Soluciones Ejercicios Tema 4

Germán Moltó Martínez
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Implementación de Métodos de LEG<E> (I)

```
public int indiceDe(E e){
   int res;
   NodoLEG<E> aux;
   for ( aux = primero, res = 0;
            aux != null &&  !aux.dato.equals(e);
                  aux = aux.siguiente, res++ ){;}
   if ( aux == null ) res = -1;
   return res;
}
```


Implementación de Métodos de LEG<E> (II)

```
public void insertar(E x, int i){
      NodoLEG<E> nuevo = new NodoLEG<E>(x); talla++;
      NodoLEG<E> aux   = primero, ant = null; int j = 0;
      while ( j < i ){ ant = aux; aux = aux.siguiente; j++; }
      if ( aux != null ) ant.siguiente = nuevo;
      else // no está x o x es el primero, por lo que aux == null
      // si es el primero, o la Lista está vacía
      if ( ant == null ) primero = nuevo;
      // sino inserto detrás del último
      else ant.siguiente = nuevo;
     nuevo.siguiente = aux;
   }
```


Implementación de Métodos de LEG<E> (III)

```
public E buscar(E x) throws ElementoNoEncontrado {
   NodoLEG<E> aux = primero;
   while ( aux != null && !aux.dato.equals(x) ){
         aux = aux.siguiente;
   }
   if (aux == null)
        throw new ElementoNoEncontrado(x + "No esta ");
   return aux.dato;
}
```

Solución: LEGOrdenada (I)

- No es **necesario** que LEGOrdenada sobrescriba el método borrar de LEG, ya que borrar de una lista ordenada NO afecta a la propiedad de ordenación de la lista.
- Sin embargo, SÍ es **conveniente** que LEGOrdenada sobrescriba el método borrar de LEG por cuestiones de eficiencia:
  - No tiene sentido seguir buscando el elemento a borrar si encontramos un elemento mayor que él en la lista ordenada.
- Ídem con el método buscar.

Solución: LEGOrdenada (III)

```
public class LEGOrdenada<E extends Comparable<E>>
   extends LEG<E>{
   public LEGOrdenada(){…}
   public void insertar(E e){…}
   public E buscar(E aBuscar) throws ElementoNoEncontrado{…}
   public E borrar(E aBorrar) throws ElementoNoEncontrado{…}
}
```


Solución: LEGOrdenada (III)

```
public E borrar(E x) throws ElementoNoEncontrado {
   NodoLEG<E>  aux = primero, ant = null;
   while ( aux != null &&  aux.dato.compareTo( x ) < 0) {
         ant = aux; aux = aux.siguiente;
   }
   if ( aux == null ||  aux.dato.compareTo(x) > 0)
      throw new ElementoNoEncontrado(x+" no está en la lista");
   if ( ant == null )  primero = aux.siguiente;
   else ant.siguiente = aux.siguiente;
   talla--;
   return aux.dato;
}
```


Herencia en LEGConUltimo

```
public class LEGConUltimo<E> extends LEG<E>{
   protected NodoLEG<E> ultimo;

   public LEGConUltimo(){
         super();  //No hace falta ponerla puesto que se hace automáticamente.
         ultimo = null;
   }
   public void insertar(E x) { ...}
   public void insertarEnFin(E x) { ...}
   public E borrar(E x) throws ElementoNoEncontrado {...}
}
```

- Heredamos la definición tanto de ***primero*** como de ***talla*** y que NO es necesario sobrescribir el método ***talla*** ni el método ***toString,*** ni el método ***buscar,*** puesto que sirve la implementación heredada.

Solución: LEGCircular

```
public boolean eliminar(E x) {
   if (ultimo == null) return false;
   NodoLEG<E> aux = ultimo.siguiente;
   NodoLEG<E> ant = ultimo;
   while ( (aux != ultimo) && (!aux.dato.equals(x))){
           ant = aux;
           aux = aux.siguiente;
   }
   if (!aux.dato.equals(x)) return false;
   if ( (aux == ultimo) && (ant == ultimo)) ultimo = null;
   else {
      ant.siguiente = aux.siguiente;
      if (aux == ultimo)  ultimo = ant;
   }
   talla--;
    return true;
} 9
```

Solución: LDEG toString

- Utilizamos dos bucles:
  - Uno para llegar al último elemento de la lista. Otro para recorrer la lista en sentido descendente hasta el primer elemento.
  - El coste computacional es lineal con el número de elementos.

```
public String toString{
   String res = “”; NodoLDEG<E> aux = null;
   if (primero != null) {
   for (aux = primero ; aux.siguiente!=null ; aux = aux.siguiente);
      // aux es una referencia al último nodo de la Lista Enlazada
      while (aux != null){
           res += aux.dato.toString();
           aux = aux.anterior;
      }//while
   }//if
   return res; }
```

10

Solución: LDEG insertarEnFin

```
public void insertarEnFin(E x){
   NodoLDEG<E> nuevo, aux;
   nuevo = new NodoLDEG<E>(x);
   aux = primero;
   if (aux == null) primero = nuevo;
   else {
          while( aux.siguiente != null) aux = aux.siguiente;
          nuevo.anterior = aux;
          aux.siguiente = nuevo;
   }
}
```

11

eliminarMayor de LEG

```
public class LEGDeComparables<E extends Comparable<E>> extends
   LEG<E> {
 public void eliminarMayor(E x){
   NodoLEG<E> aux = primero, ant = null;
   while (aux != null){
      int resC = aux.dato.compareTo(x);
      if (resC > 0){
          if (ant != null) ant.siguiente = aux.siguiente;
          else primero = aux.siguiente;
       } else ant = aux;
      aux = aux.siguiente;
   }}}
```

- El puntero ant únicamente se incrementa cuando detectamos que el elemento de aux **NO** hay que eliminarlo.

12

eliminarMayor de LEGOrdenada

```
public void eliminarMayor(E x){
   NodoLEG<E> aux = primero, ant = null;
   while (aux != null){
         int resC = aux.dato.compareTo(x);
         if (resC > 0){
                  if (ant != null) ant.siguiente = null;
                  else primero = null;
         } else {
                  ant = aux;
                  aux = aux.siguiente;
         }
   }}
```


Borra última aparición en LEG

```
public E borraUltimaAparicion(E x) throws ElementoNoEncontrado {
  NodoLEG<E> antUltEnc = null,   ant = null,  aux = primero, ultEnc=null;
   while (aux != null) {
      if (aux.dato.equals(x)) {
         antUltEnc = ant; ultEnc = aux;
      }
      ant = aux;  aux = aux.siguiente;
   }
  if (ultEnc == null)   throw new ElementoNoEncontrado(x + " no está");
  if (antUltEnc == null) primero = primero.siguiente;
  else antUltEnc.siguiente = ultEnc.siguiente;
  talla--;
  return ultEnc.dato; }
```


Solución Eliminar i-ésimo

```
public boolean eliminar(int i) {
    if ( (talla()==0) || (i<0) || (i>=talla()) ) return false;
   NodoLEG<E> aux = primero, ant = null;
   for (int j = 0; j < i ; j++) {
        ant = aux; aux = aux.siguiente;
    }
    if ( ant == null ) primero = aux.siguiente;
   else  ant.siguiente= aux.siguiente;
   talla--;
   return true;
}
```


Solución toStringOAMayoresQue

```
public String toStringOAMayoresQue(E e) throws
   ElementoNoEncontrado{
      String res =""; NodoLEG<E> aux = primero;
      while ( aux != null  &&  !(aux.dato.compareTo(e) > 0) )
         aux = aux.siguiente;
      if ( aux == null ) throw new ElementoNoEncontrado("No hay
   MAYORES QUE "+e);
      while ( aux != null ){
            res += aux.dato.toString()+"\n";
            aux = aux.siguiente;
      }
      return res;
   }
```

Solución esMediana en LEGOrdenada

```
public boolean esMediana(E x) {
boolean res = false;
int menores = 0, mayores = 0;
NodoLEG<E> aux = primero;
while ( aux != null  && aux.dato.compareTo(x) < 0 ) {
    menores++; aux = aux.siguiente; }
if ( aux != null ){
        mayores = this.talla – menores;
        if (aux.dato.compareTo(x) == 0)  mayores--;
         res = ( mayores == menores )
}
return res;
}
```