UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Tema 5. Tipos de datos lineales

Programación (PRG)

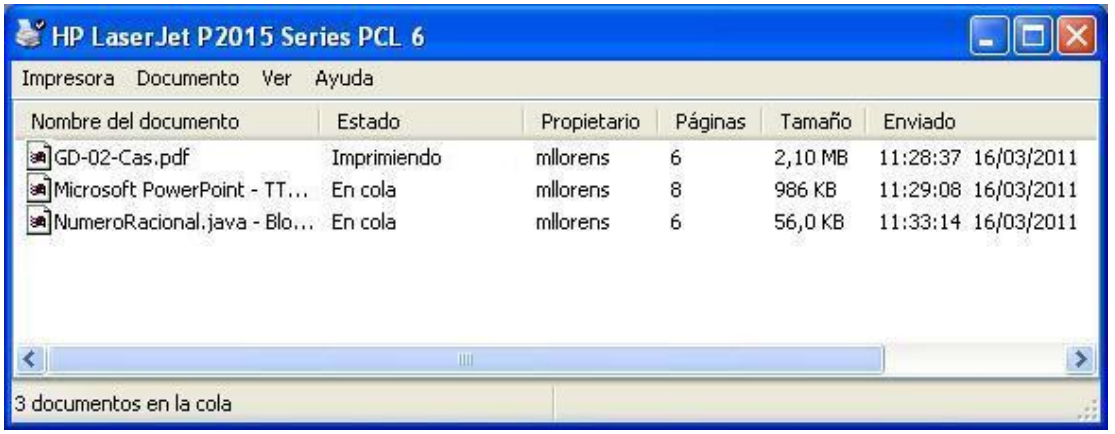
Jorge González Mollá

Departamento de Sistemas Informáticos y Computación

Índice

Definición

- Una *cola* (*queue*) es una colección de datos del mismo tipo en la que el acceso se realiza siguiendo un criterio *FIFO* (*F*irst *I*n *F*irst *O*ut).
  - El primer elemento que llega a la cola (que se inserta en la cola) será el primero en ser atendido (en ser eliminado de la cola).
- Ejemplos de colas:



Interfaz de operaciones (API)

Operación	Descripción
public **Queue**()	Crea una nueva **Queue** vacía
public void **add**(Tipo x)	Añade el dato al final de la **Queue**
public Tipo **remove**()	Extrae el dato de la cabeza de la **Queue** y lo devuelve *
public Tipo **element**()	Devuelve (sin desencolarlo) el dato de la cabeza de la **Queue** *
public boolean **isEmpty**()	Devuelve `true` si la **Queue** está vacía y `false` en caso contrario
public int **size**()	Devuelve el nº de datos de la **Queue** (≥0)

* Lanza `NoSuchElementException` si la **Queue** está vacía.

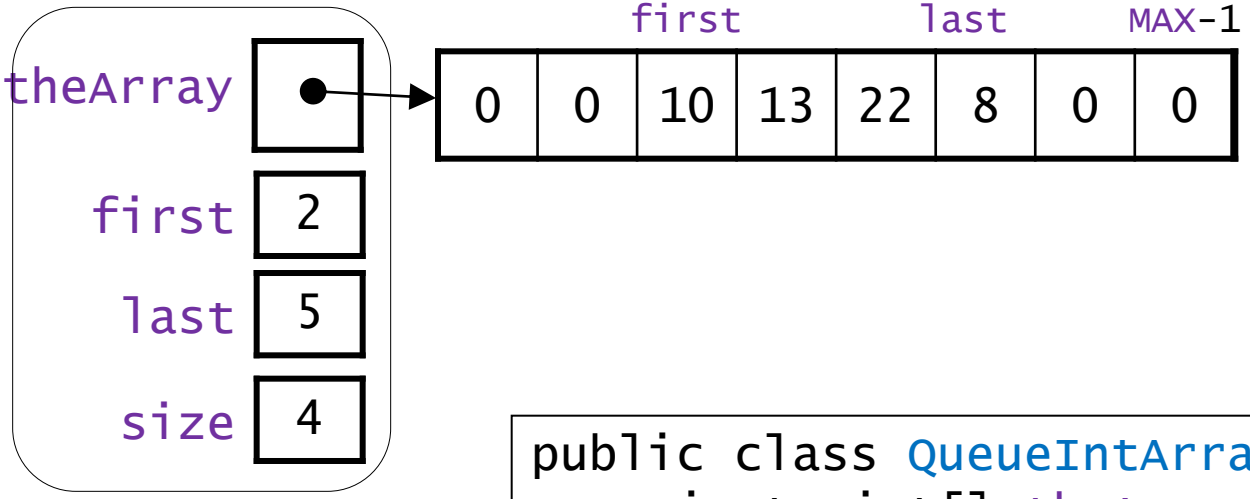
Se puede conseguir implementar estas operaciones con coste constante.

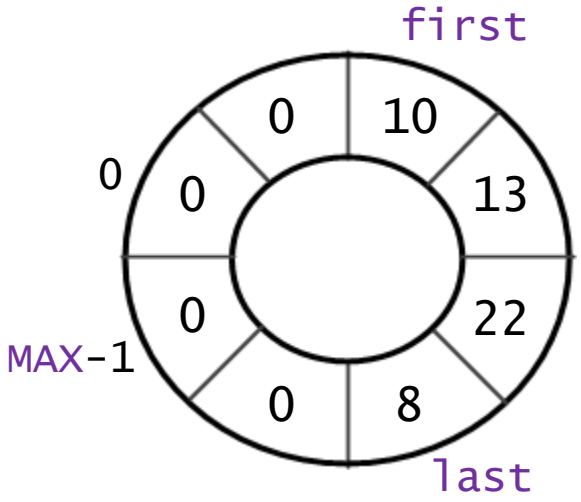
Implementación mediante arrays

- Una cola se puede implementar utilizando un array (`theArray`) que almacene los datos, dos índices (`first` y `last`) que marquen los dos puntos de acceso a la cola y una constante que defina la dimensión máxima del array (`MAX`).
- Para poder reutilizar todas las posiciones del array sin tener que desplazar elementos y así llegar a su final con todas sus posiciones ocupadas, se considera el array como una *estructura circular*, sin principio ni fin, donde después del último elemento va el primero.
- Además, para poder distinguir las situaciones de una cola sin datos (cola vacía) y una cola con el máximo número de datos (cola llena), se añade un atributo (`size`) que sea el número de datos de la cola.

Implementación mediante arrays

- Cola cuyos datos son de tipo `int`:






```
public class QueueIntArray {
   private int[] theArray;
   private int first, last, size;
   private static final int MAX = ...;

   // Implementación de las operaciones:
   ...
}
```

Implementación mediante arrays

- Constructor `QueueIntArray`: Crea el array e inicializa los puntos de acceso, indicando que la cola está vacía.

```
public QueueIntArray() {
   theArray = new int[MAX];
   size = first = 0; last = -1;
}
```

- Operación `add`:

```
public void add(int x) {
   if (size == theArray.length) {
      duplicateArray(); }
   else {
      last = (last + 1) % theArray.length;
      theArray[last] = x;
      size++;
   }
}
```

Implementación mediante arrays

- Operaciones `remove` y `element`:

```
public int remove() {
    if (size == 0) { throw new NoSuchElementException(); }
    int x = theArray[first];
    first = (first + 1) % theArray.length;
    size--;
    return x;
}

public int primero() {
    if (size == 0) { throw new NoSuchElementException(); }
    return theArray[first];
}
```

Implementación mediante arrays

- Operación `isEmpty`:

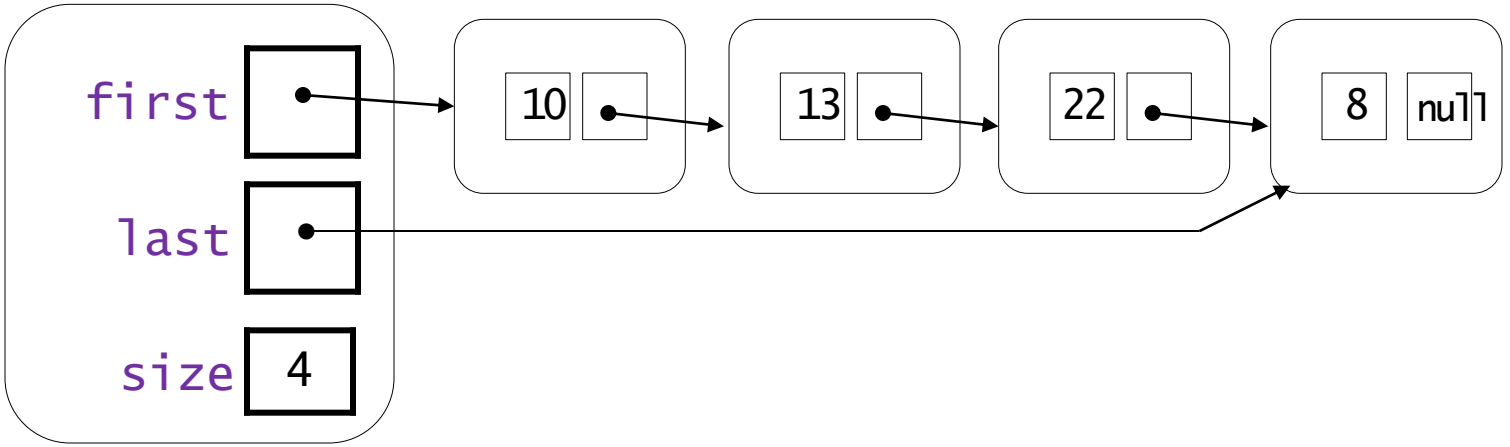
```
public boolean isEmpty() {
    return size == 0;
}
```

- Operación `size`:

```
public int size() {
    return size;
}
```

Implementación enlazada

- En una implementación enlazada, la cola se gestiona por medio de dos atributos que dan acceso directo al principio y al final de la cola.




```
public class QueueIntLinked {
    private NodeInt first, last;
    private int size;

    // Implementación de las operaciones:
    ...
}
```

Implementación enlazada

- Constructor `QueueIntLinked`: Asigna `null` a las referencias `first` y `last` e inicializa `size` a 0.

```
public QueueIntLinked() {
   first = null;
   last  = null;
   size  = 0;
}
```

- Operación `add`:

```
public void add(int x) {
   NodeInt nuevo = new NodeInt(x);
   if (last != null) { last.next = nuevo; }
   else { first = nuevo; }
   last = nuevo;
   size++;
}
```

Implementación enlazada

- Operaciones `remove` y `element`:

```
public int element() {
    if (size == 0) { throw new NoSuchElementException(); }
    int x = first.data;
    first = first.next;
    if (first == null) { last = null; }
    size--;
    return x;
}

public int primero() {
    if (size == 0) { throw new NoSuchElementException(); }
    return first.data;
}
```

Implementación enlazada

- Operación `isEmpty`:

```
public boolean isEmpty() {
   return first == null;
}
```

- Operación `size`:

```
public int size() {
   return size;
}
```

Comparación de implementaciones

- La complejidad temporal de todas las operaciones en las dos representaciones estudiadas es independiente de la talla del problema: $\Theta(1)$.
- En cuanto a la complejidad espacial, la implementación con arrays presenta el inconveniente de la estimación del tamaño máximo del array y la reserva de espacio que en muchos casos no se utilizará, igual que sucedía en el caso de las pilas.
- Pero igual que en las pilas, este consumo adicional de espacio no tendrá demasiada importancia si el tipo de las componentes de `theArray` es relativamente pequeño, como es el caso de una cola de `int`, o de colas de objetos (las componentes del array son entonces referencias).
- También en este caso, la representación enlazada requiere un espacio de memoria adicional para los enlaces.