Actividad 4.4

Lista doblemente enlazada y circular

Algoritmos y Estructuras de Datos

Tema 4: listas, pilas y colas

Modificar la actividad 4.3 para que la lista ahora sea doblemente enlazada y circular. Para el usuario de la clase no habrá ninguna diferencia de utilización en comparación con la clase de la actividad 4.3, por lo cual las cabeceras de los métodos públicos (nombre, número y tipo de parámetros, retornos) serán idénticas a los de la actividad 4.3.

En concreto, realizaremos las siguientes modificaciones:

- Ahora la estructura Nodo es diferente. Tiene un puntero tanto al anterior nodo como al siguiente.
- En la clase ListaEnlazada vamos a añadir dos atributos más: "posicionUltimoNodoAccedido" y "punteroUltimoNodoAccedido". Representan la posición y el puntero del último nodo que se devolvió la última vez que se llamó a "getNodo()". Los pondremos a -1 y a NULL respectivamente en el constructor. Igualmente, cada vez que llamemos a eliminar() o a insertar() también los pondremos a -1 y a NULL, para evitar complicaciones (esto podría mejorarse mucho).
- Si el método "getNodo()" tiene que devolver un nodo que es el inmediatamente siguiente o anterior al último nodo accedido (o incluso es el mismo nodo al que se accedió la última vez), usará los dos atributos descritos para acceder a él de forma inmediata, en O(1), en vez de empezar por el principio. De este modo sucesivas llamadas a getNodo() con elementos en posiciones sucesivas (como por ejemplo en imprimirListaEnlazada(), o en concatenar(), o en buscar()) tendrán una complejidad temporal de O(1) cada una de las llamadas, en vez de O(n). De este modo, por ejemplo, imprimirListaEnlazada() ahora será O(n) en vez de O(n^2), sin necesidad de haber tocado nada en el código interno de imprimirListaEnlazada().
- En caso de que el nodo a buscar no esté pegado al último accedido, el método "getNodo()" buscará "hacia delante" o "hacia atrás" según la posición del nodo cuyo puntero queramos obtener. Ej: si la lista tiene 100 elementos, y queremos acceder al nodo 30, iremos iterando "hacia delante" (nodo 0, nodo 1, nodo 2… así hasta el nodo 30). Sin embargo, si quisiéramos acceder al nodo 60, iremos iterando "hacia atrás" (nodo 0, nodo 99, nodo 98, nodo 97… hasta nodo 60), pues de esta manera pasamos por menos nodos que yendo hacia delante.
- Un método "concatenar()", con la misma cabecera y características que en la actividad 4.2. Mejorará mucho su complejidad temporal debido a las mejoras en getNodo(), descritas anteriormente.

- Un método “buscar()”, con la misma cabecera y características que en la actividad 4.2. Igualmente, su complejidad temporal permanecerá en O(n) debido a las mejoras en getNodo(), descritas anteriormente.

Para probarlo, amplía el main de la actividad 4.3 para probar también la concatenación y la búsqueda, tal cual como aparece en la siguiente prueba de caja negra: