*Informe problema 2 - Trabajo Práctico N\*2*

En este proyecto, mi compañera y yo desarrollamos una base de datos en memoria principal llamada Temperaturas\_DB, implementada con un árbol AVL para el almacenamiento de datos. Este árbol nos permitió optimizar las operaciones de consulta, inserción y eliminación de registros de temperatura al mantener balanceada la estructura en cada operación. Diseñamos este sistema pensando en las necesidades de nuestro usuario, Kevin Kelvin, un científico que requiere registrar y consultar temperaturas asociadas a fechas específicas. El sistema permite realizar consultas clave, como la temperatura máxima y mínima en un rango de fechas, así como gestionar las inserciones y eliminaciones de registros de manera eficiente.

Análisis del Orden de Complejidad (Big-O)

Al basarnos en un árbol AVL, garantizamos que las operaciones sean balanceadas, lo que resulta en tiempos de ejecución eficientes aun cuando el conjunto de datos crece. A continuación, desglosamos la complejidad de cada método en la clase Temperaturas\_DB:

| Método | Complejidad (Big-O) | Explicación |
| --- | --- | --- |
| | guardar\_temperatura() | | --- |  |  | | --- | | | O(log n) | | --- |  |  | | --- | | | Inserta un nodo en el árbol AVL, realizando búsqueda, posibles rotaciones y actualizando el balance del árbol. | | --- |  |  | | --- | |
| | devolver\_temperatura() | | --- |  |  | | --- | | | O(log n) | | --- |  |  | | --- | | | Realiza una búsqueda en el árbol AVL, aprovechando su estructura balanceada para una consulta logarítmica. | | --- |  |  | | --- | |
| | max\_temp\_rango() | | --- |  |  | | --- | | | O(m + log n) | | --- |  |  | | --- | | | Recorre los nodos en un rango de fechas específico y devuelve el valor máximo. | | --- |  |  | | --- | |
| | min\_temp\_rango() | | --- |  |  | | --- | | | O(m + log n) | | --- |  |  | | --- | | | Similar a max\_temp\_rango(), pero devuelve el valor mínimo. | | --- |  |  | | --- | |
| | temp\_extremos\_rango() | | --- |  |  | | --- | | | O(m + log n) | | --- |  |  | | --- | | | Encuentra tanto el valor mínimo como el máximo en un rango, con dos comparaciones adicionales. | | --- |  |  | | --- | |
| | borrar\_temperatura() | | --- |  |  | | --- | | | O(log n) | | --- |  |  | | --- | | | Elimina un nodo del árbol y realiza rotaciones si es necesario, manteniendo el balance general del árbol. | | --- |  |  | | --- | |
| | devolver\_temperaturas() | | --- |  |  | | --- | | | O(m + log n) | | --- |  |  | | --- | | | Recorre los nodos en un rango dado y los retorna en una lista ordenada, donde m es la cantidad de nodos en el rango. | | --- |  |  | | --- | |
| | cantidad\_muestras() | | --- |  |  | | --- | | | O(1) | | --- |  |  | | --- | | Devuelve el número total de registros guardado como variable de clase, sin necesidad de recorrer el árbol. |

Explicación de Complejidades

* Inserción (guardar\_temperatura): En un árbol AVL, la inserción de un nodo puede requerir rotaciones para mantener el balance del árbol. Esto permite que la operación se mantenga en una complejidad de O(log n), asegurando eficiencia en el almacenamiento.
* Búsqueda (devolver\_temperatura): La búsqueda de una fecha específica es O(log n) debido a la estructura balanceada del árbol, lo que facilita la realización de consultas frecuentes.
* Consultas en Rango (max\_temp\_rango, min\_temp\_rango, temp\_extremos\_rango, devolver\_temperaturas): Estas operaciones implican un recorrido del árbol en un rango de fechas, con una complejidad de O(m + log n), donde m es la cantidad de nodos en ese rango. El término log n representa el tiempo necesario para ubicar el nodo de inicio del rango.
* Eliminación (borrar\_temperatura): La eliminación también puede requerir rotaciones para mantener el balance, con una complejidad de O(log n), similar a la inserción.
* Conteo (cantidad\_muestras): Dado que cantidad\_muestras simplemente devuelve un contador que se actualiza con cada inserción o eliminación, esta operación tiene una complejidad constante O(1).

Entonces, para cerrar la idea y a modo de conclusión, aclaro que la decisión de implementar Temperaturas\_DB usando un árbol AVL fue clave para cumplir con las necesidades de nuestro usuario en términos de rendimiento y eficiencia, especialmente al realizar consultas en un conjunto de datos en constante crecimiento. Con esta estructura, logramos que las operaciones críticas, como inserción, eliminación y búsqueda, mantuvieran complejidad logarítmica, lo cual asegura un rendimiento estable del sistema a medida que aumenta el número de registros de temperatura.