BIOINGENIERÍA

Algoritmos y Estructuras de Datos

Trabajo práctico N°1:
Aplicaciones de TADs: Problema 1

Autora:

Dinamarca Daiana

24/10/2025

Problema 1

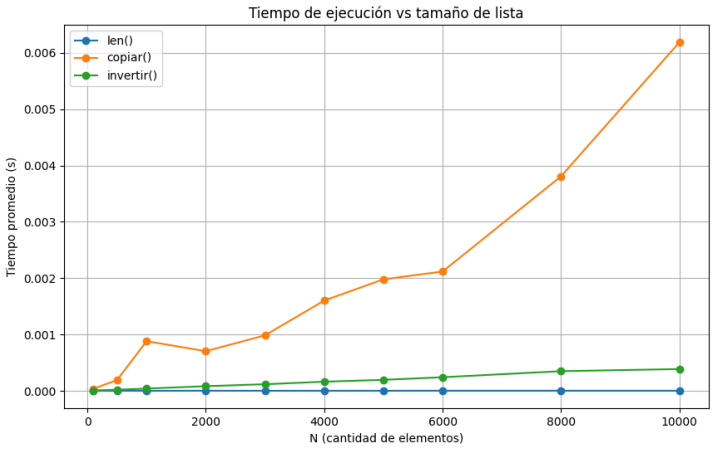
El objetivo de este ejercicio es implementar una **Lista Doblemente Enlazada** y sus operaciones básicas de manera eficiente, evitando el uso de almacenamiento adicional innecesario o estructuras de Python que dupliquen los datos. Se desarrollaron los métodos principales del TAD, incluyendo agregar al inicio, agregar al final, insertar en posición, extraer, copiar, invertir, concatenar y la sobrecarga de operadores como + y len().

La lista se implementó utilizando las clases *Nodo* y *ListaDobleEnlazada*. Cada nodo contiene un valor y punteros a los nodos siguiente y anterior. La lista mantiene referencias a la cabeza y a la cola, así como un atributo *tamanio* que permite acceder al tamaño de manera constante. La implementación evita el uso de listas de Python para operaciones internas y realiza las manipulaciones directamente sobre los nodos.

El método copiar() recorre cada nodo para crear uno nuevo con el mismo valor, asegurando que la operación se realice con complejidad lineal respecto al tamaño de la lista. El método invertir() intercambia los punteros siguiente y anterior de cada nodo recorriendo la lista una sola vez, manteniendo la integridad de la estructura. Los métodos concatenar() y __add__() ajustan cuidadosamente los punteros de cabeza y cola para garantizar que la lista resultante tenga la cabeza con el puntero anterior en None y se mantenga la correcta conexión de los nodos.

Se llevaron a cabo pruebas unitarias de todos los métodos implementados, verificando que las operaciones básicas funcionen correctamente y que los punteros siguiente y anterior se mantengan consistentes. También se comprobó que los operadores + y len() devuelvan los resultados esperados. Por ejemplo, len() devuelve el tamaño correcto de la lista, concatenar() mantiene la cabeza con anterior = None, y invertir() retorna la lista con los nodos en orden inverso, cumpliendo con la integridad estructural de la lista doblemente enlazada.

Análisis de tiempos de ejecución: Se midió el tiempo de ejecución de los métodos len(), copiar() e invertir() en listas de distintos tamaños. La gráfica de N (cantidad de elementos) versus tiempo de ejecución muestra que len() se mantiene constante para cualquier tamaño de lista, confirmando que su complejidad es aproximadamente O(1). Por su parte, invertir() crece de forma lineal con N, lo que indica una complejidad O(N). En cuanto al método copiar(), el tiempo de ejecución aumenta con N, pero presenta algunas irregularidades en ciertos rangos, especialmente alrededor de N=5000. Este comportamiento no altera la complejidad teórica lineal O(N) del método y se explica por la gestión de memoria interna de Python y la recolección de basura, que puede afectar el tiempo de ejecución en rangos específicos. Se presenta a continuación la gráfica (disponible en la carpeta *data*):



El código utilizado para medir los tiempos de ejecución y generar las gráficas se encuentra separado en una aplicación principal, independiente del módulo que implementa la lista doblemente enlazada. Esta estructura permite mantener una organización modular del proyecto, donde la lógica del TAD y la ejecución de experimentos permanecen claramente diferenciadas.

Conclusión

Los resultados coinciden con la complejidad teórica esperada para cada método, confirmando la eficiencia de la implementación. El comportamiento constante de len() y el crecimiento lineal de copiar() e invertir() validan que los algoritmos fueron diseñados correctamente. Las pequeñas variaciones observadas se atribuyen a factores externos de ejecución y no a deficiencias del código. En conjunto, la lista doblemente enlazada cumple con los criterios de eficiencia y uso adecuado de memoria establecidos en la consigna.