



## 4. Colas (TDA e implementación con listas ligadas)

Carlos Zerón Martínez

Universidad Nacional Autónoma de México

*zeron@ciencias.unam.mx*

Martes 10 de Noviembre de 2020

# Introducción

El término cola (*Queue* en inglés) se relaciona con cosas de la vida cotidiana como por ejemplo:

- ▶ las filas para comprar en las tiendas
- ▶ las filas para abordar un medio de transporte público

La primera persona que se formó es la primera que se atiende.

Las personas que van llegando se tienen que incorporar a la fila de modo que siempre la última en llegar quede hasta atrás.

Propiedad FIFO (*First In First Out*):

El primer elemento que entra a la cola es el primero que sale de ella.

Las colas tienen dos posiciones fundamentales: la parte delantera, llamada **frente** y la parte trasera, denominada **fondo**.

## Características de la cola

La cola es una estructura de datos posicional, homogénea y dinámica, como la lista y la pila, que además cuenta con estas características:

- ▶ Se puede determinar si la cola es vacía.
- ▶ Se puede determinar el número de elementos de la cola.
- ▶ Se puede insertar un elemento al final de la cola (en el fondo)
- ▶ Se puede sacar el elemento ubicado al inicio (en el frente) de la cola y acceder a él para procesarlo de alguna forma

# Algunas aplicaciones de las colas

## Servicios en recursos compartidos

Para que varios usuarios puedan acceder a recursos como una impresora en red, o lo que se conoce como calendarización de procesos (asignar el procesador a un proceso o programa en ejecución por cierto intervalo de tiempo). Los usuarios obtienen acceso al recurso en un orden establecido.

## Acceso a servidores en Internet

La resolución de peticiones por parte de usuarios o clientes que interactúan a través de sus navegadores, en orden de llegada.

## Sistemas de atención telefónica

Se mantiene en espera a las personas en el orden en que se recibe la llamada, hasta que alguien puede otorgarles servicio.

# Especificación del TDA Cola

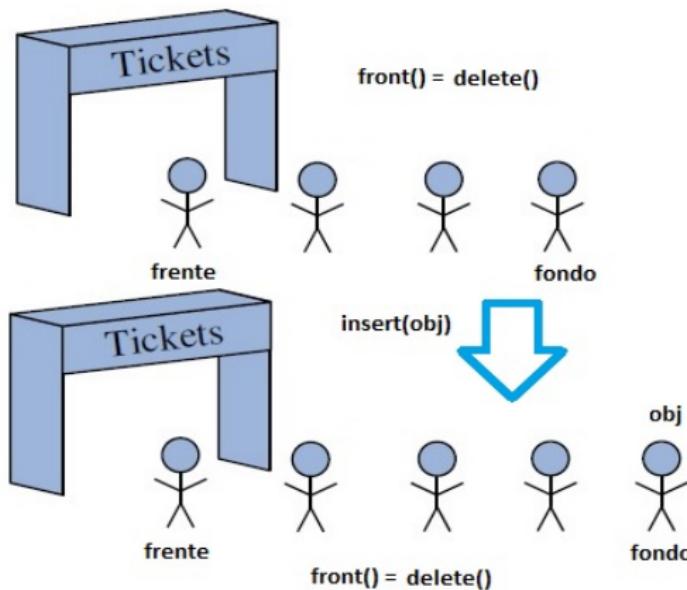
**Datos:** Objetos (una cantidad entera no negativa de ellos)

**Operaciones:**

- ▶  $\text{isEmpty}() : \{V, F\}$ . Devuelve un valor lógico que indica si la cola tiene elementos o no.
- ▶  $\text{size}() : \mathbb{Z}^+ \cup \{0\}$ . Devuelve el número de objetos en la cola.
- ▶  $\text{insert}(obj) : \emptyset$ . Inserta el objeto  $obj$  al fondo de la cola.  
Alternativa al nombre de la operación:  $\text{enqueue}(obj) : \emptyset$ .
- ▶  $\text{delete}() : Objeto$ . Devuelve y elimina el objeto en el frente de la cola. Si la cola es vacía, regresa un objeto nulo.  
Alternativa al nombre de la operación:  $\text{dequeue}() : Objeto$ .
- ▶  $\text{front}() : Objeto$ . Devuelve el objeto que se encuentra en el frente de la cola. Si la cola es vacía, regresa un objeto nulo.  
Alternativa al nombre de la operación:  $\text{first}() : Objeto$ .

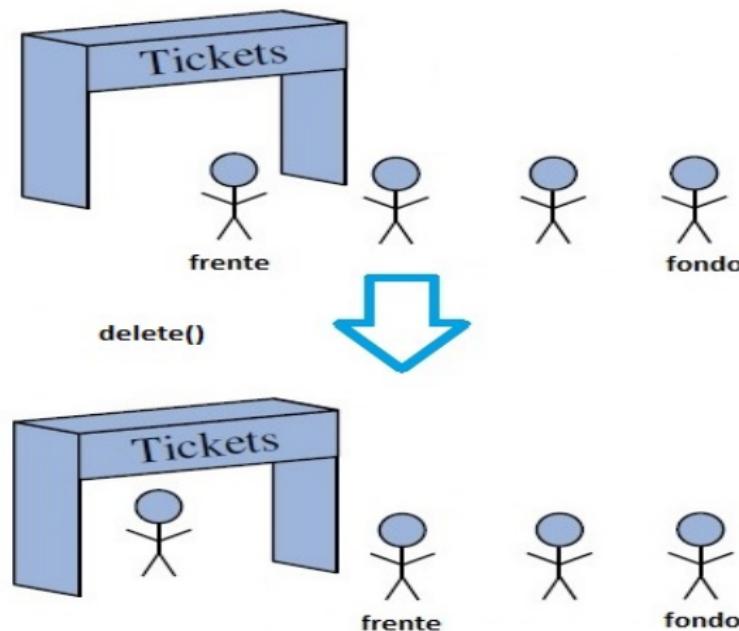
## Axiomas

La inserción de un objeto en el fondo de la cola incrementa en una unidad su tamaño y si ésta no estaba vacía antes de la inserción, entonces al efectuar un acceso al frente de la cola para eliminación o devolución, se accede al mismo objeto antes y después de la inserción.



## Axiomas

La eliminación de un objeto del frente de la cola decrementa el tamaño de la cola en una unidad y el elemento del frente es aquel elemento que fue insertado inmediatamente después del objeto eliminado de la cola.



# Interfaz para el TDA Cola

```
public interface Cola {  
  
    /**  
     * Devuelve un valor logico que indica si la cola tiene elementos o no.  
     *  
     * @return true si la cola no tiene elementos y false en caso contrario.  
     */  
    boolean isEmpty();  
  
    /**  
     * Devuelve el número de elementos de la cola.  
     */  
    int size();  
  
    /**  
     * Inserta el objeto dado al fondo de la cola.  
     *  
     * @param obj El objeto a insertar.  
     */  
    void insert(Object obj);
```

# Interfaz para el TDA Cola

```
/*
 * Devuelve y elimina el objeto en el frente de la cola.
 *
 * @return El objeto en el frente. Null si la cola es vacía.
 */
Object delete();

/*
 * Devuelve el objeto que se encuentra en el frente de la cola.
 *
 * @return El objeto en el frente. Null si la cola es vacía.
 */
Object front();

}
```

## Implementación de una cola basada en una lista ligada

Nuevamente mantenemos una lista ligada como único atributo, el frente de la cola será la primera posición de la lista ligada y el fondo, la última posición.



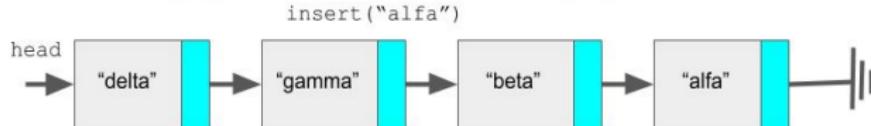
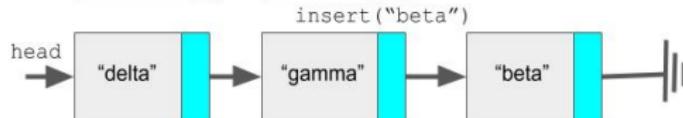
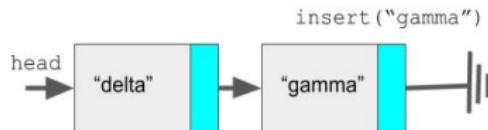
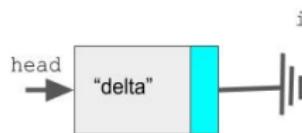
En la inserción de un objeto al fondo se considera la posición `size()`, mientras que la eliminación y acceso del objeto al frente queda en la posición 0.

# Implementación de una cola basada en una lista ligada

```
public class ColaLista implements Cola {  
  
    private final ListaLigada lista;  
  
    /*  
     * Construye una cola sin elementos.  
     */  
    public ColaLista() {  
        lista = new ListaLigada();  
    }  
  
    @Override  
    public boolean isEmpty() {  
        return lista.isEmpty();  
    }  
  
    @Override  
    public int size() {  
        return lista.size();  
    }  
  
    @Override  
    public void insert(Object obj) {  
        lista.add(lista.size(), obj);  
    }  
}
```

# Secuencia de operaciones insert

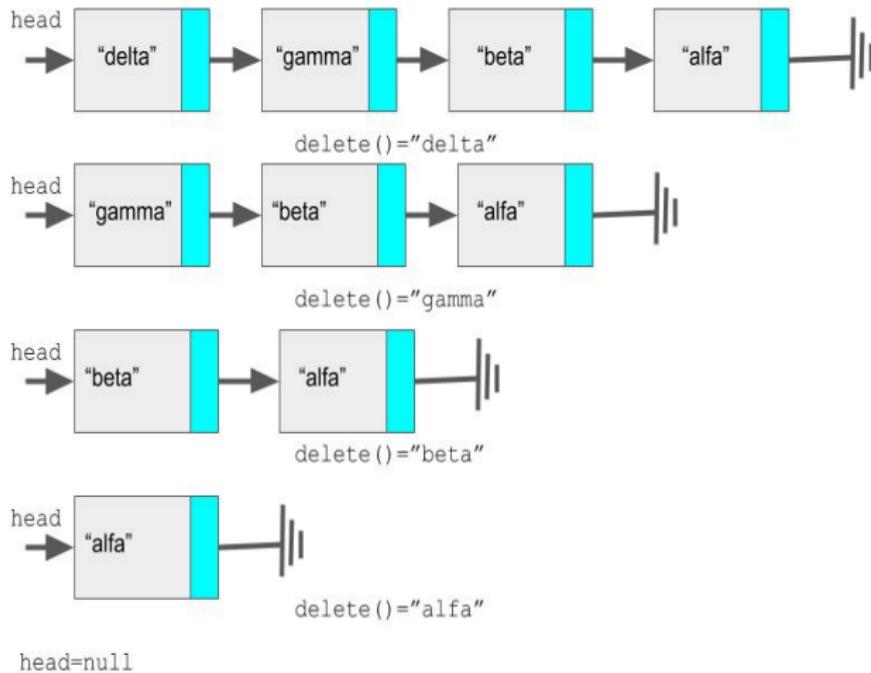
head=null



## Implementación de una cola basada en una lista ligada

```
@Override  
public Object delete() {  
    if (isEmpty()) {  
        return null;  
    }  
    Object frente = lista.get(0);  
    lista.remove(0);  
    return frente;  
}  
  
@Override  
public Object front() {  
    return isEmpty()? null : lista.get(0);  
}  
}
```

# Secuencia de operaciones delete



## Análisis de complejidad

Considerando como tamaño de la cola  $n$  el tamaño de la lista ligada que se emplea como atributo:

- ▶ Las operaciones `size()`, `isEmpty()` tienen complejidad constante porque el número de operaciones no dependen de  $n$
- ▶ La operación `front` requiere en el peor caso un acceso a la cabeza de la lista ligada, lo cual es  $O(1)$ .
- ▶ La operación `delete` requiere en el peor caso, además del acceso a la cabeza de la lista, una eliminación en esa misma posición, ambas operaciones toman tiempo constante, por lo que toda la operación también tiene complejidad  $O(1)$ .
- ▶ La operación `insert` necesita en el peor caso el recorrido de  $n$  nodos, lo que corresponde al peor caso de la inserción en la lista ligada, que es  $O(n)$ .

A partir de la implementación de una lista ligada, podemos definir los métodos de la cola.