Santiago de cali

Taller 4

Ingenieria en sistemas

Uniremington

Sebastián Casañas Salcedo 1005707698.

Iván González 1234193301.

Danna Michelle Caicedo 1086634456.

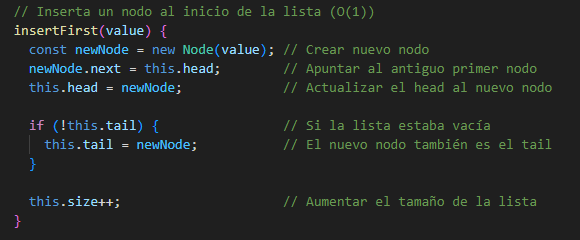
Angela María Medina Ruiz 1088034273

Profesor Jorge Gamarra

Analisis

1. Identifica cuáles operaciones son O(1) y cuáles son O(n).

R// O(1)



La operación insertFirst tiene complejidad O(1) porque solo realiza pasos fijos (crear nodo, ajustar punteros y actualizar tamaño) sin importar el tamaño de la lista. No necesita recorrer la lista, por eso su tiempo es constante.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

La operación insertLast tiene complejidad O(1) porque agrega el nuevo nodo directamente al final de la lista usando el puntero tail, sin necesidad de recorrer la lista.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

La función removeFirst tiene complejidad O(1) porque elimina el primer elemento en un número fijo de pasos: actualiza el puntero head al siguiente nodo, ajusta tail si la lista queda vacía y disminuye el tamaño. Estas operaciones no dependen de la longitud de la lista, por lo que el tiempo es constante.

O(N)

Texto

El contenido generado por IA puede ser incorrecto.

La operación insertAt tiene complejidad O(n) debido a que, en el peor caso, debe recorrer la lista desde el inicio hasta la posición indicada (índice) para insertar el nuevo nodo. Este recorrido hace que el tiempo de ejecución crezca linealmente con el tamaño de la listaTexto

El contenido generado por IA puede ser incorrecto.

La operación removeLast tiene complejidad O(n) porque para eliminar el último nodo debe recorrer la lista desde el inicio hasta el penúltimo nodo. Este recorrido hace que el tiempo de ejecución crezca linealmente con el tamaño de la lista.

Texto

El contenido generado por IA puede ser incorrecto.

La función removeAt tiene una complejidad O(n) porque necesita desplazarse a través de la lista hasta llegar al nodo en la posición indicada para poder eliminarlo.

1. **Compara los tiempos medidos y confirma si los resultados coinciden con la teoría**

Los resultados medidos muestran un incremento proporcional al tamaño de la lista, lo cual contradice la teoría y lo implementado en el codigo, donde ambas inserciones deberían ser operaciones de tiempo constante. Por lo tanto, los tiempos medidos NO coinciden con la teoría.

**Preguntas de reflexión**

**1. ¿Qué ventajas y desventajas tiene una lista simplemente enlazada frente a un arreglo tradicional en términos de rendimiento y uso de memoria?**

**R**// Las listas enlazadas ofrecen inserciones y eliminaciones rápidas y tamaño dinámico, pero tienen acceso lento y usan más memoria por los punteros. Los arreglos permiten acceso rápido y mejor uso de memoria, pero su tamaño es fijo y modificarlos es costoso en cuanto a sacrificar rendimiento.

1. **Si se quisiera implementar la eliminación de un nodo específico por valor y no por índice, ¿cómo afectaría eso la complejidad temporal?**

**R//** Eliminar un nodo por valor en una lista simplemente enlazada requiere buscarlo primero, lo que toma tiempo lineal O(n). Luego, la eliminación es O(1), pero en total la operación sigue siendo O(n) debido a la búsqueda.

1. **¿Cómo cambiaría la complejidad temporal si en lugar de agregar al final de la lista se insertara un nodo en una posición aleatoria en cada operación?**

**R//** Si en lugar de agregar un nodo al final de la lista se inserta en una posición aleatoria, la complejidad temporal cambia a O(n). Esto ocurre porque hay que recorrer la lista hasta esa posición para hacer la inserción, y recorrer la lista crece linealmente con el tamaño de la misma. En cambio, agregar al final usando un puntero tail es O(1) porque no requiere recorrido.