CLASE 6

Los Tipos Abstractos de Datos (TADs) se definen por su **interfaz** (qué operaciones ofrecen) y no por su implementación (cómo se realizan esas operaciones). Esto genera distintos roles con diferentes responsabilidades:

- **Diseñador**: Define la interfaz del TAD.
- **Usuario**: Utiliza el TAD a través de su interfaz, sin conocer los detalles internos.
- **Implementador**: Elige las estructuras de datos y algoritmos para que las operaciones sean eficientes.

La **eficiencia** de las operaciones de un TAD es crucial y generalmente se mide en el peor de los casos, en función del número de elementos (n) de la estructura. Las complejidades comunes son:

- **Constante O(1)**: El costo es siempre el mismo, sin importar el tamaño.
- **Lineal O(n)**: El costo crece proporcionalmente al número de elementos.
- **Cuadrática O(n2)**: El costo crece de forma cuadrática con el número de elementos.

Punto Clave: Es fundamental tener claro qué representa "n" en cada implementación, ya que puede variar y llevar a cálculos de costos engañosos (por ejemplo, en un Set, "n" podría ser el total de elementos o el número de elementos únicos).

TADs Clásicos y sus Implementaciones

1. Stack (Pila)

- **Concepto**: Una colección de elementos con acceso LIFO (Last-In, First-Out).
- **Operaciones clave**: push (agregar), pop (quitar), top (consultar el último).
- **Implementación común**: Usando una lista, donde push y pop se realizan en la cabeza de la lista, logrando una eficiencia de **O(1)**.

2. Queue (Cola)

- **Concepto**: Una colección de elementos con acceso FIFO (First-In, First-Out).
- **Operaciones clave**: enqueue (encolar), dequeue (desencolar), firstQ (consultar el primero).
- **Implementaciones y su eficiencia**:
    - **Una lista (agregando adelante)**: enqueue es O(1), pero dequeue (que debe quitar el último) es O(n).
    - **Una lista (agregando atrás)**: enqueue es O(n), pero dequeue es O(1).
    - **Dos listas (frente y fondo)**: Esta es una implementación más eficiente que puede lograr que la mayoría de las operaciones sean O(1) en promedio (costo amortizado).
- **Extensión de la interfaz**: Si se quiere agregar una operación lenQ (longitud), la implementación recursiva simple tiene un costo de O(n). Para que sea O(1), se debe modificar la representación interna para almacenar el tamaño explícitamente.

3. Set (Conjunto)

- **Concepto**: Una colección de elementos únicos, sin un orden particular.
- **Operaciones clave**: adds (agregar), belongs (pertenencia), removeS (quitar), unions (unión).
- **Implementaciones**:

- **Lista con repetidos**: adds es O(1), pero belongs y removeS son O(n). La unión puede ser costosa.
- **Lista sin repetidos (con invariante de representación)**: adds requiere verificar si el elemento ya existe, por lo que su costo es O(n), al igual que belongs y removeS.

4. Priority Queue (Cola de Prioridad)

- **Concepto**: Una colección donde cada elemento tiene una prioridad. Al eliminar, siempre se obtiene el elemento de máxima prioridad (el mínimo, según la definición).
- **Operaciones clave**: insertPQ, findMinPQ (encontrar mínimo), deleteMinPQ (eliminar mínimo).
- **Usos**: Salas de espera de hospitales (paciente más grave) , planificadores de sistemas operativos.
- **Implementaciones y su eficiencia**:
  - **Lista arbitraria**: insertPQ es O(1), pero findMinPQ y deleteMinPQ requieren recorrer toda la lista, costando **O(n)**.
  - **Lista ordenada**: findMinPQ y deleteMinPQ son O(1), pero insertPQ debe encontrar la posición correcta, costando **O(n)**.
- **Mejora**: Para mejorar la eficiencia se necesitan estructuras más avanzadas como los árboles.

5. Map (Diccionario o Mapa)

- **Concepto**: Una colección de pares clave-valor, donde cada clave es única y se asocia a un valor.
- **Operaciones clave**: assocM (asociar), lookupM (buscar), deleteM (eliminar).
- **Usos**: Agendas, diccionarios, y cualquier aplicación que requiera buscar información a partir de una clave.
- **Implementaciones y su eficiencia**:
  - **Lista de pares (clave, valor) sin claves repetidas**: Todas las operaciones principales (assocM, lookupM, deleteM) requieren, en el peor caso, buscar a lo largo de la lista, resultando en una eficiencia de **O(n)**.
- **Mejora**: Al igual que con las Priority Queues, se requieren estructuras como los árboles para obtener mejores costos.

6. Multiset (Multiconjunto o Bag)

- **Concepto**: Similar a un Set, pero permite que los elementos aparezcan múltiples veces.
- **Operaciones clave**: addMS (agregar), ocurrencesMS (contar ocurrencias).
- **Implementaciones**:
  - **Con una lista de elementos**: Simple, pero operaciones como ocurrencesMS tienen un costo de O(n).
  - **Con un Map de elementos a enteros**: Se utiliza otro TAD para implementarlo. La clave es el elemento y el valor es el número de ocurrencias. La eficiencia dependerá de la implementación del Map subyacente.