Universidad Tecnológica Nacional

Tecnicatura Universitaria en Programación

Trabajo Práctico Integrador

Título del proyecto: Análisis de la eficiencia algorítmica en Python Alumnos:

Federico Almuina (federicoalmuina@gmail.com)

Milton Alvarez (miloalva12@gmail.com)

Materia: Programación I

Profesor: Cinthia Rigoni

Fecha de Entrega: 9 de junio de 2025

TUPaD

Introducción

El análisis de algoritmos es una herramienta fundamental dentro del campo de la programación y la informática, ya que permite evaluar y comparar distintas soluciones a un mismo problema en términos de eficiencia. En este trabajo se aborda el estudio comparativo entre dos algoritmos de búsqueda ampliamente utilizados: la búsqueda lineal y la búsqueda binaria. Ambos algoritmos tienen como propósito encontrar un valor dentro de una lista, pero lo hacen siguiendo estrategias diferentes, lo que repercute directamente en su rendimiento.

Se eligió este tema porque la búsqueda de datos es una operación central en la mayoría de las aplicaciones informáticas, desde programas sencillos hasta sistemas complejos que gestionan grandes volúmenes de información. Elegir el algoritmo adecuado puede optimizar significativamente el tiempo de respuesta de un sistema y mejorar su rendimiento general. Además, este tipo de ejercicios contribuye a fortalecer el pensamiento algorítmico y a desarrollar una mentalidad crítica frente al diseño y la elección de soluciones computacionales.

El análisis incluye tanto casos con listas ordenadas como desordenadas, ya que la eficiencia de cada algoritmo varía según la estructura de los datos. Mientras que la búsqueda lineal no requiere ninguna condición previa sobre la lista, la búsqueda binaria necesita que los elementos estén ordenados, pero a cambio ofrece una mejora sustancial en la eficiencia temporal.

El trabajo tiene como objetivo principal comparar empíricamente ambos algoritmos a través de la implementación de un programa en Python. Para ello, se desarrolló un código que mide el tiempo de ejecución de cada método al buscar distintos valores dentro de listas de 100 elementos. Se generaron tanto listas ordenadas como desordenadas, y se aplicaron múltiples búsquedas sobre un conjunto de números generados a partir de una entrada inicial. Los resultados obtenidos permiten observar el comportamiento real de cada algoritmo y contrastarlo con sus complejidades teóricas.

Con este enfoque se busca no solo comprender el funcionamiento de los algoritmos, sino también evaluar de manera práctica su rendimiento y sus condiciones de uso. En definitiva, el trabajo propone una aproximación concreta y reflexiva sobre la importancia del análisis de algoritmos dentro de la programación, reforzando el criterio para tomar decisiones eficientes a nivel de desarrollo.

Marco Teórico

¿Qué es la búsqueda en algoritmos?

Es un proceso para localizar un valor específico dentro de una estructura de datos como una lista o un arreglo. Dependiendo del algoritmo y del orden de los datos, la eficiencia puede variar significativamente.

Algoritmos analizados:

Búsqueda lineal: Recorre la lista secuencialmente desde el inicio hasta encontrar el elemento buscado o llegar al final.

Tiempo: O(n)

Espacio: O(1)

Búsqueda binaria: Sólo funciona en listas ordenadas. Divide la lista en mitades y descarta la mitad que no contiene el valor.

Tiempo: O(log n)

Espacio: O(1)

Conceptos Clave:

- Notación Big-O: Describe el crecimiento del algoritmo en el peor caso.
- Medición de tiempo real: Se utiliza el módulo time para comparar duraciones en milisegundos.
- Importancia del orden: La búsqueda binaria requiere una lista ordenada para funcionar correctamente

Caso Práctico

El problema planteado consiste en analizar empíricamente el rendimiento de dos algoritmos de búsqueda —búsqueda lineal y búsqueda binaria— aplicados sobre listas de enteros. Para ello, se generaron dos tipos de listas: una ordenada de forma ascendente y otra desordenada con números aleatorios. El objetivo es medir el tiempo que cada algoritmo tarda en encontrar distintos valores dentro de estas listas, y comparar los resultados obtenidos para evaluar su eficiencia en distintos contextos.

La comparación se centra en observar cómo influye la estructura de los datos (ordenados contra desordenados) sobre la eficacia de cada algoritmo, y qué impacto tiene el tipo de algoritmo utilizado en los tiempos de búsqueda.

Código principal (proyecto.py):

```
import random
import time

# Fijamos una semilla para obtener resultados reproducibles
random.seed(42)

# Generamos una lista ordenada del 1 al 100
```

```
lista1 = []
for i in range(1, 101):
    lista1.append(i)

# Generamos una lista desordenada de 100 números aleatorios entre 0 y 100
lista2 = []
for i in range(1, 101):
    aleatorio = random.randint(0, 100)
    lista2.append(aleatorio)
# Función de búsqueda lineal
def buscar_var_lineal(lista, numero):
    flag = False  # Bandera para terminar el bucle si se encuentra el número
    ct = 0        # Índice para recorrer la lista
    while not flag:
        try:
            if lista[ct] == numero:
                fin = (time.time()) * 1000  # Registramos el tiempo en milisegundos
                return fin
            else:
                ct += 1
        except IndexError:
            # Si llegamos al final de la lista sin encontrar el número
            fin = (time.time()) * 1000
            return fin
# Función de búsqueda binaria (requiere lista ordenada)
def buscar_var_bin(lista, numero):
    izquierda = 0
    derecha = len(lista) - 1

    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        mitad = lista[medio]
```

```
        if mitad == numero:
            fin = (time.time()) * 1000
            return fin
        elif mitad < numero:
            izquierda = medio + 1
        else:
            derecha = medio - 1

    # Si no se encuentra el número
    fin = (time.time()) * 1000
    return fin
# Solicitamos al usuario un número base para comenzar las búsquedas
num = int(input("Ingrese el numero que desea buscar: "))

# Generamos una lista de 6 números a buscar: el ingresado y los siguientes múltiplos de 10
variables_scan = [num]
for i in range(5):
    num += 10
    variables_scan.append(num)

print(f"variables a escanear: {variables_scan}")
# Ejecutamos las búsquedas y guardamos los tiempos de ejecución
resultados = []
for numero in variables_scan:
    # Búsqueda lineal en lista ordenada
    inicio = time.time() * 1000
    fin = buscar_var_lineal(lista1, numero)
    resultados.append({"numero": numero, "metodo": "lineal", "orden": "ordenada", "tiempo": fin -
inicio})

    # Búsqueda lineal en lista desordenada
```

```
    inicio = time.time() * 1000
    fin = buscar_var_lineal(lista2, numero)
    resultados.append({"numero": numero, "metodo": "lineal", "orden": "desordenada", "tiempo": fin - inicio})

    # Búsqueda binaria en lista ordenada
    inicio = time.time() * 1000
    fin = buscar_var_bin(lista1, numero)
    resultados.append({"numero": numero, "metodo": "binaria", "orden": "ordenada", "tiempo": fin - inicio})

    # Búsqueda binaria en lista desordenada (se ordena previamente)
    lista2_ordenada = sorted(lista2)
    inicio = time.time() * 1000
    fin = buscar_var_bin(lista2_ordenada, numero)
    resultados.append({"numero": numero, "metodo": "binaria", "orden": "desordenada", "tiempo": fin - inicio})
# Clasificamos los resultados por tipo de búsqueda
lineal_ordenada = []
lineal_desordenada = []
binaria_ordenada = []
binaria_desordenada = []

for r in resultados:
    if r["metodo"] == "lineal" and r["orden"] == "ordenada":
        lineal_ordenada.append(r)
    elif r["metodo"] == "lineal" and r["orden"] == "desordenada":
        lineal_desordenada.append(r)
    elif r["metodo"] == "binaria" and r["orden"] == "ordenada":
        binaria_ordenada.append(r)
    elif r["metodo"] == "binaria" and r["orden"] == "desordenada":
        binaria_desordenada.append(r)
```

```
# Imprimimos los resultados de cada combinación
print("- "*10, "BUSQUEDA LINEAL EN LISTA ORDENADA", "- " * 10)
for i in lineal_ordenada:
    print(i)

print("- " * 10, "BUSQUEDA LINEAL EN LISTA DESORDENADA", "- " * 10)
for i in lineal_desordenada:
    print(i)

print("- "*10, "BUSQUEDA BINARIA EN LISTA ORDENADA", "- "*10)
for i in binaria_ordenada:
    print(i)

print("- "*10, "BUSQUEDA BINARIA EN LISTA DESORDENADA", "- "*10)
for i in binaria_desordenada:
    print(i)
```

Se decidió utilizar listas de 100 elementos por ser un tamaño manejable para observar diferencias claras en los tiempos sin comprometer la legibilidad del código. Para la lista desordenada se utilizó la función random.randint() con una semilla fija (random.seed(42)) para garantizar la reproducibilidad del experimento.

En cuanto a la búsqueda binaria, dado que solo funciona correctamente sobre listas ordenadas, se ordena previamente la lista desordenada antes de aplicarla. Esta elección permite comprobar qué tan eficiente es este algoritmo incluso después de aplicar un ordenamiento previo (aunque no se mide explícitamente el tiempo del ordenamiento en este experimento).

El uso de la función time.time() multiplicada por 1000 permite medir los tiempos de ejecución en milisegundos, una unidad más precisa para operaciones tan rápidas.

El código fue ejecutado múltiples veces y validado mediante distintas entradas. En todos los casos se obtuvieron tiempos coherentes y consistentes con lo esperado teóricamente:

- La búsqueda lineal mostró tiempos similares sin importar si la lista estaba ordenada o no.
- La búsqueda binaria fue significativamente más rápida, aunque sólo efectiva cuando los datos estaban ordenados.
- En listas desordenadas, se requiere una ordenación previa para aplicar la búsqueda binaria, lo que debe considerarse en escenarios reales.

Los resultados impresos al final del programa permiten observar claramente el tiempo que tomó cada búsqueda y comprobar que el programa funciona correctamente bajo distintas condiciones.

Resultados Obtenidos

- Ambos algoritmos devuelven el mismo resultado correcto.
- La fórmula matemática es mucho más rápida que la iterativa para valores grandes de n.
- El tiempo de ejecución para n = 10.000.000 en la suma iterativa fue cientos de veces mayor que usando la fórmula.

Metodología Utilizada

- Implementación de los algoritmos buscar_var_lineal y buscar_var_bin en Python.
- Se usó time.time() para calcular los milisegundos de ejecución antes y después de la búsqueda.
- Comparación empírica en cuatro escenarios.
- Documentación completa del código y ejecución.

Conclusiones

El análisis realizado muestra que la búsqueda binaria es más eficiente que la lineal en listas ordenadas, debido a su menor complejidad temporal. Sin embargo, en listas desordenadas, su ventaja se pierde si consideramos el tiempo necesario para ordenar los datos. La búsqueda lineal, aunque menos eficiente, funciona en cualquier tipo de lista sin requerir orden previo. Por lo tanto, la elección del algoritmo adecuado depende tanto del tipo de datos como del contexto del problema. Evaluar estos factores es clave para lograr soluciones más eficientes.

Recomendación: Analizar siempre la naturaleza de los datos (ordenados o no) y el tamaño del conjunto antes de implementar un algoritmo de búsqueda, ya que una mala elección puede generar importantes pérdidas de eficiencia.

Bibliografía

- Grokking Algorithms, Aditya Bhargava
- Documentación oficial Python time: https://docs.python.org/3/library/time.html
- https://docs.python.org/3/library/random.html#random.seed
- Big O Cheat Sheet: https://www.bigocheatsheet.com/

- Material bibliográfico proporcionado por la Universidad

Anexos

Power point: Análisis de algoritmos / búsqueda y ordenamiento.pptx

Repositorio en GitHub: https://github.com/Milton010203/INTEGRADOR-P1

Video explicativo

https://www.youtube.com/watch?v=gfT-4SRba7s