Trabajo práctico Integrador N°2
“Análisis de algoritmos de Búsqueda y Ordenamiento”

Alumnos: Cristian AYALA / Ramiro APARICIO

Materia: Programación I

Profesor: AUS Bruselario, Sebastián

Fecha de entrega: 9/06/2025

Índice

Introducción

En programación, un algoritmo es un conjunto ordenado y finito de instrucciones que una computadora sigue para realizar una tarea o resolver un problema. En esto los algoritmos de búsqueda y ordenamiento son fundamentales, ya que nos permiten manejar eficientemente grandes conjuntos de datos, tanto sea en su ordenamiento como en la búsqueda, es con este trabajo que intentaremos mostrar desde la parte teórica y practica su relevancia en la programación.

Marco Teórico

Con frecuencia los programadores necesitan encontrar un valor que coincida con un cierto valor clave que se encuentra en un arreglo o registro, a este proceso se llama búsqueda,De igual manera necesita clasificar los datos de un arreglo o colección de datos en un cierto orden a este proceso denominamos Ordenamiento.

En ambos casos se dispone de una variedad de algoritmos que pasaremos a detallar de manera sucinta

Algoritmo de Búsqueda.

Un algoritmo de búsqueda es un conjunto de pasos o instrucciones que se utilizan para encontrar un elemento o un conjunto de elementos dentro de una estructura de datos, como una lista, un árbol o un grafo. Su objetivo principal es localizar rápidamente la información deseada de manera eficiente. Por ejemplo, un algoritmo de búsqueda puede ayudarte a encontrar un número específico en una lista de números o una palabra en un diccionario digital.

Algoritmos de Ordenamiento.

Los algoritmos de ordenamiento son conjuntos de instrucciones que se utilizan para organizar u ordenar un conjunto de datos, como números, palabras o cualquier tipo de información, en un orden específico, ya sea ascendente o descendente. Su objetivo es facilitar la búsqueda, comparación o análisis de los datos, haciendo que la información sea más fácil de manejar y entender. Algunos ejemplos comunes de algoritmos de ordenamiento son el bubble sort, el quicksort y el mergesort

Ventajas de usar Algoritmos de Búsqueda y Ordenamiento

1. **Mayor eficiencia:** Permiten encontrar y organizar datos rápidamente, lo que hace que las aplicaciones funcionen de manera más ágil y eficiente, especialmente cuando manejan grandes volúmenes de información.
2. **Facilitan la gestión de datos:** Ordenar los datos ayuda a visualizarlos y analizarlos mejor, además de simplificar tareas como buscar, eliminar o actualizar información específica.
3. **Mejor rendimiento:** Al usar algoritmos optimizados, se reduce el tiempo y los recursos necesarios para realizar operaciones con datos, lo que resulta en programas más rápidos y con menor consumo de recursos.
4. **Automatización de tareas:** Permiten automatizar procesos complejos de organización y búsqueda, ahorrando tiempo y esfuerzo manual.
5. **Base para otros algoritmos:** Muchos algoritmos avanzados y técnicas de programación dependen de búsquedas y ordenamientos eficientes para funcionar correctamente.

Tipos de Ordenamiento

Dentro de la programación, los métodos más comunes son:

1. **Ordenamiento de burbuja (Bubble Sort):** Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Es simple, pero no muy eficiente para grandes conjuntos de datos.
2. **Ordenamiento por selección (Selection Sort):** Encuentra el elemento más pequeño (o más grande) y lo coloca en la posición correcta, repitiendo el proceso para el resto de la lista.
3. **Ordenamiento por inserción (Insertion Sort):** Construye la lista ordenada de uno en uno, insertando cada nuevo elemento en su lugar correcto.
4. **Ordenamiento rápido (Quick Sort):** Divide y vencerás, seleccionando un pivote y particionando la lista en dos partes, ordenando cada una recursivamente. Es muy eficiente en la mayoría de los casos.

5. **Ordenamiento por mezcla (Merge Sort):** Divide la lista en partes más pequeñas, las ordena y luego las combina. Es muy estable y eficiente para listas grandes.
6. **Ordenamiento por montículo** (HeapSort): HeapSort es un algoritmo de ordenamiento que utiliza la estructura de datos Heap (montón) para ordenar una lista de elementos. Funciona construyendo un Heap a partir de la lista y luego extrae el elemento más grande (o más pequeño, según el Heap) de la raíz en cada iteración, colocando el elemento extraído al final de la lista, hasta que la lista esté ordenada.

A continuación, un caso práctico de un algoritmo de Ordenamiento:

C: > Users > Ramiro > Desktop > tp integrador > Ordenamiento_burbuja_interactivo.py > ...

```python
def burbuja(arr):
    n = len(arr)
    # Mostramos el array inicial
    print(f"Array inicial: {arr}")

    for i in range(n):
        # Creamos una copia del array para mostrar cambios
        arr_copia = arr.copy()

        # Bandera para verificar si hubo intercambios
        hay_cambios = False

        # Recorremos desde el primer elemento hasta el (n-i-1)ésimo
        for j in range(0, n - i - 1):
            # Si el elemento actual es mayor que el siguiente
            if arr[j] > arr[j + 1]:
                # Intercambiamos los elementos
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                hay_cambios = True

        # Mostramos el estado después de cada iteración
        print(f"\nIteración {i+1}:")
        print(f"Estado final: {arr}")

        # Si no hubo cambios en la iteración, el array ya está ordenado
        if not hay_cambios:
            print("\nEl array ya está ordenado!")
            break

    return arr

# Ejemplo con números desordenados
numeros = [30,16,89,1,47,36,24,]
resultado = burbuja(numeros)
print(f"\nArray final ordenado: {resultado}")
```

Tipos de Algoritmo de Búsqueda:

1- Búsqueda Lineal (Linear Search)
Es el algoritmo de búsqueda más simple, que recorre cada elemento del conjunto de datos de forma secuencial hasta encontrar el elemento deseado.

2. Búsqueda Binaria (Binary Search)
Es eficiente y funciona en conjuntos de datos ordenados. Divide el conjunto de datos en dos mitades y busca el elemento deseado en la mitad correspondiente. Repite este proceso hasta encontrar el elemento o determinar que no está en el conjunto de datos.

3. Búsqueda de interpolación:
Mejora la búsqueda binaria al estimar la posición del elemento deseado en función de su valor. Puede ser más eficiente que la búsqueda binaria para conjuntos de datos grandes con una distribución uniforme de valores.

4. Búsqueda por salto (Jump Search)
También requiere lista ordenada. Salta en bloques de tamaño √n y luego realiza búsqueda lineal dentro del bloque adecuado.

5. Búsqueda Exponencial (Exponential Search)
Diseñada para listas grandes con acceso secuencial o cuando no se conoce el tamaño exacto. Busca un rango exponencialmente y luego hace búsqueda binaria-

Ideal para listas enormes, especialmente si se desconoce su tamaño. Es una combinación inteligente entre búsqueda rápida y acotada.

6. Búsqueda Hash (Hash Search)
Utiliza una función hash para asignar cada elemento a una ubicación única en la tabla hash.

7. Búsqueda Binaria Recursiva

Divide repetidamente el array ordenado en mitades, comparando el valor objetivo con el valor medio, y descartando la mitad en la que el objetivo no puede estar. Se implementa usando recursos.

Metodología utilizada

- Se procedió a buscar información en fuentes confiables
- Se hicieron pruebas de funcionamiento con diferentes tipos de datos
- Implementación en Python varios de los algoritmos

Resultados Obtenidos

- Se entendieron los conceptos de ordenamiento y búsqueda
- Se verificaron los resultados al obtener una lista ordenada
- Se comprendieron en cada caso y de acuerdo a cada necesidad la implementación de los distintos algoritmos

Conclusión

Como conclusión determinamos lo siguiente:

- El ordenamiento es una inversión inicial que mejora significativamente las búsquedas posteriores
- Un conjunto ordenado permite usar búsqueda binaria (O(log n)) en lugar de lineal (O(n))
- Esta relación justifica el uso de algoritmos de ordenamiento más costosos cuando se anticipan muchas búsquedas

Bibliografía

https://realpython.com/

https://jorgeantilefblog.wordpress.com/algoritmos-de-busqueda-y-ordenamiento/

https://realpython.com/sorting-algorithms-python/#measuring-quicksorts-big-o-complexity

https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/