10 | Lista doblemente ligada

Meta

Que el alumno domine el manejo de información almacenada en una *Lista*.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Visualizar cómo se almacena una lista en la memoria de la computadora mediante el uso de nodos con referencias a su elemento anterior y su elemento siguiente.
- Programar dicha representación en un lenguaje orientado a objetos.

Antecedentes

Definición 10.1

Una lista es:

$$\text{Lista} = \begin{cases} \text{Lista vacía} \\ \text{Dato seguido de otra lista} \end{cases}$$

Alternativamente:

Definición 10.2

Una **lista** es una secuencia de cero a más elementos **de un tipo determinado** (que por lo general se denominará tipo-elemento). Se representa como una sucesión de

elementos separados por comas:

$$a_0, a_1, ..., a_{n-1}$$

donde $n \geqslant 0$ y cada a_i es del tipo **tipo-elemento**.

- Al número n de elementos se le llama *longitud* de la lista.
- a_0 es el *primer elemento* y a_{n-1} es el *último elemento*.
- Si $n = 0$, se tiene una **lista vacía**, es decir, que no tiene elementos. Aho, Hopcroft y Ullman 1983, pp. 427

En este caso utilizaremos como definición del tipo de datos abstracto, la interfaz definida por Oracle:

`http://docs.oracle.com/javase/8/docs/api/java/util/List.html`.

Actividad 10.1

Lee la definición de la interfaz `List<E>`. ¿Te queda claro lo qué debe hacer cada método? Si no, pregunta a tu ayudante.

Actividad 10.2

Elige los métodos que concideres más importantes y dibuja cómo te imaginas que se ve la lista antes de mandar llamar un método y qué le sucede cuando éste es invocado.

Una **lista doblemente ligada** es una implementación de la estructura de datos lista, que se caracteriza por:

1. Guardar los datos de la lista dentro de nodos que hacen referencia al nodo con el dato anterior y al nodo con el siguiente dato.
2. Tener una referencia al primer y último nodo.
3. Tener un tamaño dinámico, pues el número de datos que se puede almacenar está limitado únicamente por la memoria de la computadora y el tamaño de la lista se incrementa y decrementa conforme se insertan o eliminan datos de ella.
4. Es fácil recorrerla de inicio a fin o de fin a inicio.

Desarrollo

Para implementar el TDA *Lista* se deberá extender la clase:

`ColeccionAbstracta<E>`

programada anteriormente e implementar la interfaz `List<E>`. Esto se deberá hacer en una clase llamada

`ListaDoblementeLigada<E>`.

En tu implementación eres libre de escoger la forma en la que manejes las referencias al inicio y fin de la lista, pero éstas deben estar presentes y debes mantener su consistencia correctamente.

1. Dentro del paquete correspondiente, `ed.estructuras.lineales`, programa `ListaDoblementeLigada<E>` según lo indicado. Observa que varios de los métodos ya los implementaste en `ColeccionAbstracta<E>`.
2. Programa los métodos faltantes. Sólamente `sublist()` es opcional, los demás son obligatorios.
3. Sobrecarga el método `clear()` de manera que su complejidad sea $\mathcal{O}(\mathbf{1})$

Preguntas

1. Explica la diferencia conceptual entre los tipos `Nodo<E>` y `E`.
2. ¿Por qué `ListIterator` sólo permite remover, agregar o cambiar datos después de llamar `previous` o `next`?
3. Si mantenemos los elementos ordenados alfabéticamente, por ejemplo, ¿cuándo sería más eficiente agregar un elemento desde el inicio o el final de la lista?
4. ¿En qué casos sería más eficiente obtener un elemento desde el inicio de la lista o desde el final de la lista?