UNIVERSIDAD TECNOLOGICA NACIONAL

TRABAJO INTEGRADOR - PROGRAMACIÓN I

Propuesta de Investigación

Métodos de Búsqueda y Ordenamiento

Grupo 7
Alumnos:
Monardes, Agustín
Pérez Campos, Joaquín

Materia: Programación I
Profesor: Ing. Laura Fernández

UNIVERSIDAD TECNOLOGICA NACIONAL

Índice

Introducción

En la programación, y en el mundo de la informática en general, los algoritmos de búsqueda y ordenamiento son fundamentales para la gestión eficiente de grandes volúmenes de datos. Los algoritmos de búsqueda son claves para recuperar y localizar datos de forma rápida y efectiva, mientras que los algoritmos de ordenamiento son cruciales para organizar estos datos según un criterio específico, lo que facilita su procesamiento y búsqueda.

Comprender cómo se implementan y en qué situaciones se aplican es clave para el desarrollo de software efectivo; un algoritmo de búsqueda y de organización efectivo reduce considerablemente el uso de recursos y el tiempo de procesamiento. Muchos algoritmos y estructuras de datos se basan directa o indirectamente de su capacidad de buscar y ordenar datos; desde las bases de datos hasta la automatización de las inteligencias artificiales.

La implementación de estos algoritmos son ejercicios fundamentales para desarrollar habilidades de programación, depuración y optimización de código, fomentando también una mentalidad de resolución de problemas, ya que introduce a los programadores a diferentes paradigmas de diseño de algoritmos.

Este trabajo explora estos algoritmos, incluyendo una explicación especial sobre la búsqueda binaria y el ordenamiento por burbujeo, por su relevancia en estructuras de datos ordenadas.

Marco Teórico

En su base, un algoritmo es una secuencia finita y definida de instrucciones para realizar una tarea; los algoritmos de búsqueda y de ordenamiento tienen como objetivo localizar un elemento específico o de reorganizar un conjunto de datos según un criterio.
La implementación de algoritmos y que tan eficientes puedan ser a menudo dependen de la estructura de datos en la que trabajan. Las colecciones lineales como listas y arrays son las estructuras más comunes para aplicar búsqueda y ordenamiento, pero también existen, por ejemplo, estructuras jerárquicas que intrínsecamente necesitan una lógica diferente al momento de buscar datos.
El diseño de un algoritmo no es solo una cuestión de repetición o de un seguimiento de pasos, sino de elegir un paradigma adecuado para abordar un problema. Así es como es que existen varios enfoques comunes para la resolución de problemas, como lo son:

- **La iteración**: es una repetición de un proceso hasta que se cumpla una condición deseada. Por ejemplo: el ordenamiento de burbuja es uno de los algoritmos de organización más comunes, ya que compara repetidamente el elemento adyacente y los intercambia si están en el orden incorrecto. En cada repetición, el elemento más grande o pequeño se va “burbujeando” hasta su posición final.

También es un buen ejemplo la búsqueda lineal; este algoritmo de búsqueda se utiliza para buscar un elemento específico en un lista, independientemente de si esa lista está ordenada o no. Lo que hace es recorrer la lista elemento por elemento hasta encontrar el objetivo o llegar al final de la lista.
Similar al burbujeo, el ordenamiento por inserción se utiliza en una lista ordenada en la que se construye un elemento a la vez. Cada nuevo elemento se toma de la parte no ordenada y se “inserta” en su posición correcta dentro de la parte ya ordenada.

- **La recursión**: sucede cuando una función se llama a sí misma para resolver el problema, habiendo un cambio cada vez que lo hace. Por ejemplo: una suma recursiva, sumando números del 1 a X hasta que 1 alcance el valor de X y cumplir con algún criterio.

Quizá el algoritmo de búsqueda más conocido sea el de búsqueda binaria, que funciona dentro de una lista ordenada y reduce a la mitad el espacio de búsqueda con cada paso para poder encontrar el índice de un elemento específico. Al estar ordenada la lista, si el objetivo que buscamos es menor al central, lo busca en la mitad izquierda; si es mayor, lo busca en la mitad derecha. Este paso se repite hasta que el elemento central sea el objetivo que estamos buscando.

- La **Fuerza Bruta:** es el enfoque más directo, que consiste en probar todas las posibles soluciones hasta encontrar la solución deseada. Es útil y fácil de implementar para problemas pequeños, cuando la simplicidad es más importante que la eficiencia. Podríamos decir que los algoritmos de búsqueda de burbuja y de inserción aplican también fuerza bruta en el sentido de que comparan y mueven elementos de forma repetitiva; no es que hay un salto directo para que la lista esté ordenada, sino que se realizan muchas comparaciones e intercambios.

La fuerza bruta puede ser utilizada también para el descubrimiento de una contraseña, probando todas las combinaciones posibles de caracteres hasta encontrar la correcta. Como esta forma explora todos los espacios posibles de caracteres, si la contraseña buscada es de una longitud considerable, el algoritmo puede tardar años hasta encontrar la contraseña buscada.

Divide y vencerás: este enfoque divide el problema en dos o más partes más pequeñas, resolviendo recursivamente cada problema y luego combinan sus soluciones para obtener la solución al problema original. Por ejemplo:

El ordenamiento por fusión divide la lista a ordenar en dos mitades, ordena recursivamente las mitades y las fusiona en dos listas ordenadas.

El ordenamiento rápido es similar al método anterior, pero elige un punto específico que parte la lista en dos.

Como se mencionó antes, la búsqueda binaria divide el espacio de la búsqueda a la mitad en cada paso, por lo que puede considerarse también un algoritmo que divide y vence.

UNIVERSIDAD TECNOLOGICA NACIONAL

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.
Código de ejemplo:

```
def ordenar_alumnos(lista):
    x = len(lista)
    print(f"Iniciando ordenamiento por burbujeo. Lista original:
{lista}")

    for i in range (x-1):
        cambio=False
        for y in range(0, x-i-1):
            if lista[y] > lista[y+1]:
                lista[y], lista[y+1]=lista[y+1], lista[y]
                cambio=True

        if not cambio:
            break
```

Método de ordenamiento burbuja:
Este algoritmo de ordenamiento, toma como argumento una lista (en este caso de nombres) y utiliza una variable *x*, a la que se le asigna el valor de la longitud de la lista a ordenar. Luego, mediante el uso de dos estructuras *for,* la primera desde el índice *i* hasta el valor de *x,* menos 1, y la segunda desde el índice *y* hasta el valor de la *x,* menos *i*, menos uno. En cada paso, va a comparar el valor del elemento en el índice *y,* con el valor del elemento en el índice *y+1,* de ser mayor el primero, va a intercambiar el valor en cada posición.
La variable *cambio*, iniciada como *False,* va a servir para que cuando no haya más cambios posibles, el ciclo *for* corte, mediante la sentencia *break.*

```python
def busqueda_binaria_alumno(lista_ordenada, nombre_objetivo):
    print(f"Buscando alumno: {nombre_objetivo}")

    inicio=0
    fin=len(lista_ordenada)-1
    paso=0

    while inicio <= fin:
        paso+=1
        medio=(inicio+fin)//2
        nombre_medio=lista_ordenada[medio]

        print(f"Paso {paso}: chequeando índice {medio} por
{nombre_medio}")

        if nombre_medio == nombre_objetivo:
            print(f"{nombre_objetivo} encontrado en el índice {medio}.
Busqueda finalizada en {paso} pasos")
            return nombre_medio, medio
        elif nombre_medio<nombre_objetivo:
            print(f"{nombre_objetivo} es alfabéticamente posterior a
{nombre_medio}")
            inicio=medio+1
        else:
            print(f"{nombre_objetivo} es alfabéticamente anterior a
{nombre_medio}")
            fin=medio-1
    print(f"{nombre_objetivo} no se encontro en la lista.")
    return None, -1
```

Método de búsqueda binaria:
Este algoritmo toma como argumento la lista donde vamos a buscar (lista_ordenada*)* y el valor que vamos a buscar (nombre_objetivo). Primero se debe definir las variables que indiquen las posiciones que vamos a recorrer en la lista ya ordenada, en este caso: *inicio*, iniciada en 0, indica que vamos a recorrer desde la primera posición de la lista; y *fin*, que va a tomar el valor de la longitud de la lista, menos uno, indicando la última posición de la lista. La variable *paso,* iniciada en 0 va a servir para imprimir mensajes que indiquen cuántos pasos nos toma encontrar el nombre objetivo. Luego, dentro de una estructura *while,* con la

condición *“mientras inicio sea menor o igual a fin”,* se crea la variable *medio,* inicializada cómo la división entera por dos de la suma de los valores de *inicio* y *fin,* y la variable *nombre_medio* al que se le asigna el valor de posición *medio* de la lista que vamos a comparar con el valor buscado *nombre_objetivo.* Dentro de esta estructura, vamos a sumar uno a la variable *paso;* un *if* va a evaluar la condición *“el valor del elemento en la posición medio de la lista es igual a nombre_objetivo”*, de cumplirse retornará el valor de *nombre_objetivo y el* valor de posición en el que se encontró; sino (sección *elif*) si el valor del elemento en la posición *medio* de la lista, es menor alfabeticamente que el valor objetivo *nombre_objetivo*, lo informará en un mensaje y cambiará el valor de *inicio* a la suma de *medio + 1* para “achicar” el rango de búsqueda en la siguiente iteración; sino (sección *else*), si el valor de *medio* es alfabeticamente anterior a *nombre objetivo*, lo informará en un mensaje en pantalla y cambiará el valor de *fin* a *medio - 1*. De esta forma, se busca recorrer toda la lista hasta encontrar el valor buscado, de encontrarlo, retornará el valor del elemento encontrado y el valor de posición en la que se encuentran, y de no encontrar el valor objetivo, retornará un *None* y “-1” indicando que no se encontró.

UNIVERSIDAD TECNOLOGICA NACIONAL

Metodología Utilizada

Para la realización del trabajo decidimos organizar reuniones virtuales, a través de Meet, para compartir ideas y plantear los pasos a seguir para concreción de este Trabajo Integrador.
Primeramente decidimos buscar material teórico, en documentación confiable, sobre el tema elegido, para luego plasmar la información, en el marco teórico de este trabajo.
Una vez definido el marco teórico, pasamos a plantear un caso de ejemplo donde se necesite aplicar métodos de búsqueda y ordenamiento, para lo cual planteamos el caso de un archivo, donde se necesita un programa que, buscando por apellidos, devuelva en qué espacio de archivo se encuentra la documentación del estudiante. El programa a desarrollar deberá tomar una lista desordenada de apellidos y ordenarla, para luego buscar por apellido su posición en la lista, que corresponderá a un espacio de archivo específico donde guardar o buscar la documentación de dicho estudiante. Para este fin decidimos crear un algoritmo de ordenamiento burbuja y un algoritmo de búsqueda binaria.
El código fue realizado en base al caso práctico de ejemplo, brindado por la cátedra, haciendo las modificaciones necesarias para trabajar con cadenas de texto, en lugar de números enteros.

Probamos el programa pasando como entrada, listas de cadenas de caracteres con distintas longitudes, no ordenadas alfabéticamente, que contengan cadenas de texto con mayúsculas y minúsculas, para probar la función de ordenamiento; y para probar la función de búsqueda, pasamos como entrada cadenas de texto que sabíamos que estaban incluidas en la lista, y cadenas que sabíamos que no, con diferencias de mayúsculas y minúsculas.
Como prueba de entrada ingresamos la lista:

```
lista_nombres=["Agustin", "Sebastian", "Camila", "Diego", "Fabian",
"Carla", "Lucas", "Juan", "Roman", "Tatiana", "Hugo", "Eugenia",
"Daiana"]
```

Y buscamos la cadena “Juan” con la función de búsqueda binaria.

Durante las pruebas, se realizó el registro de los resultados y la captura de pantallas para plasmar en este informe, y una vez comprobado el funcionamiento del programa, se realizó la grabación, edición y carga a YouTube del video explicativo del programa.

UNIVERSIDAD TECNOLOGICA NACIONAL

Resultados Obtenidos

El programa ordena correctamente la lista de nombres ingresada y realiza correctamente la búsqueda, informando la posición en la lista en que se encuentra el nombre buscado, o informa que no se ha encontrado el nombre en la lista, en caso de que no esté incluído.
Esto nos permitió ver cómo funciona paso a paso los algoritmos con los que elegimos trabajar: en el ordenamiento por burbujeo pudimos entender visualmente las comparaciones y los intercambios adyacentes entre elementos en una lista para poder ser ordenadas.
También comprobamos lo útil que es el algoritmo de la búsqueda binaria. A diferencia de la búsqueda lineal que recorre la lista elemento por elemento, la búsqueda binaria descarta la mitad de los elementos de una lista con cada paso que realiza, dividiendo repetidamente el problema, lo que resulta en un número menor de comparaciones.
El programa ilustra cómo los algoritmos trabajan juntos y, en parte, cómo la eficiencia de uno depende también de la eficacia del resultado del otro, ya que en este caso el algoritmo de búsqueda no sería posible si no hubiese un algoritmo de ordenamiento que justamente ordene la lista.
Los conceptos de iteración y recursión son dos enfoques que se ven en el programa detallado y son fundamentales para la resolución del problema, tanto la repetición controlada del burbujeo para lograr ordenar la lista, como el proceso de reducción progresiva en la búsqueda binaria.

UNIVERSIDAD TECNOLOGICA NACIONAL

Conclusiones

Los algoritmos de búsqueda y ordenamiento son pilares esenciales de la informática. Comprender los principios algoritmos es lo que sustenta una programación eficaz/
Este trabajo reforzó la importancia de seleccionar el algoritmo adecuado según el contexto y el tipo de datos a procesar; un algoritmo es la base de un buen software. No todos son igual de eficientes, por lo que uno debe determinar cuál es el mejor a implementar para que el rendimiento de un programa sea lo mejor posible.
Aplicar estos conceptos en un contexto un poco más realista, cómo lo es la gestión de una lista de nombres de alumnos, hace que esos algoritmos abstractos sean soluciones prácticas que, al fin y al cabo, es la base fundacional de la programación: poder resolver un problema y mejorar su proceso.

UNIVERSIDAD TECNOLOGICA NACIONAL

Bibliografía

- Problem Solving with Algorithms and Data Structures using Python - Miller, B., & Ranum, D.
- Python Oficial: https://docs.python.org/3/library/
- Khan Academy: https://es.khanacademy.org/computing/computer-science/algorithms

UNIVERSIDAD TECNOLOGICA NACIONAL

Anexos

Captura de pantalla del código completo del programa

```
def ordenar_alumnos(lista):
    x = len(lista)
    print(f"Iniciando ordenamiento por burbujeo. Lista original:
{lista}")

    for i in range (x-1):
        cambio=False
        for y in range(0, x-i-1):
            if lista[y] > lista[y+1]:
                lista[y], lista[y+1]=lista[y+1], lista[y]
                cambio=True

        if not cambio:
            break

    print(f"Lista ordenada: {lista}")

def busqueda_binaria_alumno(lista_ordenada, nombre_objetivo):
    print(f"Buscando alumno: {nombre_objetivo}")

    inicio=0
    fin=len(lista_ordenada)-1
    paso=0

    while inicio <= fin:
        paso+=1
        medio=(inicio+fin)//2
        nombre_medio=lista_ordenada[medio]

        print(f"Paso {paso}: chequeando índice {medio} por
{nombre_medio}")

        if nombre_medio == nombre_objetivo:
            print(f"{nombre_objetivo} encontrado en el índice {medio}.
Busqueda finalizada en {paso} pasos")
            return nombre_medio, medio
        elif nombre_medio<nombre_objetivo:
```

```
            print(f"{nombre_objetivo} es alfabéticamente posterior a
{nombre_medio}")
            inicio=medio+1
        else:
            print(f"{nombre_objetivo} es alfabéticamente anterior a
{nombre_medio}")
            fin=medio-1
    print(f"{nombre_objetivo} no se encontro en la lista.")
    return None, -1


lista_nombres=["Agustin", "Sebastian", "Camila", "Diego", "Fabian",
"Carla", "Lucas", "Juan", "Roman", "Tatiana", "Hugo", "Eugenia",
"Daiana"]

print(f"Lista de alumnos sin ordenar: {lista_nombres}")

lista_ordenada=lista_nombres[:]
ordenar_alumnos(lista_ordenada)

buscar_alumno="Juan"

nombre_encontrado,
indice_encontrado=busqueda_binaria_alumno(lista_ordenada,
buscar_alumno)

if indice_encontrado !=-1:
    print(f"El nombre {nombre_encontrado} se encuentra en el archivo
bajo el índice {indice_encontrado}.")
else:
    print(f"{buscar_alumno} no se encuentra en el archivero")

print("Ahora, ingrese un nombre para buscar en el archivero.")

nombre_buscado = input("Ingrese el nombre a buscar: ").strip()

nombre_encontrado_original, indice_encontrado =
busqueda_binaria_alumno(lista_ordenada, nombre_buscado)

if indice_encontrado != -1:
```

```
    print(f"El alumno {nombre_encontrado_original} se encuentra en el archivo en el índice {indice_encontrado}.")
else:
    print(f"{nombre_buscado} no se encuentra en el archivero.")
```

Impresiones en consola al correr el programa

```
PS C:\Users\Usuario> & C:/Users/Usuario/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/Usuario/Documents/TEC PROG UTN/PROGRAMACION I/python/TP-algoritmos.py"
Lista de alumnos sin ordenar: ['Agustin', 'Sebastian', 'Camila', 'Diego', 'Fabian', 'Carla', 'Lucas', 'Juan', 'Roman', 'Tatiana', 'Hugo', 'Eugenia', 'Daiana']
Iniciando ordenamiento por burbujeo. Lista original: ['Agustin', 'Sebastian', 'Camila', 'Diego', 'Fabian', 'Carla', 'Lucas', 'Juan', 'Roman', 'Tatiana', 'Hugo', 'Eugenia', 'Daiana']
Lista ordenada: ['Agustin', 'Camila', 'Carla', 'Daiana', 'Diego', 'Eugenia', 'Fabian', 'Hugo', 'Juan', 'Lucas', 'Roman', 'Sebastian', 'Tatiana']
Buscando alumno: Juan
Paso 1: chequeando índice 6 por Fabian
Juan es alfabéticamente posterior a Fabian
Paso 2: chequeando índice 9 por Lucas
Juan es alfabéticamente anterior a Lucas
Paso 3: chequeando índice 7 por Hugo
Juan es alfabéticamente posterior a Hugo
Paso 4: chequeando índice 8 por Juan
Juan encontrado en el índice 8. Busqueda finalizada en 4 pasos
El nombre Juan se encuentra en el archivo bajo el índice 8.
Ahora, ingrese un nombre para buscar en el archivero.
Ingrese el nombre a buscar: Roman
Buscando alumno: Roman
Paso 1: chequeando índice 6 por Fabian
Roman es alfabéticamente posterior a Fabian
Paso 2: chequeando índice 9 por Lucas
Roman es alfabéticamente posterior a Lucas
Paso 3: chequeando índice 11 por Sebastian
Roman es alfabéticamente anterior a Sebastian
Paso 4: chequeando índice 10 por Roman
Roman encontrado en el índice 10. Busqueda finalizada en 4 pasos
El alumno Roman se encuentra en el archivo en el índice 10.
PS C:\Users\Usuario>
```

Repositorio en GitHub: https://github.com/JoacoPerezCampos/TPI_ProgramacionI_g7
Video explicativo en Youtube: https://youtu.be/WRTWaQGR890