

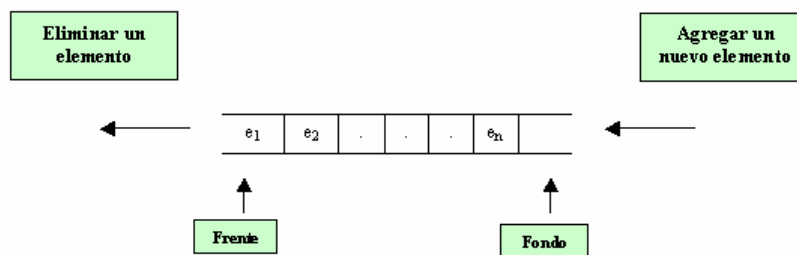


TAD: Cola

Un objeto **cola** del Tipo Abstracto de Dato **Cola** (*queue*) es un conjunto de elementos del mismo tipo en la cual todos los elementos se agregan por un extremo y las eliminaciones y accesos se realizan en el otro extremo.

Las colas trabajan como una estructura FIFO (first in, first out) similar a una cola de personas en un banco. En un banco existe un lugar reservado para que las personas se formen, lo que equivale a la operación de crear una cola. Si no hay nadie en la cola decimos que la cola está vacía. Cuando llega una persona al banco se coloca al final de la cola, esto corresponde a una operación de agregar. En el momento que una persona es atendida, deja de formar parte de la cola, esto es una eliminación de un elemento de la cola. Cuando el cajero termina de atender a una persona puede ver quién es la persona que se encuentra al frente de la cola.

La representación gráfica de una cola es:



El último elemento agregado se denomina **fondo** (*back*) y el primero **frente** (*front*). El único elemento visible y por consiguiente al único que se puede acceder es al frente. A esta operación la denominaremos **verFrente** (*getFront*). Una cola sin elementos es una cola vacía que la representaremos con el símbolo ϕ , y una operación analizadora importante es saber si una cola está o no vacía. Las operaciones que modifican el estado de una cola son agregar un nuevo elemento a la cola y sacar el elemento más antiguo. A la primera operación la denominaremos **agregar** (*enqueue*) y a la segunda, **eliminar** (*dequeue*).

Las colas se utilizan para simular situaciones que se encuentran frecuentemente, como por ejemplos, las colas que se forman frente a las cajas de los bancos, supermercados, cines, etc. También se utilizan para administrar los recursos de una computadora, que es una de las principales funciones de un sistema operativo, por ejemplo, controlar la cola de impresión.

Especificación del TAD Cola:

Nombre del TAD: Cola

Constructor:

- **Cola:** - \rightarrow **Cola**
 - Construye una cola vacía, esto es solicita memoria para poder almacenar los elementos en la cola.
 - precondiciones: -
 - postcondiciones: Cola $c = \phi$

Modificadoras:

- **agregar (enqueue) :** Cola X Elemento \rightarrow Cola
 - Agrega a la Cola c un nuevo elemento e
 - precondiciones: Cola $c = \langle e_1, e_2, e_3, \dots, e_n \rangle$ y elemento $e \neq e_i$

Cola $c = \phi$ y elemento e

⦿ postcondiciones: Cola $c = \langle e_1, e_2, e_3 \dots, e_n, e \rangle$ ó

Cola $c = \langle e \rangle$

• **eliminar (dequeue): Cola → Cola**

⦿ Saca de la Cola c el elemento insertado más antiguamente.

⦿ precondiciones: Cola $c \neq \phi$ o sea $c = \langle e_1, e_2, e_3 \dots, e_n \rangle$

⦿ postcondiciones: Cola $c = \langle e_2, e_3 \dots, e_n \rangle$

Analizadoras:

• **verFrente (getFrente): Cola → Elemento**

⦿ Recupera el valor del elemento que está en el frente.

⦿ precondiciones: Cola $c \neq \phi$ o sea $c = \langle e_1, e_2, e_3 \dots, e_n \rangle$

⦿ postcondiciones: Elemento e_1

• **esVacia (isEmpty): Cola → boolean**

⦿ Informa si la Cola c está o no vacía.

⦿ precondiciones: Cola c

⦿ postcondiciones: VERDADERO si $c = \phi$; FALSO si $c \neq \phi$.

Interface en Java

```
public interface Cola {
    void agregar(Object x);
    void eliminar();
    Object verFrente();
    boolean esVacia();
}
```

Ejercicios

📖 Ejercicio 1

Realizar el seguimiento de las instrucciones que se dan a continuación. Suponer que C es una cola de números enteros y las variables x , y y z son tipo entero.

- Crear C ; agregar el número 7; agregar el 8; mostrar el frente; remover; agregar el número 9; mostrar el frente; agregar el 10; agregar el 2; mostrar el frente; remover.
- Crear C ; agregar el número 7; guardar el frente en x ; remover; agregar el número 9; agregar el valor de x ; mostrar el frente; remover; agregar el 2; mostrar el frente; agregar el 3; mostrar el frente.

📖 Ejercicio 2

Especificar formalmente y desarrollar los siguientes métodos, utilizando sólo las operaciones primitivas de cola (cuando sea necesario se pueden utilizar las de pila):

- Calcular la longitud de una cola.
- Concatenar dos colas dejando el resultado en la primera de ellas.
- Informar si un elemento dado se encuentra en la cola.
- Agregar un nuevo elemento en una posición dada (colarse).

- e) Dado un elemento sacarlo de la cola todas las veces que aparezca.
- f) Dada una cola de enteros y un número entero X, construir dos colas de manera que en una queden todos elementos menores a X y en la otra todos los mayores.
- g) Invertir los elementos de una cola.
- h) Colocar el elemento elem como primero en la cola.



Ejercicio 3

El estacionamiento de las avionetas de un aeródromo es en línea, con una capacidad de hasta 12 avionetas. Las avionetas llegan por el extremo izquierdo y salen por el extremo derecho. Cuando llega un piloto a recoger su avioneta, si ésta no está justamente en el extremo de salida (derecho), todas las avionetas a su derecha han de ser retiradas, sacar la suya y las retiradas colocadas de nuevo en el mismo orden relativo en que estaban. La salida de una avioneta supone que las demás son movidas hacia delante, de tal forma que los espacios libres del estacionamiento están en la parte izquierda.

El programa para emular este estacionamiento tiene como entrada un carácter que indica una acción sobre la avioneta, y la matrícula de la avioneta. La acción puede ser llegada (E) o salida (S) de avioneta. En la llegada puede ocurrir que el estacionamiento esté lleno; si es así, la avioneta espera hasta que se quede una plaza libre, o hasta que se de la orden de retirada (salida).

Almacenamiento en memoria

Igual que hemos visto en el TAD Pila, los elementos de una cola los podemos guardar en memoria utilizando dos tipos de almacenamiento:

-  memoria estática
-  memoria dinámica

Descripción de las distintas implementaciones

Vamos a estudiar distintos tipos de implementaciones, unas con memoria estática y otras con memoria dinámica y encontraremos ventajas y desventajas de cada una de ellas.

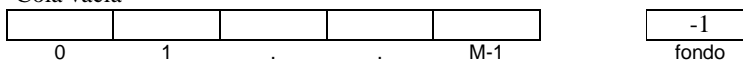
Memoria estática o contigua

Primer método:

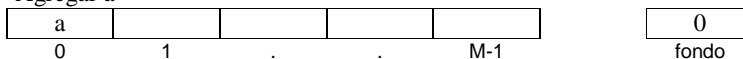
Se usa un arreglo como soporte de la cola de la siguiente forma: la primera posición es siempre el frente de la cola y para agregar nuevos elementos se incrementa el índice. La operación de agregar un nuevo elemento no es costosa en tiempo de procesamiento, pero la operación de remover el elemento del frente puede resultar un poco lenta si la cola tiene muchos elementos porque cada vez que se elimine el primer elemento se deberá hacer un corrimiento de los restantes elementos de la cola para ocupar la posición vacante dejada al remover el frente.

En el siguiente diagrama mostramos el estado de la cola y el valor de frente y fondo a medida que se realizan operaciones de agregar un nuevo elemento y eliminar el que está en el frente. M es el tamaño del arreglo. Además del arreglo donde se almacenan los elementos de la cola deberá tener una variable entera para indicar donde está el fondo. El frente es siempre cero.

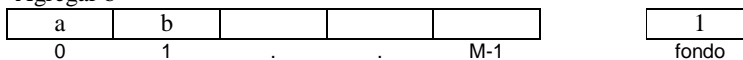
Cola vacía



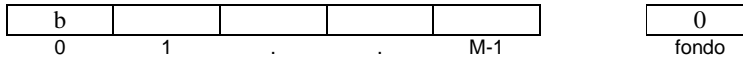
Agregar a



Agregar b



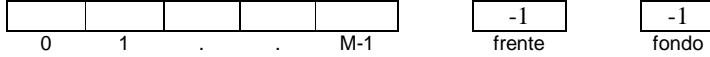
Eliminar a



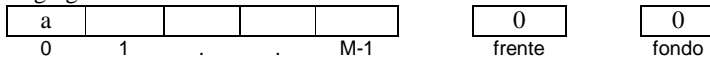
Segundo método:

Se puede mejorar el método explicado dejando que el frente sea movable, es decir, que no quede fijo en la primera posición sino que a medida que los elementos van saliendo, el frente se mueva a la posición siguiente. La desventaja de este método es que cuando se vacían las primeras celdas del arreglo no vuelven a ser ocupadas con el consiguiente desperdicio de espacio. La cola además del arreglo deberá tener dos variables enteras para indicar donde está el frente y donde está el fondo.

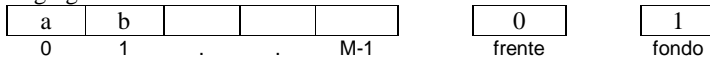
Cola vacía



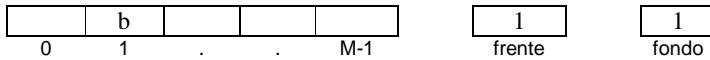
Agregar a



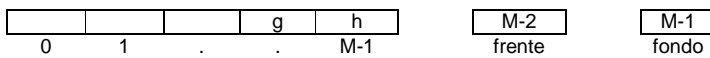
Agregar b



Eliminar a



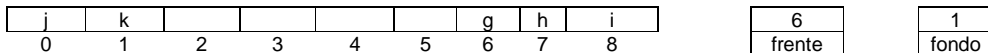
Si por ejemplo llegamos a tener



En esta situación, no se puede agregar un nuevo elemento, pues el índice del último está indicando que ya está ocupada la última posición, aunque haya M-2 lugares disponibles (los M-2 primeros lugares del arreglo).

Tercer método:

Los dos métodos anteriores son ampliamente aventajados si en vez de pensar en un arreglo como algo lineal que tiene un comienzo y un fin, pensamos que es un anillo que no tiene ni principio ni fin. A esta representación se la denomina **arreglo circular**, donde la última celda del arreglo “apunta” a la primera. Esto se logra algorítmicamente, puesto que no hay manera de lograr físicamente que esto sea así.



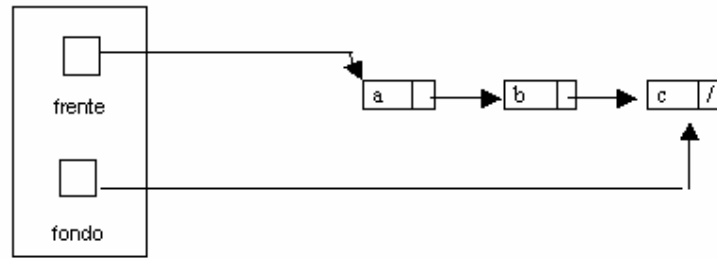
El problema es determinar cuando el arreglo está lleno, lo que se soluciona agregando una variable entera que indique la cantidad de elementos que hay realmente en la cola. En el caso anterior dirá 5. Si el valor de tal variable fuera 9 y hay que agregar un nuevo elemento entonces habría que agrandar el arreglo.

Memoria dinámica

Cuarto método:

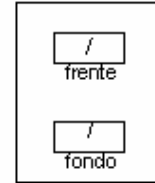
Los elementos de la cola se almacenan en nodos los cuales hacen referencia al siguiente elemento de la cola. Además se necesitan dos variables frente y fondo pero ahora deben ser de tipo Nodo no enteras como en el caso de los tres métodos anteriores.

Por ejemplo si los elementos ingresaron en la cola en el siguiente orden: a, b y c; la representación en memoria de la misma sería:



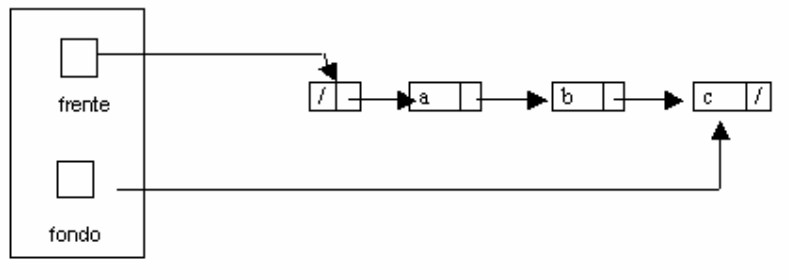
En una cola vacía tanto frente como fondo apuntan a null.

Esta representación tiene la desventaja que cada vez que se agrega un nuevo elemento habría que consultar si la cola está o no vacía. Cada vez que se agrega un nuevo elemento se modifica por supuesto el valor de fondo, pero si la cola está vacía también hay que modificar el valor de frente.



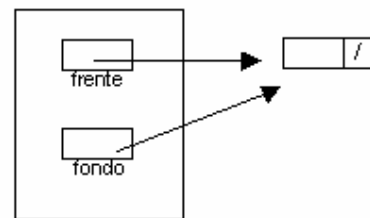
Quinto método:

Para solucionar el problema comentado en el método anterior, usaremos un **nodo cabecera**. Un nodo cabecera es un nodo de la estructura pero que no lleva información. El nodo cabecera ocupa la primera posición física de la cola mientras que el segundo nodo físico es el primero lógicamente. A partir del segundo nodo se encuentran los datos de la cola. Con esta representación la cola anterior se almacena así:



El uso del nodo cabecera tiene la ventaja que no se deberá preguntar cada vez que se agrega un nuevo elemento a la cola si la cola está vacía. Al realizar una eliminación, lo que se elimina es el nodo cabecera quedando como nuevo nodo cabecera el nodo que contiene la información que se quiere eliminar. ¿Por qué parece más eficiente eliminar físicamente el nodo cabecera en lugar del primer nodo que tiene información?

Finalmente notemos que la cola vacía sólo tiene el nodo cabecera y tanto frente como fondo apuntan a él.



Ejercicio 4

- Implementar el TDA cola utilizando un arreglo circular.
- Implementar el TDA cola utilizando memoria dinámica con nodo cabecera.
- Utilizando cualquier implementación programar el ejercicio 3.

Ejercicio 5

- Una bicola es una estructura lineal en la cual los elementos pueden ser adicionados, eliminados y consultados por ambos extremos. Especificar formalmente el TDA bicola.

- b) En una cola de prioridad cada elemento tiene asignada una prioridad. Para sacar un elemento, lo mismo que para consultarlo se toma el más antiguo de mayor prioridad. Especificar formalmente el TDA cola de prioridad.