Algoritmos de Búsqueda y Ordenamiento

Alumnos:

Montenegro Sergio – smontene22@gmail.com

Morales Bruno - brunomorales2303@outlook.es

Materia: Programación I

Profesor Coordinador: Alberto Cortez

Profesor a cargo comisión: Cinthia Rigoni

Nicolas Quirós Ricci

Julieta Trape

Sebastián Bruselario

Ariel Enferrel

Fecha de Entrega:

Índice

1. Introducción

Los **algoritmos de búsqueda y ordenamiento** son fundamentales en informática y se utilizan para organizar y encontrar datos de manera eficiente, lo que es esencial para optimizar el rendimiento de las aplicaciones.

Debemos comprender como se implementan y en que situaciones pueden aplicarse para realizar un desarrollo de software efectivo.

En este trabajo mencionaremos los aspectos más relevantes de estos algoritmos

2. Marco Teórico

El ordenamiento es un proceso fundamental en la programación que consiste en organizar un conjunto de elementos (letras, números, objetos) siguiendo un criterio especifico, como de menor a mayor o de mayor a menor. Esta tarea puede parecer simple, pero es esencial para muchas aplicaciones, desde la búsqueda de información hasta la visualización de datos.

Los algoritmos de ordenación se pueden clasificar en función de los siguientes parámetros:

1. **La cantidad de intercambios o inversiones requeridas:** Es la cantidad de veces se realiza un intercambio de elementos para ordenar la entrada. La ordenación por selección requiere el número mínimo de intercambios.
2. **El número de comparaciones:** Las veces que el algoritmo compara elementos para realizar el orden. Usando la notación Big-O, se puede estudiar el peor de los casos, el mejor y el caso promedio.
3. **Ya sea que usen recursividad o no:** Algunos algoritmos de ordenación, como la ordenación rápida, usan técnicas recursivas para ordenar la entrada. Otros algoritmos de ordenación, como la ordenación por selección o la ordenación por inserción, utilizan técnicas no recursivas. Por último, algunos algoritmos de ordenación, como la ordenación por fusión, utilizan técnicas tanto recursivas como no recursivas para ordenar la entrada.
4. **Ya sean estables o inestables:** Los algoritmos de ordenación estables mantienen el orden relativo de los elementos con valores iguales o claves. Los algoritmos de ordenación inestables no mantienen el orden relativo de los elementos con valores/claves iguales.
5. **La cantidad de espacio adicional requerido:** Algunos algoritmos de ordenación pueden ordenar una lista sin crear una lista completamente nueva. Estos se conocen como algoritmos de ordenación en el lugar y requieren un O(1) espacio adicional constante para la ordenación. Mientras tanto, los algoritmos de ordenación fuera de lugar crean una nueva lista durante la ordenación.

Existen diferentes algoritmos de ordenamiento, cada uno con sus ventajas y desventajas en términos de velocidad y uso de memoria. Algunos de los más conocidos son:

- **Bubble Sort (ordenamiento burbuja)**: Compara elementos adyacentes e intercambia si están desordenados. Es fácil de entender, pero poco eficiente.
- **Selection Sort (ordenamiento por selección)**: Encuentra el menor elemento en cada iteración y lo coloca en la posición correcta.
- **Insertion Sort (ordenamiento por inserción)**: Inserta cada elemento en su lugar dentro una lista ordenada. Es útil para conjuntos pequeños.

- **Merge Sort (ordenamiento por mezcla)**: Divide y vencerás: separa la lista en partes más pequeñas, las ordena y las une de nuevo.
- **Quick Sort(ordenamiento rápido)**: Elige un elemento "pivote" y reordena la lista en dos partes. Es uno de los más rápidos en la práctica.
- **Heap Sort(ordenamiento por montículo)**: Usa una estructura de datos llamada heap para ordenar elementos eficientemente.

La **búsqueda** es el proceso de encontrar un elemento específico dentro de una estructura de datos, como una lista, un arreglo o una base de datos. Es fundamental para optimizar el rendimiento de programas y algoritmos.

Existen varios tipos de algoritmos de búsqueda, entre ellos:

- **Búsqueda lineal**: Recorre la estructura de datos elemento por elemento hasta encontrar el objetivo. Es simple, pero puede ser lenta para grandes volúmenes de datos.
- **Búsqueda binaria**: Divide la lista ordenada en mitades y descarta la mitad donde no está el elemento buscado. Es mucho más eficiente, pero requiere que los datos estén ordenados previamente.
- **Búsqueda por interpolación**: Similar a la binaria, pero usa una estimación matemática para decidir por dónde empezar la búsqueda. Ajusta su posición de búsqueda en función del valor clave buscado. Funciona bien con datos distribuidos de manera uniforme.
- **Búsqueda por saltos**: También conocida como búsqueda por bloques, está diseñada específicamente para matrices ordenadas. A diferencia de la búsqueda lineal, que examina cada elemento secuencialmente, la búsqueda por saltos salta en incrementos de tamaño fijo, omitiendo múltiples elementos para realizar búsquedas más rápidas.

3. Caso Práctico

Se desarrolló un programa en Python para ordenar una lista de números utilizando los algoritmos Bubble Sort y para buscar un número específico mediante búsqueda binaria.

```python
import random

def quick_sort(arr):
    if len(arr) <= 1:
        return arr  # Caso base: lista de 1 elemento ya está ordenada
    else:
        pivot = arr[len(arr) // 2]  # Elegimos el pivote
        izquierda = [x for x in arr if x < pivot]  # Elementos menores al pivote
        centro = [x for x in arr if x == pivot]  # Elementos iguales al pivote
        derecha = [x for x in arr if x > pivot]  # Elementos mayores al pivote
        return quick_sort(izquierda) + centro + quick_sort(derecha)  # Ordenamos recursivamente


def busqueda_binaria(arr, target):
    izquierda, derecha = 0, len(arr) - 1
    while izquierda <= derecha:
        centro = (izquierda + derecha) // 2
        if arr[centro] == target:
            return centro  # Se encontró el elemento
        elif arr[centro] < target:
            izquierda = centro + 1  # Buscar en la mitad derecha
        else:
            derecha = centro - 1  # Buscar en la mitad izquierda
    return -1  # Elemento no encontrado

# Prueba del algoritmo


lista = [random.randint(1, 1000) for _ in range(100)]

lista_ordenada = quick_sort(lista)
objetivo = int(input("Ingrese el elemento a buscar: "))
resultado = busqueda_binaria(lista_ordenada, objetivo)
print(f"La lista aleatoria es {lista}")
print(f"La lista ordenada es {lista_ordenada}")
print(f"El elemento {objetivo} está en la posición {resultado}" if resultado != -1 else "Elemento no se creo en la lista aleatoria")
```

4. Metodología Utilizada

La elaboración del trabajo se realizó en las siguientes etapas:

- Recolección de información teórica en documentación confiable.

- Implementación en Python de los algoritmos estudiados.

- Pruebas con pocos datos para luego pasar elevar la cantidad.

- Registro de resultados y validación de funcionalidad.

- Elaboración de este informe y preparación de anexos.

5. Resultados Obtenidos

- Crear una lista de números en forma aleatoria

- El programa ordenó correctamente la lista de números

- La búsqueda binaria localizó de forma eficiente el número especificado.

- Se comprendieron las diferencias en complejidad entre los algoritmos estudiados.

- Se valoró la importancia de tener los datos ordenados para aplicar búsqueda binaria.

6. Conclusiones

Los algoritmos de búsqueda y ordenamiento son pilares esenciales de la informática. Entender bien sus ventajas y desventajas permite optimizar los procesos de búsqueda y organización de datos. En este trabajo destacamos la búsqueda binaria, pero para poder utilizarla realizamos un ordenamiento rápido (quick sort). Ambos fueron muy efectivos.

7. Bibliografía

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

- "**Algoritmos de ordenación explicados con ejemplos en JavaScript, Python, Java y C++":** https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/

- "**Entendiendo los Distintos Tipos de Algoritmos de Búsqueda": https://www.luigisbox.es/blog/tipos-de-algoritmos-de-busqueda/**

8. Anexos

- Repositorio en GitHub: SMontene22/TP-Integrador---Programacion-I

- Video explicativo: TP Integrador Programación I