

la

**ALGORITMOS Y ESTRUCTURAS DE DATOS**

# Trabajo Práctico 2

**Integrantes del Grupo: Filchtinsky Tomas y Gabirondo Lucas**

## Objetivos

* Aplicar conceptos teóricos sobre estructuras jerárquicas, grafos y sus algoritmos asociados.

## Consignas

#### 1. Sala de Emergencias

El **triaje** es un proceso que permite una gestión del riesgo clínico para poder manejar adecuadamente y con seguridad los flujos de pacientes cuando la demanda y las necesidades clínicas superan a los recursos. El siguiente proyecto de software posee una sencilla simulación de esta situación: [enlace a código](https://github.com/Grupo-de-Computacion-de-la-FI-UNER/ayed-tps-codigos-iniciales-para-estudiantes/blob/26f98faefe753ce0bd7c1f6e4d452a36879eb06e/TP2_problema2/main.py).

En este proyecto para simplificar existen pacientes con tres niveles de riesgo para su salud: 1: crítico, 2: moderado, 3: bajo. Actualmente la simulación muestra que los pacientes se atienden según el orden en que llegan al centro de salud.

**Se solicita:**

1. Seleccionar, programar y aplicar una estructura de datos adecuada para almacenar los pacientes conforme ingresan al centro de salud **de modo tal que cuando se atiende un paciente siempre sea aquel cuyo nivel de riesgo es el más delicado** en comparación con el resto de los pacientes que restan por ser atendidos. Si dos pacientes poseen el mismo nivel de riesgo, adoptar un segundo criterio para seleccionar uno de ellos. Recordar que la estructura de datos debe ser genérica, es decir, debe poder almacenar cualquier tipo de dato, y no ser específica para alojar pacientes (separar implementación de aplicación).
2. Fundamentar en el informe, en no más de una página, la estructura seleccionada indicando el orden de complejidad *O* de inserciones y de eliminaciones en la estructura seleccionada.

**Observación:** Pueden realizarse todas las modificaciones al código que crean necesarias siempre que se respete el propósito del problema planteado.

**RESPUESTAS:**

En esta implementación, se ha seleccionado una cola de prioridad que utiliza un montículo binario (también conocido como heap) como estructura subyacente. Un montículo binario es una estructura de datos eficiente para administrar elementos con prioridad, donde se busca acceso rápido al elemento de mayor o menor prioridad (dependiendo del tipo de montículo).

La elección del montículo binario permite que la cola de prioridad tenga un acceso óptimo al elemento de mayor o menor prioridad en la parte superior, con tiempos de ejecución eficientes tanto para inserciones como para eliminaciones.

**Inserciones (insertar)**

Al insertar un nuevo elemento, este se agrega al final del montículo y luego se ajusta la estructura mediante el proceso de "infiltración hacia arriba" (infiltArriba). Este proceso asegura que se mantenga la propiedad de montículo: el padre siempre es menor que sus hijos en un min-montículo.

Complejidad: el ajuste tiene una complejidad de O(log n), ya que en el peor de los casos el elemento debe moverse desde el último nivel hasta la raíz, recorriendo la altura del montículo, que es O(log n).

**Eliminaciones (extraer\_mayor\_prioridad)**

La eliminación del elemento de mayor prioridad se realiza extrayendo la raíz (primer elemento) del montículo y reemplazándola con el último elemento. Después, se aplica el proceso de "infiltración hacia abajo" (infiltAbajo) para restaurar la propiedad del montículo.

Complejidad: similar a la inserción, el proceso de infiltración tiene una complejidad de O(log n), ya que en el peor caso el elemento debe desplazarse desde la raíz hasta una hoja.

#### 2. Temperaturas\_DB

Kevin Kelvin es un científico que estudia el clima y, como parte de su investigación, debe consultar frecuentemente en una base de datos la temperatura del planeta tierra dentro de un rango de fechas. A su vez, el conjunto de medidas crece conforme el científico registra y agrega **mediciones** a la base de datos.

Una medición está conformada por el valor de temperatura en ºC (flotante) y la fecha de registro, la cual deberá ser ingresada como “dd/mm/aaaa” (string). (Internamente sugerimos emplear objetos de tipo *datetime*).

Ayude a Kevin a realizar sus consultas eficientemente implementando una base de datos en memoria principal “**Temperaturas\_DB**”que utilice internamente un árbol AVL. La base de datos debe permitir realizar las siguientes operaciones (interfaz de Temperaturas\_DB):

**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.

Adicionalmente realizar las siguientes actividades:

* Escriba una tabla con el análisis del orden de complejidad Big-O para cada uno de los métodos implementados para su clase “Temperaturas\_DB”, explique brevemente el análisis de los mismos.

**RESPUESTAS:**

|  |  |  |
| --- | --- | --- |
| guardar\_temperatura | O(log n) | Este método llama a agregar en el árbol AVL, que tiene complejidad O(log n) debido al reequilibrado tras la inserción, manteniendo el árbol balanceado. |
| devolver\_temperatura | O(log n) | obtener en el árbol AVL también tiene O(log n) al buscar un nodo por su clave. La estructura balanceada garantiza esta eficiencia de búsqueda. |
| max\_temp\_rango | O(n) | Itera en orden todos los nodos del árbol, y en el peor caso, debe verificar cada nodo en el rango dado para encontrar la temperatura máxima. |
| min\_temp\_rango | O(n) | Similar a max\_temp\_rango, recorre todos los nodos en el rango para encontrar la mínima temperatura, y podría visitar hasta todos los nodos en el peor caso. |
| temp\_extremos\_rango | O(n) | Combina las funciones de max\_temp\_rango y min\_temp\_rango, por lo que recorre los nodos en el rango para obtener ambas temperaturas extremas. |
| borrar\_temperatura | O(log n) | Utiliza el método eliminar del árbol AVL, que tiene complejidad O(log n) debido al reequilibrio necesario para mantener la estructura balanceada. |
| devolver\_temperaturas | O(n) | Recorre los nodos en orden dentro del rango dado y, en el peor caso, podría visitar todos los nodos, resultando en O(n). |
| cantidad\_muestras | O(1) | Simplemente devuelve el tamaño actual almacenado en tamano, que es una operación de acceso directo. |

#### 3. Palomas mensajeras

Desde la antigüedad y hasta la invención del telégrafo en el año 1835 por Morse, las palomas mensajeras fueron el sistema de comunicación más rápido entre ciudades. La premisa que sustenta este tipo de comunicación es que una paloma mensajera criada en un lugar y liberada en otro, volverá a su lugar de origen; si se aprovecha este viaje haciendo que el animal lleve consigo un mensaje, la información se podrá replicar y transmitir de un lugar a otro.

William, como dueño de la agencia de noticias “Palomas William”, conoce el tema y entiende que hay varios caminos para entregar un mensaje a un lugar. Sin embargo, todavía no está seguro de cómo encontrar la mejor forma de enviar las noticias a las demás aldeas ahorrando recursos. Él tiene a cargo toda la comunicación desde su aldea “Peligros” a un grupo de otras 21 aldeas, todas con un palomar propio. Cada aldea, incluída Peligros, sólo puede enviar mensajes a sus aldeas vecinas de las cuales tiene alguna paloma mensajera. La lista de aldeas vecinas y la distancia entre cada par de ellas está disponible en el archivo [aldeas.txt](https://drive.google.com/file/d/166xJP_JfKhJDC_EVay1tB7GodLdPP7ER/view?usp=drive_link). Cada ruta en la lista es una tupla  donde  y  son, respectivamente, los nombres (string) de la aldea de inicio y final de cada posible viaje de una paloma mensajera, y  representa la distancia en leguas entre esas aldeas (entero positivo).

Ayude a William a encontrar la forma más eficiente de llevar un mensaje desde su aldea a todas las demás, donde cada aldea reciba solamente una vez la noticia. Cada aldea, al recibir la noticia, puede replicarla y enviarla a tantas aldeas vecinas como quiera.

Entregables:

* Mostrar la lista de aldeas en orden alfabético.
* Para cada aldea, mostrar de qué vecina debería recibir la noticia, y a qué vecinas debería enviar réplicas, siendo que se está enviando el mensaje de la forma más eficiente a las 21 aldeas. Tomar en cuenta que desde Peligros solamente se envían noticias a una o más aldeas vecinas.
* Para el envío de una noticia, mostrar la suma de todas las distancias recorridas por todas las palomas enviadas desde cada palomar.

**RESPUESTAS:**