Fase COMPRENDE

Prompt 1. Visualización de Lista Doblemente Enlazada

Lista inicial:

null ← [7] ↔ [15] ↔ [23] ↔ [42] → null

↑ ↑

cabeza cola

1. Insertar el valor 3 al INICIO

null ← [3] ↔ [7] ↔ [15] ↔ [23] ↔ [42] → null

↑ ↑

(nueva) cabeza cola

Complejidad: O(1) (constante)

2. Insertar el valor 50 al FINAL

null ← [3] ↔ [7] ↔ [15] ↔ [23] ↔ [42] ↔ [50] → null

↑ ↑

cabeza (nueva) cola

Complejidad: O(1)

3. Eliminar el nodo con valor 15 (con referencia directa)

null ← [3] ↔ [7] [15] [23] ↔ [42] ↔ [50] → null

↑ ↑

cabeza cola

Lista final:

null ← [3] ↔ [7] ↔ [23] ↔ [42] ↔ [50] → null

↑ ↑

cabeza cola

Explicación de diferencia de complejidades

Para eliminar un nodo es necesario conocer el nodo que le antecede. En una lista simple solo se tienen referencias al siguiente nodo, pero no hay referencia alguna hacia el nodo anterior. Por eso en las listas simples es necesario recorrer la lista hasta el nodo anterior al nodo que se desea eliminar, mientras que en las listas doblemente enlazadas siempre se conoce cual es el nodo anterior gracias al puntero extra que posee. Así pues, en las listas simples primero se recorre y luego se reasignan punteros, y en las listas doblemente enlazadas solo es necesario hacer un cambio de punteros, sin recorrer la lista.

Checkpoint metacognitivo – Fase COMPRENDE

¿Puedo dibujar de memoria la estructura de una lista doble con 3 nodos, incluyendo cabeza, cola, y todos los punteros?

Sí, ya lo hice en el ejercicio anterior.

¿Puedo explicar con mis palabras por qué eliminarFinal() es O(1) en lista doble pero O(n) en lista simple?

Porque en la lista simple no se conoce el nodo anterior, y es por eso que debe recorrerse toda la lista hasta llegar al penúltimo nodo. En las lista doble sí se conoce el nodo anterior y es por eso que solo es necesario cambiar punteros, sin precisar recorrer toda la lista.

¿Entiendo los 5 invariantes listados arriba? ¿Puedo dar un ejemplo de qué pasa si alguno se rompe?

Sí los comprendo. Si cabeza.anterior = null no se cumple entonces la lista sería infinita, o circular dependiendo de a que nodo esté haciendo referencia.

Fase APLICA

Prompt 2. Proyecto: Playlist Bidireccional con Lista Doble propia

Código fuente

```
class NodoDoble {
constructor(valor) {
this.valor = valor;
this.siguiente = null;
this.anterior = null;
}
}

class ListaDoble {
constructor() {
this.cabeza = null;
```

```
this.cola = null;
this.actual = null;
this.tamaño = 0;
}

// Insertar al inicio
insertarInicio(valor) {
const nuevoNodo = new NodoDoble(valor); // Crea un nuevo nodo

if (this.cabeza === null) { // Si la lista está vacía:
this.cabeza = nuevoNodo; // El nuevo nodo se convierte en la cabeza
this.cola = nuevoNodo; // El nuevo nodo se convierte en la cola
} else { // Si la lista no está vacía:
nuevoNodo.siguiente = this.cabeza; // El puntero siguiente del nuevo nodo apunta a la cabeza
this.cabeza.anterior = nuevoNodo; // El puntero anterior de la cabeza apunta al nuevo nodo
this.cabeza = nuevoNodo; // El nuevo nodo es la nueva cabeza
}

this.tamaño++; // Se le suma 1 al tamaño de la lista
}

// Insertar al final (O(1))
insertarFinal(valor) {
const nuevoNodo = new NodoDoble(valor); // Crea un nuevo nodo

if (this.cabeza === null) { // Si la lista está vacía los punteros de cabeza y cola apuntan al nuevo nodo
this.cabeza = nuevoNodo;
this.cola = nuevoNodo;
} else { // Si la lista no está vacía:
nuevoNodo.anterior = this.cola; // El puntero anterior del nuevo nodo apunta a la cola actual
this.cola.siguiente = nuevoNodo; // El puntero siguiente de la cola actual apunta al nuevo nodo
this.cola = nuevoNodo; // El puntero de cola apunta al nuevo nodo
}

this.tamaño++; // Se le suma 1 al tamaño de la lista
}

// Eliminar al inicio
eliminarInicio() {
if (this.cabeza === null) return null; // Lista vacía: devuelve null

const valorEliminado = this.cabeza.valor; // Guarda el valor del nodo a eliminar

if (this.cabeza.siguiente === null) { // Si la lista solo tiene un nodo la lista queda vacía
this.cola = null;
this.cabeza = null;
}
```

```
else { // Si la lista tiene más de un nodo:
this.cabeza = this.cabeza.siguiente; // La cabeza se mueve un nodo hacia adelante
this.cabeza.anterior = null; // El puntero anterior de la nueva cabeza apunta a null
}

this.tamaño--; // Se reduce el tamaño en 1
return valorEliminado; // Se devuelve el valor del nodo eliminado
}

// Eliminar al final
eliminarUltimo() {
if (this.cabeza === null) return null; // Lista vacía devuelve null
const valorEliminado = this.cola.valor; // Guarda el valor del nodo a eliminar

if (this.cabeza.siguiente === null) { // Si solo hay un nodo la lista queda vacía
this.cola = null;
this.cabeza = null;
}
else { // Si hay más de un nodo
this.cola = this.cola.anterior; // La cola se mueve al nodo anterior
this.cola.siguiente = null; // El puntero siguiente de la nueva cola apunta a null
}

this.tamaño--;

return valorEliminado;
}

// Eliminar nodo intermedio
eliminarNodo(nodo) {
if (nodo === null) return null; // Si el nodo está vacío, devuelve null
if (this.cabeza === nodo) return this.eliminarInicio(); // Si es el primer nodo usa el método
correspondiente
if (this.cola === nodo) return this.eliminarUltimo(); // Si es el último nodo usa el método correspondiente

const valorEliminado = nodo.valor;

nodo.anterior.siguiente = nodo.siguiente; // El nodo anterior al nodo a eliminar se enlaza al nodo que va
después de este último
nodo.siguiente.anterior = nodo.anterior; // El nodo siguiente al nodo a eliminar se enlaza al nodo que
va antes de este último
// El nodo a eliminar es aislado

// Los punteros del nodo eliminado se limpian
nodo.anterior = null;
nodo.siguiente = null;
```

```
this.tamaño--;

return valorEliminado;
}

buscar(valor) {
let actual = this.cabeza; // Guarda el nodo cabeza

while (actual !== null) { // Mientras haya elementos sin recorrer o la lista esté vacía
if (actual.valor === valor) return actual; // Compara el valor del nodo con el valor dado, si es igual
entonces devuelve el nodo actual
actual = actual.siguiente // Recorre una posición hacia adelante
}

return null; // Si no se encontró el valor no devuelve nada
}

recorrerAdelante() {
const arreglo = []; // Crea un arreglo vacío

let actual = this.cabeza; // Guarda el nodo cabeza

while (actual !== null) { // Mientras haya elementos sin recorrer
arreglo.push(actual.valor); // Guarda el valor del nodo en un arreglo
actual = actual.siguiente // Guarda el siguiente nodo en la variable
}

return arreglo; // Devuelve el arreglo
}

// Lo mismo que arriba pero usando el puntero anterior
recorrerAtras() {
const arreglo = [];

let actual = this.cola;

while (actual !== null) {
arreglo.push(actual.valor);
actual = actual.anterior;
}

return arreglo;
}
}

class Playlist extends ListaDoble {
constructor() {
```

```
super();
}

reproducirActual() {
return this.actual === null ? null : this.actual.valor; // Si el nodo actual está vacío no devuelve nada, si no lo está devuelve el valor del nodo
}

reproducirSiguiente() {
if (this.cabeza === null) return null; // Lista vacía devuelve null
if (this.actual === null) { // Si el nodo está vacío:
this.actual = this.cabeza; // Apunta la referencia actual hacia la cabeza
return this.actual.valor; // Devuelve el valor de la cabeza
}

if (this.actual.siguiente !== null) { // Si hay más de un nodo:
this.actual = this.actual.siguiente; // Apunta el puntero actual hacia el siguiente nodo
return this.actual.valor; // Devuelve el valor del siguiente nodo
}

return null;
}

reproducirAnterior() {
if (this.cabeza === null) return null; // Lista vacía devuelve null

if (this.actual.anterior !== null) { // Si hay un nodo anterior:
this.actual = this.actual.anterior; // Mueve el puntero actual al nodo anterior
return this.actual.valor; // Devuelve el puntero anterior
}

return null;
}

agregarCancion(nombre) {
this.insertarFinal(nombre); // Llama al método insertarFinal

if (this.actual === null) this.actual = this.cabeza; // Si actual está vacío entonces apunta hacia la cabeza
}

eliminarActual() {
if (this.actual === null) return null;

const nodoAEliminar = this.actual; // Guarda la canción a eliminar

if (this.actual.siguiente !== null) { // Si hay un nodo siguiente:
this.actual = this.actual.siguiente; // Mueve el puntero "actual" hacia el siguiente nodo
```

```
} else { // Si no:
this.actual = this.actual.anterior; // Apunta "actual" hacia el nodo anterior
}

return this.eliminarNodo(nodoAEliminar); // Devuelve el nodo a eliminar
}

mostrarPlaylist() {
let actualNodo = this.cabeza; // Guarda la canción cabeza

while (actualNodo !== null) { // Mientras haya canciones sin recorrer
if (actualNodo === this.actual) { // Si la canción es igual a la canción actual:
console.log(actualNodo.valor + " ← reproduciendo"); // Indica que se está reproduciendo esa canción
} else { // Si no:
console.log(actualNodo.valor); // Solo muestra el nombre de la canción
}

actualNodo = actualNodo.siguiente; // Avanza a la siguente canción
}
}
}

// Crear playlist
const playlist = new Playlist();

console.log("=== Agregando canciones ===");
playlist.agregarCancion("The Adults Are Talking");
playlist.agregarCancion("Ode to the Mets");
playlist.agregarCancion("Not The Same Anymore");
playlist.agregarCancion("Why are Sundays so Depressing");

playlist.mostrarPlaylist();


// ========================
// Navegación hacia adelante
// ========================

console.log("\n=== Navegando hacia adelante ===");

console.log("Actual:", playlist.reproducirActual());
console.log("Siguiente:", playlist.reproducirSiguiente());
console.log("Siguiente:", playlist.reproducirSiguiente());
console.log("Siguiente:", playlist.reproducirSiguiente());
console.log("Siguiente (ya no hay más):", playlist.reproducirSiguiente());

// ========================
```

```
// Navegación hacia atrás
// ========================

console.log("\n=== Navegando hacia atrás ===");

console.log("Anterior:", playlist.reproducirAnterior());
console.log("Anterior:", playlist.reproducirAnterior());
console.log("Anterior:", playlist.reproducirAnterior());
console.log("Anterior (ya no hay más):", playlist.reproducirAnterior());


// ========================
// Eliminar canción actual
// ========================

console.log("\n=== Eliminando canción actual ===");

console.log("Eliminada:", playlist.eliminarActual());

console.log("\nPlaylist después de eliminar:");
playlist.mostrarPlaylist();


// ========================
// Recorrido completo
// ========================

console.log("\n=== Recorrido completo adelante ===");
console.log(playlist.recorrerAdelante());

console.log("\n=== Recorrido completo atrás ===");
console.log(playlist.recorrerAtras());
```

Demostración

```
=== Agregando canciones ===
The Adults Are Talking  ← reproduciendo
Ode to the Mets
Not The Same Anymore
Why are Sundays so Depressing

=== Navegando hacia adelante ===
Actual: The Adults Are Talking
Siguiente: Ode to the Mets
Siguiente: Not The Same Anymore
Siguiente: Why are Sundays so Depressing
Siguiente (ya no hay más): null

=== Navegando hacia atrás ===
Anterior: Not The Same Anymore
Anterior: Ode to the Mets
Anterior: The Adults Are Talking
Anterior (ya no hay más): null

=== Eliminando canción actual ===
Eliminada: The Adults Are Talking

Playlist después de eliminar:
Ode to the Mets  ← reproduciendo
Not The Same Anymore
Why are Sundays so Depressing

=== Recorrido completo adelante ===
▶ Array(3) [ "Ode to the Mets", "Not The Same Anymore", "Why are Sundays so Depressing" ]

=== Recorrido completo atrás ===
▶ Array(3) [ "Why are Sundays so Depressing", "Not The Same Anymore", "Ode to the Mets" ]
```

Comentarios

Comente la mayoría de las líneas de código, véase el código fuente de arriba.

Checkpoint Metacognitivo - Fase APLICA

¿Puedo implementar eliminarNodo() sin mirar el pseudocódigo? ¿Qué parte me cuesta más?

Implementarlo en un lenguaje real quizás no, pero sí podría demostrar la lógica en pseudocódigo. Si recuerdo bien lo que tengo que hacer es apuntar el puntero siguiente del nodo anterior al que quiero eliminar hacia el nodo que va después de este, y hacer lo mismo a la inversa con el puntero anterior del nodo siguiente.

¿Cuántos punteros debo actualizar al insertar en una lista doble vs. una simple? ¿Puedo listarlos?

Depende de la posición en la que se inserte. Al inicio y al final serían 3 punteros, y en un nodo intermedio serían 2.

Inicio y final: puntero siguiente de la antigua cola o cabeza, puntero anterior de la nueva cola o cabeza y finalmente el puntero cola o cabeza.

Intermedio: Puntero siguiente del nodo anterior y puntero anterior del nodo siguiente al nodo a eliminar

Si la IA me dio código, ¿lo entiendo línea por línea o hay partes que "confío" sin verificar?

Sí comprendo lo que hacen la mayoría de las líneas de código, aunque probablemente no podría hacer la estructura completa por mí mismo.

Fase REFLEXIONA

Reflexión Guiada

Para el sistema de playlist que implementaste: ¿Qué pasaría si hubieras usado una lista simple? Las interacciones básicas del usuario tomarían mucho tiempo o serían imposibles de realizar. ¿Cuáles operaciones serían más lentas y por cuánto? Reproducir la canción anterior pasaría de ser O(1) a ser O(n) porque tendría que recorrer toda la lista de nuevo. ¿El usuario notaría la diferencia con 50 canciones? Puede que no ¿Y con 10,000? Seguramente sí

Sobre memoria: Si cada canción ocupa 200 bytes de metadatos y un puntero ocupa 8 bytes, ¿qué porcentaje de memoria extra usa la lista doble vs. la simple para una playlist de 1,000 canciones? 4% ¿Es significativo? No parece serlo.

Sobre tu proceso: ¿Cuál fue el bug más difícil que encontraste al implementar la lista doble? Ninguno relacionado con punteros. ¿Tenía que ver con olvidar actualizar un puntero? No ¿Cómo lo detectaste? Le pedí a la IA que me ayudara.

Checkpoint Metacognitivo – Fase Reflexiona

¿Puedo explicar a un compañero cuándo elegir lista doble vs. simple vs. arreglo, con ejemplos concretos?

Podría repetir los ejemplos que ya he aprendido, pero no sé si podría pensar en unos nuevos.

¿Qué patrón descubrí al implementar? ¿Hay una "receta" para las operaciones de lista doble?

Siempre deben actualizarse los punteros que puedan ser afectados por una operación determinada.

Si tuviera que enseñar "listas dobles" a alguien, ¿por dónde empezaría y qué diagrama dibujaría primero?

Diría que es lo mismo que las listas simples, con la diferencia de que se añade un puntero extra que apunta hacia el nodo anterior. Dibujaría un diagrama que exponga la simetría bidireccional entre dos nodos.

Fase VALIDA

Prompt 3. El Auditor de Listas Dobles

Primer método

```
insertarFinal(valor) {
const nuevo = new Nodo(valor);

// Caso lista vacía
if (!this.cabeza) {
this.cabeza = nuevo;
this.cola = nuevo;
return;
}

// Conectar el nuevo nodo al final
this.cola.siguiente = nuevo;
nuevo.anterior = this.cabeza;

// Actualizamos la cola
this.cola = nuevo;
}
```

Mi análisis

En la línea *nuevo.anterior = this.cabeza* es donde se encuentra el error. Es incorrecto porque debería hacer referencia al nodo que es la cola actual. Si se hace de esta manera se pierde la simetría bidireccional.

Código corregido

```
insertarFinal(valor) {
const nuevo = new Nodo(valor);

// Caso lista vacía
if (!this.cabeza) {
this.cabeza = nuevo;
this.cola = nuevo;
return;
}

// Conectar el nuevo nodo al final
this.cola.siguiente = nuevo;
nuevo.anterior = this.cola;

// Actualizamos la cola
this.cola = nuevo;
}
```

Segundo método

```
eliminarNodo(valor) {
if (!this.cabeza) return;

let actual = this.cabeza;

while (actual) {

if (actual.valor === valor) {

// Caso: nodo intermedio o final
if (actual.anterior) {
actual.anterior.siguiente = actual.siguiente;
}

if (actual.siguiente) {
actual.siguiente.anterior = actual.anterior;
}

// Si es la cabeza, la movemos
if (actual === this.cabeza) {
this.cabeza = actual.siguiente;
```

```
}

// Si es la cola, la movemos
if (actual === this.cola) {
this.cola = actual.anterior;
}

return;
}

actual = actual.siguiente;
}
}
```

Mi análisis

El código no toma en cuenta el caso en el que solo exista un nodo. Esto puede provocar que se rompa la simetría bidireccional cuando eso ocurra, aunque funcinará perfectamente mientras no lo haga.

Código corregido

```
eliminarNodo(valor) {
if (!this.cabeza) return;

let actual = this.cabeza;

while (actual) {

if (actual.valor === valor) {

// Caso: único nodo
if (this.cabeza === this.cola) {
this.cabeza = null;
this.cola = null;
return;
}

// Caso: eliminar cabeza
if (actual === this.cabeza) {
this.cabeza = actual.siguiente;
this.cabeza.anterior = null;
return;
}

// Caso: eliminar cola
if (actual === this.cola) {
this.cola = actual.anterior;
this.cola.siguiente = null;
```

```
return;
}

// Caso: nodo intermedio
actual.anterior.siguiente = actual.siguiente;
actual.siguiente.anterior = actual.anterior;

return;
}

actual = actual.siguiente;
}
}
```

Tercer método

```
insertarDespues(valorObjetivo, nuevoValor) {
if (!this.cabeza) return;

let actual = this.cabeza;

while (actual) {

if (actual.valor === valorObjetivo) {

const nuevo = new Nodo(nuevoValor);

// Conectamos el nuevo nodo
nuevo.siguiente = actual.siguiente;
nuevo.anterior = actual;

// Enlazamos el actual con el nuevo
actual.siguiente = nuevo;

// Si no es el último nodo, actualizamos la cola si es necesario
if (!nuevo.siguiente) {
this.cola = nuevo;
}

return;
}

actual = actual.siguiente;
}
}
```

Mi análisis

En caso de que haya un nodo siguiente, el puntero anterior de este no se actualiza, lo que provoca que se rompa la simetría bidireccional.

Código corregido

```
insertarDespues(valorObjetivo, nuevoValor) {
if (!this.cabeza) return;

let actual = this.cabeza;

while (actual) {

if (actual.valor === valorObjetivo) {

const nuevo = new Nodo(nuevoValor);

nuevo.siguiente = actual.siguiente;
nuevo.anterior = actual;

actual.siguiente = nuevo;

//      Corrección clave:
if (nuevo.siguiente) {
nuevo.siguiente.anterior = nuevo;
} else {
// Si se insertó al final
this.cola = nuevo;
}

return;
}

actual = actual.siguiente;
}
}
```

Checkpoint Metacognitivo – Fase VALIDA

¿Implementé la función verificarIntegridad() y la ejecuté después de cada operación? ¿Encontró algún error?

Sí la ejecuté, no encontró ningún error.

¿Pude detectar los errores en el código de la IA sin ayuda? ¿Qué estrategia usé?

No pude, me equivoqué al intentar deducir el error y la IA tuvo que explicarlo.

¿Escribí pruebas yo mismo o solo usé las que me dieron? ¿Puedo pensar en un caso de prueba que no está en la lista?

Solo usé las que me dieron.

Fase PROFUNDIZA

¿Puedo identificar al menos 3 aplicaciones reales donde una lista doble es mejor que una simple?

Para una playlist, para una cola de espera que pueda cambiar prioridades, para un videojuego en donde se necesite gestionar turnos.

¿Entiendo por qué la lista circular elimina los punteros null y qué riesgo introduce?

Sí, no hay punteros null porque el último elemento hace referencia al primero, por lo que se crea una lista cerrada. El riesgo que presenta esto es que debe declararse explícitamente cuando debe parar de recorrerse la lista.

¿Puedo explicar cómo mi ListaDoble se conecta con la implementación de Pilas de la próxima semana?

No sé muy bien aún lo que son las pilas, así que no.