Integrador Programación I: Algoritmos de busqueda y ordenamiento

Integrante: Prados Gonzalo

Materia: Programación I

Docente: Nicolás Quirós

Fecha de entrega: 09/06/2025

Introducción

Este trabajo pretende ser un primer acercamiento a los algoritmos de búsqueda y ordenamiento: cómo funcionan, cuáles son los más efectivos y en qué momentos conviene usar uno u otro.

Cuando hablamos de algoritmos de búsqueda, hablamos de una serie de instrucciones detalladas para poder retirar o adquirir cierta información dentro de una estructura de datos. Los algoritmos de ordenamiento, por otro lado, son una serie de instrucciones para imponer orden dentro de una serie, grupo o conjunto. Una búsqueda efectiva implica mejores tiempos de respuesta y una mejor optimizacion del tiempo de cómputo.

En este trabajo nos vamos a enfocar en las dos formas más primitivas de estos algoritmos: la búsqueda lineal y la búsqueda binaria. Hablo de formas primitivas porque claramente hay formas más avanzadas y complejas, pero estas sientan las bases conceptuales y fundamentos de esas otras formas.

Marco Teórico

Búsqueda Lineal

Los algoritmos de búsqueda lineal, también conocidos como búsqueda secuencial, implican recorrer una lista de elementos uno por uno hasta encontrar un elemento específico. Este algoritmo es muy sencillo de implementar en código y extremadamente intuitivo, pero puede ser muy ineficiente dependiendo del largo de la lista y la ubicación donde se encuentra el elemento.

Las ventajas principales de este tipo de búsqueda son su sencillez y su flexibilidad. Por un lado, la búsqueda lineal es uno de los algoritmos más simples y fáciles de implementar y solo requiere iterar a través de la lista de elementos uno por uno hasta encontrar el objetivo. También puede aplicarse a cualquier tipo de lista, independientemente de si está ordenada o no.

Sus desventajas radican en su ineficiencia al manejar grandes volúmenes de datos y en que no es el método óptimo para listas ordenadas. La búsqueda lineal es

extremadamente ineficiente en listas grandes, ya que compara cada elemento uno por uno, lo cual implica que su tiempo de ejecución crezca de manera lineal con el tamaño de la lista. La otra dificultad es que aunque puede funcionar en listas no ordenadas, no es particularmente eficiente para listas ordenadas, en esos casos es muchas veces preferible la busqueda binaria.

Búsqueda Binaria

El algoritmo de búsqueda binaria es un algoritmo muy eficiente que se aplica solo a listas ordenadas. Funciona dividiendo repetidamente la lista en dos mitades y comparando el elemento objetivo con el elemento del medio. Esto reduce dramáticamente la cantidad de comparaciones necesarias.

Las dos ventajas principales de este método son su eficiencia y su reducido número de comparaciones. Su tiempo de ejecución es de O(log n), lo que significa que disminuye rápidamente a medida que el tamaño de la lista aumenta. Comparado con la búsqueda lineal, la búsqueda binaria realiza menos comparaciones en promedio, lo que la hace más rápida para encontrar el objetivo.

Su desventaja principal es que requiere una lista ordenada y en segundo lugar que acarrea una mayor complejidad a la hora de la implementación. Si la lista no está ordenada, se debe realizar una operación adicional para ordenarla antes de usar este método, lo cual implica un costoa dicional de cómputo. Comparado con la búsqueda lineal, la búsqueda binaria también es más compleja de implementar debido a su naturaleza recursiva.

Caso Práctico

ESCENARIO 1:

Para este caso de estudio vamos a hacer la comparativa entre ambos algoritmos utilizando una lista ordenada con cien mil elementos. También estoy utilizando el módulo `time` para poder medir cuánto tiempo tarda cada llamada.

Búsqueda Lineal

Acá presentamos un algoritmo de búsqueda lineal extremadamente simple en Python (también disponible en el repositorio bajo el nombre `algoritmosBusqueda.py`).
Nuestro código se limita a buscar un número en una lista y nos devuelve la posición en la que se encuentra el número. Ya sea que se encuentre o no el número deseado, nos devuelve un mensaje con el resultado de la búsqueda. Comencé por definir la función principal, el código lo que hace es ir elemento por elemento de la lista hasta que encuentra el que estamos buscando. Para este primer escenario vamos a buscar el número 4900, que se encuentra en una lista con cien mil elementos.

```
def busquedaLineal(lista, obj):
```

```
    for i in range(len(lista)):
        if lista[i] == obj:
            return i
    return -1
```

Resultado Búsqueda Lineal:

```
[Búsqueda Lineal ] El número 4900 se encuentra en la posición real: 4900, Tiempo: 0.61962 ms
```

Búsqueda Binaria

Este es nuestro algoritmo de búsqueda binaria (también disponible en el repositorio bajo el nombre `algoritmosBusqueda.py`).
En este código implementamos la búsqueda binaria de forma recursiva. Primero ordenamos la lista luego vamos dividiendo el rango de búsqueda a la mitad comparando el valor del medio con el número buscado. Esto permite encontrar el elemento rápidamente en listas ordenadas, con un menor número de comparaciones que la búsqueda lineal.

```
def busquedaBinaria(lista, objetivo, inicio, fin):
    if inicio > fin:
        return -1

    centro = (inicio + fin) // 2
    if lista[centro] == objetivo:
        return centro
    elif lista[centro] < objetivo:
        return busquedaBinaria(lista, objetivo, centro + 1, fin)
    else:
        return busquedaBinaria(lista, objetivo, inicio, centro - 1)
```

Resultado Búsqueda Binaria:

```
[Búsqueda Binaria] El número 4900 se encuentra en la posición real: 4900, Tiempo: 0.02377 ms
```

ESCENARIO 2:

El segundo escenario voy a plantearlo con el número 13, el cual se encuentra casi al comienzo de la lista y por una cuestión de simpleza, solamente voy a copiar los resultados.

Resultado Búsqueda Lineal:

```
[Búsqueda Lineal ] El número 13 se encuentra en la posición real: 13, Tiempo: 0.00678 ms
```

Resultado Búsqueda Binaria:

```
[Búsqueda Binaria] El número 13 se encuentra en la posición real: 13, Tiempo: 0.02514 ms
```

ESCENARIO 3:

El tercer escenario voy a plantearlo con el número 100000, el cual se encuentra al final de la lista.

Resultado Búsqueda Lineal:

```
[Búsqueda Lineal ] El número 100000 se encuentra en la posición real: 100000, Tiempo: 14.45768 ms
```

Resultado Búsqueda Binaria:

```
[Búsqueda Binaria] El número 100000 se encuentra en la posición real: 100000, Tiempo: 0.06502 ms
```

Metodología Utilizada

La metodología incluye una investigación exhaustiva en diversas fuentes, presentes en la seccion de Bibliografía. Luego procedí a generar algoritmos de prueba para verificar de forma empírica los tiempos de cómputo y compararlos.
Como herramienta de trabajo estoy utilizando Visual Studio Code como IDE, GitHub Copilot para ayudarme a corregir mi código y GitHub como plataforma en la que voy a publicar mi trabajo. Gamma me asistió a la hora de crear la presentación. Incluyo el módulo `time` para poder medir los tiempos de ejecución y el método `sort` para ordernar mi lista.

Resultados Obtenidos

A primera vista salta el hecho de la longitud del código. Mientras que nuestro algoritmo de búsqueda lineal tiene 5 líneas, el de búsqueda binaria tiene 11 más del doble de código. Respecto de que la búsqueda lineal es más simple, no hay duda alguna.

Los resultados obtenidos en los distintos escenarios muestran cómo se comportan ambos algoritmos en la práctica, según la posición del número que se desea encontrar:

En el escenario 1, donde el número 4900 está en una posición avanzada de la lista (posición 4900 de 100000), la búsqueda binaria fue mucho más rápida que la búsqueda lineal.

Búsqueda Lineal: 0.61962 ms

Búsqueda Binaria: 0.02377 ms

Esto demuestra la ventaja de la búsqueda binaria cuando trabajamos con listas ordenadas muy grandes

En el escenario 2, el número 13 está casi al principio de la lista. En este caso, la búsqueda lineal fue más eficiente porque encontró el número en los primeros pasos, mientras que la búsqueda binaria necesitó hacer varias divisiones antes de llegar a él.

Búsqueda Lineal: 0.00678 ms

Búsqueda Binaria: 0.02514 ms

Este resultado confirma que, cuando el dato buscado está al principio la búsqueda lineal puede ser más rápida.

En el escenario 3, el número 100000 está al final de la lista. Aquí se observa claramente la gran desventaja de la búsqueda lineal, que tuvo que recorrer casi toda la lista hasta encontrar el número, mientras que la búsqueda binaria lo localizó rápidamente.

Búsqueda Lineal: 14.45768 ms

Búsqueda Binaria: 0.06502 ms

En este caso, la diferencia de rendimiento es drástica, con la búsqueda binaria siendo más de 200 veces más rápida.

Conclusión Final

La busqueda lineal y la busqueda binaria son elementos extremadamente básicos pero no dejan de ser utiles en ciertos contextos. Si estamos hablando de listas de pocos elementos desordenados la busqueda lineal es una buena opcion de implementar, sin embargo su rendimiento decrece de forma drástica a medida que los elementos de la lista van creciendo o el elemento en cuestion se encuentra al final de la lista. La búsqueda binaria demuestra ser mucho más eficiente en listas ordenadas, reduciendo el tiempo de búsqueda gracias a la estrategia de divide y conquistarás. No obstante, requiere que la lista esté ordenada previamente, lo cual puede implicar un costo computacional extra.

Los resultados de mi investigación arrojan que la búsqueda lineal es una buena opción para volumenes pequeños de datos pero que la busqueda binaria es la más eficiente a medida que se van agregando elementos en la lista. Comparativamente hablando ambos algoritmos tienen sus fortalezas y debilidades, ninguno es ampliammente superior al otro ni ninguno deja al otro obsoleto sino que es el contexto el que prima en estas ocasiones. Ambas son herramientas útiles en el repertorio de todo desarrollador.

Bibliografía

Algoritmos de Ordenamiento y Búsqueda en Python: Optimizando la Gestión de Datos. (2025, April 25). 4Geeks. https://4geeks.com/es/lesson/algoritmos-de-ordenamiento-y-busqueda-en-python

Krishna, A. (2024, August 14). Search algorithms – linear search and binary search code implementation and complexity analysis. freeCodeCamp.org. https://www.freecodecamp.org/news/search-algorithms-linear-and-binary-search-explained/

TutorialsPoint. (2025, March 25). Searching algorithms. https://www.tutorialspoint.com/data_structures_algorithms/searching_algorithms.htm

TutorialsPoint. (2025a, March 25). Linear Search Algorithm. https://www.tutorialspoint.com/data_structures_algorithms/linear_search_algorithm.htm

TutorialsPoint. (2025a, March 25). Binary search algorithm. https://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm

Anexo

Presentacion de Youtube

Link al repositorio