Universidad Nacional de Entre Ríos

➢*Facultad:* Facultad de Ingeniería.

➢*Carrera:* Licenciatura en Bioinformática y Bioingeniería.

➢*Cátedra:* Algoritmos y estructuras de datos

➢*Docentes:* Javier E. Diaz Zamboni, Jordán F. Insfrán, Juan F. Rizzato

➢*Título del trabajo:* "Aplicaciones de estructuras jerárquicas y grafos -"*Problema 2 ""*.

➢*Alumnos:*

- Almirón Spahn, María Paz
- Leiva, Giuliana
- Saravia, Lucía Milagros

➢*Fecha de entrega:* 31 de Octubre del 2025

Problema 2:

En esta actividad implementamos una base de datos en memoria "Temperaturas_DB" que almacena mediciones de temperatura asociadas a fechas (objetos "datetime") usando un **árbol AVL** como estructura subyacente. Asimismo, se implementan operaciones para guardar, recuperar, borrar y consultar rangos (máximo, mínimo, extremos y listado), además de devolver la cantidad de muestras.

Algunas de las especificaciones lógicas implementadas fueron:

- **guardar_temperatura(temperatura, fecha)**: guarda la medida de temperatura asociada a la fecha.
- **devolver_temperatura(fecha)**: devuelve la medida de temperatura en la fecha determinada.
- **max_temp_rango(fecha1, fecha2)**: devuelve la temperatura máxima entre los rangos fecha1 y fecha2 inclusive (fecha1 < fecha2). Esto no implica que los intervalos del rango deban ser fechas incluidas previamente en el árbol.
- **min_temp_rango(fecha1, fecha2):** devuelve la temperatura mínima entre los rangos fecha1 y fecha2 inclusive (fecha1 < fecha2). Esto no implica que los intervalos del rango deban ser fechas incluidas previamente en el árbol.
- **temp_extremos_rango(fecha1, fecha2)**: devuelve la temperatura mínima y máxima entre los rangos fecha1 y fecha2 inclusive (fecha1 < fecha2).
- **borrar_temperatura(fecha):** recibe una fecha y elimina del árbol la medición correspondiente a esa fecha.
- **devolver_temperaturas(fecha1, fecha2):** devuelve un listado de las mediciones de temperatura en el rango recibido por parámetro con el formato "dd/mm/aaaa: temperatura ºC", ordenado por fechas.
- **cantidad_muestras():** devuelve la cantidad de muestras de la BD.

Objetivos:

1. Diseñar e implementar una base de datos en memoria llamada "Temperaturas_DB" para consultas eficientes por fecha.
2. Permitir operaciones: insertar, consultar por fecha, obtener máximos/mínimos/en rango, borrar y conocer la cantidad de muestras.
3. Analizar complejidades y justificar la elección de estructura.

Estructura:

Para esta actividad utilizamos un árbol AVL para mantener las claves ordenadas (fechas) y garantizar complejidad O (log n) para búsquedas, inserciones y borrados en el

peor caso. Esto es apropiado cuando se requieren consultas por rangos y acceso ordenado por fecha, ya que esta estructura:

- ➔ Mantiene las claves ordenadas de forma natural.
- ➔ Permite buscar el rango de fechas con coste dependiente del número de resultados en lugar de recorrer todo el árbol.

Análisis del orden de complejidad para cada método:

La clase “Temperatura_DB” fue implementada utilizando como estructura base un árbol AVL, una variante balanceada de los árboles binarios de búsqueda. El uso de este tipo de estructuras garantiza que, después de cada inserción o eliminación, la **altura** del árbol se mantenga dentro de **O(log n)**, lo que asegura una eficiencia elevada en la mayoría de las operaciones, incluso con un número grande de registros.

A continuación, se presenta una tabla con el análisis de complejidad temporal (Big-O) de cada uno de los métodos principales de la clase, junto con una explicación detallada de su comportamiento y justificación.

Método	Complejidad	Explicación
guardar_temperatura(temperatura, fecha):	**O(log n)**	Este método inserta una nueva medición en el árbol AVL. Primero convierte la fecha a un formato manejable (con datetime) y luego recorre el árbol comparando fechas hasta encontrar la posición correcta. La altura del árbol es logarítmica respecto al número de nodos (n), por lo que la inserción es O(log n). En caso de necesitar rebalanceo, las rotaciones son operaciones constantes (O(1)).
devolver_temperatura(fecha):	**O(log n)**	Busca la medición correspondiente a una fecha específica. El recorrido desde la raíz hasta la hoja correspondiente implica comparar fechas a lo largo de los niveles del árbol. Como el árbol está balanceado, su altura es O(log n), garantizando una búsqueda rápida.
max_temp_rango(fecha1, fecha2):	**O(log n + k)**	Devuelve la temperatura máxima registrada entre dos fechas. Se recorre el árbol hasta alcanzar el inicio del rango (O(log n)) y luego se recorren los k nodos comprendidos en ese intervalo para determinar el valor máximo (O(k)). Por lo tanto, el costo total es O(log n + k).
min_temp_rango(fecha1, fecha2):	**O(log n + k)**	Similar al método anterior, pero devuelve la temperatura mínima. Se realiza una búsqueda en el rango definido (O(log n)) y luego se determina el valor

		mínimo recorriendo los k nodos comprendidos (O(k)).
temp_extremos_rango(fecha1, fecha2):	**O(log n + k)**	Devuelve la temperatura mínima y máxima dentro del rango de fechas indicado. Para esto, se genera una lista con las mediciones del rango (O(log n + k)) y luego se calcula el mínimo y máximo con un recorrido lineal (O(k)). La complejidad total sigue siendo O(log n + k).
borrar_temperatura(fecha):	**O(log n)**	Elimina la medición correspondiente a una fecha determinada. Primero localiza el nodo que contiene la fecha (O(log n)) y luego lo elimina. Si el borrado causa un desbalance, el árbol se reequilibra mediante rotaciones, que son operaciones de tiempo constante (O(1)).
devolver_temperaturas(fecha1, fecha2):	**O(log n + k)**	Retorna una lista con las mediciones comprendidas entre dos fechas. El árbol se recorre parcialmente: primero se localiza el inicio del rango (O(log n)) y luego se recorren solo los k nodos del intervalo (O(k)). Si el rango incluye todas las fechas, el costo puede llegar a O(n).
cantidad_muestras():	**O(1)**	Devuelve la cantidad total de mediciones almacenadas en la base. Esta información se mantiene actualizada mediante un contador o atributo, por lo que la operación no requiere recorrer el árbol. Es constante.

Es posible afirmar que, en definitiva, la clase “Temperatura_DB” presenta una estructura eficiente y escalable gracias al uso del árbol AVL como estructura de datos subyacente. En conjunto, la clase logra un equilibrio entre la rapidez a la hora de acceder a ella, el uso eficiente de memoria y el mantenimiento del orden temporal de las mediciones