

**Instituto Tecnológico de Mérida**

*Campus Poniente*

***“Estructura de datos”***

Alumno: Luis Alberto Salazar Canché

Matricula: **E23080244**

Carrera: Ingeniería en sistemas computacionales

Fecha: 16/03/24

Materia: Estructura de datos

Docente: Ing. Armando López Valadez

***Búsqueda Secuencial y Binaría***

Explicará en qué consiste cada ejercicio

Por qué usa ese método de búsqueda

Si sé pudiera mejorar con otro método de búsqueda

Conclusiones

***Búsqueda Secuencial***

***Texto

Descripción generada automáticamente***

***¿Cómo funciona?***

La función busquedaSecuencial busca un elemento en una lista de manera secuencial, recorriendo cada elemento hasta encontrar el objetivo o llegar al final de la lista. Inicializa una variable pos en 0 para rastrear la posición y una variable booleana encontrado en False. A través de un bucle while, compara cada elemento en la posición actual de la lista con el elemento buscado. Si lo encuentra, establece encontrado en True; si no, incrementa pos en 1 y continúa la búsqueda. Finalmente, la función retorna True si encontró el elemento y False si no. Este método es simple y eficaz para listas pequeñas, pero puede ser ineficiente para listas grandes. Métodos alternativos como la búsqueda binaria, que requiere listas ordenadas y ofrece una complejidad de O(log n), o estructuras avanzadas como tablas hash y árboles de búsqueda, pueden mejorar significativamente la eficiencia en listas grandes.

***Por qué se usa este método de búsqueda***

La búsqueda secuencial es uno de los métodos más simples para buscar un elemento en una lista. Se usa por las siguientes razones:

*Simplicidad*: Es fácil de entender e implementar.

*Generalidad*: Funciona con cualquier tipo de lista, sin importar si está ordenada o no.

*Pequeños conjuntos de datos*: Es eficaz para listas pequeñas donde el tiempo de búsqueda no es un problema significativo.

***Búsqueda Binaría***

Texto

Descripción generada automáticamente

¿Cómo funciona?

La función busqueda\_binaria inicia con dos variables, inicio y final, que representan el rango de búsqueda en la lista ordenada lista. Utiliza un bucle while que continúa hasta que inicio sea mayor que final. En cada iteración, calcula el medio del rango actual y compara el elemento en lista[medio] con buscado. Si son iguales, retorna la posición medio. Si lista[medio] es menor que buscado, ajusta el rango de búsqueda a la mitad superior del rango actual incrementando inicio en medio + 1. Si lista[medio] es mayor que buscado, ajusta el rango de búsqueda a la mitad inferior del rango actual decrementando final en medio - 1. Si el elemento no se encuentra después de recorrer todo el bucle, retorna None. Este método es altamente eficiente con una complejidad de O(log n), ideal para listas grandes y ordenadas como la proporcionada en el ejemplo.`

***Por qué se usa este método de búsqueda***

Eficiencia: La búsqueda binaria tiene una complejidad temporal de O(log n), lo que la hace muy eficiente, especialmente en listas grandes. Esto se debe a que divide repetidamente el espacio de búsqueda por la mitad, reduciendo significativamente el número de comparaciones necesarias en comparación con la búsqueda secuencial (que tiene una complejidad de O(n)).

Ordenamiento requerido: La búsqueda binaria requiere que la lista esté ordenada previamente. Sin embargo, una vez que la lista está ordenada, la búsqueda binaria puede ser mucho más rápida que otros métodos como la búsqueda secuencial.

Aplicabilidad: Es especialmente útil cuando se necesita buscar elementos en grandes conjuntos de datos ordenados, como listas o matrices. Esto la hace ideal para bases de datos y sistemas de gestión de archivos donde se deben realizar búsquedas eficientes.

***Mejoras con otros métodos de búsqueda***

Si bien la búsqueda binaria es muy eficiente para listas ordenadas, existen escenarios donde otros métodos podrían ser más apropiados:

Búsqueda secuencial: Para listas pequeñas o no ordenadas, la búsqueda secuencial puede ser más sencilla de implementar y suficientemente eficiente.

Búsqueda por hashing: En estructuras de datos como tablas hash, donde se puede acceder directamente al elemento deseado en tiempo promedio O(1). Esto es ideal cuando se conoce la función hash y no es necesario recorrer todos los elementos.

Árboles de búsqueda: Estructuras como los árboles binarios de búsqueda o los árboles AVL pueden ser más eficientes en ciertos casos, especialmente cuando se necesita mantener una estructura de datos ordenada y realizar operaciones de inserción y eliminación además de la búsqueda.