13 | 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.

Código Auxiliar 13.1: Lista doblemente ligada

https://github.com/computacion-ciencias/ed-lista-doblemente-ligada-cs

Antecedentes

Estructura

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 con datos.
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.

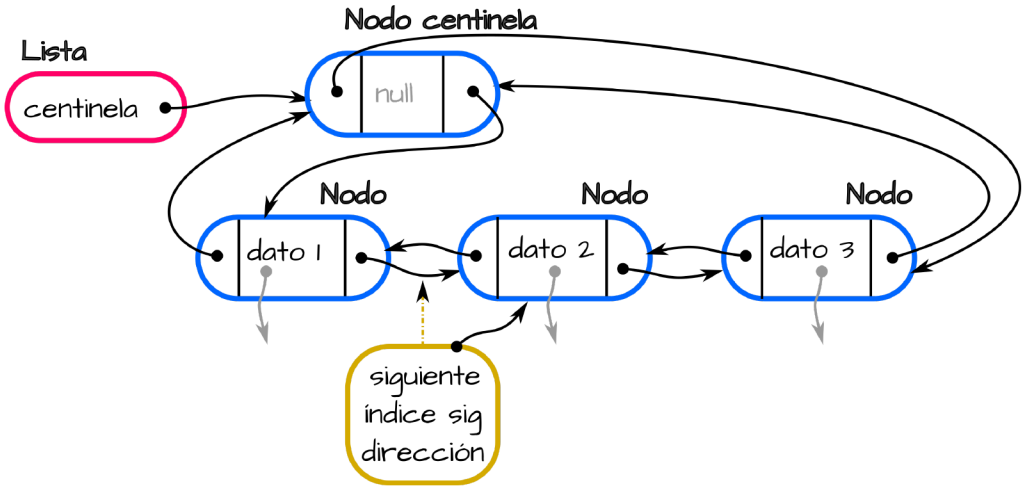

Figura 13.1 Lista doblemente ligada con nodos, centinela e iterador.

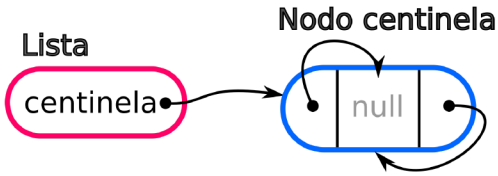

Figura 13.2 Lista con centinela vacía. El centinela apunta a si mismo como primer y último elemento.

4. Es fácil recorrerla de inicio a fin o de fin a inicio.

En particular, un forma de mantener las referencias al primer y último nodo es utilizar un nodo *centinela*. La clase lista tendrá un atributo que siempre hace referencia al centinela. Este nodo no guarda ningún dato en su interior, pero su nodo siguiente será el primer nodo de la lista y su nodo anterior será el último, como muestra la Figura 13.1; si la lista está vacía el centinela apunta a sí mismo en ambas direcciones Figura 13.2.

Con esta implementación las propiedades `Anterior` y `Siguiente` de los nodos nunca son `null`, lo que facilitará mucho la programación de los métodos para agregar y remover elementos de la lista, pues agregar o remover de los extremos se ve exactamente igual que si se estuviera en medio de la lista.

Iterador

Conceptualmente, el iterador para una lista en C#se encuentra colocado en el elemento que será devuelto por la propiedad `Current`, para colocarlo sobre el primer elemento e irlo desplazando hacia los demás se utiliza el método `MoveNext`, que devolverá `true` si logró desplazarse y `false` si ya había llegado al final de la lista. El iterador para esta práctica también puede desplazarse en reversa con el método `MovePrevious`, el cual le

permite seleccionar al elemento anterio al que estaba señalando.

Actividad 13.1

En el código auxiliar lee la definición de la interfaz `IEnumeradorLista<T>`, luego revisa la documentación de la interfaz `IList<T>`. ¿Te queda claro lo qué debe hacer cada método? Si no, pregunta a tu ayudante.

Como la definición de la lista permite acceder a todos sus elementos, el iterador también tendrá la capacidad de insertar y remover datos conforme se desplaza a través de la estructura. Para ello, el objeto iterador debe contar con atributos que le permitan acceder a los nodos que serían devueltos por llamadas a los métodos `MoveNext` y `MovePrevious` y recordar qué movimiento fue el último que realizó. Además, como se vió en la definición de lista, cada elemento en ella tiene asociado un índice que indica su posición en la estructura; por ello el iterador también debe llevar la cuenta del número de elemento que devolverían cualquiera de las llamadas a los métodos anteriores, de modo que la propiedad `Índice` pueda devolver la posición del elemento `Current` cuando éste sea válido.

La Figura 13.1 muestra cómo podemos visiualizar la colocación del objeto iterador. Observese que, a través de las diferentes referencias tanto en el iterador, como en los nodos de la lista, es posible acceder directa o indirectamente a todas las piezas de información que requieren los métodos anteriores y reasignar variables acordemente cuando el iterador sea desplazado.

Desarrollo

Para implementar el TDA *Lista* se deberá implementar la interfaz `IListа<T>` del proyecto `IColección`. Esto se deberá hacer en una clase llamada

`ListaDoblementeLigada<T>`.

dentro del espacio de nombres

`ED.Estructuras.Lineales.Lista`

Observa que, en esta ocasión, sí permitiremos el almacenamiento de datos `null` en la estructura, ten especial cuidado en los métodos donde operarás con los datos almacenados.

1. Dentro del proyecto correspondiente crea la clase `ListaDoblementeLigada<T>` y dentro de ella programa una clase con acceso privado para representar al nodo con

el dato a almacenar y las referencias a los nodos anteriores y siguientes. Observa que, al tratarse de una clase interna, puede utilizar la variable genérica de la clase que la contiene. Agrega los constructures que consideres necesarios y propiedades para acceder a sus información.

2. Ahora programa `ListaDoblementeLigada<T>` según lo indicado, comienza por el constructor para una lista vacía; en él deberás crear al centinela y asignar sus referencias como se indicó anteriormente.

3. A continuación programa el iterador. Observa que en esta ocasión se debe implementar `IEnumeradorLista<T>`. Prueba que los métodos del iterador funcionen, si programas de una vez el método `Add(T item)` de la lista, las pruebas unitarias te ayudarán a ver si vas bien.

4. Programa ahora los demás métodos de la lista, observa que si usas el iterador `Contains` y `CopyTo` pueden ser idénticos a los de la pila que programaste. Para los métodos que faltan usa el iterador cada vez que puedas, te ahorrará mucho trabajo y hará más fácil que tu código sea correcto.

   Las propiedades a implementar son:

   - `Count`
   - `IsReadOnly`, que devuelve `false`.

   Y los métodos a programar son:

   - `public T this[int index]` Es la sobrecarga del operador `[]`, como una propiedad tiene asociados `get` y `set`, de modo que se pueda utilizar con esta lista la misma notación que se usa para acceder a los elementos de un arreglo.
   - `public IEnumerator<T> GetEnumerator())`
   - `public IEnumerator IEnumerable.GetEnumerator()`
   - `public IEnumerator<T> GetEnumerator(int index)`
     Este método en particular te ayudará a programar varios de los métodos siguientes. Si necesitas colocar el iterador en alguna posición que está más cerca del final que del principio, inicialo al final y retrocede tantos lugares como necesites.
   - `public void Add(T item)`
   - `public void Clear()`
   - `public int IndexOf(T item)`
   - `public void Insert(int index, T item)`
   - `public bool Remove(T item)`
   - `public void RemoveAt(int index)`
   - `public bool EstáVacía()`

Preguntas

1. Explica la diferencia conceptual entre los tipos `Nodo<E>` y `E`.
2. ¿Por qué `IEnumeradorLista<T>` sólo permite remover o cambiar datos después de llamar `MovePrevious` o `MoveNext`?
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?