Temario:
Double-Ended Priority Queues

Estructuras de Datos Avanzadas

1. Introducción

En esta parte damos el contexto general para que se entienda por qué existen las double-ended priority queues.

- Repaso rápido de una *priority queue* normal.
  - Definición básica.
  - Operaciones clásicas: `getMin`/`getMax`, `insert`, `deleteMin`/`deleteMax`.
- Motivación de las **double-ended priority queues (DEPQ)**.
  - Limitaciones de una cola de prioridad de un solo extremo.
  - Casos donde necesitamos acceder al mínimo y al máximo de forma eficiente.

2. Definición formal de una DEPQ

- Modelo abstracto.
  - Conjunto de elementos con una prioridad o valor.
  - Operaciones principales:
    - `getMin`, `getMax`
    - `insert` / `put`
    - `removeMin`, `removeMax`
- Objetivo de diseño (de forma ideal).
  Aclarar que el objetivo típico es de usar una DEPQ ES:
  - `getMin` y `getMax` en $O(1)$.
  - Inserciones y borrados en $O(\log n)$.
  - Uso de memoria en $O(n)$.

3. Aplicación: QuickSort externo

Aquí explicamos una aplicación concreta para que no se vea tan abstracto.

- Recordatorio rápido de QuickSort interno.

    - Explicar en palabras simples qué es quicksort interno (en memoria) y su idea de pivot, subarreglo izquierdo, pivot, subarreglo derecho

- Problema cuando los datos están en disco (QuickSort externo).
  Describir el problema cuando los datos no caben en memoria.
- Uso de una DEPQ en QuickSort externo.
  Contar la idea general del external quicksort usando una DEPQ
    - La DEPQ guarda un "grupo medio" que sí cabe en memoria.
    - Elementos muy pequeños se van a L, muy grandes a R.
    - Los que caen "entre" el min y el max de la DEPQ se manejan con intercambios.

4. Visión general de implementaciones de DEPQ

- Idea general: casi todas se basan en *heaps* (árboles binarios completos).
- Objetivo común:
    - `getMin`/`getMax` en $O(1)$.
    - `insert`, `removeMin`, `removeMax` en $O(\log n)$.
- Estructuras que voy a mencionar:
    - Symmetric Min-Max Heaps.
    - Interval Heaps.
    - Min-Max Heaps.
    - Deaps.

5. Symmetric Min-Max Heaps (SMMH)

- Representación.
    - Árbol binario completo.
    - La raíz está vacía, los demás nodos tienen un elemento.
- Propiedades clave (relaciones entre hijos y abuelos).
- Localización de mínimos y máximos.
    - `getMin`: hijo izquierdo de la raíz.
    - `getMax`: hijo derecho de la raíz.
- Idea general de las operaciones.
    - Inserción: "burbujear" hacia arriba manteniendo las propiedades.
    - `removeMin`/`removeMax`: variantes de *trickle down* de heaps.

6. Interval Heaps

- Definición intuitiva.
  - Cada nodo guarda un intervalo $[a, b]$ con $a \leq b$.
  - Los hijos representan intervalos contenidos en el del padre.
- Conexión con heaps.
  - Extremos izquierdos forman un min-heap.
  - Extremos derechos forman un max-heap.
- `getMin`/`getMax`:
  - Min: extremo izquierdo de la raíz.
  - Max: extremo derecho de la raíz.
- Idea de inserción y eliminación.
  - Dependiendo de si el nuevo valor cae dentro o fuera del intervalo del padre.
  - Reacomodo usando la lógica de min-heap o max-heap según el caso.

7. Min-Max Heaps

- Idea general.
  - Árbol binario completo con niveles alternados:
    - Niveles min: el nodo es el mínimo de su subárbol.
    - Niveles max: el nodo es el máximo de su subárbol.
- Localización de mínimos y máximos.
  - Min: en la raíz.
  - Max: en uno de los hijos de la raíz.
- Operaciones básicas.
  - Inserción con *trickle-up* usando reglas de min-heap o max-heap según el nivel.
  - Eliminación con *trickle-down* comparando con hijos y nietos.

8. Deaps

- Definición.
  - Árbol binario completo con raíz vacía.
  - Subárbol izquierdo: min-heap.
  - Subárbol derecho: max-heap.
- Propiedad de correspondencia.

    - Cada nodo del lado min tiene un nodo correspondiente del lado max.
    - El valor en el min-heap es $\leq$ al valor del nodo correspondiente del max-heap.
- `getMin`/`getMax`.
    - Min: raíz del min-heap (hijo izquierdo).
    - Max: raíz del max-heap (hijo derecho).
- Idea de inserción y eliminación.

9. Métodos genéricos para construir DEPQs

- Dual priority queues.
    - Mantener dos colas de prioridad: una min y una max.
    - Cada elemento está en ambas, con punteros de correspondencia.
- Total correspondence.
- Leaf correspondence.
    - Solo se relacionan las hojas de cada heap.
    - Sirve para estructuras tipo leftist heaps, pairing heaps, etc.

10. Meldable DEPQs (MDEPQ)

- Nueva operación: `meld(Q1, Q2)`.
    - Combinar dos DEPQs en una sola.
- Cuándo importa tener `meld`.
    - Algoritmos donde se fusionan muchas colas de prioridad.
- Ideas de implementación.
    - Adaptar leftist heaps, pairing heaps, Fibonacci heaps, etc., al caso doble.

11. Cierre

- Tabla o resumen (para la diapositiva final).
    - Complejidades de cada estructura.
    - Qué tan difícil es implementar cada una.
    - Si soportan o no `meld`.
- Comentarios finales.
    - Cuándo vale la pena usar una DEPQ.
    - Qué implementación recomendaría yo en la práctica y por qué.