Informe del Trabajo Práctico N°2

Aplicaciones de estructuras jerárquicas y grafos

Materia: Algoritmos y Estructuras de Datos - FIUNER

Integrantes:

Flores, Valentina

Aguilar Gonzales, Andrea Fernanda

Lell, Camila Luciana

Fecha de entrega: 6 de junio del 2025

ACTIVIDAD 2

Temperaturas_DB

En esta segunda actividad se planteó el diseño de una base de datos que almacena mediciones de temperatura asociadas a fechas, con el objetivo de realizar búsquedas eficientes y consultas por rangos. Para abordar este problema, decidimos utilizar un Árbol AVL, una estructura jerárquica que garantiza que los datos se mantengan siempre ordenados y balanceados.

Un árbol AVL (O(log n)) es una variante del árbol binario de búsqueda que se auto-balancea tras cada inserción o eliminación. Esto permite asegurar que la altura del árbol se mantenga logarítmica, lo cual es clave para mantener la eficiencia en operaciones de acceso, búsqueda y modificación de los datos. En nuestro caso, cada nodo del árbol almacena una fecha como clave (mediante objetos datetime) y una temperatura como valor (en grados Celsius).

Métodos implementados y comportamiento observado

Tabla:

Método	Complejidad	Descripción
guardar_temperatura	O(log n)	Inserción o actualización en el árbol AVL.
devolver_temperatura	O(log n)	Búsqueda por fecha en el árbol.
borrar_temperatura	O(log n)	Eliminación con rebalanceo.
max_temp_rango	O(log n + k)	Búsqueda del rango más recorrido de k nodos.
min_temp_rango	O(log n + k)	Similar a max_temp_rango.
temp_extremos_rango	O(log n + k)	Un solo recorrido para mínimo y máximo.
devolver_temperaturas	O(log n + k)	Recorrido in-order de k nodos en el rango.
cantidad_muestras	O(1)	Acceso al atributo tamano del árbol.

La solución se implementó en Python con los siguientes componentes:

- **arbol.py**: Define las clases NodoArbol y ArbolAVL. Cada nodo almacena una clave (datetime para fechas) y un valor (float para temperaturas). El árbol AVL soporta inserciones, búsquedas, eliminaciones, y balanceo en O(log n).
- **temperaturas.py**: Implementa la clase TemperaturasDB, que usa un árbol AVL para gestionar las mediciones. Incluye métodos para validar fechas, manejar duplicados (actualizando temperaturas existentes), y realizar consultas en rango mediante un recorrido in-order.
- **main.py**: Simula operaciones básicas (inserciones, consultas, eliminaciones) para verificar el funcionamiento.

- **test_temperaturas.py**: Contiene pruebas unitarias, cubriendo casos como fechas inválidas, rangos vacíos, y actualizaciones.

Pruebas

- **Simulación en main.py**: Inserción de 10 mediciones, consultas de temperaturas específicas, rangos, y eliminación de una fecha. Todas las operaciones produjeron resultados correctos.
- **Pruebas unitarias en test_temperaturas.py**: 10 pruebas que cubren:
    - Inserción y consulta de temperaturas.
    - Actualización de fechas existentes.
    - Consultas de máximo, mínimo, y extremos en rangos.
    - Eliminación de mediciones.
    - Listado de temperaturas en rango.
    - Cantidad de muestras.
    - Casos inusuales: fechas inválidas, rangos vacíos, y rangos inválidos

Resultados:

- Todas las pruebas unitarias pasaron correctamente.
- El árbol AVL mantuvo el balanceo tras inserciones y eliminaciones, asegurando O(log n) en operaciones básicas.
- Las consultas en rango recorrieron solo los nodos necesarios, con complejidad O(log n + k).
- El uso de datetime garantizó comparaciones precisas y manejo robusto de fechas.

Observaciones

- **Validación robusta**: Se implementó validación de fechas en _parse_fecha, lanzando ValueError para formatos inválidos. Los métodos de rango verifican que fecha1 <= fecha2.
- **Manejo de duplicados**: guardar_temperatura actualiza la temperatura si la fecha ya existe, asegurando una sola medición por fecha.
- **Eficiencia**: No se usaron estructuras auxiliares, optimizando memoria y tiempo.

Conclusión

La implementación de TemperaturasDB con un árbol AVL resolvió eficientemente los requisitos del ejercicio. La estructura permitió inserciones, búsquedas, eliminaciones, y consultas en rango con complejidades óptimas (O(log n) para operaciones básicas, O(log n + k) para rangos). Las pruebas unitarias confirmaron la robustez, cubriendo casos inusuales y validaciones.