Universidad Americana UAM

Fundamentos de Robótica, grupo 7

Implementación y Análisis de Algoritmos de Búsqueda en Sistemas de Gestión Bibliográfica

Estudiante

Roberto Fabio Tercero Membreño (24014121)

Docente

MSc. José Durán García

23 de noviembre de 2025

Índice

1. Introducción

En la era actual de la información, la gestión eficiente de datos es uno de los pilares fundamentales de la ingeniería de software. A medida que los volúmenes de datos crecen, la capacidad de recuperar información específica de manera rápida y precisa se convierte en un desafío técnico crítico. Los algoritmos de búsqueda representan las herramientas matemáticas y lógicas diseñadas para resolver este problema, permitiendo la navegación a través de estructuras de datos para localizar elementos que cumplan con criterios predefinidos.

El presente informe documenta el desarrollo de un módulo de software denominado "BusquedaLibros", construido en el lenguaje de programación C#. Este proyecto tiene como objetivo implementar y comparar diferentes estrategias de recuperación de información aplicadas a un contexto de gestión bibliográfica. Específicamente, se abordan cuatro problemas clásicos de búsqueda: la búsqueda lineal en estructuras no ordenadas, la búsqueda binaria en listas ordenadas, la identificación de valores extremos (máximos y mínimos) y la búsqueda de patrones de texto dentro de cadenas de caracteres.

A través de este desarrollo, se busca no solo demostrar la funcionalidad técnica de los algoritmos, sino también comprender sus implicaciones en términos de eficiencia computacional y sus requisitos previos, como la necesidad de ordenamiento de datos para estrategias logarítmicas.

2. Marco Teórico

Los algoritmos de búsqueda son procedimientos paso a paso utilizados para localizar datos específicos dentro de una colección. La eficiencia de estos algoritmos se

mide generalmente utilizando la notación Big O (O), que describe cómo el tiempo de ejecución o el espacio requerido crece en relación con el tamaño de la entrada (n).

2.1 Búsqueda Lineal

La búsqueda lineal o secuencial es el método más simple de recuperación de información. Consiste en recorrer una estructura de datos, elemento por elemento, desde el inicio hasta el final, comparando cada ítem con el valor objetivo (Cormen et al., 2009).

- **Complejidad:** En el peor de los casos, el elemento está al final o no existe, resultando en una complejidad de tiempo de O(n).
- **Aplicabilidad:** Es ideal para listas pequeñas o datos no ordenados donde el costo de ordenar superaría el beneficio de una búsqueda más avanzada.

2.2 Búsqueda Binaria

La búsqueda binaria es un algoritmo eficiente basado en el paradigma de "divide y vencerás". Requiere que la colección de datos esté previamente ordenada. El algoritmo compara el valor objetivo con el elemento central de la colección; si no son iguales, determina en qué mitad (superior o inferior) debe continuar la búsqueda, descartando la otra mitad (Knuth, 1998).

Complejidad: Dado que el espacio de búsqueda se reduce a la mitad en cada iteración, su complejidad es O(\log n), lo que lo hace significativamente más rápido que la búsqueda lineal para grandes volúmenes de datos.

2.3 Búsqueda de Extremos (Min/Max)

La identificación de valores mínimos y máximos en una colección no ordenada requiere, inevitablemente, examinar cada elemento al menos una vez para garantizar que no se omita un valor extremo. Por lo tanto, este proceso tiene una cota inferior de complejidad de \Omega(n) (Sedgewick & Wayne, 2011).

2.4 Búsqueda de Patrones en Cadenas (String Matching)

La búsqueda de patrones consiste en encontrar la ocurrencia de una subcadena (patrón) dentro de una cadena principal (texto). El enfoque "ingenuo" o de fuerza bruta alinea el patrón con el texto y compara los caracteres uno a uno. Si hay una discrepancia, el patrón se desplaza una posición a la derecha y se reinicia la comparación (Cormen et al., 2009). Aunque existen algoritmos más eficientes (como Knuth-Morris-Pratt), el enfoque ingenuo es fundamental para comprender la manipulación de arreglos y caracteres.

3. Explicación de Algoritmos Implementados

El sistema fue desarrollado en C# utilizando el paradigma de Programación Orientada a Objetos (POO), separando la lógica de acceso a datos (DAO) de la lógica de negocio y la interfaz de usuario. A continuación, se detalla la implementación técnica de cada requerimiento.

3.1 Búsqueda Lineal en Lista de Libros

Para localizar un libro por su título, se implementó una búsqueda lineal. Debido a que los libros se agregan a la lista en el orden de llegada (cronológico de inserción) y no alfabético, no se podía garantizar el orden necesario para algoritmos más complejos.

- **Implementación:** Se itera a través de la lista LibroDAO.Libros. En cada ciclo, se compara el título ingresado por el usuario con la propiedad Nombre del objeto Libro.
- **Resultado:** El algoritmo devuelve un valor booleano indicando el éxito y, en caso afirmativo, imprime la posición (índice) donde fue hallado.

3.2 Búsqueda Binaria en Lista de Autores

Para la búsqueda de autores, se optó por el algoritmo de búsqueda binaria.

- **Precondición:** Dado que la lista de autores en AutorDAO es dinámica y podría estar desordenada, se implementó un paso previo de ordenamiento utilizando el método .Sort() de las listas genéricas de C#, aplicando un comparador alfabético (StringComparison.OrdinalIgnoreCase).
- **Lógica:** Se definieron punteros inicio y fin. Dentro de un bucle while, se calcula el índice medio. Si el nombre del autor en el medio coincide con el buscado, se termina el proceso. Si el nombre buscado es "menor" alfabéticamente, se ajusta el fin a medio - 1; de lo contrario, se ajusta el inicio a medio + 1.

3.3 Búsqueda del Libro Más Reciente y Más Antiguo

Este algoritmo se implementó mediante un único recorrido lineal (O(n)) sobre la lista de libros para optimizar el rendimiento.

- **Lógica:** Se inicializan dos variables de referencia, masReciente y masAntiguo, asignándoles el primer libro de la lista. Se recorre el resto de la

colección comparando la propiedad FechaPublicacion. Si la fecha del libro actual es mayor que la de masReciente, se actualiza la referencia; análogamente se procede con masAntiguo. Esto permite obtener ambos resultados en una sola pasada.

3.4 Búsqueda de Coincidencias en Descripciones

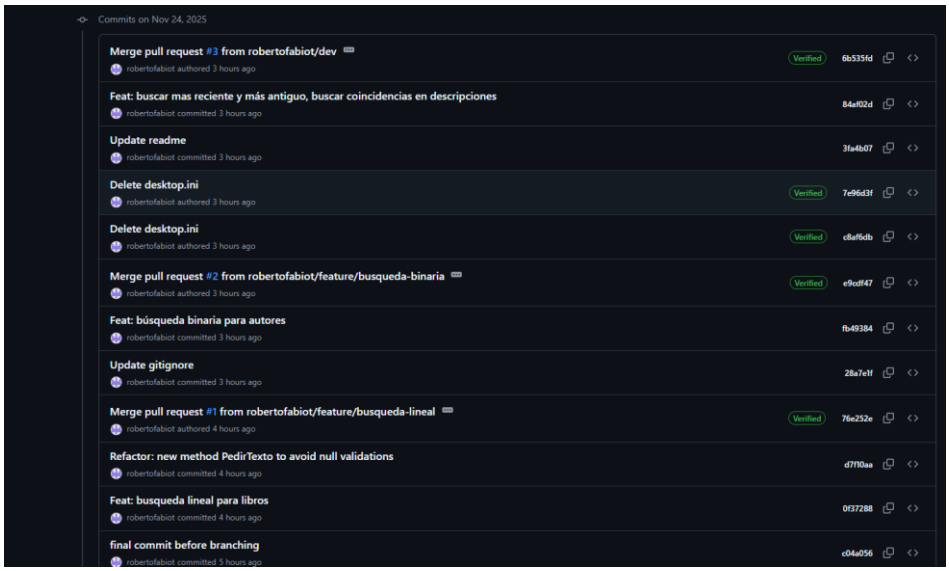
Se desarrolló un algoritmo de búsqueda de texto manual (fuerza bruta) para encontrar coincidencias parciales dentro de las descripciones de los libros, sin depender de métodos preconstruidos como .Contains().

- **Lógica:** Se implementó la función auxiliar ContieneManual. Esta utiliza dos bucles anidados. El bucle externo recorre la descripción del libro (texto fuente), y el bucle interno recorre la palabra buscada (patrón). Se comparan los caracteres en las posiciones correspondientes ($i + j$ vs j). Si todos los caracteres del patrón coinciden, se considera encontrado.
- **Sensibilidad:** Se aplicó una conversión a minúsculas (char.ToLower) en tiempo real durante la comparación para hacer la búsqueda insensible a mayúsculas y minúsculas (case-insensitive).

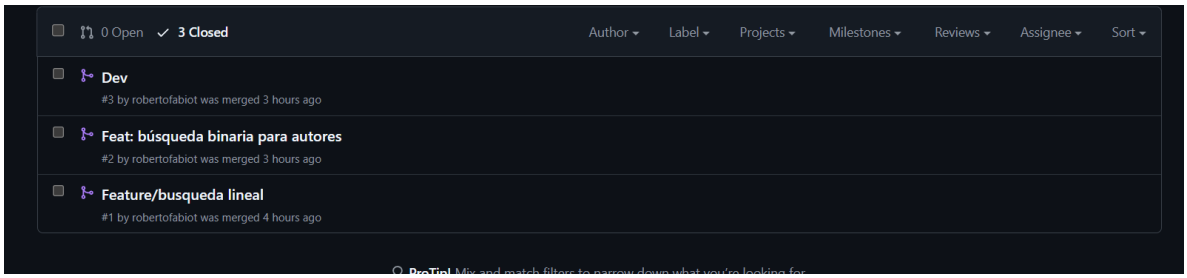
4. Evidencia del Repositorio

A continuación se presentan las evidencias del trabajo colaborativo y la gestión de versiones en GitHub.

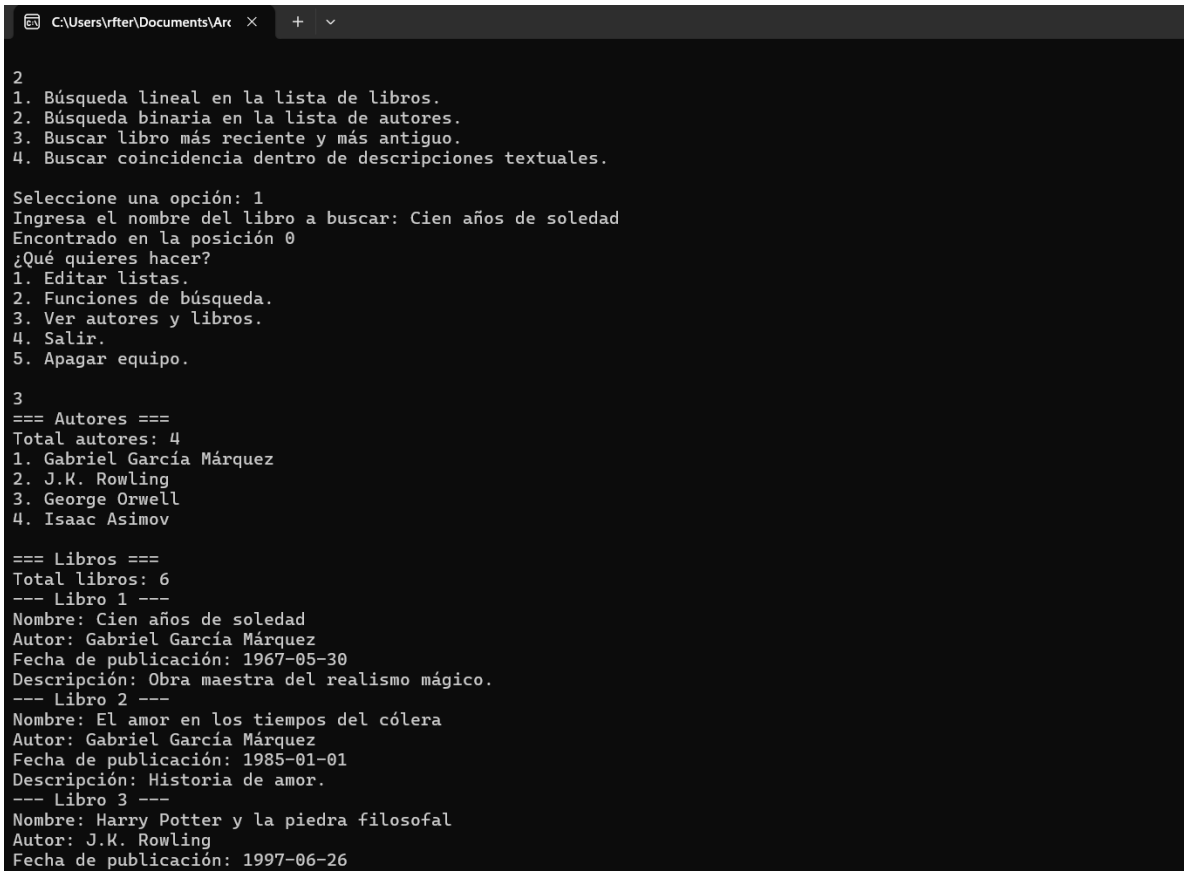
4.1 Historial de Commits y Ramas



4.2 Pull Requests Revisados



4.3 Ejecución del Programa (Consola)



5. Conclusiones

La implementación de este módulo de búsqueda ha permitido contrastar en la práctica las diferencias teóricas entre los distintos algoritmos de recuperación de datos.

1. **Eficiencia vs. Preprocesamiento:** Se evidenció que, si bien la **búsqueda binaria** es notablemente superior en eficiencia para la recuperación de datos, impone un costo operativo adicional: el ordenamiento previo de la lista. Esto la hace ideal para conjuntos de datos estáticos o donde las lecturas superan masivamente a las escrituras. Por el contrario, la **búsqueda lineal**

demostró ser más flexible para datos dinámicos como la lista de libros, donde el orden no está garantizado.

2. **Manipulación de Datos a Bajo Nivel:** La implementación manual de la búsqueda de cadenas (string matching) reforzó la comprensión sobre cómo los datos textuales son, en esencia, arreglos de caracteres, y cómo la complejidad computacional aumenta drásticamente (a $O(n \times m)$) cuando se realizan comparaciones anidadas.
3. **Trabajo Colaborativo:** El uso de una estrategia de ramas (*Gitflow* simplificado) facilitó el desarrollo paralelo de las funcionalidades lineales y binarias, evitando conflictos de código y permitiendo una integración limpia mediante Pull Requests.

En conclusión, la elección del algoritmo adecuado no depende solo de la velocidad teórica, sino de la naturaleza de los datos (ordenados vs. desordenados) y de la frecuencia con la que estos se modifican.

6. Referencias

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3ra ed.). MIT Press.

Knuth, D. E. (1998). *The art of computer programming: Volume 3: Sorting and searching* (2da ed.). Addison-Wesley Professional.

Microsoft. (2023). *Documentación de C# - List<T>.Sort Método*. Microsoft Learn. https://learn.microsoft.com/es-es/dotnet/api/system.collections.generic.list-1.sort

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4ta ed.). Addison-Wesley Professional.