Informe del Trabajo Práctico N°1

Aplicaciones de Tipos Abstractos de Datos (TADs)

Materia: Algoritmos y Estructuras de Datos - FIUNER

Integrantes:

Flores, Valentina

Aguilar Gonzales, Andrea Fernanda

Lell, Camila Luciana

Fecha de entrega: 2 de mayo del 2025

ACTIVIDAD 2

Estructura de Datos Implementada

En el desarrollo de la segunda actividad, se implementó una estructura de datos denominada ListaDobleEnlazada. Esta estructura se compone de nodos que mantienen referencias tanto al nodo anterior como al siguiente, lo que permite insertar y eliminar elementos de manera eficiente en ambos extremos.

Se cumplieron todas las restricciones impuestas por el enunciado, tales como evitar el uso de listas auxiliares de Python para copiar, invertir o extraer elementos, y garantizar que los métodos desarrollados presenten un comportamiento eficiente en tiempo y uso de memoria.

Métodos Analizados

esta_vacia(): Retorna True si la lista no contiene elementos.

- *Complejidad:* O(1)

len(): Devuelve la cantidad de elementos presentes en la lista. Se implementó con un contador que se actualiza en cada inserción o extracción, evitando recorrer toda la estructura.

- *Complejidad:* O(1)

agregar_al_inicio(item) y **agregar_al_final(item):** Permiten añadir un elemento al comienzo o al final de la lista, respectivamente, actualizando correctamente las referencias de los nodos.

- *Complejidad:* O(1)

insertar(item, posicion): Inserta un elemento en una posición específica, o al final si no se indica la posición.

- *Complejidad:* O(1) para extremos, O(n) en posiciones intermedias.

extraer(posicion): Elimina y retorna el elemento en una posición dada, o el último si no se especifica ninguna.

- *Complejidad:* O(1) en extremos, O(n) en posiciones internas.

copiar(): Crea una nueva instancia de ListaDobleEnlazada, copiando cada nodo uno a uno sin utilizar estructuras auxiliares.

- *Complejidad:* O(n)

invertir(): Reorganiza los nodos de la lista en orden inverso, invirtiendo las referencias de cada uno.

- *Complejidad:* O(n)

concatenar(otra_lista): Añade los elementos de otra lista al final de la lista actual, copiándolos nodo por nodo.

- *Complejidad:* O(m), siendo *m* la longitud de la lista que se concatena.

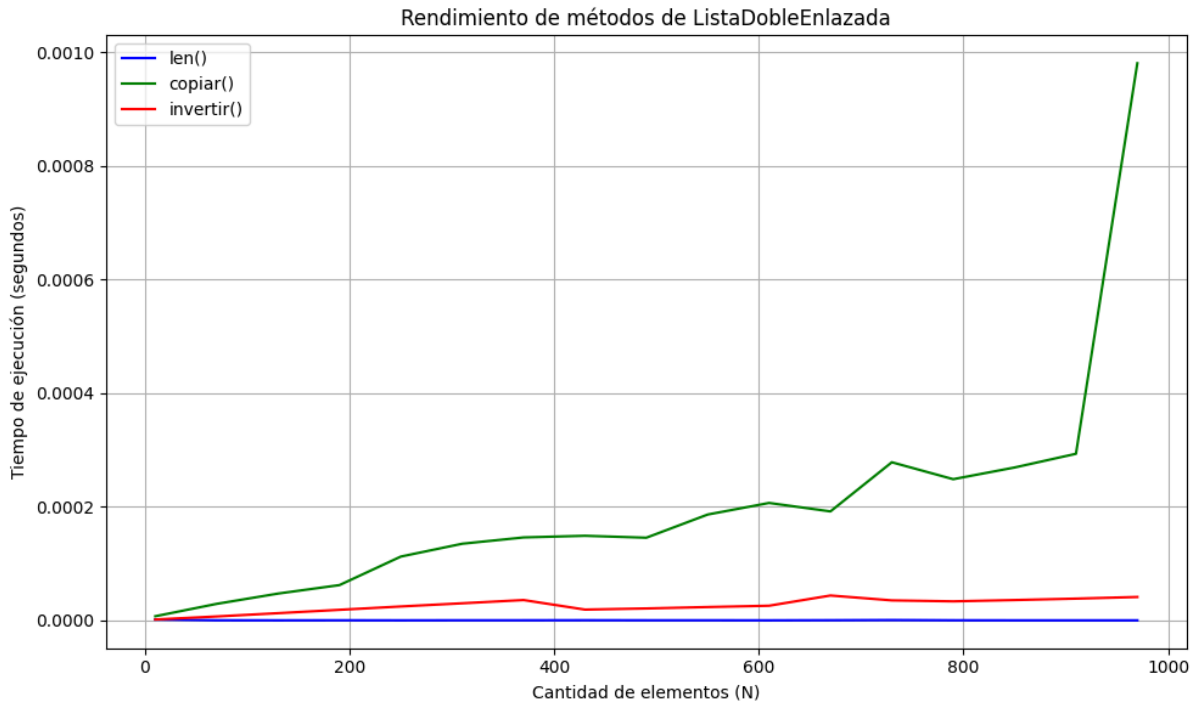
add(otra_lista): Sobrecarga del operador + para combinar dos listas doblemente enlazadas. Utiliza el método concatenar() internamente.

- *Complejidad:* O(n + m)

Resultados Experimentales

Se realizaron mediciones de tiempo para los métodos __len__, copiar() e invertir() utilizando listas de diferentes tamaños, alcanzando hasta 1000 elementos. Estas mediciones permitieron validar el comportamiento esperado de cada método en relación con su complejidad teórica.

Gráfica: Tiempo de ejecución de los métodos len, copiar e invertir según el tamaño de la lista.



Observaciones

- **len():** El tiempo de ejecución se mantuvo constante sin importar la longitud de la lista, lo que es consistente con su complejidad O(1).
- **copiar():** El tiempo de ejecución aumentó proporcionalmente al tamaño de la lista, reflejando una complejidad O(n).
- **invertir():** Mostró un crecimiento similar al de copiar(), validando también su comportamiento lineal. Las leves variaciones observadas podrían deberse a factores del entorno de ejecución como el rendimiento del procesador.

Conclusiones

- La estructura ListaDobleEnlazada desarrollada cumple con los requisitos establecidos en cuanto a eficiencia computacional y uso adecuado de la memoria.
- Los métodos __len__, copiar() e invertir() presentan las complejidades esperadas:

    ○ __len__(): O(1)

    ○ copiar() e invertir(): O(n)

- No se hizo uso de estructuras auxiliares innecesarias, tal como lo indicaba el enunciado.
- La implementación superó correctamente las pruebas unitarias proporcionadas por la cátedra, lo cual respalda su correcto funcionamiento.