Universidad Internacional del Ecuador

Nombre: Lander Gonzalez
Docente: Ing. Charlie Cárdenas
Materia: Estructura de datos
Parte 1

Caso borde	**¿Qué significa?**	**¿Por qué es crítico?**
Lista vacía	La lista no contiene nodos head == null, tail == null	Evita NullPointerException al eliminar, invertir o buscar
Insertar en lista vacio	Primer nodo insertado	Define correctamente head y tail
Un solo elemento	head == tail	Situación critica al borrar o invertir la lista
Eliminar head	Se elimina el primer nodo	Debe garantizar head.prev === null.
Eliminar tail	Se elimina el ultimo nodo	Debe garantizar tail.next === null
Eliminar único nodo	Lista queda vacia	Debe quedar head == null y tail == null
Eliminar nodo intermedio	Nodo entre dos nodos	Se debe volver a conectar prev.next y next.prev
Invertir lista vacia	reverse() sin elementos	Es necesario que se mantenga el estado sin errores

Parte 2

ID	Precondición	Acción	Resultado esperado	Postcondición
TC-DLL-001	Lista vacía	Crear lista	isEmpty = true	head == null, tail == nuil
TC-DLL-002	Lista vacía	deleteByValue (10)	Retorna false	Lista sigue vacia
TC-DLL-003	Lista vacía	reverse()	No cambia	Lista sigue vacia
TC-DLL-004	Lista vacía	insertAtEnd(10)	Lista = [10]	Head == tail, extremos nulos
TC-DLL-005	Lista vacía	Insertar 10,20,30	Lista = [10,20,30]	Head.prev == null, tail.next == null
TC-DLL-006	Lista = [10]	deleteByValue (10)	Lista vacia	Head == null, tail == null
TC-DLL-007	Lista = [10,20,30]	deleteByValue (10)	Lista = [20,30]	Head.prev === null
TC-DLL-008	Lista = [10,20,30]	deleteByValue (30)	Lista = [10,20]	tail.next == null
TC-DLL-009	Lista = [10,20,30]	deleteByValue (20)	Lista = [10,30]	10.next == 30 y 30.prev == 10
TC-DLL-0010	Lista = [10,20,30]	deleteByValue (99)	Retorna false	Lista no cambia
TC-DLL-0011	Lista = [10]	reverse()	Lista = [10]	Punteros correctos
TC-DLL-0012	Lista = [10,20,30, 40]	reverse()	Lista = [40,30,20,10]	Extremos correctos
TC-DLL-0013	Lista vacia	search(10)	Retorna -1	Lista no cambia
TC-DLL-0014	Lista = [10,20,30]	search(99)	Retorna -1	Lista no cambia

Parte 3

Se llevaron a cabo las pruebas en java dentro del archivo DoublyLinkedListTest.java por lo cual cada caso de prueba que se valida

- Estado lógico de la lista
- Integridad de los punteros prev y next
- Comportamiento correcto de delete, insert, reverse y search

Desde el archivo MainTest.java se ejecutan las pruebas, que producen la salida y un resumen final

Parte 4

Cobertura de las pruebas

Las pruebas abarcan las operaciones mas significativas que se pueden realizar con la lista doblemente enlazada:

- Inserción (insertAtEnd) en una lista vacia y en otra con muchos elementos
- Eliminacion (deleteByValue) en diversos caso: lista vacia, elemento único, head,tail, nodo intermedio y nodo que no existe
- Inversión (reverse) en lista que están vacias, que tienen un solo elemento o mas de uno
- Busqueda (search) en valores que no existen y en lista vacia

Así mismo se comprobó que los punteros prev y next sean coherentes en cada operación y que la lista conserve su integridad

Gaps identificados

- No se realizaron pruebas de rendimiento con listas muy extensas
- No se incorporaron pruebas para insertar en lugares concretos como insertAtPosition o insertAtStart.

Propuestas de mejora

- Incorporar pruebas que incluyan mas de 100 nodos con el fin de evaluar la eficiencia y prevenir ciclos
- Incluir ensayos de inserción en lugares determinados
- Aplicar ensayos automatizados con JUNIT para llevar a cabo pruebas de integración continua