Algoritmos y Estructuras de Datos

Departamento de Informática
LAS - TUP

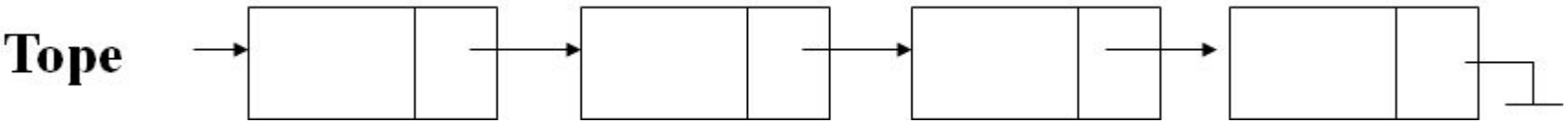
Contenedores lineales

Objetivos:

- Introducir conceptos de contenedores lineales.
- Pilas
- Colas
- Listas
- Implementación:
  - Mediante arreglos
  - Mediante lista enlazada simple
  - **Mediante lista enlazada doble**
- **Contenedores particulares**

Contenedores lineales - Listas

Una lista es una colección lineal de elementos llamados nodos donde el orden de los mismos se establece mediante referencias y existe una referencia especial (tope) para localizar al primer elemento.



Contenedores lineales - Listas

Un nodo se divide en 2 partes:

- **Información**: Contiene la información del elemento.
- **Enlace**: Contiene la dirección del siguiente nodo de la lista.

Contenedores lineales - Listas

Lista enlazada simple

- Colección lineal de elementos llamados nodos.
- Existe un elemento llamado **tope** que apunta al primer elemento de la lista.
- Cada nodo contiene **un campo de enlace** que apunta al siguiente elemento.
- El último elemento de la lista en su campo enlace apunta a nulo.
- Al principio el apuntador **tope** apunta a nulo.

Contenedores lineales - Listas

Operaciones con lista enlazada simple

- Insertar: Agrega un elemento a la lista.
- Eliminar: Retira un elemento de la lista.
- Vacía: Indica si la lista contiene o no elementos.
- Tamaño: Indica el número de elementos de la lista.
- Buscar: Busca un elemento en la lista.
- Recorrer: Visita todos los elementos de la lista.

Contenedores lineales - Listas

Lista enlazada simple en JAVA

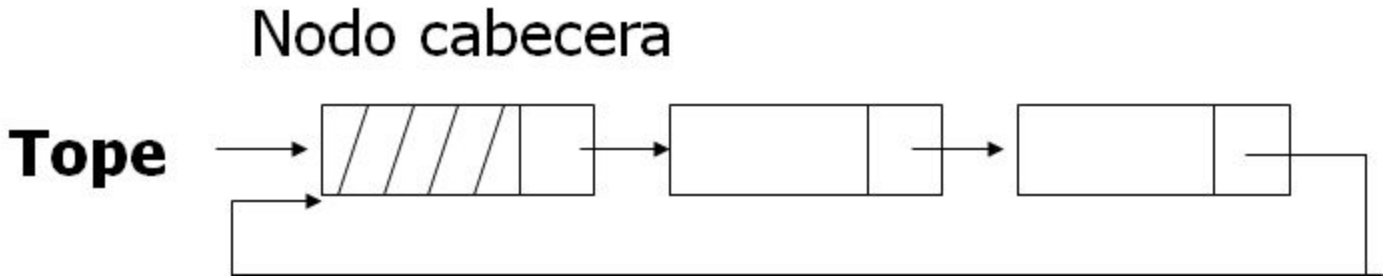
- Especificar: Interfaz
- Implementar: Clase
- Probar: Instancias y operaciones.

Unidad N° 5: Listas enlazada doble

En algunas aplicaciones podríamos necesitar recorrer la lista hacia adelante y hacia atrás, o dado un elemento, podríamos necesitar conocer rápidamente los elementos anterior y siguiente.

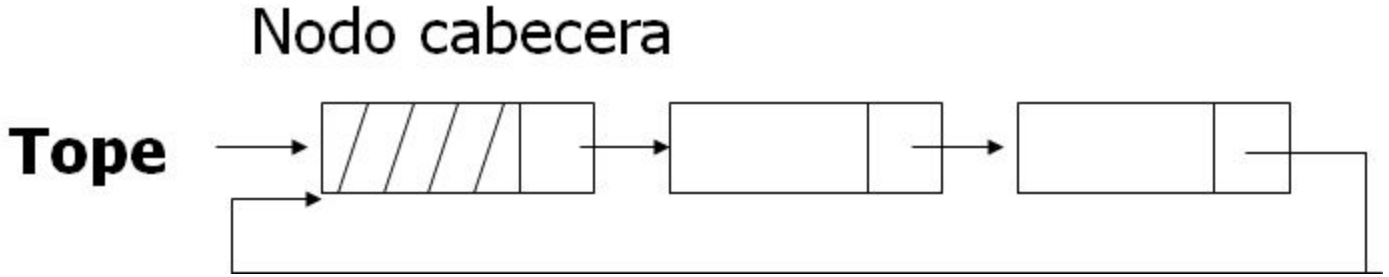
Contenedores lineales - Listas

Lista circular:



Contenedores lineales - Listas

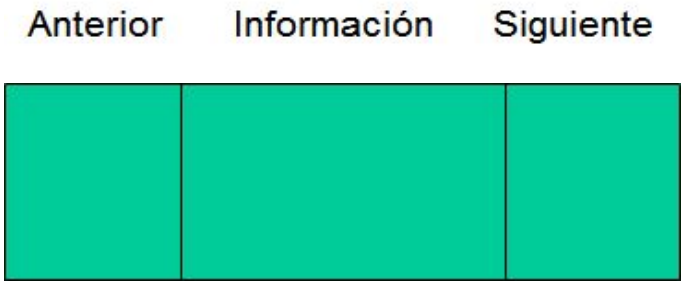
Lista circular:



Sin embargo, en otras situaciones podríamos darle a cada nodo de la lista un enlace a los nodos siguiente y anterior en la lista.

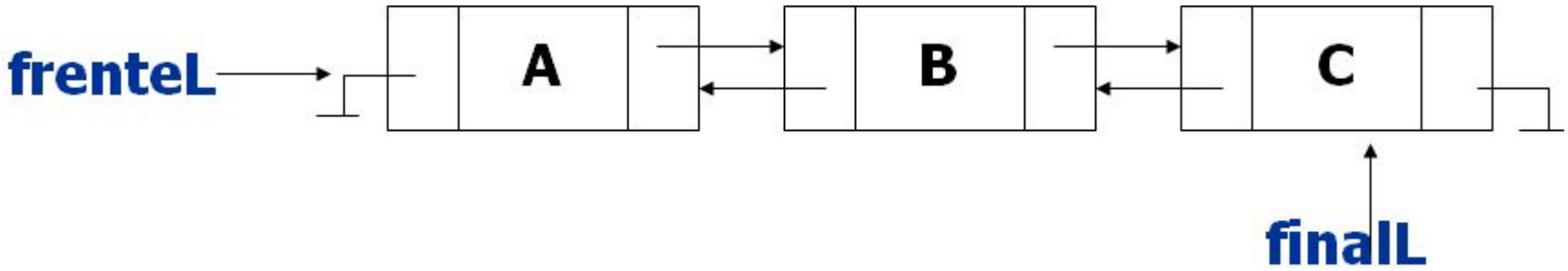
Unidad N° 5: Listas enlazada doble

Una **lista doble** es una estructura lineal de elementos llamados nodos los cuales contienen dos campos de enlace: uno al elemento **anterior** y otro al elemento **siguiente** de la lista.



Unidad N° 5: Listas enlazada doble

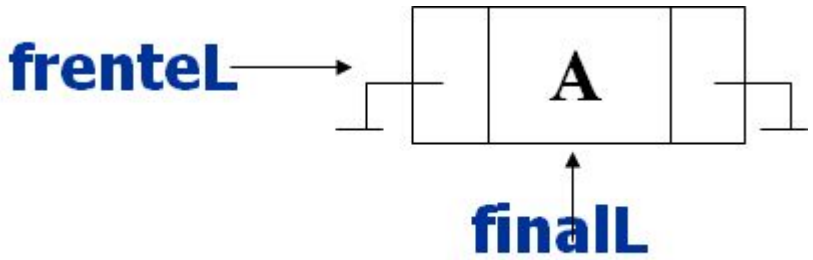
El primer nodo de la lista contiene nulo en su enlace al elemento anterior y el último nodo de la lista contiene nulo en su enlace al elemento siguiente.



Unidad N° 5 Listas enlazada doble

Lista vacía:

Lista de un sólo elemento:



Unidad N° 4: Listas enlazada doble

```
public class NodoDoble {
    private Object nodoInfo;
    private NodoDoble prevNodo, nextNodo;
    public NodoDoble(Object nodoInfo){
        this(nodoInfo,null,null);
    }
    public NodoDoble(Object nodoInfo, NodoDoble nextNodo){
        this(nodoInfo,null,nextNodo);
    }
    public NodoDoble(Object nodoInfo, NodoDoble prevNodo, NodoDoble nextNodo){
        this.nodoInfo=nodoInfo;
        this.prevNodo=prevNodo; this.nextNodo=nextNodo;
    }
    public void setPrevNodo(NodoDoble prevNodo){        this.prevNodo=prevNodo; }
    public NodoDoble getPrevNodo(){     return this.prevNodo;   }
    public void setNextNodo(NodoDoble nextNodo){    this.nextNodo=nextNodo; }
    public NodoDoble getNextNodo(){     return this.nextNodo;   }
    public void setNodoInfo(Object nodoInfo){       this.nodoInfo=nodoInfo;     }
    public Object getNodoInfo(){        return this.nodoInfo;   }
}
```

Lista enlazada doble: Especificación

```java
public interface OperacionesCL2 {
    public int buscar(Object elemento);
    public Object devolver(int posicion);
    public void eliminar(int posicion);
    public void limpiar();
    public boolean estaVacia();
    public int tamanio();
}
```

Lista enlazada doble: Implementación

```
public abstract class Lista0DLinkedL implements OperacionesCL2 {
    protected NodoDoble frenteL, finalL;
    protected int ultimo;

    public Lista0DLinkedL() {
        this.limpiar();
    }

    public void limpiar() {
        this.frenteL = this.finalL = null;
        this.ultimo = -1;
    }

    public boolean estaVacia() {
        return (this.frenteL == null);
    }

    public int tamanio() {
        return (this.ultimo + 1);
    }
```

Lista enlazada doble: Implementación - eliminar

```
public void eliminar(int posicion) {
    .
    .
    .

}
```

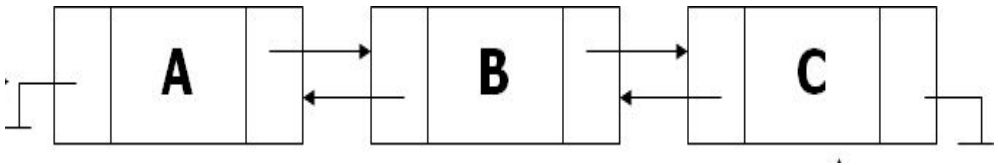
Controlar que la lista no esté vacía.

La posición sea válida

Es el único elemento

Es el primer elemento

Es el último elemento



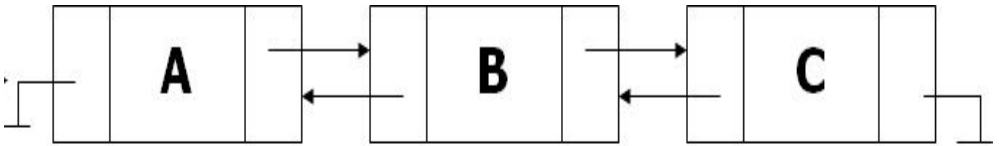
Llegar hasta la posición y actualizar las referencias del nodo previo y siguiente.

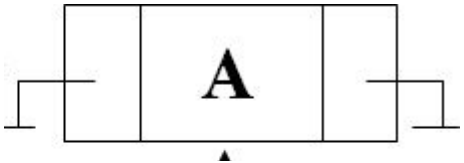
Lista enlazada doble: Implementación - eliminar

```
public void eliminar(int posicion) {
    if (estaVacia()) {
        System.out.println("Error eliminar. Lista vacia...");
    }else{
        if (posicion >= tamanio() || posicion < 0) {
            System.out.println("Error eliminar. Posicion inexistente ");
        }else{
            if (posicion == 0) {
                if (this.frenteL == this.finalL) {
                    limpiar();
                }else{
                    this.frenteL = this.frenteL.getNextNodo();
                    this.frenteL.setPrevNodo(null);
                    this.ultimo--;
                }
            }else{
                if (posicion == tamanio() - 1) {
                    this.finalL = this.finalL.getPrevNodo();
                    this.finalL.setNextNodo(null);
                }else{
                    NodoDoble prev, next;
                    prev = this.frenteL;
                    next = this.frenteL.getNextNodo();
                    for (int counter = 1; counter < posicion; counter++) {
                        prev = prev.getNextNodo();
                        next = next.getNextNodo();
                    }

                    next = next.getNextNodo();
                    prev.setNextNodo(next); // actualizamos referencias
                    next.setPrevNodo(prev);
                }
                this.ultimo--;
            }
        }
    }

}
```





Cuestiones de interés

- En la interfaz faltan operaciones.

Lista enlazada doble: Especificación

```java
public interface OperacionesCL2 {
    public int buscar(Object elemento);
    public Object devolver(int posicion);
    public void eliminar(int posicion);
    public void limpiar();
    public boolean estaVacia();
    public int tamanio();
}
```

Lista enlazada doble: Especificación

```java
public interface OperacionesCL2 {
    public int buscar(Object elemento);
    public Object devolver(int posicion);
    public void eliminar(int posicion);
    public void limpiar();
    public boolean estaVacia();
    public int tamanio();
}
```

```java
public interface OperacionesCL3 {
    public void insertar(Object elemento, int posicion);
    public void reemplazar(Object elemento, int posicion);
}
```

```java
public interface OperacionesCL4 {
    public void insertar(Object elemento);
}
```

Lista enlazada doble: Implementación

```
public abstract class Lista0DLinkedL implements OperacionesCL2 {
    protected NodoDoble frenteL, finalL;
    protected int ultimo;

    public Lista0DLinkedL() {
        this.limpiar();
    }

    public void limpiar() {
        this.frenteL = this.finalL = null;
        this.ultimo = -1;
    }

    public boolean estaVacia() {
        return (this.frenteL == null);
    }

    public int tamanio() {
        return (this.ultimo + 1);
    }
```

Lista enlazada doble: Implementación

Extender!

```
// implementando una lista simple
public abstract class Lista1DLinkedL extends Lista0DLinkedL implements OperacionesCL3 {
    public void insertar(Object elemento, int posicion) {
        NodoDoble nodo;
        if (posicion > tamanio() || posicion < 0) {
            System.out.println("Error insertar. Posicion inexistente ");
        }else{
            if (posicion == 0) { // insercion al comienzo
                if (!estaVacia()) {
                    this.frenteL = new NodoDoble(elemento, null, this.frenteL);
```

Lista enlazada doble: Implementación

Extender!

```
// lista ordenada
public abstract class Lista2DLinkedL extends Lista0DLinkedL implements OperacionesCL4 {

    // insercion ordenada
    public void insertar(Object elemento) {
        NodoDoble nodo;

        if (estaVacia()) {
            this.frenteL = this.finalL = new NodoDoble(elemento);

        } else{
            if (esMenor(elemento, this.frenteL.getNodoInfo())) {        //insercion al frent
                this.frenteL = new NodoDoble(elemento, null, this.frenteL); // nuevo frente
```

Lista enlazada doble: Implementación

Razones:

- Reutilizar código.
- Buscar generalidad.
- Reducir tiempo de mantenimiento.

Cuestiones de interés

- En la interfaz faltan operaciones.
- Clases abstractas

Lista enlazada doble: Implementación

Método abstracto:

```
public abstract class Lista0DLinkedL implements OperacionesCL2 {

        .

        .

        public abstract int buscar(Object elemento);

        .

        .

}
```

Lista enlazada doble: Implementación

Método abstracto:

Las subclases (Lista1DLinkedL y Lista2DLinkedL) implementa el método abstracto “buscar”

Entonces al instanciar un objeto de alguna de estas dos clases, ya tenemos la implementación de los métodos definidos en la interfaz!

Pero las subclases también son abstractas!!!?

Lista enlazada doble: Implementación

Las subclases tienen métodos abstractos: por ejemplo “iguales”

```
public abstract boolean iguales(Object elementoL, Object elemento);

public int buscar(Object elemento) {
    int posicion = -1; int contador = 0;
    Object unElemento;
    NodoDoble temp;

    temp = this.frenteL;
    while (temp != null && posicion == -1) {
        unElemento = temp.getNodoInfo();
        if (iguales(unElemento,elemento)) {
            posicion = contador;
        }else{
            temp = temp.getNextNodo();
            contador++;
        }
    }
    return posicion;
}
```

Lista enlazada doble: Implementación

Al momento de utilizar el contenedor, en nuestro caso creamos una lista de enteros

```
public class ListaEnteros extends Lista1DLinkedL {
    public boolean iguales(Object elementoL, Object elemento) {
        if (((Integer)elementoL).intValue() == ((Integer)elemento).intValue()) {
            return true;
        }else {
            return false;
        }
    }

    public void muestra() {
        NodoDoble temp;

        if (!estaVacia()){
            temp=this.frenteL; // el primero
            while (temp != null) {

                System.out.println(temp.getNodoInfo().toString());
                temp = temp.getNextNodo();
            }
        }else{
            System.out.println("Error muestra. Lista vacia");
        }

    }
}
```

Lista enlazada doble: Implementación

Al momento de utilizar el contenedor, en nuestro caso creamos una lista de enteros

```
public class Test {
    public static void main(String[] args) {
        Integer obj1, obj2, obj3, obj4, obj5, obj6;

        obj1 = new Integer(100);
        obj5 = new Integer(55);
        obj6 = new Integer(77);


        ListaEnteros lista = new ListaEnteros();

        lista.insertar(obj1, 0); // inserto inicio
        lista.insertar(obj3, 0);
        lista.insertar(obj4, 3); // inserto fin

        lista.muestra();

        lista.eliminar(4);
        lista.muestra();

        System.out.println(" 90 en que posicion esta? " + lista.buscar(new Integer(90)));
        System.out.println("-90 en que posicion esta? " + lista.buscar(new Integer(-90)));

        //System.out.println("mostrando posicion 3 " + lista.devolver(3).toString());
        lista.reemplazar(new Integer(666), 0);

        lista.muestra();
```

Lista enlazada doble: Lista ordenada

Es una lista especial donde los elementos aparecen ordenados respecto al campo info.

Vimos que hay algunos métodos que tienen una especificación diferente al caso lista:

- En una lista vacía
- Al inicio
- Al final
- Al medio

Lista enlazada doble: Lista ordenada - insertar

```
: abstract class Lista2DLinkedL extends Lista0DLinkedL implements OperacionesCL4 {

' insercion ordenada
iblic void insertar(Object elemento) {
   NodoDoble nodo;

   if (estaVacia()) {
       this.frenteL = this.finalL = new NodoDoble(elemento);

   } else{
       if (esMenor(elemento, this.frenteL.getNodoInfo())) {        //insercion al frente
           this.frenteL = new NodoDoble(elemento, null, this.frenteL); // nuevo frente
           this.frenteL.getNextNodo().setPrevNodo(this.frenteL); // vamos al 2do y conectamos con el 1ero

       }else{

           if (esMayor(elemento, this.finalL.getNodoInfo()) || iguales(elemento, this.finalL.getNodoInfo())) {
               //Completar!!!

           }else{
               // al medio
               NodoDoble temp = this.frenteL;
               boolean flag = false;
               while (temp.getNextNodo() != null && !flag) {
                   if (esMayor(elemento, temp.getNextNodo().getNodoInfo())) {
                       temp = temp.getNextNodo();
                   }else{
                       flag = true;
                   }
               }

               // insercion al medio. entre temp y temp.next
               nodo = new NodoDoble(elemento, temp, temp.getNextNodo());
               temp.getNextNodo().setPrevNodo(nodo);
               temp.setNextNodo(nodo);
           }
       }
   }
```

Cola de prioridad

Es un tipo especial de cola donde un elemento de mayor prioridad será desencolado antes de un elemento de mayor prioridad.

Organizar la implementación de Colas para evitar solapamiento de código y especificación entre ambos tipos de Colas.

Conjunto

Es un tipo contenedor especial que se caracteriza por no tener elementos repetidos y no existe un orden.

Operaciones: meter, sacar, unión, intersección, etc.

Una lista con los elementos del conjunto.

El recorrido de la lista en cada operación?

Conclusiones

La implementación de Listas mediante arreglos, requiere que se especifique el tamaño máximo de lista durante la compilación (o al instanciar el objeto). Esto no es necesario si se usa lista enlazada.

Los tiempos computacionales asociados al uso de las operaciones básicas en listas no son uniformes. Las operaciones INSERTAR y ELIMINAR para una lista enlazada, ocasionan menor costo computacional que si se comparan con la implementación usando arreglos.

Conclusiones

La implementación de listas con arreglos, puede malgastar espacio en memoria, debido al espacio que debe reservarse de antemano. No obstante, la manipulación de los elementos de la lista (operaciones matemáticas, actualización) no generan demasiado costo adicional.

La implementación de Listas mediante lista enlazada, usa sólo el espacio necesario para almacenar los elementos de la lista. Sin embargo, para cada celda/nodo se almacena la dirección a la siguiente celda. Además, su uso trae aparejado un mayor esfuerzo durante la programación y prueba.

Conclusiones

No es necesario implementar (y copiar) una lista ordenada, cola de prioridad, etc cuando tenemos una buena organización y especificación previa de lo de desarrollado.

La abstracción y la genericidad son fundamentales para un buen desarrollo de programas.

POO y Java facilitan el desarrollo de programas genéricos, abstractos, reusables y extensibles.