ALGORITMOS DE BUSQUEDA

MARCO TEORICO

Los algoritmos de búsqueda permiten encontrar un elemento específico dentro de una lista. Según cómo estén organizados los datos, se elige el algoritmo más adecuado: si la lista está ordenada, se puede usar la **búsqueda binaria**, que es más eficiente. En cambio, si la lista está desordenada, se debe aplicar la **búsqueda lineal**, que revisa cada elemento uno por uno. Ambos algoritmos son fundamentales en programación y se usan frecuentemente para resolver problemas relacionados con estructuras de datos.

Además, es importante considerar la **eficiencia** de cada algoritmo, ya que afecta directamente al **tiempo de ejecución** del programa, especialmente cuando se trabaja con grandes volúmenes de datos. Por eso, elegir el algoritmo correcto puede marcar una gran diferencia en el rendimiento de una aplicación.

**BUSQUEDA 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, pero puede ser muy ineficiente dependiendo del largo de la lista y la ubicación donde está el elemento.

**Ventajas y Desventajas del Algoritmo de Búsqueda Lineal**

**Ventajas:**

* **Sencillez**: La búsqueda lineal es uno de los algoritmos de búsqueda más simples y fáciles de implementar. Solo requiere iterar a través de la lista de elementos uno por uno hasta encontrar el objetivo.
* **flexibilidad**: La búsqueda lineal puede aplicarse a cualquier tipo de lista, independientemente de si está ordenada o no.

**Desventajas:**

* **Ineficiencia en listas grandes**: La principal desventaja de la búsqueda lineal es su ineficiencia en listas grandes. Debido a que compara cada elemento uno por uno, su tiempo de ejecución crece de manera lineal con el tamaño de la lista.
* **No es adecuada para listas ordenadas**: Aunque puede funcionar en listas no ordenadas, la búsqueda lineal no es eficiente para listas ordenadas. En tales casos, algoritmos de búsqueda más eficientes, como la búsqueda binaria, son preferibles.

CASO PRACTICO

*def* linear\_search (*lista*, *objetivo*):

    # Recorremos todos los elementos de la lista uno por uno

    for i in range(len(lista)):

        # Si el elemento actual coincide con el objetivo, devolvemos su índice

        if lista[i] == objetivo:

            return i

    # Si terminamos de recorrer la lista y no encontramos el objetivo, devolvemos -1

    return -1

# Lista de números ordenada (aunque la búsqueda lineal no necesita que lo esté)

lista = [1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 15, 20, 27, 34, 39, 50]

# Número que queremos buscar

numero\_objetivo = 39

# Llamamos a la función de búsqueda lineal y guardamos el resultado

resultado = linear\_search (lista, numero\_objetivo)

# Verificamos si el número fue encontrado y mostramos el resultado

if resultado! = -1:

    print(*f*"El número {numero\_objetivo} se encuentra en la posición: {resultado}")

else:

    print (*f*"El número {numero\_objetivo} NO se encuentra en la lista.")

En este ejemplo de código, necesitamos buscar el número **39**, para buscarlo de forma lineal simplemente recorremos la lista con la ayuda de una estructura de bucle for y luego preguntamos si el elemento actual es igual a el elemento que estamos buscando, de ser así retornamos el índice del elemento y terminamos el bucle pero si el bucle termina y no retorno ningún elemento significa que el número que buscamos no se encuentra en la lista por lo que retornamos **-1**. Este algoritmo puede ser útil para recorrer listas pequeñas o listas desordenadas pero no es eficiente para recorrer listas demasiado largas.

**BUSQUEDA 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 significativamente la cantidad de comparaciones necesarias.

**Ventajas y Desventajas del Algoritmo de Búsqueda Binaria**

**Ventajas:**

* **Eficiencia de listas ordenadas**: La principal ventaja de la búsqueda binaria es su eficiencia en listas ordenadas. Su tiempo de ejecución es de **O(log n)**, lo que significa que disminuye rápidamente a medida que el tamaño de la lista aumenta.
* **Menos comparaciones**: Comparado con la búsqueda lineal, la búsqueda binaria realiza menos comparaciones en promedio, lo que lo hace más rápido para encontrar el objetivo.

**Desventajas:**

* **Requiere una lista ordenada**: La búsqueda binaria sólo es aplicable a listas ordenadas, Si la lista no está ordenada, se debe realizar una operación adicional para ordenarla antes de usar la búsqueda binaria.
* **Mayor complejidad de implementación**: Comparado con la búsqueda lineal, la búsqueda binaria es más compleja de implementar debido a su naturaleza recursiva.

CASO PRACTICO

def binary\_search(lista, objetivo, inicio, fin):

# Condición de corte: si el inicio supera al fin, el elemento no está en la lista

if inicio > fin:

return -1

# Calculamos el índice del elemento central

centro = (inicio + fin) // 2

# Si el valor del centro es igual al objetivo, lo encontramos

if lista[centro] == objetivo:

return centro

# Si el valor del centro es menor al objetivo, buscamos en la mitad derecha

elif lista[centro] < objetivo:

return binary\_search(lista, objetivo, centro + 1, fin)

# Si el valor del centro es mayor, buscamos en la mitad izquierda

else:

return binary\_search(lista, objetivo, inicio, centro - 1)

# --- Ejemplo de uso del algoritmo ---

# Lista ordenada (requisito fundamental para aplicar búsqueda binaria)

lista = [1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 15, 20, 27, 34, 39, 50]

# Número que queremos buscar

numero\_objetivo = 27

# Definimos el rango de búsqueda inicial (toda la lista)

inicio\_busqueda = 0

fin\_busqueda = len(lista) - 1

# Llamamos a la función y guardamos el resultado

resultado = binary\_search(lista, numero\_objetivo, inicio\_busqueda, fin\_busqueda)

# Mostramos el resultado

if resultado != -1:

print(f"El número {numero\_objetivo} se encuentra en la posición {resultado}.")

else:

print(f"El número {numero\_objetivo} NO se encuentra en la lista.")

En este ejemplo, hacemos uso de un algoritmo de búsqueda binario para encontrar el número **27** en una lista de elementos ordenados, para poder encontrar el elemento que buscamos podemos hacer uso de una función [**recursiva**](https://4geeks.com/es/lesson/que-es-la-recursividad-en-python), en esta función el caso base sería si el número de la lista en la posición centro es igual al número que buscamos, de ser así retornamos el valor de la variable centro este sería el índice del número, de lo contrario, dividimos la lista en dos mitades y hacemos el llamado recursivo hasta encontrar el número que buscamos pero si el número no se encuentra en la lista retornamos **-1**.

FUENTES

* <https://4geeks.com/es/lesson/algoritmos-de-ordenamiento-y-busqueda-en-python>
* <https://www.geeksforgeeks.org/searching-algorithms/>