

Resolución de problemas enlazando distintas estructuras

Algoritmos y Estructuras de Datos

1^{er} cuatrimestre 2025

- ▶ Venimos viendo distintas estructuras de datos, que nos permiten modelar TADs clásicos.
- ▶ Distintas estructuras de datos nos permiten resolver problemas particulares de forma eficiente.
- ▶ Pero a veces no alcanza con una sola estructura.
- ▶ ¿Qué pasa si quiero acceder a datos según múltiples criterios?

Ejemplo 1

Una empresa tiene un deposito al que le llegan pedidos. Estos pedidos vienen de distintos clientes, pidiendo productos del depósito, y cuentan con un número identificador. Queremos un sistema que nos permita administrar los pedidos, para ir cumpliéndolos en orden de llegada. Además, necesitamos poder ver la información de los pedidos.

Ejemplo 1 - TAD

```
Pedido ES struct<id:N,  
                cliente:N,  
                producto:N>  
TAD SistemaPedidos {  
  obs cola:seq<Pedido>  
  
  proc nuevoSistema() : SistemaPedidos  
    requiere ...  
    asegura ...  
  proc agregarPedido(inout s: SistemaPedidos,  
                    in p: Pedido)  
  proc proximo(inout s: SistemaPedidos) : Pedido  
  proc infoPedido(in s: SistemaPedidos,  
                 in idPedido: N): Pedido  
}
```

Ejemplo 1 - Clase Java

```
class SistemaPedidos {  
    SistemaPedidos() {...}  
    void agregarPedido(Pedido p) {...}  
    Pedido proximo() {...}  
    Pedido infoPedido(int idPedido) {...}  
}
```

Ejemplo 1 - Clase Java

```
class SistemaPedidos {  
    SistemaPedidos() {...}  
    void agregarPedido(Pedido p) {...}  
    Pedido proximo() {...}  
    Pedido infoPedido(int idPedido) {...}  
}
```

¿Nos alcanza con una sola estructura conocida para resolver este problema eficientemente?

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    ...  
}
```

- ▶ SistemaPedidos()
- ▶ void agregarPedido(Pedido p)
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p)
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(1)$
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(1)$
- ▶ Pedido proximo() $\rightarrow O(1)$
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(1)$
- ▶ Pedido proximo() $\rightarrow O(1)$
- ▶ Pedido infoPedido(int idPedido) $\rightarrow O(n)!!$

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(1)$
- ▶ Pedido proximo() $\rightarrow O(1)$
- ▶ Pedido infoPedido(int idPedido) $\rightarrow O(n)!!$

¿Podemos mejorar infoPedido?

Ejemplo 1 - Estructura

```
class SistemaPedidos {  
    // Usamos la cola para resolver los pedidos en orden  
    ColaSobreLista<Pedido> colaDePedidos;  
    // Y guardamos los pedidos tambien en un diccionario  
    // para acceder a su info  
    DiccionarioLog<int, Pedido> infoPedidos;  
    ...  
}
```

Ejemplo 1 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, Pedido> infoPedidos;  
    ...  
}
```

- ▶ SistemaPedidos()
- ▶ void agregarPedido(Pedido p)
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, Pedido> infoPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p)
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, Pedido> infoPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, Pedido> infoPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo() $\rightarrow O(\log(n))$
- ▶ Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, Pedido> infoPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo() $\rightarrow O(\log(n))$
- ▶ Pedido infoPedido(int idPedido) $\rightarrow O(\log(n))$

Ejemplo 1 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, Pedido> infoPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo() $\rightarrow O(\log(n))$
- ▶ Pedido infoPedido(int idPedido) $\rightarrow O(\log(n))$

Algunas operaciones aumentaron su complejidad.
¿Vale la pena? Depende.

Ejemplo 1 - Invariante de Representación

¿Qué debe cumplir la estructura interna en todo momento?

Ejemplo 1 - Invariante de Representación

¿Qué debe cumplir la estructura interna en todo momento?

- ▶ Los pedidos en `colaDePedidos` no tienen ids repetidos.
- ▶ Los pedidos en `colaDePedidos` y los valores de `infoPedidos` son los mismos.
- ▶ Cada pedido en `infoPedidos` tiene como clave el id del pedido.

Ejemplo 1 - Código

```
public class SistemaPedidos {  
    ColaSobreLista<Pedido> cola;  
    DiccionarioLog<Integer, Pedido> dicc;  
  
    public SistemaPedidos() {  
        cola = new ColaSobreLista<>();  
        dicc = new DiccionarioLog<>();  
    }  
  
    public void agregar(Integer id, Pedido p) {  
        cola.encolar(p);  
        dicc.definir(id, p);  
    }  
  
    ...  
}
```

Ejemplo 1 - Código

```
public class SistemaPedidos {  
    ColaSobreLista<Pedido> cola;  
    DiccionarioLog<Integer, Pedido> dicc;  
  
    ...  
  
    public Pedido siguiente() {  
        Pedido p = cola.proximo();  
        dicc.eliminar(p.id);  
        return p;  
    }  
  
    public Pedido infoPedido(Integer id) {  
        return dicc.obtener(id);  
    }  
}
```


- ▶ Vimos que a veces conviene guardar los datos en múltiples lugares.
- ▶ ¡Pero a veces eso tampoco alcanza!
- ▶ Caso más común: queremos acceder rápidamente a un lugar que la estructura sola no nos permite.

Ejemplo 2

El deposito puede tomarse su tiempo, por lo que le da la opción a los clientes de cancelar pedidos.

Ejemplo 2

El deposito puede tomarse su tiempo, por lo que le da la opción a los clientes de cancelar pedidos.

```
TAD SistemaPedidos {  
    ...  
    proc cancelarPedido(inout s: SistemaPedidos,  
                        in id: IN) : Pedido  
}
```

Ejemplo 2

El deposito puede tomarse su tiempo, por lo que le da la opción a los clientes de cancelar pedidos.

```
TAD SistemaPedidos {  
  ...  
  proc cancelarPedido(inout s: SistemaPedidos,  
                      in id: IN) : Pedido  
}
```

¿Qué problema surge con nuestra estructura?

Ejemplo 2

El deposito puede tomarse su tiempo, por lo que le da la opción a los clientes de cancelar pedidos.

```
TAD SistemaPedidos {  
  ...  
  proc cancelarPedido(inout s: SistemaPedidos,  
                      in id: IN) : Pedido  
}
```

¿Qué problema surge con nuestra estructura? Eliminar un elemento arbitrario de una lista enlazada es lento :(


Ejemplo 2

- ▶ Nos gustaría poder “llegar rápido” a un elemento del medio de la lista.


Ejemplo 2

- ▶ Nos gustaría poder “llegar rápido” a un elemento del medio de la lista.
- ▶ ¿Podemos guardarnos referencias a los nodos?

Ejemplo 2

- ▶ Nos gustaría poder “llegar rápido” a un elemento del medio de la lista.
- ▶ ¿Podemos guardarnos referencias a los nodos?
- ▶  NO, ¡romperíamos el encapsulamiento!

Ejemplo 2

- ▶ Nos gustaría poder “llegar rápido” a un elemento del medio de la lista.
- ▶ ¿Podemos guardarnos referencias a los nodos?
- ▶  NO, ¡romperíamos el encapsulamiento!
- ▶ Pero, parecido a los iteradores, podemos encapsular una referencia a un nodo.

- ▶ Un Handle es una **referencia abstracta** a un recurso manejado por otro sistema.
- ▶ Podemos pensarlo como un “iterador que no se mueve”, solo apunta ¹.
- ▶ Lo implementaremos como un objeto que, en su representación interna, guarda un puntero al Nodo de interés.
- ▶ Podría no tener métodos públicos, como usuarios solo nos interesa almacenarlo y usarlo como parámetro de métodos de nuestro TAD.
- ▶ Vamos a **cambiar la interfaz** de nuestra clase para que use Handles.

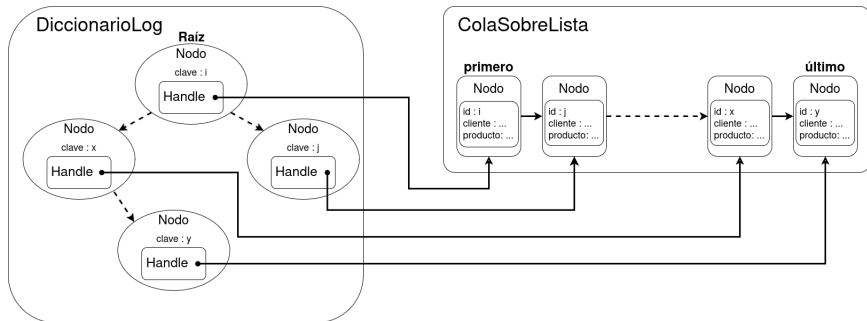
¹en algunos lenguajes, como C++, se suele usar un iterador para esta finalidad.

Handles - Implementación

```
public class ColaSobreLista<T> {  
    private int _longitud;  
    private Nodo _primero;  
    private Nodo _ultimo;  
  
    public class Handle {  
        private Nodo nodoApuntado;  
        public Handle(Nodo n) {  
            this.nodoApuntado = n;  
        }  
    }  
    ...  
}
```

```
public class ColaSobreLista<T> {  
    public ColaSobreLista() {...}  
    public boolean vacia() {...}  
    public Handle encolar(T x) {...}  
    public T desencolar() {...}  
    public T proximo() {...}  
    // Con el handle, nos evitamos el costo lineal de  
    // buscar el nodo  
    public void eliminar(Handle h) {...}  
}
```

Handles - Diagrama



También el Handle podría tener sus propios métodos.

```
public class ColaSobreLista<T> {  
    ...  
    public class Handle {  
        private Nodo nodoApuntado;  
        public Handle(Nodo n) {...}  
        // Obtiene el dato apuntado  
        public T obtener() {...}  
        // Elimina el dato apuntado de la cola  
        public void eliminar() {...}  
    }  
}
```

Ejemplo 2 - Estructura

```
class SistemaPedidos {  
    // Usamos la cola para resolver los pedidos en orden  
    ColaSobreLista<Pedido> colaDePedidos;  
    // Con los handles de la cola podemos:  
    // 1. Obtener info del Pedido  
    // 2. Eliminarlo eficientemente  
    DiccionarioLog<int, ColaSobreLista<Pedido>.Handle>  
        handlesPedidos;  
    ...  
}
```

Ejemplo 2 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, ColaSobreLista<Pedido>.Handle>  
        handlesPedidos;  
    ...  
}
```

- ▶ SistemaPedidos()
- ▶ void agregarPedido(Pedido p)
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)
- ▶ void cancelarPedido(int id)

Ejemplo 2 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, ColaSobreLista<Pedido>.Handle>  
        handlesPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p)
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)
- ▶ void cancelarPedido(int id)

Ejemplo 2 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, ColaSobreLista<Pedido>.Handle>  
        handlesPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo()
- ▶ Pedido infoPedido(int idPedido)
- ▶ void cancelarPedido(int id)

Ejemplo 2 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, ColaSobreLista<Pedido>.Handle>  
        handlesPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo() $\rightarrow O(\log(n))$
- ▶ Pedido infoPedido(int idPedido)
- ▶ void cancelarPedido(int id)

Ejemplo 2 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, ColaSobreLista<Pedido>.Handle>  
        handlesPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo() $\rightarrow O(\log(n))$
- ▶ Pedido infoPedido(int idPedido) $\rightarrow O(\log(n))$
- ▶ void cancelarPedido(int id)

Ejemplo 2 - Estructura

```
class SistemaPedidos {  
    ColaSobreLista<Pedido> colaDePedidos;  
    DiccionarioLog<int, ColaSobreLista<Pedido>.Handle>  
        handlesPedidos;  
    ...  
}
```

- ▶ SistemaPedidos() $\rightarrow O(1)$
- ▶ void agregarPedido(Pedido p) $\rightarrow O(\log(n))$
- ▶ Pedido proximo() $\rightarrow O(\log(n))$
- ▶ Pedido infoPedido(int idPedido) $\rightarrow O(\log(n))$
- ▶ void cancelarPedido(int id) $\rightarrow O(\log(n))$