cabeza

|

▼ [7|●]—▶[15|●]—▶[23|●]—▶[42|●]—▶null 1

Insertar el valor 3 al INICIO

Estado ANTES

cabeza

|

▼ [7|●]—▶[15|●]—▶[23|●]—▶[42|●]—▶null Paso

1: crear el nuevo nodo

nuevo

|

▼

[3|●] [7|●]—▶[15|●]—▶[23|●]—▶[42|●]—▶null Paso 2: el

nuevo nodo apunta al antiguo primero nuevo

|

▼ [3|●]—▶[7|●]—▶[15|●]—▶[23|●]—▶[42|●]—▶null Paso

3: cabeza apunta al nuevo nodo

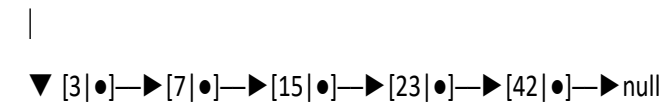
cabeza

|

▼ [3|●]—▶[7|●]—▶[15|●]—▶[23|●]—▶[42|●]—▶null Estado

DESPUÉS

cabeza



Complejidad: O(1)

Porque solo se cambian punteros, no se recorre la lista.

2 Insertar el valor 50 al FINAL Estado

ANTES

cabeza

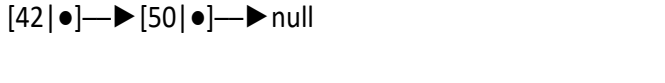
Paso 1: recorrer la lista hasta el final

actual

(Se recorrieron todos los nodos)

Paso 2: crear el nuevo nodo nuevo

Paso 3: el último nodo apunta al nuevo



Estado DESPUÉS

cabeza

|

▼ [3|●]—▶[7|●]—▶[15|●]—▶[23|●]—▶[42|●]—▶[50|●]—▶null

Complejidad: O(n)

Porque hay que recorrer toda la lista hasta encontrar el último nodo.

3 Eliminar el nodo con valor 15

Estado ANTES

cabeza

|

▼ [3|●]—▶[7|●]—▶[15|●]—▶[23|●]—▶[42|●]—▶[50|●]—▶null Paso 1:

recorrer buscando el nodo

anterior actual

| |

▼ ▼

[7|●]—▶[15|●]—▶[23|●] Paso

2: saltar el nodo 15

anterior.next = actual.next

Visualmente:

[7|●]————▶[23|●] X

[15|●]

Paso 3: eliminar el nodo

[15|●] (liberado)

Estado DESPUÉS

cabeza

|

▼ [3|●]—▶[7|●]—▶[23|●]—▶[42|●]—▶[50|●]—▶null

Complejidad: O(n)

Porque es necesario recorrer la lista para encontrar el nodo.

¿Por qué insertar al inicio es O(1)?

Porque no importa cuántos nodos tenga la lista:

solo se crea un nodo y se cambian dos referencias.

2 ¿Por qué insertar al final es O(n)?

Porque se debe recorrer toda la lista para encontrar el último nodo antes de insertar.

3 ¿Qué problema surge al eliminar el primer nodo?

Que no existe un nodo anterior, así que no se puede “reconectar” la lista como en otros casos.

La solución es mover directamente la referencia cabeza.

¿Puedo explicar en mis propias palabras por qué una lista enlazada no permite acceso O(1) por índice, aunque los arreglos sí?

Sí.

En una lista enlazada los nodos no están en posiciones contiguas de memoria y cada nodo solo conoce al siguiente.

Por eso, si quiero llegar al "nodo 4", tengo que empezar desde cabeza y recorrer nodo por nodo hasta llegar a él, lo que toma O(n).
En cambio, en un arreglo, los elementos están almacenados de forma continua, así que el sistema puede calcular directamente la posición del índice y acceder en O(1).

Si me piden insertar un elemento entre el nodo 3 y el nodo 4 de una lista, ¿sé qué punteros debo modificar sin consultar código?

Sí.

Solo debo modificar dos punteros
El nuevo nodo debe apuntar al nodo que antes era el siguiente (nuevo.next = actual.next)
El nodo anterior debe apuntar al nuevo nodo (actual.next = nuevo)
No es necesario mover ni copiar otros nodos; solo se ajustan las referencias.

¿Entiendo por qué necesito referencia al nodo ANTERIOR para eliminar un nodo en una lista simple?

Sí.

En una lista enlazada simple, cada nodo solo apunta al siguiente, no al anterior.

Para eliminar un nodo, necesito que el nodo anterior apunte directamente al siguiente del nodo que quiero eliminar, "saltándolo".

Si no tengo referencia al nodo anterior, no hay forma de reconectar la lista, excepto en el caso especial de eliminar el primer nodo, donde se mueve directamente la referencia cabeza.

En una lista enlazada simple, cada nodo solo apunta al siguiente, no al anterior.

Para eliminar un nodo, necesito que el nodo anterior apunte directamente al siguiente del nodo que quiero eliminar, “saltándolo”.

Si no tengo referencia al nodo anterior, no hay forma de reconectar la lista, excepto en el caso especial de eliminar el primer nodo, donde se mueve directamente la referencia cabeza.