Programación II – ALGORTIRMOS Y ESTRUCTURAS DE DATOS II

**Docente:** Esp. Lic. PEREZ, Nicolás Ignacio

[nicoperez@uade.edu.ar](mailto:nicoperez@uade.edu.ar)

Repositorio de la materia: <https://github.com/NicolasPerezUNLaSMN/PROG_II_UADE_JAVA>

Facultad de Ingeniería y Ciencias Exactas

Universidad Argentina de la Empresa

Logotipo, nombre de la empresa

El contenido generado por IA puede ser incorrecto.

Resumen de Implementaciones de Estructuras de Datos y su Eficiencia

Este documento ofrece un resumen detallado sobre el análisis de la eficiencia de diversas estructuras de datos implementadas en Java, tanto estáticas como dinámicas. Las estructuras analizadas son listas, pilas, colas, diccionarios, con especial enfoque en el rendimiento tanto en términos de tiempo (complejidad algorítmica) como de uso de memoria.

# 1. Listas

Se implementaron dos tipos de listas: una lista estática, basada en un array de tamaño fijo, y una lista dinámica, que utiliza nodos enlazados. A continuación, se presenta un análisis de su eficiencia.

## Eficiencia de las Listas

**1. \*\*Lista Estática\*\* (con array):**

- \*\*Inserción\*\*: O(1) si se inserta al final (sin redimensionar el array).

- \*\*Búsqueda\*\*: O(n), debido a la necesidad de recorrer el array.

- \*\*Eliminación\*\*: O(n), al mover los elementos después de eliminar.

- \*\*Memoria\*\*: La lista estática reserva memoria fija, lo que significa que su uso de memoria es muy eficiente, aunque no puede crecer más allá de su capacidad inicial.

2. \*\*Lista Dinámica\*\* (con nodos enlazados):

- \*\*Inserción\*\*: O(1) al insertar en el inicio o final de la lista.

- \*\*Búsqueda\*\*: O(n), debido a la necesidad de recorrer la lista enlazada.

- \*\*Eliminación\*\*: O(n) en el peor caso, ya que es necesario recorrer la lista para encontrar el nodo a eliminar.

- \*\*Memoria\*\*: La lista dinámica utiliza menos memoriam pero dato a dato ocupa más que el array debido al overhead de los objetos `Nodo` (cada uno tiene un puntero y otros datos).

# Pilas

Se implementaron pilas tanto estáticas como dinámicas. Las pilas están basadas en las mismas ideas de implementación que las listas, pero con un enfoque LIFO (Last In, First Out).

## 2.1. Eficiencia de las Pilas

1. \*\*Pila Estática\*\* (con array):

- \*\*Apilar\*\*: O(1), ya que el elemento se añade al final del array.

- \*\*Desapilar\*\*: O(1), ya que se elimina el último elemento del array.

- \*\*Memoria\*\*: Similar a la lista estática, la pila estática tiene un uso de memoria eficiente pero limitado por la capacidad inicial del array.

2. \*\*Pila Dinámica\*\* (con nodos enlazados):

- \*\*Apilar\*\*: O(1), añadiendo el nuevo nodo al principio de la pila.

- \*\*Desapilar\*\*: O(1), eliminando el primer nodo.

- \*\*Memoria\*\*: Aunque la pila dinámica es flexible, usa más memoria debido a los objetos `Nodo`, lo que incrementa el overhead.

# Colas

La implementación de colas también se realizó con versiones estáticas y dinámicas. Las colas siguen el principio FIFO (First In, First Out).

## 3.1. Eficiencia de las Colas

1. \*\*Cola Estática\*\* (con array):

- \*\*Encolar\*\*: O(1), añadiendo al final del array.

- \*\*Desencolar\*\*: O(1), eliminando el primer elemento del array.

- \*\*Memoria\*\*: La cola estática tiene una eficiencia en memoria muy alta, ya que la memoria se asigna de forma continua.

2. \*\*Cola Dinámica\*\* (con nodos enlazados):

- \*\*Encolar\*\*: O(1), añadiendo un nodo al final de la cola.

- \*\*Desencolar\*\*: O(1), eliminando el primer nodo.

- \*\*Memoria\*\*: La cola dinámica usa más memoria debido al almacenamiento de punteros en cada nodo, lo que incrementa el uso de memoria.

# 4. Diccionario Dinámico

El diccionario dinámico fue implementado utilizando una lista enlazada para almacenar pares clave-valor, y permite operaciones como insertar, obtener, eliminar y verificar la existencia de una clave.

## 4.1. Eficiencia del Diccionario Dinámico

1. \*\*Diccionario Dinámico\*\* (con lista enlazada):

- \*\*Insertar\*\*: O(n), ya que se busca la clave antes de insertar o actualizar el valor.

- \*\*Obtener\*\*: O(n), debido a la necesidad de recorrer la lista para encontrar la clave.

- \*\*Eliminar\*\*: O(n), debido a que se recorre la lista para eliminar el nodo correspondiente.

- \*\*Memoria\*\*: Cada par clave-valor se almacena como un objeto `Nodo`, lo que implica un mayor uso de memoria en comparación con una solución basada en arrays.

# 5. Conclusión

En general, las estructuras de datos dinámicas ofrecen más flexibilidad pero con un coste mayor en términos de algoritmos, major respuesta y major rendimiento (salvo cuando el tamaño estatico es el correcto). Las implementaciones estáticas son más eficientes en cuanto a memoria, pero con capacidades fijas, lo que las hace menos flexibles.

# 6. Resumen

| **Operación** | **Estática (Array)** | **Dinámica (Nodos)** | **Observación** |
| --- | --- | --- | --- |
| Insertar al inicio | O(n) | O(1) | Dinámica es mejor |
| Insertar al final | O(1) | O(1)\* | Ambas eficientes si dinámica guarda puntero al final |
| Insertar en posición | O(n) | O(n) | Iguales |
| Eliminar al inicio | O(n) | O(1) | Dinámica es mejor |
| Eliminar al final | O(1) | O(n) | Estática es mejor |
| Eliminar en posición | O(n) | O(n) | Iguales |
| Buscar por valor | O(n) | O(n) | Iguales |
| Buscar (binaria) | O(log n) | O(n log n) | Estática gana ampliamente, dinámica es ineficiente, aunque posible |
| Acceder por posición | O(1) | O(n) | Estática es mucho mejor |
| Ordenar (burbuja) | O(n²) | O(n²) | Iguales |
| Memoria por elemento | 4 bytes | ~28 bytes | Estática mucho más eficiente en espacio |
| Crecimiento dinámico | No | Sí | Dinámica permite crecer sin redimensionar |