muestras.

devolver_temperatura: complejidad O(logN). La búsqueda de un elemento en un árbol AVL requiere recorrer el árbol desde la raíz hasta el nodo deseado. Debido a que el árbol está balanceado, la altura máxima es O(logN), lo que hace que la búsqueda sea muy eficiente.

max_temp_rango, in_temp_rango, temp_extremos_rango: : complejidad O(logN). La búsqueda de las fechas límite del rango (fecha1 y fecha2) tiene una complejidad de O(log n). Luego, la búsqueda dentro del subárbol correspondiente (o nodos en el rango) puede ser de O(n) en el peor caso, pero en un árbol AVL bien balanceado, esta búsqueda suele ser mucho más rápida (aproximadamente log n), lo que hace que la complejidad general de las funciones sea O(log n)

borrar_temperatura: complejidad O(logN). La eliminación en un árbol AVL implica buscar el nodo a eliminar, realizar la eliminación (que puede requerir encontrar un sucesor in-order) y luego reequilibrar el árbol mediante rotaciones. Todas estas operaciones, al igual que la inserción, tienen una complejidad de O(logN) debido al balanceo del árbol.

devolver_temperaturas:complejidad O(1). Mantener un contador en la clase `Temperaturas_DB` permite devolver la cantidad de muestras en tiempo constante, ya que solo se necesita acceder a una variable.

O(logN): Este orden de complejidad se debe a la naturaleza de los árboles AVL. Al mantener el árbol balanceado, la altura del árbol siempre es logarítmica respecto al número de nodos (N). Esto significa que las operaciones que recorren un camino desde la raíz hasta una hoja (como inserción, eliminación y búsqueda de un solo elemento) son muy eficientes, ya que el número de pasos es proporcional a logN.

O(N): Cuando una operación necesita potencialmente visitar todos los nodos del árbol (o una parte significativa que no está limitada por la altura), su complejidad se vuelve lineal con respecto al número de nodos. Aunque nuestros métodos de rango usan propiedades de subárboles para optimizar, en el peor de los casos, un rango podría abarcar casi todo el árbol, haciendo que la complejidad se acerque a O(N).

O(1): Las operaciones que no dependen del tamaño de la entrada se consideran de tiempo constante.