3. Listas (TDA e implementación con arreglos)

Carlos Zerón Martínez

Universidad Nacional Autónoma de México

zeron@ciencias.unam.mx

Martes 20 al Jueves 22 de Octubre de 2020

Introducción

El concepto de mayor relevancia relacionado con listas es la posición.

$\Rightarrow$ Podemos saber cual es el primer elemento, el segundo y así sucesivamente

Una lista es una colección ordenada o posicional de los mismos, refiriendo al orden en el sentido de que cada elemento tienen una posición:

- lista de números telefónicos
- registros de pago
- artículos de tienda
- entre otros ejemplos

Características de una lista

1. **Posicional (Ordenada)**. Los elementos mantienen una posición en la cual son almacenados: se coloca un elemento detrás de otro.

2. **Homogénea**. El conjunto de datos que almacena la lista pertenecen al mismo dominio genérico.

3. **Dinámica**. El espacio requerido para el almacenamiento de elementos no es fijo, sino variable, es decir, se puede almacenar un número finito variable de elementos.

Operaciones de una lista

1. Insertar nuevos elementos en la lista en cualquier posición, incluyendo antes de la primera (inicio de la lista) y después de la última (final de la lista).

2. Eliminar elementos de cualquier posición existente de la lista.

3. Acceder a cualquier elemento existente en la lista.

4. Sustituir cualquier elemento dentro de la lista.

5. Determinar si la lista tiene o no elementos.

6. Determinar el número de elementos en la lista.

Especificación del TDA Lista

Datos:

Un número no negativo de objetos de un mismo tipo genérico

Operaciones:

- isEmpty() : $\{V, F\}$. Devuelve un valor lógico que indica si la lista tiene elementos o no.
- size() : $\mathbb{N} \cup \{0\}$. Devuelve el número de elementos de la lista (tamaño).
- get(i) : Objeto. Devuelve el objeto que se encuentra en la posición i-ésima de la lista, donde $0 \leq i \leq \text{size}() - 1$.

Especificación del TDA Lista

Operaciones:

- set(i, obj) : Objeto. Sustituye el objeto que se encuentra en la posición i-ésima de la lista con el objeto obj, donde $0 \leq i \leq \text{size}() - 1$, y regresa el objeto sustituido.
- add(i, obj) : $\varnothing$. Inserta un objeto obj en la i-ésima posición de la lista, donde $0 \leq i \leq \text{size}()$.
- remove(i) : Objeto. Elimina y devuelve el elemento ubicado en la i-ésima posición de la lista, donde $0 \leq i \leq \text{size}() - 1$.

Axiomas:

- Si se agrega un elemento en una posición i entonces el tamaño de la lista se incrementa en una unidad.
- Si se elimina un elemento de la posición i, el tamaño de la lista se decrementa en una unidad.

Especificación del TDA Lista

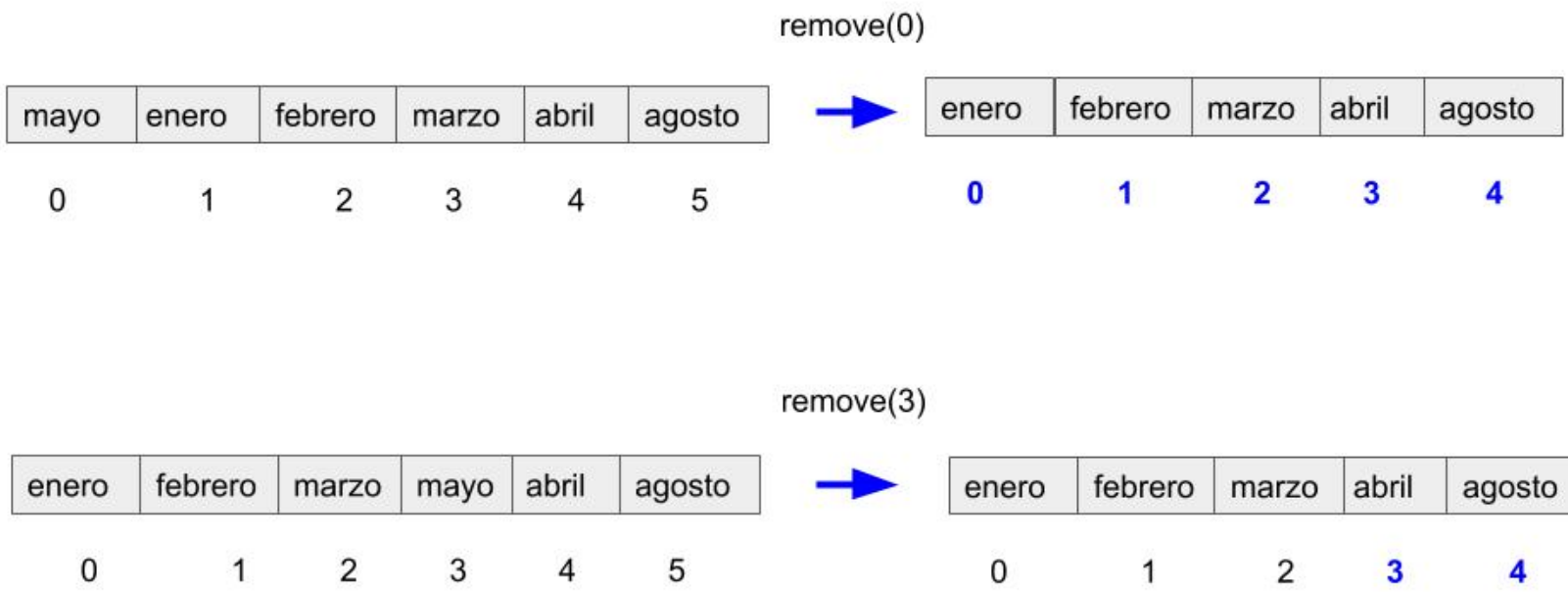
Axiomas:

- Si se agrega un elemento en una posición i, donde $0 \leq i \leq \text{size}() - 1$, entonces los elementos que originalmente estaban en las posiciones iguales o superiores a la i-ésima incrementan su posición en una unidad.

Especificación del TDA Lista

Axiomas:

- Si se elimina un elemento de la posición i, entonces los elementos que originalmente estaban en las posiciones superiores a la i-ésima disminuyen su posición en una unidad.



Interfaz para el TDA Lista (parte 1)

```
public interface Lista {

    /**
     * Devuelve un valor lógico que indica si la lista
     * tiene elementos o no.
     */
    boolean isEmpty();

    /**
     * Devuelve el número de elementos de la lista.
     */
    int size();

    /**
     * Devuelve el objeto que se encuentra en la posición
     * i-ésima de la lista.
     */
    Object get(int i) throws IndexOutOfBoundsException;
```

Interfaz para el TDA Lista (parte 2)

```
    /**
     * Inserta el objeto obj en la i-ésima posición de la lista.
     */
    void add(int i, Object obj) throws IndexOutOfBoundsException;

    /**
     * Sustituye el objeto en la i-ésima posición de la lista por el objeto obj
     * y regresa el objeto sustituido.
     */
    Object set(int i, Object obj) throws IndexOutOfBoundsException;

    /**
     * Elimina y devuelve el elemento ubicado en la i-ésima posición de la lista.
     */
    Object remove(int i) throws IndexOutOfBoundsException;

}
```

Implementación basada en un arreglo

Inconvenientes:

- limitación predeterminada del espacio en memoria al instanciar un arreglo
- necesidad de conocer un límite sobre el número de localidades para almacenar elementos para inserciones futuras

Atributos

- **Arreglo de objetos**.
- **tamaño físico de la lista**. Número máximo de elementos que pueden almacenarse en el arreglo.
- **tamaño lógico de la lista**. Número de elementos almacenados en el arreglo que corresponde al número real de elementos en la lista
- **tamaño físico de la lista por omisión**. Número máximo de localidades del arreglo por omisión.

Implementación de lista como arreglo (Código)

```java
/**
 * Implementación de una Lista basada en un arreglo.
 */
public class ListaArreglo implements Lista {

    // máximo tamaño fisico por omisión
    private static final int MAXIMO_ELEMENTOS = 1000;

    // tamaño físico
    private int maximoElementos;

    // tamaño lógico
    private int numeroElementos;

    // arreglo que se usa para el almacenamiento
    private Object[] elementos;
```

Implementación de lista como arreglo (Código)

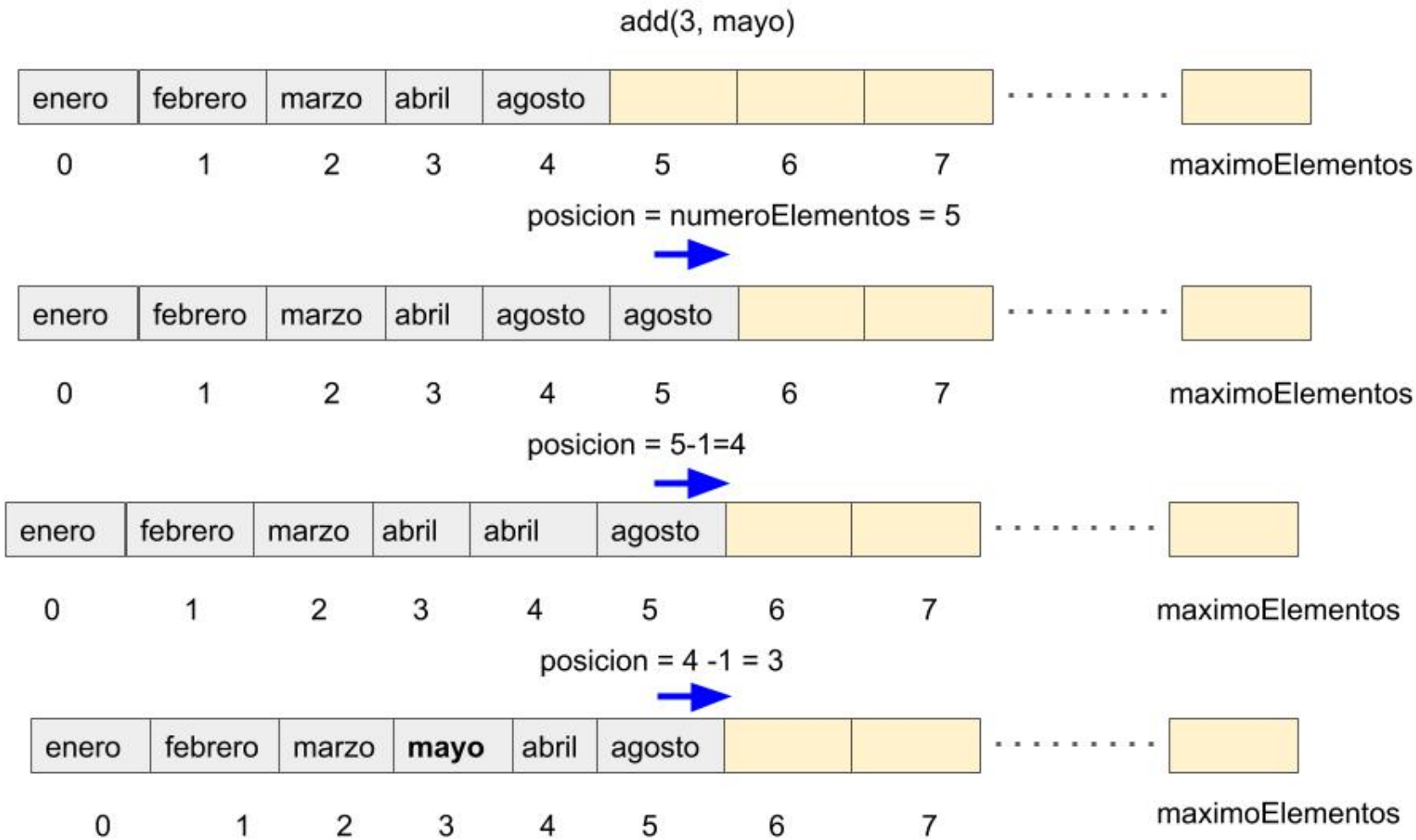
```java
/**
 * Constructor de una lista con un máximo
 * número de elementos por omisión.
 */
public ListaArreglo(){
    this(MAXIMO_ELEMENTOS);
}

/**
 * Constructor que recibe el número de elementos máximo
 * que va a almacenar la lista.
 *
 * @param num El número de elementos máximo.
 */
public ListaArreglo(int num) {
    maximoElementos = num;
    numeroElementos = 0;
    elementos = new Object[num];
}

@Override
public boolean isEmpty() {
    return numeroElementos == 0;
}

@Override
public int size() {
    return numeroElementos;
}
```

Implementación de lista como arreglo (Código)

```
@Override
public Object get(int i) throws IndexOutOfBoundsException {
    if (numeroElementos == 0) {
        throw new IndexOutOfBoundsException("Indice invalido. Lista vacía");
    }
    if (i < 0 || i > numeroElementos - 1) {
        throw new IndexOutOfBoundsException("Indice invalido");
    }
    return elementos[i];
}
```

Implementación de la inserción

add(i, obj). Se aumenta una posición a los elementos a partir del que aparece en la última posición hasta el de la i-ésima y después se coloca el elemento obj en la i-ésima posición.



Implementación de lista como arreglo (Código)

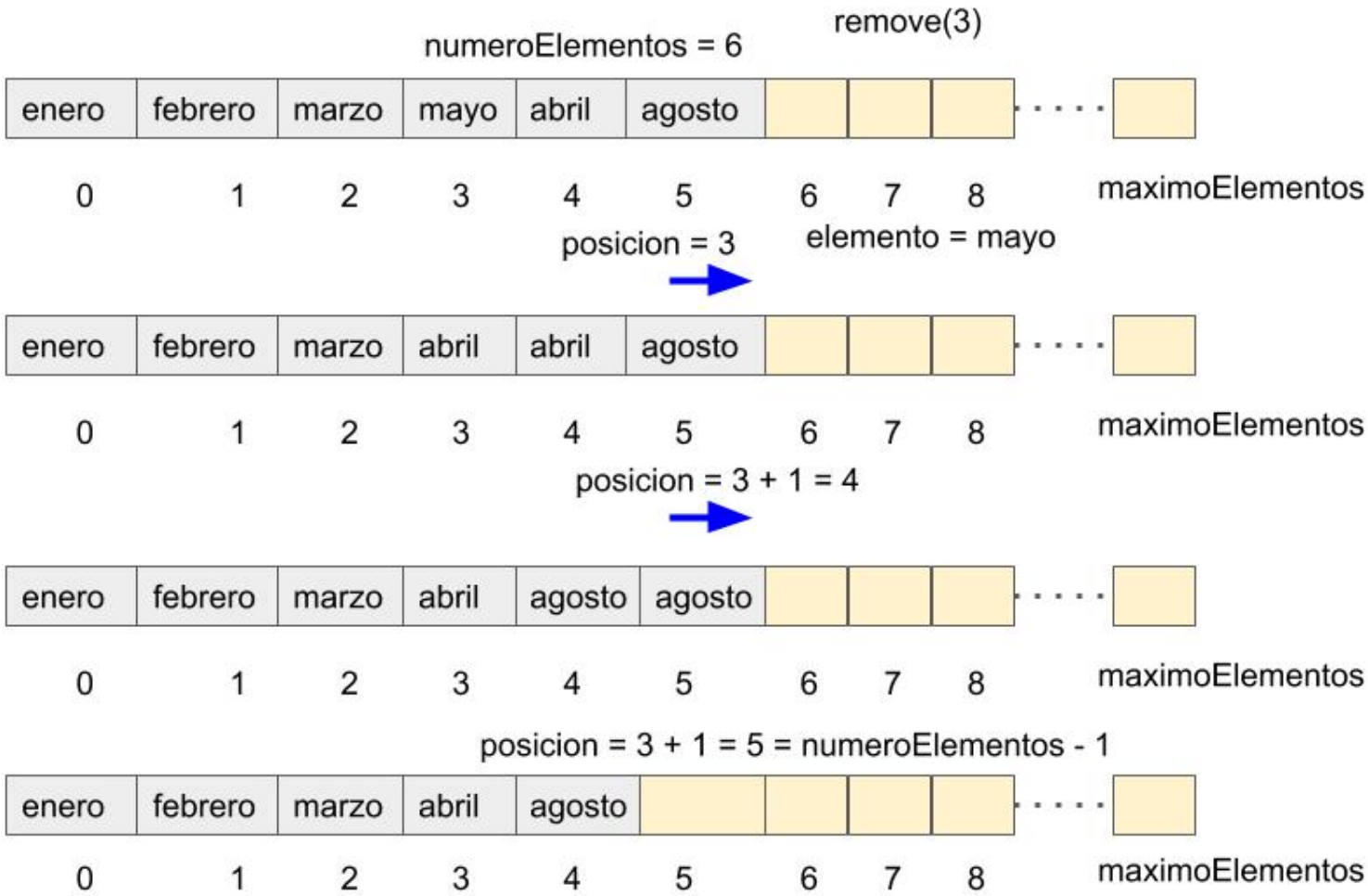
```
@Override
public void add(int i, Object obj) throws IndexOutOfBoundsException, IllegalStateException {
    if (numeroElementos == maximoElementos) {
        throw new IllegalStateException("Memoria insuficiente");
    }

    if (i < 0 || i > numeroElementos) {
        throw new IndexOutOfBoundsException("Indice inválido");
    }

    // recorremos una posición hacia adelante los elementos
    // en las posiciones comprendidas desde numElems - 1 hasta i
    int posicion = numeroElementos;
    while (posicion > i) {
        elementos[posicion] = elementos[posicion - 1];
        posicion--;
    }
    elementos[posicion] = obj;
    numeroElementos++;
}
```

Implementación de lista como arreglo (Código)

```java
@Override
public Object set(int i, Object obj) throws IndexOutOfBoundsException {
    if (numeroElementos == 0) {
        throw new IndexOutOfBoundsException("Indice invalido. Lista vacía");
    }
    if (i < 0 || i > numeroElementos - 1) {
        throw new IndexOutOfBoundsException("Indice invalido");
    }
    Object elemento = elementos[i];
    elementos[i] = obj;
    return elemento;
}
```

Implementación de la eliminación

remove(i). Se disminuye una posición a los elementos a partir de la posición $(i+1)$ para que el elemento en la posición i-ésima ya no esté en el arreglo. Nulificamos la última posición ocupada.



Implementación de lista como arreglo (Código)

```
@Override
public Object remove(int i) throws IndexOutOfBoundsException {
    if (numeroElementos == 0) {
        throw new IndexOutOfBoundsException("Indice inválido. Lista vacía");
    }

    if (i < 0 || i > numeroElementos - 1) {
        throw new IndexOutOfBoundsException("Indice inválido");
    }

    // recorremos un lugar hacia la izquierda los elementos
    // en las posiciones comprendidas desde i+1 hasta numElems
    int posicion = i;
    Object elemento = elementos[posicion];
    while (posicion < numeroElementos - 1) {
        elementos[posicion] = elementos[posicion + 1];
        posicion++;
    }
    elementos[posicion] = null;
    numeroElementos--;
    return elemento;
}
```

Análisis de complejidad

- Suponiendo que el tamaño lógico de la lista es n, esto es, el número de elementos es n:
- Los métodos isEmpty(), size(), get(i) y set(i, obj) tienen tiempo constante $O(1)$.
- El peor caso de los métodos add(i, obj) y remove(i) es $O(n)$, cuando se opera en la primera posición del arreglo.