Informe de temperaturas- Árbol AVL

UNIVERSIDAD NACIONAL DE ENTRE RÍOS
Facultad de Ingeniería
Carrera: T.U.P.E.D

Materia: Fundamentos de Algoritmos y Estructuras de Datos

Estudiante: Franco Tomiozzo

Profesor: Dr. Javier E. Diaz Zamboni

Bioing. Jordán F. Insfrán

Esp. Bioing. Francisco Rizzato

Comisión: 1

Fecha: Octubre 2025

Introducción

Implementé una base de datos de temperaturas diarias usando un árbol AVL.
Cada registro se guarda con una fecha como clave y una temperatura como valor.
El objetivo fue mantener operaciones rápidas de inserción, búsqueda, borrado y consultas por rangos de fechas, aprovechando el balanceo automático del AVL para sostener un rendimiento eficiente incluso con un número considerable de registros.

Diseño e implementación

Los nodos almacenan la fecha, la temperatura, la altura y referencias a hijo izquierdo y derecho.
Sobre esta estructura, la clase Temperaturas_DB provee métodos para:

- Guardar o actualizar una temperatura por fecha.
- Buscar por fecha.
- Borrar un registro.
- Obtener el mínimo y el máximo de temperatura en un rango de fechas.
- Listar cronológicamente todas las temperaturas dentro de un intervalo.
- Cargar datos desde un archivo de texto para pruebas masivas.

La carga desde archivo lee el archivo "muestras.txt" con codificación **UTF-16 LE**, aceptando tanto el separador punto y coma **;** como la coma **,** .
Las fechas pueden estar en formato **dd/mm/yyyy** o **yyyy-mm-dd**, lo que facilita compatibilidad con distintas fuentes.
Un ejemplo válido: **2025-01-02;37.9.**

Resultados principales

líneas_procesadas: 120, registros_cargados: 120

cantidad_muestras: 120

devolver_temperatura('15/03/2025'): 25.0

min_temp_rango('01/02/2025','28/02/2025'): 35.1

max_temp_rango('01/02/2025','28/02/2025'): 39.2

temp_extremos_rango('01/02/2025','28/02/2025'): (35.1, 39.2)

devolver_temperaturas('01/02/2025','28/02/2025') (n=28):

listado cronológico desde 01/02/2025: 36.4 °C hasta 28/02/2025: 35.7 °C

borrar_temperatura('15/03/2025'): True

devolver_temperatura('15/03/2025'): None

cantidad_muestras: 119

Método	Descripción	Complejidad temporal	Complejidad espacial	Explicación
guardar_o_actualizar(fecha, temp)	Inserta o modifica una temperatura	**O(log n)**	O(1)	Recorre el árbol hasta el punto adecuado y rebalancea si es necesario.
devolver_temperatura(fecha)	Busca una fecha	**O(log n)**	O(1)	Búsqueda típica de árbol binario de búsqueda AVL.
borrar_temperatura(fecha)	Elimina un registro	**O(log n)**	O(1)	Puede requerir una o más rotaciones, pero siempre logarítmico.
min_temp_rango(desde, hasta)	Mínimo en rango	**O(log n + k)**	O(1)	Recorre solo nodos dentro del intervalo; k = visitados.
max_temp_rango(desde, hasta)	Máximo en rango	**O(log n + k)**	O(1)	Similar al mínimo, pero enfocándose en los valores superiores.

temp_extremos_rango(desde, hasta)	Mínimo y máximo simultáneamente	**O(log n + k)**	O(1)	Reutiliza las mismas ramas relevantes.
devolver_temperaturas(desde, hasta)	Listado cronológico del rango	**O(log n + m)**	O(m)	Recorrido inorden podado; m = resultados que caen dentro del rango.
min_fecha() / max_fecha()	Fecha más antigua / reciente	**O(log n)**	O(1)	Basta recorrer la rama más a la izquierda o derecha.
cantidad_muestras()	Retorna cantidad de registros	**O(1)**	O(1)	Utiliza contador interno.
cargar_desde_archivo(ruta)	Importa dataset desde archivo	**O(L × log n)**	O(1)**	L = líneas válidas; cada línea genera una inserción AVL.

Análisis de complejidad Big-O Conclusión del análisis:

El árbol AVL mantiene la altura balanceada, garantizando tiempos cercanos a O(log n) en las operaciones principales.

Las consultas por rango o listados aprovechan recorridos podados, manteniendo una excelente relación entre precisión y velocidad.

Método implementado para carga desde archivo

Se incorporó un método robusto para la lectura y carga de muestras desde un archivo plano.

Incluye validación de formato, flexibilidad en separadores y formatos de fecha, control de errores y un resumen final de la carga:

```
def cargar_desde_archivo(self, ruta="muestras.txt",
encoding="utf-16-le", saltar_encabezado=False,
reportar_errores_max=10):
    """
    Lee un archivo de muestras (fecha;temperatura o
fecha,temperatura) y las carga en la base.
    Acepta fechas en dd/mm/yyyy o yyyy-mm-dd. Temperaturas con coma o
punto decimal.
    Complejidad: O(L*log n)
    """
    lineas_procesadas = 0
    registros_cargados = 0
    errores = 0
    errores_detalle = []

    with open(ruta, "r", encoding=encoding) as f:
        for idx, linea in enumerate(f, start=1):
            if saltar_encabezado and idx == 1:
                continue
            if not linea.strip() or linea.lstrip().startswith("#"):
                continue
            lineas_procesadas += 1
            try:
                sep = ';' if ';' in linea else (',' if ',' in linea
else None)
                if not sep:
                    raise ValueError("Sin separador ';' o ','.")
                fecha_str, temp_str = map(str.strip,
linea.split(sep))
                fecha = self._to_dt(fecha_str)
                temp = float(temp_str.replace(",", "."))
                self.guardar_o_actualizar(fecha, temp)
                registros_cargados += 1
            except Exception as e:
                errores += 1
                if len(errores_detalle) < reportar_errores_max:
                    errores_detalle.append({"línea": idx, "error":
str(e)})

    return {
        "líneas_procesadas": lineas_procesadas,
        "registros_cargados": registros_cargados,
        "errores": errores,
```

```
        "errores_detalle": errores_detalle,
        "cantidad_muestras": self.cantidad_muestras()
    }
```

Características destacables:

- Soporta **dos formatos de fecha** ("**`%d/%m/%Y`**" y "**`%Y-%m-%d`**").
- **Separadores alternativos** (`;` o `,`) para mayor compatibilidad.
- Ignora líneas vacías o de comentario (`#`).
- Reporta **errores de línea** con detalle (hasta 10 por defecto).
- Retorna métricas generales que permiten validar la importación.

Ejemplo de resultado esperado tras la carga:

{'líneas_procesadas': 120, 'registros_cargados': 120, 'errores': 0, 'cantidad_muestras': 120}

Aspectos técnicos destacables

- El **rebalanceo automático** garantiza que la altura sea Θ(log n), manteniendo un rendimiento uniforme frente a inserciones ordenadas o desordenadas.
- Las **rotaciones simples y dobles** (**LL, RR, LR, RL**) son constantes (**O(1)**) y se aplican solo cuando se detecta un desequilibrio.
- El **borrado con dos hijos** reemplaza por el sucesor inorden derecho, preservando el orden del árbol.
- La **flexibilidad de la carga** (diversos separadores y formatos) demuestra tolerancia a distintas fuentes, aunque en un entorno productivo convendría estandarizar un formato único y validar exhaustivamente entradas erróneas.

Conclusiones

La implementación cumple los objetivos planteados:

- Inserta, consulta, lista, calcula extremos y elimina datos con **eficiencia logarítmica**.
- La estructura se comporta estable y coherentemente en todas sus operaciones.
- La carga desde archivo facilita pruebas y expansión, y los resultados experimentales indican un funcionamiento correcto con 120 registros.

Para producción, se recomienda:

- Estandarizar formato de entrada.
- Manejar errores de parseo de forma más explícita.
- Agregar validación de rango físico de temperaturas.

El sistema queda así preparado para escalar en volumen de datos manteniendo un desempeño excelente y una base sólida en su diseño algorítmico.