Trabajo Práctico 2

Integrantes: Ferrer Paira Iñaki, Spaggiari Fiamma

Respuestas

Problema 1 (Sala de Emergencias)

Para garantizar un flujo de atención óptimo de los pacientes en la sala de emergencias, se implementó un sistema que utiliza una Cola de Prioridad. Con esta estructura nos aseguramos que los pacientes sean atendidos según su nivel de riesgo clínico, priorizando a los casos más delicados. Para poder manejar los datos nos basamos en una estructura de datos fundamental: **Heap Binario (Min-Heap)**, el cual fue implementado de forma manual para esta implementación.

Lo que tuvimos en cuenta para realizar nuestro código fue:

1. **Atención al Paciente: Criterio de Prioridad y Desempate:** El objetivo principal es atender siempre al paciente cuyo nivel de riesgo sea el más delicado.
   - Nivel de Riesgo: Se asignan niveles de riesgo numéricos donde: 1 → crítico, 2 → moderado, 3 → bajo. En nuestra implementación organizamos los elementos de forma que el valor más pequeño es el que se encuentra en la raíz del árbol (es el primero en ser extraído).
   - Orden de llegada: Se introdujo un mecanismo de desempate basado en el orden de llegada. Al insertar un paciente en la cola, se almacena una tupla (prioridad, contador, elemento_paciente). El contador es un valor incremental que registra el orden cronológico de ingreso y se utiliza como segundo criterio de ordenamiento, garantizando que el paciente que llegó primero (menor valor de contador) sea atendido antes.
2. **Estructura de Datos Genérica y Reutilizable:** La implementación de la Cola de Prioridad es genérica y desacoplada de la clase Paciente (funciona con cualquier tipo de objeto que se le inserte)

Pasando al orden de complejidad (O):

- **Inserciones (Ingreso de un Paciente):** Esta operación implica dos pasos principales:
  1. Añadir el nuevo elemento al final de la lista subyacente que representa el Heap → complejidad constante, O(1).
  2. El elemento recién añadido debe "subir" en el árbol para restablecer la propiedad del Heap (donde cada nodo es menor o igual que sus hijos). En el peor de los casos, este elemento puede tener que recorrer desde la hoja hasta la raíz → O(logN).

El orden de complejidad de una inserción es O(logN).

- **Eliminaciones (Atención de un Paciente):** Esta operación implica los siguientes pasos:
  1. El elemento raíz es removido y se guarda para ser retornado.
  2. El último elemento de la última hoja del Heap es movido a la posición de la raíz vacía.
  3. Este nuevo elemento debe "bajar" a través del árbol para restablecer la propiedad del Heap, asegurando que el menor de sus hijos suba. En el peor de los casos, este elemento puede tener que descender hasta una de las hojas → O(logN).

El orden de complejidad de una eliminación en un Heap es O(logN).

Problema 2 (Temperaturas_DB)

El código de este problema se basa en un sistema de gestión de mediciones de temperatura, diseñado para almacenar y consultar datos de temperatura asociados a fechas. La solución se basa en una estructura de datos de árbol AVL

La solución se compone de tres clases:

- **NodoAVL**
- **ArbolAVL**
- **Temperaturas_DB**

Además cuenta con un test (test_temperaturas_db) para probar el correcto funcionamiento de los métodos en Temperaturas_DB y un main donde se hacen pruebas del código hecho.

Pasando a la complejidad temporal de los métodos principales de Temperaturas_DB:

MÉTODO	DESCRIPCIÓN	COMPLEJIDAD	ANÁLISIS
guardar_temperatura(fecha, temp)	Guarda/Actualiza una medición	**O(log n)**	Inserción/Actualización en AVL → operaciones con altura logarítmica
devolver_temperatura(fecha)	Busca una temperatura popt fecha	**O(log n)**	Búsqueda en AVL → operaciones con altura logarítmica
max_temp_rango(f1, f2)	Máxima temperatura en un rango	**O(log n + k)**	Búsqueda en AVL → O(log n) y recorrido de k elementos en el rango
min_temp_rango(f1, f2)	Mínima temperatura en un rango	**O(log n + k)**	Búsqueda en AVL → O(log n) y recorrido de k elementos en el rango
temp_extremos_rango(f1, f2)	Mínima y máxima temperatura en un rango	**O(log n + k)**	2 búsquedas y recorridos en AVL → O(log n + k)
borrar_temperatura(fecha)	Elimina una medición	**O(log n)**	Eliminación en AVL → operaciones con altura logarítmica
devolver_temperaturas_rango(f1, f2)	Listado de mediciones en un rango	**O(log n + k)**	Recorrido en orden de los nodos dentro de un rango.
cantidad_muestras()	Cantidad total de mediciones	**O(n)**	Recorrido completo del árbol

Problema 3 (Palomas mensajeras)

Para encontrar una red de conexiones que una todas las aldeas de forma eficiente, se utiliza el algoritmo de Prim, el cual construye un Árbol de Expansión Mínima (MST).
Este árbol representa la forma más “económica” de conectar todas las aldeas (minimizando el total de leguas recorridas).
Puntos a tener en cuenta para la resolución:

1. **Orden alfabético de las aldeas:** Se muestra una lista ordenada alfabéticamente de todas las aldeas registradas en el sistema.
2. **Lógica de replicación del mensaje:**
    - La aldea “Peligros” inicia la cadena de replicación.
    - Cada aldea que recibe la noticia la retransmite a sus vecinas en el MST.
3. **Cálculo de la distancia total:** La distancia total recorrida por las palomas se calcula como la suma de las longitudes de todas las aristas del Árbol de Expansión Mínima construido desde Peligros.