

## Estructuras de Datos Clase 6 – Listas e Iteradores (primera parte)



Dr. Sergio A. Gómez  
<http://cs.uns.edu.ar/~sag>



Departamento de Ciencias e Ingeniería de la Computación  
Universidad Nacional del Sur  
Bahía Blanca, Argentina

### Listas enlazadas (*Node lists o linked lists*)

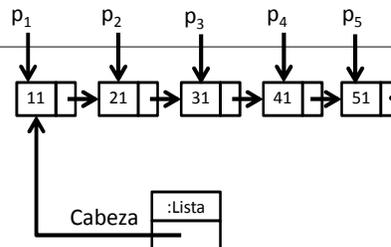
- Una lista  $L$  es una secuencia de  $n$  elementos  $x_1, x_2, \dots, x_n$
- Representaremos a  $L$  como  $[x_1, x_2, \dots, x_n]$
- Para implementarlas usaremos listas simple (o doblemente) enlazadas
- Usaremos la posición de nodo (en lugar de un índice como en arreglos) para referirnos a la “ubicación” de un elemento en la lista

## Posiciones

- `element()`: Retorna el elemento ubicado en esta posición

```
public interface Position<E>
{
    // Retorna el valor del elemento ubicado en la posición
    public E element();
}
```

En general  $p_i$  es la posición de  $x_i$  para  $i=1, \dots, n$  con  $L=[x_1, x_2, \dots, x_{n-1}, x_n]$ . Esto se llama *posición directa*.



Estructuras de datos - Dr. Sergio A. Gómez

3

## ADT Position List

### (operaciones de iteración/recorrido)

- `first()`: retorna la posición del primer elemento de la lista; error si la lista está vacía
- `last()`: retorna la posición del último elemento de la lista; error si la lista está vacía
- `prev(p)`: retorna la posición del elemento que precede al elemento en la posición  $p$ ; error si  $p = \text{first}()$
- `next(p)`: retorna la posición del elemento que sigue al elemento en la posición  $p$ ; error si  $p = \text{last}()$

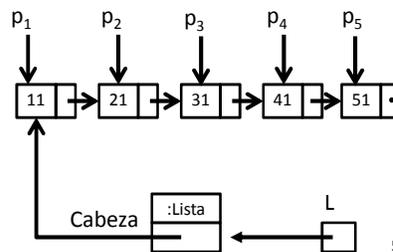
Estructuras de datos - Dr. Sergio A. Gómez

4

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## ADT Position List

- `first()`: retorna la posición del primer elemento de la lista; error si la lista está vacía
- Ejemplo:  
L.first() retorna  $p_1$

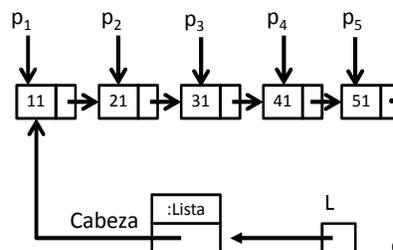


Estructuras de datos - Dr. Sergio A. Gómez

5

## ADT Position List

- `last()`: retorna la posición del último elemento de la lista; error si la lista está vacía
- Ejemplo:  
L.last() retorna  $p_5$



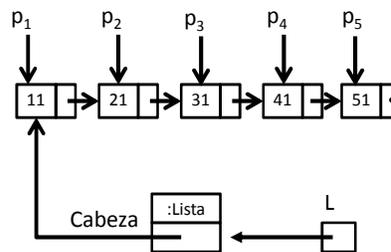
Estructuras de datos - Dr. Sergio A. Gómez

6

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## ADT Position List

- $\text{prev}(p)$ : retorna la posición del elemento que precede al elemento en la posición  $p$ ; error si  $p = \text{first}()$
- Ejemplo:
  - L. $\text{prev}(p_3)$  retorna  $p_2$
  - L. $\text{prev}(p_1)$  produce un error

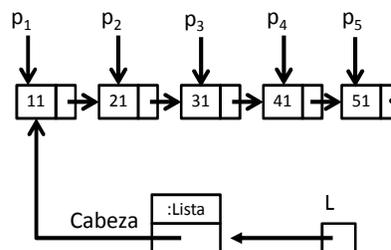


Estructuras de datos - Dr. Sergio A. Gómez

7

## ADT Position List

- $\text{next}(p)$ : retorna la posición del elemento que sigue al elemento en la posición  $p$ ; error si  $p = \text{last}()$
- Ejemplo:
  - L. $\text{next}(p_3)$  retorna  $p_4$
  - L. $\text{next}(p_5)$  produce un error



Estructuras de datos - Dr. Sergio A. Gómez

8

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## ADT Position List (métodos de actualización)

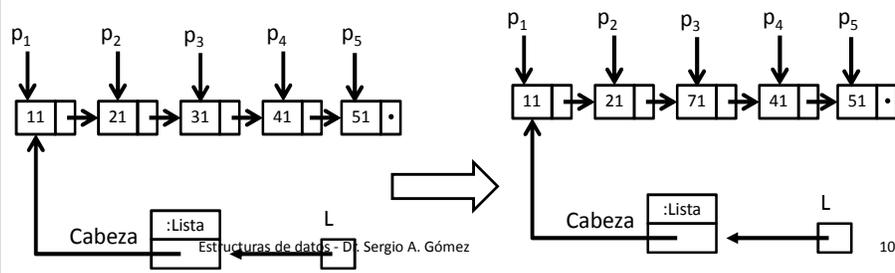
- `set(p, e)`: Reemplaza al elemento en la posición  $p$  con  $e$ , retornando el elemento que estaba antes en la posición  $p$
- `addFirst(e)`: Inserta un nuevo elemento  $e$  como primer elemento
- `addLast(e)`: Inserta un nuevo elemento  $e$  como último elemento
- `addBefore(p, e)`: Inserta un nuevo elemento  $e$  antes de la posición  $p$
- `addAfter(p, e)`: Inserta un nuevo elemento  $e$  luego de la posición  $p$
- `remove(p)`: Elimina y retorna el elemento en la posición  $p$  invalidando la posición  $p$ .

Estructuras de datos - Dr. Sergio A. Gómez

9

## ADT Position List

- `set(p, e)`: Reemplaza al elemento en la posición  $p$  con  $e$ , retornando el elemento que estaba antes en la posición  $p$
- Ejemplo:  
 $L.set(p_3, 7)$  retorna 3 y produce el siguiente efecto:

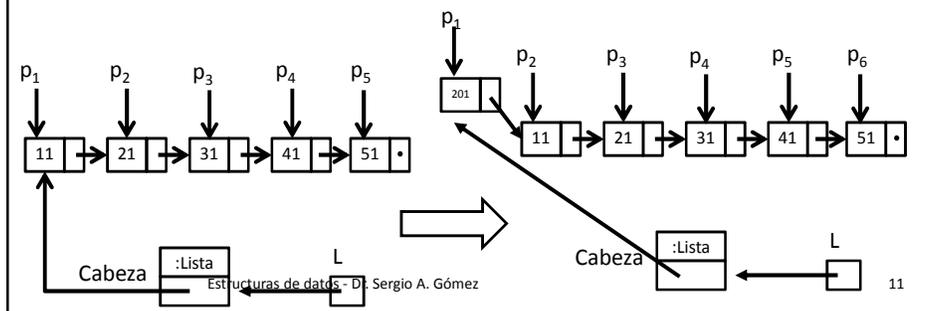


10

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

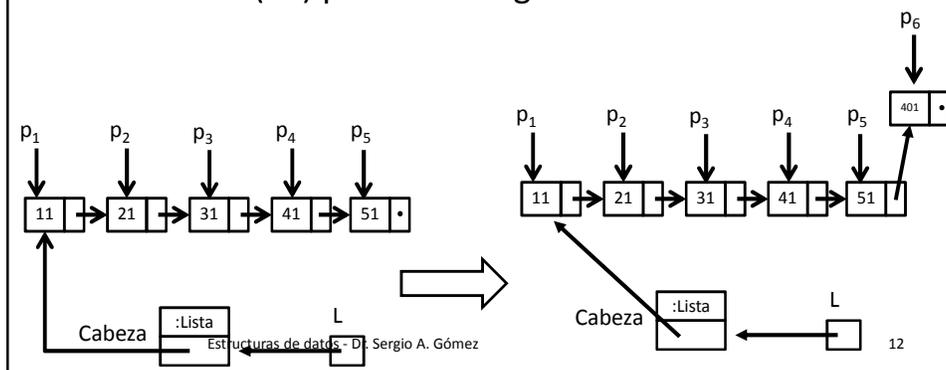
## ADT Position List

- `addFirst(e)`: Inserta un nuevo elemento `e` como primer elemento
- Ejemplo:  
`L.addFirst(20)` produce el siguiente efecto:



## ADT Position List

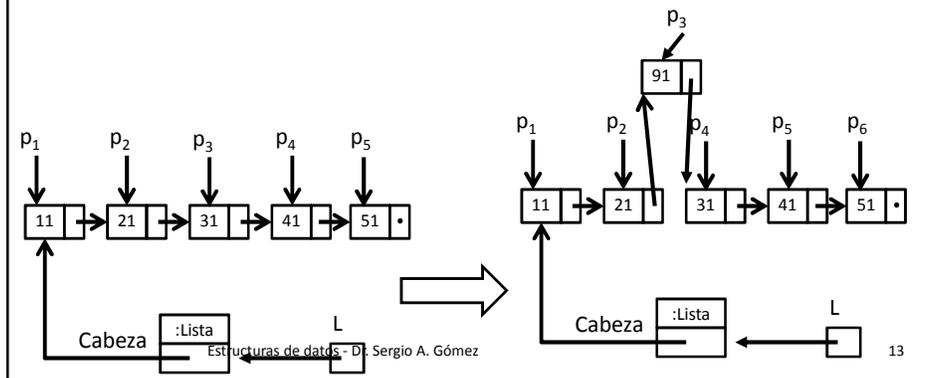
- `addLast(e)`: Inserta un nuevo elemento `e` como último elemento
- Ejemplo:  
`L.addLast(40)` produce el siguiente efecto:



El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

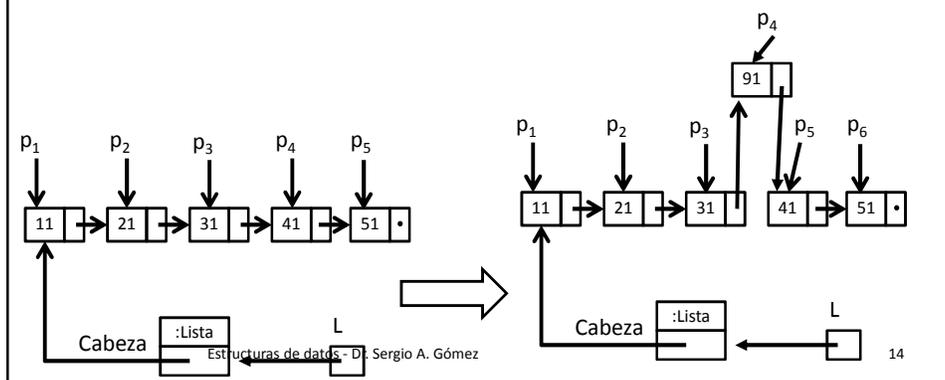
## ADT Position List

- `addBefore(p, e)`: Inserta un nuevo elemento e antes de la posición p
- Ejemplo:  
L.addBefore( $p_3$ , 9) produce el siguiente efecto:



## ADT Position List

- `addAfter(p, e)`: Inserta un nuevo elemento e luego de la posición p
- Ejemplo:  
L.addAfter( $p_3$ , 12) produce el siguiente efecto:



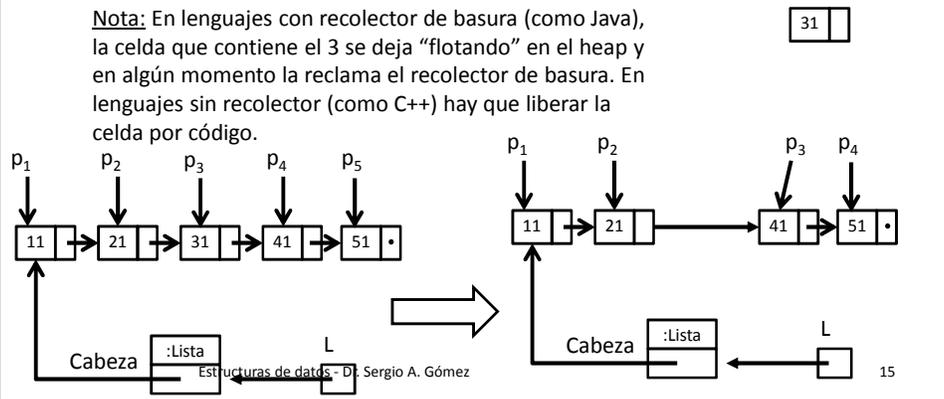
El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
"Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## ADT Position List

- `remove(p)`: Elimina y retorna el elemento en la posición  $p$  invalidando la posición  $p$ .
- Ejemplo:

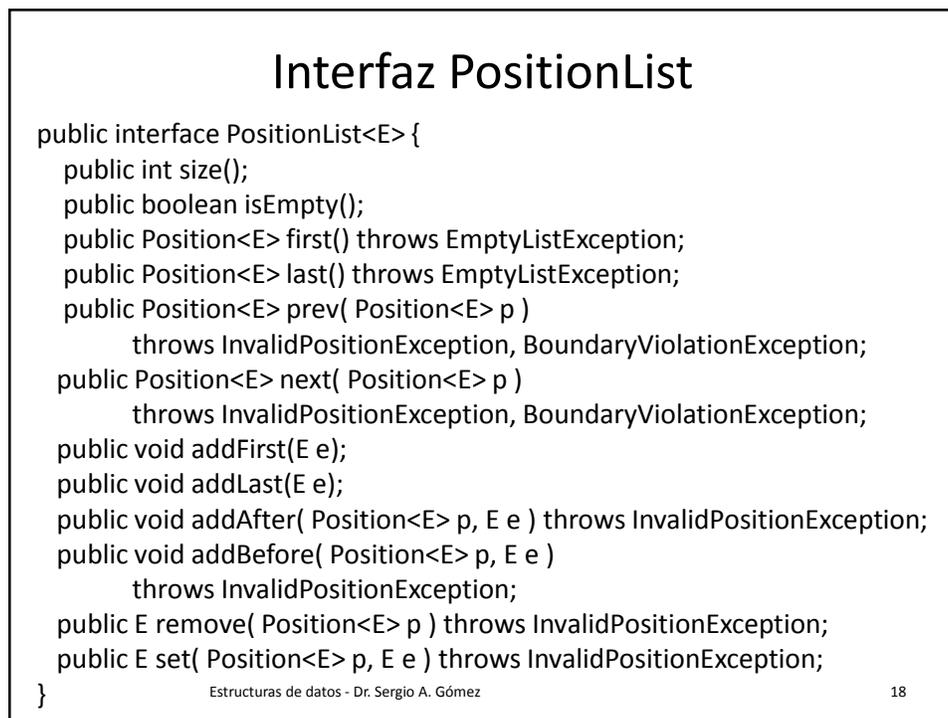
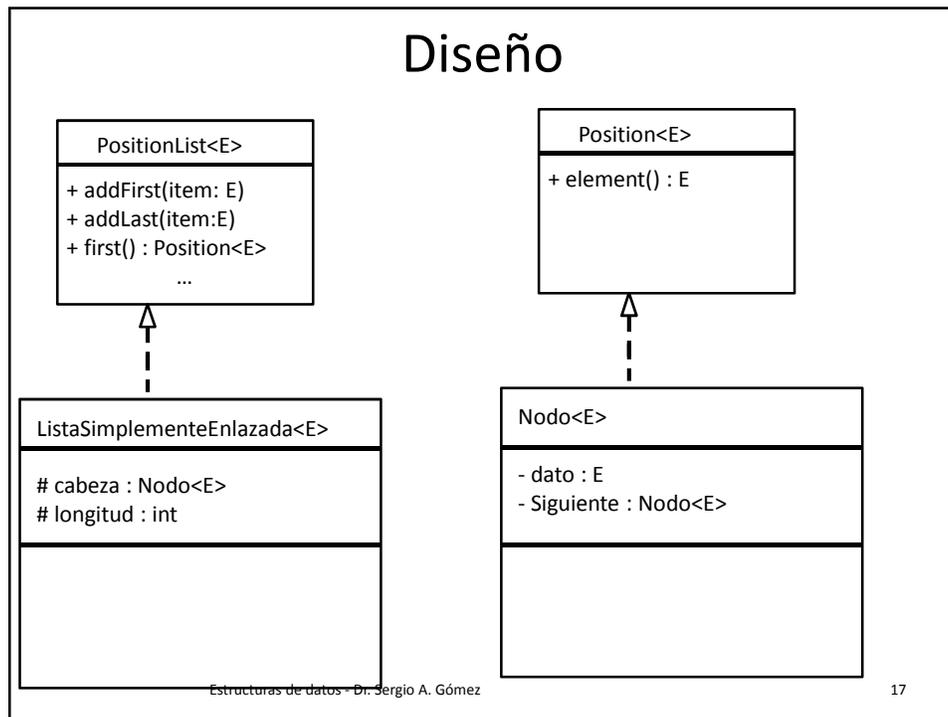
`L.remove(p3)` produce el siguiente efecto:

Nota: En lenguajes con recolector de basura (como Java), la celda que contiene el 3 se deja "flotando" en el heap y en algún momento la reclama el recolector de basura. En lenguajes sin recolector (como C++) hay que liberar la celda por código.



## Situaciones de error relacionadas a una posición $p$

1.  $p = \text{null}$
  2.  $p$  fue eliminada previamente de la lista
  3.  $p$  es una posición de una lista diferente
  4.  $p$  es la primera posición de la lista e invocamos a `prev(p)`
  5.  $p$  es la última posición de la lista e invocamos a `next(p)`
- Por cuestiones de eficiencia, al implementar el TDA PositionList no validaremos ni (2) ni (3) perdiendo un poco de robustez.



El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Ejemplo: Cargar una lista de enteros con [1,2,3,4] y luego imprimirla

```
PositionList<Integer> l = new ListaSimplementeEnlazada<Integer>();
l.addLast( 1 );
l.addLast( 2 );
l.addLast( 3 );
l.addLast( 4 );

try {
    Position<Integer> p = l.first(), ultima = l.last();
    while( p != null ) {
        System.out.println( "Elemento: " + p.element() );
        if( p != ultima ) p = l.next(p) else p = null;
    }
} catch( InvalidPositionException | BoundaryViolationException | EmptyListException e ) {
    e.printStackTrace();
}
// Observación: Capturar varias excepciones en un catch requiere nivel de compilación Java 1.8.
```

Estructuras de datos - Dr. Sergio A. Gómez

19

## Ejemplo: Buscar un elemento elem en una lista l

```
public <E> static boolean pertenece( PositionList<E> l, E elem ) {
    if( l.isEmpty() )
        return false;
    else {
        try {
            Position<E> p = l.first(), ultima = l.last();
            boolean encuentre = false;
            while( p != null && !encuentre )
                if( p.element().equals( elem ) )
                    encuentre = true;
            else
                p = ( p != ultima ) ? l.next(p) : null;
            return encuentre;
        } catch( InvalidPositionException e ) {
            System.out.println( "e: " + e.getMessage() ); return false;
        } catch( BoundaryViolationException e ) {
            System.out.println( "e: " + e.getMessage() ); return false;
        } catch( EmptyListException e ) {
            System.out.println( "e: " + e.getMessage() ); return false;
        }
    }
}
```

Nota: El operador ternario A?B:C evalúa la expresión booleana A y cuando A evalúa a true retorna B y cuando A evalúa a false retorna C. "?" tiene más precedencia que la asignación "=".

Nota:  $T_{pertenece}(n) = O(n)$

Estructuras de datos - Dr. Sergio A. Gómez

20

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Comparación de elementos

**NOTA:** Si *a* es una variable de un tipo que implementa la interfaz *Comparable*:

$$a.compareTo(b) = \begin{cases} 0 & , \text{ si } a = b \\ \text{entero negativo,} & \text{ si } a < b \\ \text{entero positivo,} & \text{ si } a > b \end{cases}$$

```
String s1 = "Sergio";      String s2 = "Martin";
String s3 = "Matias";     String s4 = "Sergio";
int valor1 = s1.compareTo(s2);
// si s1 > s2, retorna positivo; si s1 < s2, retorna negativo; si s1 = s2, retorna 0
System.out.println("valor1: " + valor1); // Imprime 6 porque "Sergio" > "Martin"

int valor2 = s2.compareTo(s3);
System.out.println("valor2: " + valor2); // Imprime -2 porque "Martin" < "Matias"

int valor3 = s1.compareTo(s4);
System.out.println("valor3: " + valor3); // Imprime 0 porque "Sergio" = "Sergio"
```

21

## Comparator

Un comparador es un objeto que abstrae la noción de comparación entre dos objetos de tipo genérico *E*.

**NOTA:** Si *c* es una variable de tipo *Comparator<E>*:

$$c.compare(a, b) = \begin{cases} 0 & , \text{ si } a = b \\ \text{entero negativo,} & \text{ si } a < b \\ \text{entero positivo,} & \text{ si } a > b \end{cases}$$

El comparador por defecto delega en el comportamiento del tipo básico *E* implementado por la función *compareTo* del tipo *E*:

```
public class DefaultComparator<E> implements java.util.Comparator<E> {
    public int compare( E a, E b ) throws ClassCastException {
        return ((Comparable<E>)a).compareTo(b);
    }
}
```

Estructuras de datos - Dr. Sergio A. Gómez

22

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Ejemplo de uso del comparador

```
// Compilar con javac Principal.java -Xlint:unchecked

public class Principal {
    public static void main( String [] args ) {

        DefaultComparator<Integer> comp = new DefaultComparator<Integer>();

        System.out.println( comp.compare( 1, 1 ) ); // Imprime 0
        System.out.println( comp.compare( 1, 2 ) ); // Imprime -1
        System.out.println( comp.compare( 2, 1 ) ); // Imprime 1

        DefaultComparator<String> comp2 = new DefaultComparator<String>();
        System.out.println( comp2.compare( "abba", "casa" ) ); // imprime -2
        System.out.println( comp2.compare( "abba", "abba" ) ); // imprime 0
        System.out.println( comp2.compare( "casa", "abba" ) ); // imprime 2
    }
}
```

Estructuras de datos - Dr. Sergio A. Gómez

23

**Problema:** Dada una lista l, insertar item en forma ordenada ascendente.

```
public static <E> void insertarOrdenado( PositionList<E> l, E item ) {
    try {
        if( l.isEmpty() ) l.addFirst( item );
        else {
            DefaultComparator<E> comp = new DefaultComparator<E>();
            Position<E> p = l.first(), ultima = l.last();
            boolean encuentrePosicion = false;
            while( p != null && !encuentrePosicion ) {
                int c = comp.compare( item, p.element() );
                if( c >= 0 ) p = p != ultima ? l.next(p) : null;
                else encuentrePosicion = true;
            }
            if( encuentrePosicion ) l.addBefore( p, item );
            else l.addLast( item );
        }
    } catch( BoundaryViolationException e ) {
        System.out.println( "e: " + e.getMessage() );
    } catch( InvalidPositionException e ) {
        System.out.println( "e: " + e.getMessage() );
    }
}
```

T(n) = O(n)

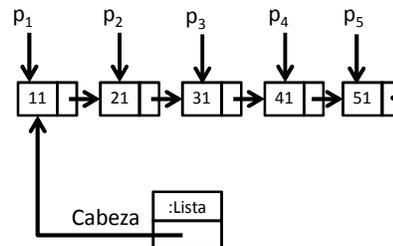
Estructuras de datos - Dr. Sergio A. Gómez

24

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Implementación con listas simplemente enlazadas (y posición directa)

- Lista simplemente enlazada: Una lista formada por nodos, donde cada nodo conoce un dato y la referencia al siguiente nodo
- La lista conoce la cabeza de la lista
- Posición directa: La posición “p” de un nodo “n” es la referencia al nodo n.



Estructuras de datos - Dr. Sergio A. Gómez

25

## Implementación con listas simplemente enlazadas y posición directa

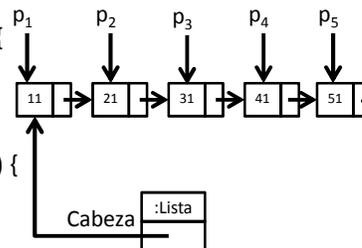
```

public class Nodo<E> implements Position<E> {
    private E elemento;
    private Nodo<E> siguiente;

    public Nodo( E elemento, Nodo<E> siguiente ) {
        this.elemento = elemento;
        this.siguiente = siguiente;    }
    public Nodo(E elemento) {    this( elemento, null );    }

    public E getElemento() { return elemento; }
    public Nodo<E> getSiguiente() { return siguiente; }
    public void setElemento( E elemento ) { this.elemento = elemento; }
    public void setSiguiente(Nodo<E> siguiente) { this.siguiente = siguiente; }
    public E element() { return elemento; }
}

```



Estructuras de datos - Dr. Sergio A. Gómez

26

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

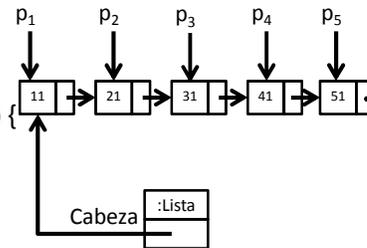
## Implementación con listas simplemente enlazadas y posición directa

```
public class ListaSimplementeEnlazada<E> implements PositionList<E>
{
    protected Nodo<E> cabeza;
    protected int longitud;

    public ListaSimplementeEnlazada() {
        cabeza = null;
        longitud = 0;
    }
    public int size() { return longitud; }
    public boolean isEmpty() { return cabeza == null; }

    // ... Operaciones de actualización y búsqueda aquí ...
}

```



Estructuras de datos - Dr. Sergio A. Gómez

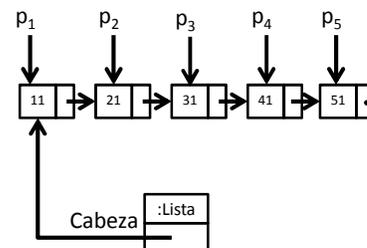
27

## Implementación con listas simplemente enlazadas y posición directa

```
public void addFirst(E e) {
    cabeza = new Nodo<E>( e, cabeza );
    longitud++;
}

public void addLast( E e ) {
    if( isEmpty() ) addFirst(e);
    else {
        Nodo<E> p = cabeza;
        while( p.getSiguiente() != null )
            p = p.getSiguiente();
        p.setSiguiente( new Nodo<E>(e) );
        longitud++;
    }
}

```



Estructuras de datos - Dr. Sergio A. Gómez

28

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

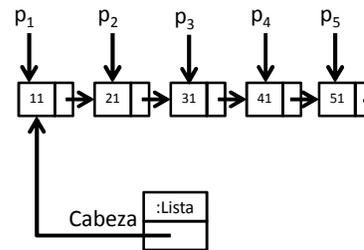
```

public Position<E> first() throws EmptyListException {
    if( cabeza == null )
        throw new EmptyListException( "Lista vacía" );
    return cabeza;
}

public Position<E> last() throws EmptyListException {
    if( isEmpty() )
        throw new EmptyListException( "last(): Quiere ejecutar last" +
            "con una lista vacía" );

    Nodo<E> p = cabeza;
    while( p.getSiguiente() != null )
        p = p.getSiguiente();
    return p;
}

```



Estructuras de datos - Dr. Sergio A. Gómez

29

```

public Position<E> next( Position<E> p )
    throws InvalidPositionException, BoundaryViolationException {
    Nodo<E> n = checkPosition(p);
    if( n.getSiguiente() == null )
        throw new BoundaryViolationException("Next: Siguiente de último");
    return n.getSiguiente();
}

private Nodo<E> checkPosition( Position<E> p )
    throws InvalidPositionException {
    try {
        if( p == null ) throw new InvalidPositionException("Posición nula");
        return (Nodo<E>) p;
    } catch( ClassCastException e ) {
        throw new InvalidPositionException( "..." );
    }
}

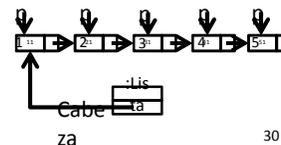
```

¿Tiempo de ejecución?

¿Qué ocurre si p es una posición de "otra" lista?

¿Qué ocurre si p es una posición de otro ADT?

¿Cómo es la implementación de "prev"?



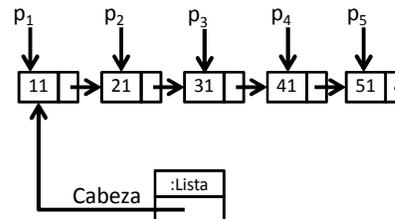
Estructuras de datos - Dr. Sergio A. Gómez

30

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Implementación con listas simplemente enlazadas y posición directa

```
public void addAfter( Position<E> p , E e )
    throws InvalidPositionException {
    Nodo<E> n = checkPosition(p);
    Nodo<E> nuevo = new Nodo<E>(e);
    nuevo.setSiguiente( n.getSiguiente() );
    n.setSiguiente( nuevo );
    longitud++;
}
```



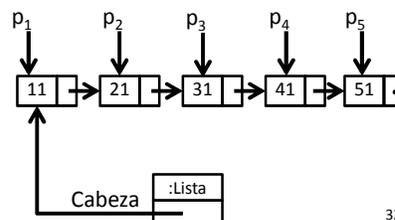
¿Tiempo de ejecución?  
 ¿Qué ocurre si p es una posición de otra lista?  
 ¿Qué ocurre con el tiempo de ejecución si testeó si p es una posición válida de la lista receptora?  
 ¿Cómo se implementa addBefore?  
 ¿Tiempo de ejecución de addBefore con esta implementación?

Estructuras de datos - Dr. Sergio A. Gómez

31

## Implementación con lista simplemente enlazada y posición directa

```
public void addBefore(Position<E> p, E e ) throws
InvalidPositionException {
    Nodo<E> n = checkPosition(p);
    if( p == first() ) addFirst(e);
    else {
        Nodo<E> anterior = (Nodo <E>) prev(p);
        Nodo<E> nuevo = new Nodo<E>( e, n );
        anterior.setSiguiente( nuevo );
        longitud++;
    }
}
```



Estructuras de datos - Dr. Sergio A. Gómez

32

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Implementación con lista simplemente enlazada y posición directa

```
public Position<E> prev( Position<E> p )
    throws InvalidPositionException, BoundaryViolationException
{
    if( p == first() )
        throw new BoundaryViolationException( "Lista::prev(): " +
            "Posición primera" );
    Nodo<E> n = checkPosition(p);
    Nodo<E> aux = cabeza;
    while( aux.getSiguiente() != n && aux.getSiguiente() != null )
        aux = aux.getSiguiente();
    if( aux.getSiguiente() == null )
        throw new InvalidPositionException( "Lista::prev(): " +
            "Posicion no pertenece a la lista" );
    return aux;
}
```

Estructuras de datos - Dr. Sergio A. Gómez

33

```
public E remove( Position<E> p ) throws InvalidPositionException {
    if( isEmpty() ) throw new InvalidPositionException( "..." );
    Nodo<E> n = checkPosition(p);
    E aux = p.element();
    if( n == cabeza )
        cabeza = cabeza.getSiguiente();
    else {
        Nodo<E> anterior = (Nodo<E>) prev(p);
        anterior.setSiguiente( n.getSiguiente() );
    }
    longitud--;
    return aux;
}
```

¿Tiempo de ejecución?

Estructuras de datos - Dr. Sergio A. Gómez

34

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Próxima clase

- Implementación con lista simplemente enlazada con enlace al primer y último elemento
- Implementación con Lista doblemente enlazada con celdas centinelas al principio y al final
- Iteradores

## Bibliografía

- Goodrich & Tamassia, capítulo 6.