Trabajo Práctico Integrador

Algoritmos de Búsqueda y Ordenamiento en Python

Alumnos:

Marcela Livio - Angel Nicolas Derito

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional

PROGRAMACIÓN I

ÍNDICE

1. INTRODUCCIÓN

En programación, los algoritmos de búsqueda y ordenamiento son herramientas muy importantes para encontrar información de forma rápida y eficiente.

Este trabajo integrador se enfoca en entender cómo funcionan los algoritmos de búsqueda y ordenamiento en Python y poner en práctica lo aprendido. Para eso, se desarrolló un programa que genera una lista de 1000 números al azar, la ordenamos usando dos métodos diferentes (Bubble Sort o Quick Sort) y luego buscamos un número dentro de esa lista ya ordenada utilizando búsqueda lineal o binaria.

Además, el programa mide y muestra los tiempos que tarda cada algoritmo, para poder ver cuál es más eficiente en cada caso.

2. MARCO TEÓRICO

Un algoritmo es un conjunto de pasos ordenados que se siguen para resolver un problema o realizar una tarea. En este trabajo se utilizaron dos algoritmos de ordenamiento:

- ☑ Bubble Sort: Este algoritmo compara dos elementos que están uno al lado del otro y los intercambia si están en el orden incorrecto. Repite este proceso muchas veces hasta que toda la lista esté ordenada. No es muy rápido para listas grandes.
- ☑ Quicksort: Este algoritmo elige un número como "pivote" y separa los menores a un lado y los mayores al otro. Es mucho más rápido que Bubble Sort para listas grandes.

Búsqueda de datos

Buscar significa encontrar un valor dentro de una lista. En este trabajo se usaron dos formas de búsqueda:

- ☑ Búsqueda lineal: Recorre la lista desde el principio hasta el final, comparando uno por uno hasta encontrar el número.
- ☑ Búsqueda binaria: Compara el número buscado con el del medio, y según si es mayor o menor, descarta la mitad de la lista y repite el proceso.

3. CASO PRÁCTICO

Se desarrolló un programa en Python para ordenar una lista de números utilizando los algoritmos Bubble Sort y Quick Sort y luego buscamos un número dentro de esa lista ya ordenada utilizando búsqueda lineal o binaria.

Tanto la explicación del código como su ejecución se mostrarán en el video.

```
import random
import time

# Algoritmos de ordenamiento
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    menores = [x for x in arr[1:] if x <= pivot]
    mayores = [x for x in arr[1:] if x > pivot]
    return quick_sort(menores) + [pivot] + quick_sort(mayores)

# Algoritmos de búsqueda
def busqueda_lineal(arr, objetivo):
    for i in range(len(arr)):
        if arr[i] == objetivo:
            return i
    return -1

def busqueda_binaria(arr, objetivo):
    izquierda, derecha = 0, len(arr) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if arr[medio] == objetivo:
            return medio
```

```
        elif arr[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1

# Selección del algoritmo de ordenamiento
def seleccionar_ordenamiento(nombre, arr):
    algoritmos = {
        "bubble sort": bubble_sort,
        "quick sort": quick_sort,
    }
    if nombre in algoritmos:
        inicio = time.perf_counter()
        resultado = algoritmos[nombre](arr.copy())
        fin = time.perf_counter()
        print(f"→ Tiempo de ordenamiento con {nombre}: {fin - inicio:.6f} segundos")
        return resultado
    else:
        print("Algoritmo no válido.")
        return arr

# Programa principal
if __name__ == "__main__":
    lista = [random.randint(0, 1000) for _ in range(10000)]
    print("Lista generada completa:")
    print(lista)

    print("Elige algoritmo de ordenamiento: bubble sort / quick sort")
    metodo = input("→ ").lower()
    ordenada = seleccionar_ordenamiento(metodo, lista)

    objetivo = int(input("¿Qué número querés buscar? "))
    print("Elige método de búsqueda: lineal / binaria")
    metodo_busqueda = input("→ ").lower()
```

```
if metodo_busqueda == "lineal":
    inicio = time.perf_counter()
    posicion = busqueda_lineal(ordenada, objetivo)
    fin = time.perf_counter()
    print(f"→ Tiempo de búsqueda lineal: {fin - inicio:.6f} segundos")
elif metodo_busqueda == "binaria":
    inicio = time.perf_counter()
    posicion = busqueda_binaria(ordenada, objetivo)
    fin = time.perf_counter()
    print(f"→ Tiempo de búsqueda binaria: {fin - inicio:.6f} segundos")
else:
    print("Método inválido.")
    exit()

if posicion != -1:
    print(f"Número encontrado en la posición {posicion}")
else:
    print(f"Numero no encontrado")
```

CASO PRÁCTICO #1: Quicksort y búsqueda lineal

```
Lista generada completa:
[690, 937, 6, 897, 336, 384, 251, 346, 244, 717, 675, 939, 733, 458, 126, 649, 290, 397, 840, 994, 291, 136, 875, 932, 993, 490
, 454, 696, 796, 124, 526, 324, 193, 451, 718, 420, 77, 961, 753, 564, 269, 446, 334, 98, 714, 87, 680, 840, 262, 956, 425, 635
, 880, 581, 181, 466, 341, 257, 825, 289, 796, 461, 772, 659, 155, 637, 817, 350, 482, 856, 612, 756, 958, 597, 213, 598, 360,
864, 16, 301, 97, 547, 94, 823, 332, 950, 321, 600, 234, 776, 767, 167, 87, 705, 328, 968, 859, 100, 626, 153, 872, 586, 401, 2
03, 990, 593, 361, 521, 533, 129, 887, 100, 750, 341, 726, 445, 496, 923, 885, 262, 319, 495, 312, 404, 417, 993, 329, 928, 184
, 914, 358, 969, 449, 649, 431, 206, 816, 208, 699, 78, 261, 416, 717, 435, 687, 843, 652, 598, 178, 819, 450, 921, 537, 617, 1
33, 850, 32, 811, 2, 113, 457, 295, 478, 734, 741, 715, 926, 493, 941, 471, 272, 240, 317, 517, 470, 189, 454, 921, 735, 248, 4
25, 534, 178, 534, 430, 239, 994, 902, 135, 18, 801, 253, 640, 166, 490, 353, 366, 176, 237, 759, 54, 170, 659, 444, 58, 666, 6
16, 85, 821, 45, 391, 326, 983, 758, 56, 939, 731, 743, 290, 478, 622, 29, 200, 295, 260, 139, 97, 347, 448, 788, 162, 252, 210
, 197, 867, 573, 571, 599, 519, 908, 209, 432, 458, 702, 117, 580, 941, 536, 872, 44, 960, 235, 55, 315, 981, 831, 245, 549, 67
3, 106, 735, 731, 961, 78, 468, 542, 617, 871, 882, 45, 7, 350, 227, 352, 465, 869, 478, 230, 218, 603, 744, 867, 765, 828, 254
```

```
Elige algoritmo de ordenamiento: bubble / insertion / quick / merge
→ quick

→ Tiempo de ordenamiento con quick: 0.040182 segundos

¿Qué número querés buscar? 21
Elige método de búsqueda: lineal / binaria
→ lineal

→ Tiempo de búsqueda lineal: 0.000028 segundos
Número encontrado en la posición 231
```

El algoritmo Quicksort ordenó la lista de 1000 elementos en aproximadamente 40 milisegundos. Quicksort es un algoritmo eficiente para ordenar grandes listas. Esto es asi porque en vez de ordenar toda la lista de una sola vez, la divide repetidamente en sublistas más pequeñas hasta que cada una es muy fácil de ordenar.

La búsqueda lineal encontró el número 21 ubicado en la posición 231 en aproximadamente 28 microsegundos.

CASO PRÁCTICO 2: Bubblesort y búsqueda binaria

```
Lista generada completa:
[21, 952, 492, 953, 185, 616, 17, 214, 73, 584, 373, 658, 977, 61, 72, 145, 482, 975, 503, 331, 824, 674, 265, 367, 949, 904, 3
25, 979, 463, 326, 96, 634, 466, 10, 867, 728, 163, 416, 614, 273, 223, 967, 569, 236, 977, 101, 506, 249, 497, 963, 927, 351,
830, 710, 596, 131, 532, 145, 678, 368, 335, 639, 727, 200, 197, 375, 269, 637, 36, 27, 420, 976, 923, 902, 979, 807, 587, 203,
 802, 770, 768, 661, 687, 744, 287, 497, 452, 810, 512, 812, 882, 670, 906, 55, 662, 345, 695, 264, 276, 620, 678, 158, 549, 52
4, 778, 176, 38, 193, 730, 221, 246, 13, 7, 201, 884, 572, 129, 514, 877, 492, 355, 792, 805, 121, 946, 21, 374, 167, 936, 977,
 569, 442, 549, 371, 444, 52, 788, 910, 467, 752, 175, 863, 80, 363, 801, 393, 945, 884, 596, 827, 587, 660, 999, 952, 712, 724
, 849, 876, 979, 880, 196, 405, 259, 408, 857, 917, 518, 582, 790, 155, 178, 394, 476, 475, 184, 612, 378, 723, 272, 990, 44, 7
5, 68, 381, 928, 791, 493, 805, 759, 505, 239, 613, 93, 497, 574, 317, 98, 732, 359, 484, 888, 752, 625, 873, 215, 603, 138, 22
, 223, 615, 830, 936, 318, 48, 963, 96, 594, 146, 471, 561, 954, 168, 337, 744, 81, 610, 295, 859, 605, 359, 685, 130, 793, 918
, 123, 116, 308, 684, 167, 935, 189, 183, 36, 750, 769, 365, 784, 12, 601, 32, 568, 781, 620, 43, 641, 320, 60, 592, 245, 968,
714, 937, 574, 465, 578, 872, 269, 69, 774, 678, 425, 594, 144, 837, 696, 447, 428, 40, 691, 720, 376, 692, 411, 976, 203, 140,

Elige algoritmo de ordenamiento: bubble / insertion / quick / merge
→ bubble

→ Tiempo de ordenamiento con bubble: 6.434915 segundos

¿Qué número querés buscar? 669
Elige método de búsqueda: lineal / binaria
→ binaria

→ Tiempo de búsqueda binaria: 0.000000 segundos
Número encontrado en la posición 6678
```

Esta vez elegimos Bubble Sort como algoritmo de ordenamiento. Tal como se puede ver, el tiempo de ejecución fue de **6.43 segundos.** Esto es asi porque bubblesort es muy lento para listas grandes, ya que compara **cada par de elementos vecinos** en la lista y los intercambia si están en el orden incorrecto.

Elegimos la búsqueda binaria, ya que el número que debíamos encontrar estaba prácticamente al final de la lista (incluso mucho más cerca del final que nuestro ejemplo anterior), tal como se muestra, tardó **0.000000 segundos** porque encontró el número casi de inmediato. Divide la lista en mitades y decide en qué mitad continuar buscando reduciendo así la necesidad de comparaciones necesarias, a diferencia de la búsqueda lineal que recorre **cada elemento de la lista uno por uno** hasta encontrar el objetivo y si el número está al final, tiene que revisar casi toda la lista.

4) METODOLOGÍA UTILIZADA

La elaboración del trabajo se realizó en las siguientes etapas:

- Elección del tema para el trabajo integrador.
- Recolección de información teórica y videos que nos ayuden a desarrollar el programa en Python.
- Implementación en Python de los algoritmos.
- Realizamos la prueba con los diferentes métodos elegidos tanto de búsqueda como de ordenamiento.
- Registramos los resultados y validamos la funcionalidad del código.
- Elaboración de informe escrito.

5) RESULTADOS OBTENIDOS

Durante la ejecución del caso práctico, se observó que cuando usamos el algoritmo **Quicksort**, el ordenamiento de la lista tardó aproximadamente **0.04 segundos**, lo cual es muy eficiente para una lista tan grande. En cambio, con el algoritmo **Bubble Sort**, el tiempo fue de alrededor de **6.4 segundos**.

En cuanto a la búsqueda, se probó buscar el número **21**, que estaba en la posición 231 en una lista de 10.000 elementos, tardando unos 0,000028 segundos. En cambio, usando **búsqueda binaria** el tiempo fue prácticamente **0.000000 segundos** a pesar de que en este caso se buscó el número 669 que se encontraba en la posición 6678.

6) CONCLUSIONES

A partir de este trabajo práctico, se observó que el algoritmo Quicksort es mucho más eficiente que Bubble Sort al momento de ordenar listas largas. Esto se debe a que Quicksort divide la lista en partes más pequeñas y las ordena de forma inteligente, sin tener que comparar cada elemento con todos los demás, como lo hace Bubble Sort. Gracias a esto, el tiempo de ordenamiento con Quicksort fue de apenas unos milisegundos (0.04 segundos), mientras que Bubble Sort tardó más de 6 segundos en hacer lo mismo.

Por otro lado, en las búsquedas realizadas, se pudo comprobar que la búsqueda binaria es mucho más rápida y eficiente que la búsqueda lineal, sobre todo cuando el número que se busca está cerca del final de la lista. En un caso concreto, al buscar el número 21 (que estaba en la posición 9998), la búsqueda binaria tardó prácticamente nada, mientras que la búsqueda lineal tuvo que recorrer casi toda la lista para encontrarlo.

Estos resultados muestran de forma clara que Quicksort es mucho más rápido que Bubble Sort al ordenar listas grandes, y que la búsqueda binaria supera a la búsqueda lineal en eficiencia.

En resumen, este trabajo permitió entender, a través de la práctica, cómo distintos algoritmos pueden afectar el rendimiento de un programa y por qué es importante elegir el algoritmo adecuado según el problema a resolver.

7) BIBLIOGRAFÍA

-”Learn Quicksort in 13 minutes” - https://www.youtube.com/watch?v=Vtckgz38QHs

- Python Oficial: https://docs.python.org/3/library/

-Chatgpt

ANEXO

Repositorio de GitHub:
https://github.com/marcelalivio/busqueda-y-ordenamiento/blob/main/algoritmos_busqueda_y_ordenamiento.py

Link al video explicativo:
https://drive.google.com/file/d/1vEn3gX_DAISRaKXVerfdeczY0HXbgwvi/view?usp=sharing