Tema 4- Representación Enlazada

Germán Moltó
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Tema 4- Representación Enlazada

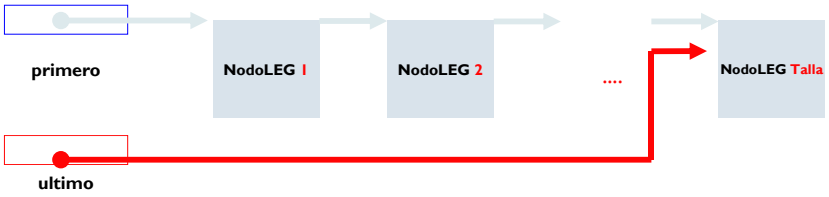
Índice general:

Objetivos

- Introducir otras implementaciones de Lista Enlazada Genérica (LEG)
- Estas nuevas implementaciones permiten realizar ciertas operaciones de manera más eficiente que con las LEG básicas.
- Otras Implementaciones de LEG
  - LEG ordenada
  - LEG con referencias al primer y último nodo.
  - LEG circular
  - Lista doblemente enlazada

LEG con Referencias al Primer y Último Nodo.

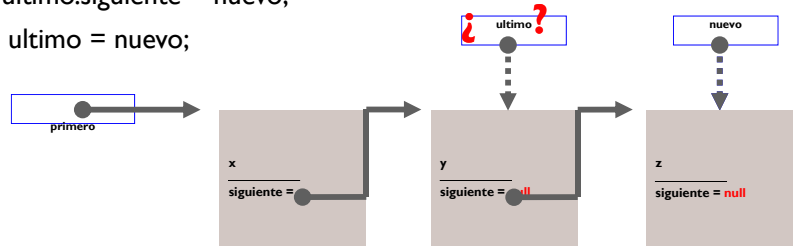
- Insertar en fin de LEG tiene un coste **lineal** con el número de nodos de la lista.
- Se puede reducir a constante (es decir, independiente del tamaño del problema).
  - Para ello, añadimos a la LEG una nueva referencia al último nodo



Inserción de Nodos al Final de una LEG con Referencias al Primer y Último Nodo

1. Crear el nodo a insertar
2. Insertar dicho nodo al final de la LEG actual

```
ultimo.siguiente = nuevo;
ultimo = nuevo;
```



La Clase LEGConUltimo (I)

```
package lineales;
import excepciones.*;
public class LEGConUltimo<E>{
   protected NodoLEG<E> primero, ultimo;
   protected int talla;
    public LEGConUltimo() {
                    primero = null; ultimo = null;  talla = 0;
     }
   public int talla() { ...}
   public void insertar(E x) { ...}
   public void insertarEnFin(E x) { ...}
   public boolean eliminar(E x) { ...}
   public String toString() { ...}
}
```


La Clase LEGConUltimo (II)

```
public void insertar(E x){
   primero = new NodoLEG<E>(x, primero);
   if ( ultimo == null ) ultimo = primero;
   talla++;
}
public void insertarEnFin(E x){
   NodoLEG<E> nuevo = new NodoLEG<E>(x);
   if (ultimo != null) ultimo.siguiente = nuevo;
   else primero = nuevo;
   ultimo = nuevo;
   talla++;
}
```


La Clase LEGConUltimo (III)

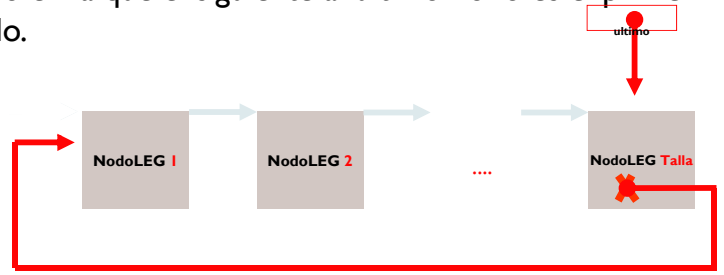
```
public boolean eliminar(E x) {
  NodoLEG<E>  aux = primero, ant = null;
  while ( aux != null && !aux.dato.equals( x ) ) {
      ant = aux;  aux = aux.siguiente;
   }
  if (aux==null) return false;
  if ( ant == null )  primero=aux.siguiente;
  else ant.siguiente = aux.siguiente;
  if ( aux.siguiente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

Herencia en LEGConUltimo

- ¿Cómo podríamos aplicar el mecanismo de la herencia en la definición de la clase LEGConUltimo?
- Para ello, se debe recordar la definición de la clase LEG.

Lista Enlazada Circular

- Una LEG Circular es una LEG con referencia al último nodo en la que el siguiente al último nodo es el primer nodo.



- El último nodo siempre referencia al primero.
  - En una Lista Enlazada Circular siempre existe el nodo anterior a uno dado.

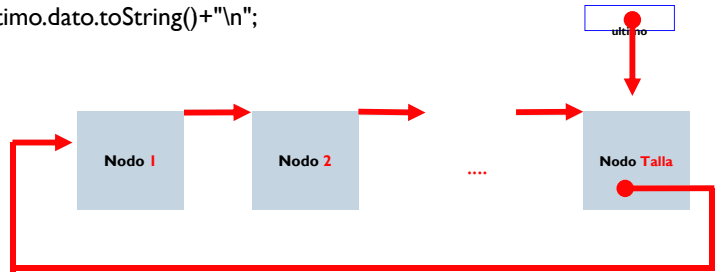
La Clase LEGCircular (I)

```
package librerias.estructurasDeDatos.lineales;
import excepciones.*;
public class LEGCircular<E>{
   protected NodoLEG<E> ultimo;
   protected int talla;
   public LEGCircular() { ... }
   public int talla() { ...}
   public void insertar(E x) { ...}
   public void insertarEnFin(E x) { ...}
   public boolean eliminar(E x) { ...}
   public String toString() { ...}
}
```


La Clase LEGCircular (II)

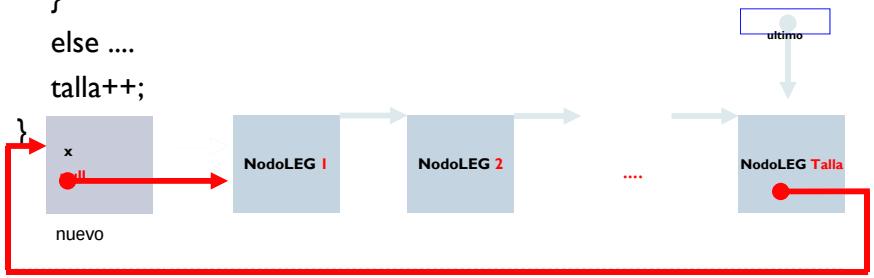
Escribir el código del método

```
public String toString(){
   String res = "";
   if ( ultimo != null ) {
    for (NodoLEG<E> aux= ultimo.siguiente ; aux != ultimo; aux =aux.siguiente)
          res += aux.dato.toString()+"\n";
     res += ultimo.dato.toString()+"\n";
   }
   return res;
}
```



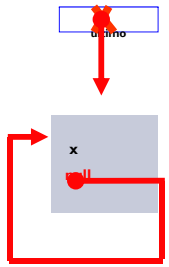
La Clase LEGCircular (III)

```
public void insertar(E x){
   NodoLEG<E> nuevo = new NodoLEG<E>(x);
    if ( ultimo != null ){
          nuevo.siguiente = ultimo.siguiente;
          ultimo.siguiente = nuevo;
    }
    else ....
    talla++;
}
```



La Clase LEGCircular (IV)

```
public void insertar(E x){
   NodoLEG<E> nuevo = new NodoLEG<E>(x);
    if ( ultimo != null ){
       nuevo.siguiente = ultimo.siguiente;
        ultimo.siguiente = nuevo;
    }
    else {
       ultimo = nuevo;
       ultimo.siguiente = nuevo;
    }
    talla++; }
```



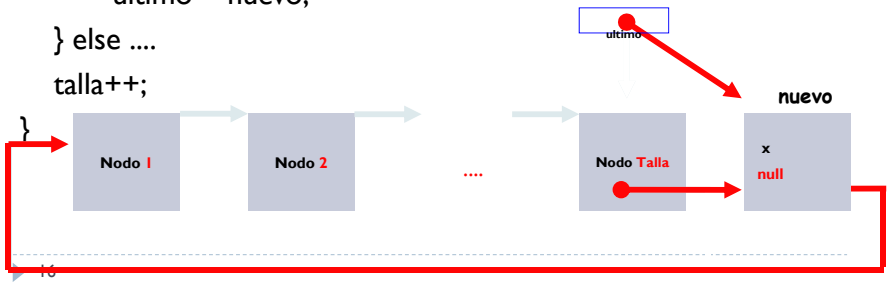
La Clase LEGCircular (V)

- Compactando el código del método insertar:

```
public void insertar(E x){
    NodoLEG<E> nuevo = new NodoLEG<E>(x);
    if ( ultimo == null ) ultimo = nuevo;
    else nuevo.siguiente = ultimo.siguiente;
    ultimo.siguiente = nuevo;
    talla++;
}
```

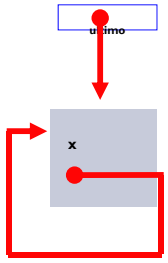

La Clase LEGCircular (VI)

```
public void insertarEnFin(E x){
   NodoLEG<E> nuevo = new NodoLEG<E>(x);
    if ( ultimo != null ){
          nuevo.siguiente = ultimo.siguiente;
          ultimo.siguiente = nuevo;
          ultimo = nuevo;
    } else ....
    talla++;
}
```



La Clase LEGCircular (VII)

```
public void insertarEnFin(E x){
    NodoLEG<E> nuevo = new NodoLEG<E>(x);
    if ( ultimo != null ){
        nuevo.siguiente = ultimo.siguiente;
        ultimo.siguiente = nuevo;
        ultimo = nuevo;
    } else  {
          ultimo = nuevo;
          ultimo.siguiente = nuevo;
    }
     talla++;}
```



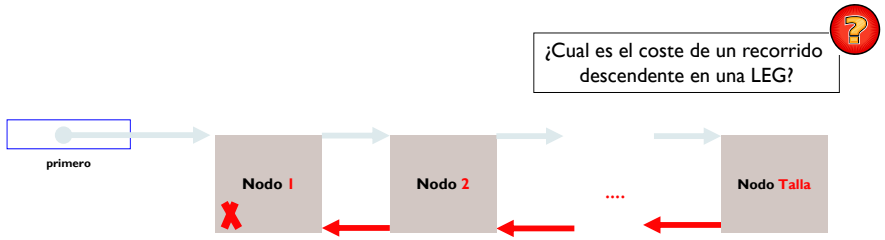
LEGCircular: Ejercicio Propuesto

- Diseñar e Implementar el método de borrado de una LEGCircular.
  - Plantear análisis de situaciones y soluciones.
  - Escribir transcripción algorítmica.

public boolean **eliminar**(E x)

Lista Genérica Doblemente Enlazada

- Una Lista Doblemente Enlazada es una LEG en la que cada nodo tiene información sobre su nodo siguiente y su nodo anterior
  - Permite el acceso en tiempo constante a los nodos anterior y siguiente a uno dado
  - Permite el recorrido y búsqueda ascendente o descendente



La clase NodoLDEG

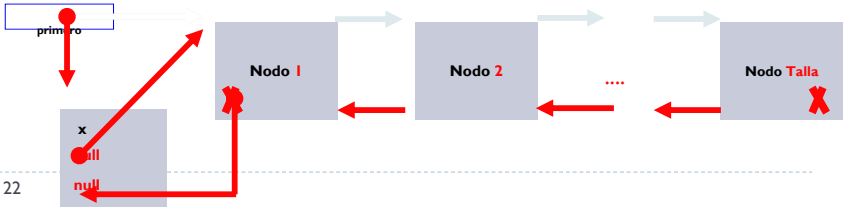
```
package librerias.estructurasDeDatos.lineales;
class NodoLDEG<E> {
   E dato;
   NodoLDEG<E> siguiente,  anterior;
   NodoLDEG(E dato) { this(dato, null , null);  }
   NodoLDEG(E dato,  NodoLDEG<E> s,  NodoLDEG<E> a) {
      this.dato = dato;
      this.siguiente = s;
      this.anterior = a;
   }
}
```

La Clase LDEG (I)

```
package librerias.estructurasDeDatos.lineales;
import excepciones.*;
public class LDEG<E>{
   protected NodoLDEG<E> primero;
   protected int talla;
   public LDEG() { ... }
   public int talla() { ...}
   public void insertar(E x) {...}
   public void insertarEnFin(E x) {...}
   public E borrar(E x) throws ElementoNoEncontrado { ...}
   public String toString() { ...}
}
```


La Clase LDEG (II)

```
public void insertar(E x){
   NodoLDEG<E> nuevo = new NodoLDEG<E>(x);
   if ( primero != null )  {
         nuevo.siguiente = primero;
         primero.anterior = nuevo;
         primero = nuevo;
   } else  primero = nuevo;
   talla++;
}
```



La Clase LDEG (III)

- Compactando el código:

```
public void insertar(E x){
   NodoLDEG<E> nuevo = new NodoLDEG<E>(x, primero, null);
   if ( primero != null ) primero.anterior = nuevo;
   primero = nuevo;
   talla++;
}
```

- El coste de la operación es constante, es decir, independiente del número de elementos en la lista.
  - El tiempo que tarda en ejecutarse la operación no depende de la cantidad de datos insertados previamente

La Clase LDEG (IV)

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

Ejercicio Propuesto

- Escribir un método toString() de LDEG que obtenga una representación textual de los Nodos de la Lista en orden descendente, del último al primero.
- Responde a las siguientes preguntas:
  1. ¿Cuál es el orden en el que se obtienen los datos, respecto al orden en el que fueron insertados?
  2. ¿Cuál es el coste temporal del algoritmo diseñado?
  3. ¿Se podría reducir más su coste temporal?, ¿Cómo?
  4. ¿Se podría reducir más su coste asintótico (complejidad temporal)?

Soluciones al Problema de Ordenación de los Datos de una LEG

1. Copiar el dato de cada nodo de la LEG en un array y ordenarlo con un método genérico de ordenación.
   - Copia: Coste **lineal** con el número de elementos (O(n))
   - Ordenación con inserción directa: Coste $O(n^2)$

¿Cual es el coste temporal de esta aproximación?

1. Construir la LEG por inserción ordenada de sus nodos → Lista Enlazada Ordenada
   - El coste de esta aproximación será **lineal** con el número de elementos (para buscar la posición de inserción).
   - Cuando hablamos de Lista Ordenada, la clase de los datos de los nodos de la lista debe implementar la interfaz **Comparable<E>.**

La Clase LEGOrdenada

```
package librerias.estructurasDeDatos.lineales;
import excepciones.*;
public class LEGOrdenada <E extends Comparable<E>> extends LEG<E>{
  public void insertar(E x){
     NodoLEG<E> nuevo  = new NodoLEG<E>(x);
      NodoLEG<E> ant = null, aux = primero;
      while ( aux != null && aux.dato.compareTo(x) < 0 ) {
         ant = aux; aux = aux.siguiente;
      }
      nuevo.siguiente = aux;
      if ( ant != null )  ant.siguiente = nuevo;
      else primero = nuevo;
      talla++;
  }}
```

¿Qué diferencia una LEGOrdenada de una LEG?

Cuestión

- ¿Es conveniente que LEGOrdenada sobrescriba también el método borrar de LEG? En caso afirmativo, indíquese por qué y realícense las modificaciones oportunas