Resolución de problemas enlazando distintas estructuras

Algoritmos y Estructuras de Datos

2^{do} cuatrimestre 2025

Motivación

- Vimos distintas estructuras de datos, las cuales nos permiten modelar TADs clásicos.
- Distintas estructuras de datos nos permiten resolver problemas de forma eficiente.
- Pero a veces no alcanza con una sola estructura.

Motivación

- Vimos distintas estructuras de datos, las cuales nos permiten modelar TADs clásicos.
- Distintas estructuras de datos nos permiten resolver problemas 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 - 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() → $O(1)$
- 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() → $O(1)$
- void agregarPedido(Pedido p) → 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() → $O(1)$
- void agregarPedido(Pedido p) → O(1)
- Pedido proximo() → O(1)
- Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

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

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

Ejemplo 1 - Estructura

Elijamos una estructura interna, pensando en las complejidades.

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

- SistemaPedidos() → $O(1)$
- void agregarPedido(Pedido p) → O(1)
- Pedido proximo() → O(1)
- Pedido infoPedido(int idPedido) → 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> diccPedidosPorId;
  ...
}
```

Ejemplo 1 - Estructura

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

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

Ejemplo 1 - Estructura

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

- SistemaPedidos() → $O(1)$
- void agregarPedido(Pedido p)
- Pedido proximo()
- Pedido infoPedido(int idPedido)

Ejemplo 1 - Estructura

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

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

Ejemplo 1 - Estructura

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

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

Ejemplo 1 - Estructura

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

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

Ejemplo 1 - Estructura

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

- SistemaPedidos() → $O(1)$
- void agregarPedido(Pedido p) → O(log(n))
- Pedido proximo() → O(log(n))
- Pedido infoPedido(int idPedido) → 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 `diccPedidosPorId` son los mismos.
- Cada pedido en `diccPedidosPorId` tiene como clave el id del pedido.

Ejemplo 1 - Código

```
public class SistemaPedidos {
  ColaSobreLista<Pedido> colaDePedidos;
  DiccionarioLog<Integer, Pedido> diccPedidosPorId;

  public SistemaPedidos() {
    colaDePedidos = new ColaSobreLista<>();
    diccPedidosPorId = new DiccionarioLog<>();
  }

  public void agregar(Pedido p) {
    colaDePedidos.encolar(p);
    diccPedidosPorId.definir(p.id, p);
  }

  ...
}
```

Ejemplo 1 - Código

```
public class SistemaPedidos {
  ColaSobreLista<Pedido> colaDePedidos;
  DiccionarioLog<Integer, Pedido> diccPedidosPorId;

  ...

  public Pedido proximo() {
    Pedido p = colaDePedidos.proximo();
    diccPedidosPorId.eliminar(p.id);
    return p;
  }

  public Pedido infoPedido(Integer id) {
    return diccPedidosPorId.obtener(id);
  }
}
```

Nuevos problemas

- 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: ℕ) : 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: ℕ) : 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: ℕ) : 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.

Handles

- Un `Handle` (del inglés para "manija") es una **referencia encapsulada** a un recurso de la estructura interna de otro sistema.
- Lo implementaremos como una clase anidada de la estructura correspondiente que, en su representación interna, guarda una referencia al objeto de interés.
- Sus atributos serán privados, podría o no tener métodos públicos. De tenerlos, estós deberían llamar a un método privado de su estructura correspondiente.
- Vamos a **cambiar la interfaz** de nuestra clase para que use Handles.

Handles - interfaz

```
interface Handle<T> {
    // Devuelve el valor del elemento
    public T valor();


    //Elimina al elemento de la coleccion
    public void eliminar();
}
```

Handles - Implementación

```
public class ColaSobreLista<T> {
  ...

  public class Handle {
    private Nodo nodoApuntado;

    private Handle(Nodo n) {
      this.nodoApuntado = n;
    }
    public T valor(){
      return nodoApuntado.valor;
    }
    public void eliminar() {
    //...
    }
  }
  ...
}
```

Handles - Implementación

```
public class ColaSobreLista<T> {
...

public class Handle implements Handle<T>{
  private Nodo nodoApuntado;

  private Handle(Nodo n) {
      this.nodoApuntado = n;
  }
  public T valor(){
      return nodoApuntado.valor;
  }
  public void eliminar() {
      eliminarNodo(nodoApuntado);
  }
}
...
// Con el handle, nos evitamos el costo lineal de
  buscar el nodo
private void eliminarNodo(Nodo h)  {...}
}
```

Handles - Signatura

```
public class ColaSobreLista<T> {
  public ColaSobreLista() {...}
  public boolean vacia()  {...}
  public Handle encolar(T x)  {...}
  public T desencolar()  {...}
  public T proximo()  {...}
}
```

Handles - Estructuras

Ahora que podemos guardarnos referencias de los elementos de la cola:

¿Cómo modificamos nuestro diseño para eliminar más rápido?

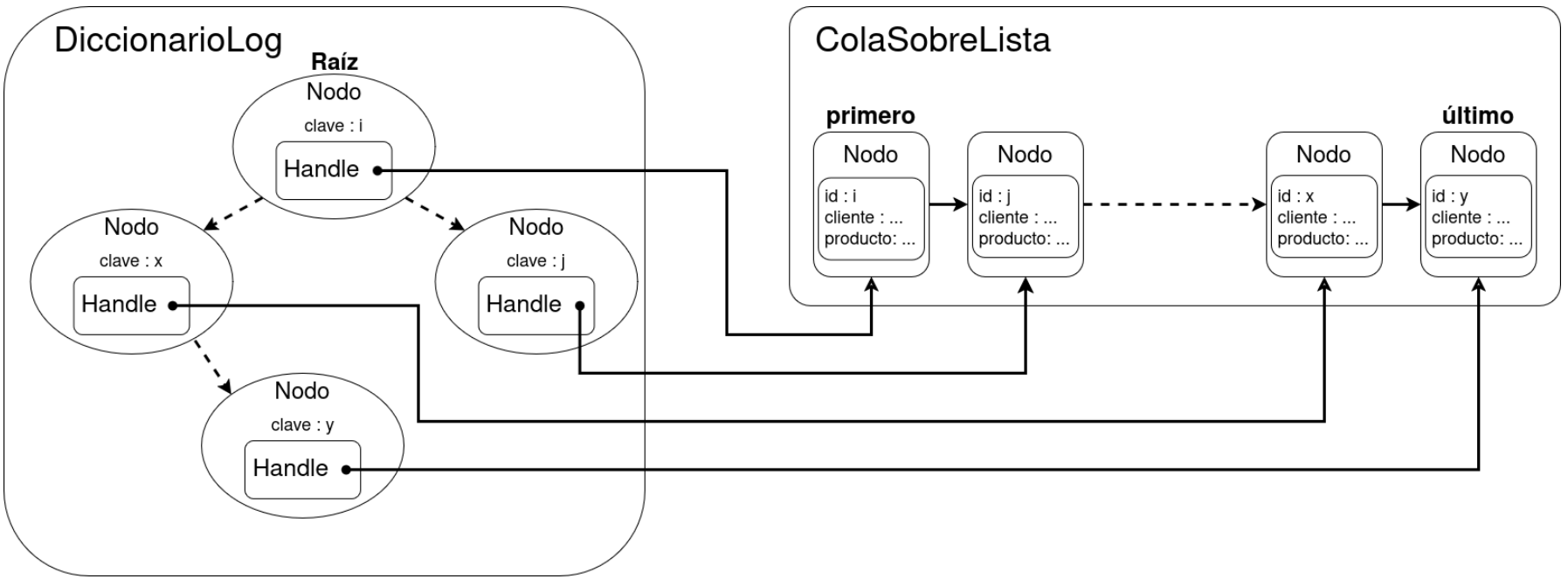
```
class SistemaPedidos {
  ColaSobreLista<Pedido> colaDePedidos;
  DiccionarioLog<int, Pedido> diccPedidosPorId;
    //...
  }
```

Handles - Estructuras

Podemos armar el diccionario con los Handles en vez de los pedidos

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

Handles - Diagrama



Ejemplo 2 - Costos Temporales

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

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

Ejemplo 2 - Costos Temporales

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

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

Ejemplo 2 - Costos Temporales

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

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

Ejemplo 2 - Costos Temporales

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

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

Ejemplo 2 - Costos Temporales

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

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

Ejemplo 2 - Costos Temporales

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

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

¡A programar!

Ya se encuentra subido el taller 5 - estructuras enlazadas y handles en el Campus.
Fecha *límite* de entrega: 09/11