Facultad de Ingeniería

Licenciatura en Bioinformática

Bioingeniería

TRABAJO PRÁCTICO N°2 - EJERCICIO 1

ALGORITMOS Y ESTRUCTURAS DE DATOS

Docentes: Rizzato, Juan - Diaz Zamboni, Javier

Alumnos:

Isaac, Priscila Rocio

Jacobo, Nahir

Nista, Nicolás

Fecha de entrega: 07/06

Año lectivo: 2025

Estructura seleccionada: Montículo Binario.

Para garantizar que los pacientes sean atendidos respetando el nivel de criticidad de su estado, se ha implementado una cola de prioridad utilizando un **montículo binario** como estructura subyacente.

Se seleccionó un montículo binario como estructura de datos porque se comporta como un árbol binario completo, lo que garantiza que todos los niveles estén completamente llenos, excepto quizás el último. Todos los niveles se completan de izquierda a derecha. Esta organización permite mantener el orden y eficiencia en las operaciones.

Sus nodos pueden ser de tipo mínimo o máximo. En un nodo mínimo, el elemento de mayor prioridad siempre se encuentra en la raíz, lo cual es ideal para sistemas de atención por prioridad (como una sala de urgencias) ya que permite atender siempre al paciente más urgente de forma rápida y eficiente. Además, y por lo que es muy útil para ordenar por prioridad, funciona con cualquier tipo de dato que pueda compararse. En este caso horario de llegada y riesgo para la salud.

La **cola prioridad** se encarga de registrar y ordenar a los pacientes de modo que siempre se atiende primero al paciente más crítico y que, en caso de empate, priorizar al que haya llegado antes (esto lo hace mediante un índice de llegada incremental).

Funciones definidas dentro de la clase Montículo:

Las funciones que realicen constantes comparaciones entre índices las consideramos de complejidad $O_{(Log\ n)}$, aquellas que solo recorran la lista 1 vez de complejidad $O_{(1)}$ y las que posean estructuras de control como bucles for / while de complejidad $O_{(n)}$.

- **__init__**: $O_{(1)}$ → Solo hay asignaciones simples
- **Extraer** → Extrae la raíz (elemento de mayor prioridad) y reorganiza el montículo hacia abajo, en el peor de los casos puede recorrer desde la raíz a hoja.
- **Insertar**: $O_{(Log\ n)}$ → Inserta al final y reorganiza hacia arriba, en el peor de los casos sube hasta la raíz.
- **Esta_vacio**: $O_{(1)}$ → Verifica si el montículo tiene elementos, para ello alcanza con recorrerlo una sola vez.
- **Cantidad**: $O_{(1)}$ → Sigue la misma lógica que la función esta_vacio.

- **Mostrar_pacientes**: $O_{(n)}$ → Recorre a todos los pacientes mediante un bucle for.
- **__Subir**: $O_{(Log\ n)}$ → Reorganiza un nodo hacia arriba hasta encontrar su lugar, siendo el peor de los casos cuando llega a subir hasta la raíz.
- **__Bajar**: $O_{(Log\ n)}$ → Sigue la misma lógica que la función _subir, siendo el peor de los casos cuando llega a bajar hasta la hoja.
- **__Intercambiar**: $O_{(1)}$ → Intercambia dos posiciones en la lista mediante asignación.