Trabajo Práctico: Gestión de Contactos con Algoritmos de Búsqueda y Ordenamiento en Python

• Alumnos: joaquín del valle lietti - delvallelietti@gmail.com
• Materia: Programación I
• Fecha de Entrega: 08-06-2025

1. Introducción

Este trabajo práctico aborda la gestión de contactos utilizando algoritmos básicos de búsqueda y ordenamiento implementados en Python. Se eligió este tema por su relevancia en la programación inicial, ya que permite comprender conceptos fundamentales como estructuras de datos simples, manejo de archivos, y algoritmos clásicos. El objetivo es desarrollar una aplicación sencilla que permite almacenar contactos, buscarlos de manera eficiente y mostrar la lista ordenada alfabéticamente, aplicando algoritmos de búsqueda secuencial y ordenamiento por burbuja.

2. Marco Teórico

Los algoritmos de búsqueda y ordenamiento son herramientas fundamentales en programación para organizar y encontrar datos dentro de colecciones.

Búsqueda secuencial: consiste en revisar uno a uno los elementos hasta encontrar el deseado. Es sencillo pero poco eficiente para grandes conjuntos.

Ordenamiento por burbuja (bubble sort): compara elementos adyacentes y los intercambia si están en orden incorrecto. Repite esto varias veces hasta que la lista queda ordenada. Su complejidad es $O(n^2)$, siendo adecuado para listas pequeñas.

3. Caso Práctico

```
def bubble_sort(contactos):
    n = len(contactos)
    for i in range(n):
        for j in range(0, n - i - 1):
            nombre1 = contactos[j].split(',')[0].lower()
            nombre2 = contactos[j + 1].split(',')[0].lower()
            if nombre1 > nombre2:
                contactos[j], contactos[j + 1] = contactos[j + 1], contactos[j]
    return contactos
```

```
def agregar_contacto(nombre, telefono):
    with open("contactos.txt", "a") as archivo:
        archivo.write(f"{nombre},{telefono}\n")
    print("Contacto agregado.\n")

def buscar_contacto(nombre):
    encontrado = False
    with open("contactos.txt", "r") as archivo:
        for linea in archivo:
            if linea.lower().startswith(nombre.lower() + ","):
                print("Contacto encontrado:", linea.strip())
                encontrado = True
                break
    if not encontrado:
        print("Contacto no encontrado.\n")

def mostrar_contactos():
    try:
        with open("contactos.txt", "r") as archivo:
            contactos = archivo.readlines()
            contactos = bubble_sort(contactos)
            print("\nContactos ordenados alfabéticamente:")
            for contacto in contactos:
                print(contacto.strip())
            print()
    except FileNotFoundError:
        print("No hay contactos aún.\n")

def menu():
    while True:
        print("1. Agregar contacto")
        print("2. Buscar contacto")
        print("3. Mostrar todos los contactos")
        print("4. Salir")
        opcion = input("Seleccione una opción: ")

        if opcion == "1":
            nombre = input("Nombre: ")
            telefono = input("Teléfono: ")
            agregar_contacto(nombre, telefono)
        elif opcion == "2":
            nombre = input("Nombre a buscar: ")
            buscar_contacto(nombre)
        elif opcion == "3":
```

```
            mostrar_contactos()
        elif opcion == "4":
            print("Programa finalizado.")
            break
        else:
            print("Opción no válida.\n")

menu()
```

Explicación del código

bubble_sort(contactos): recibe la lista de contactos (cada contacto es una línea "nombre,teléfono") y la ordena alfabéticamente por nombre. Se compara el nombre en minúsculas para evitar errores por mayúsculas/minúsculas.

Agregar_contacto: añade un contacto al final del archivo.

buscar_contacto: realiza una búsqueda secuencial en el archivo para encontrar el contacto cuyo nombre coincida.

mostrar_contactos: lee todos los contactos, los ordena usando bubble sort, y los muestra en pantalla.

menú: controla la interacción con el usuario, llamando a las funciones según la opción seleccionada.

4. Metodología Utilizada

Se realizó una investigación previa sobre algoritmos básicos de búsqueda y ordenamiento en documentación oficial de Python y algunos trabajos para cursos. Se diseñó la estructura del programa basada en funciones simples para cada tarea (agregar, buscar, mostrar). Se implementó y probó cada función individualmente para asegurar su correcto funcionamiento. Se usó un archivo de texto como almacenamiento persistente para facilitar la gestión de datos. El código fue escrito con comentarios y con estructura clara para facilitar su comprensión.

5. Resultados Obtenidos

Se logró implementar un programa funcional que permite agregar, buscar y mostrar contactos ordenados. La búsqueda secuencial encontró contactos correctamente incluso en listas con varios registros. El ordenamiento por burbuja ordenó alfabéticamente sin problemas los contactos cargados. Se detectaron y corrigieron errores menores en la comparación de cadenas para

evitar problemas de mayúsculas y minúsculas. El manejo de archivo funcionó correctamente incluso cuando el archivo estaba vacío o no existía.

6. Conclusiones

El trabajo permitió comprender la importancia de los algoritmos básicos en problemas cotidianos de programación, como la gestión de datos. La implementación de búsqueda secuencial y ordenamiento burbuja mostró la relación entre teoría y práctica en Python. Además, se valoró el uso de archivos para almacenamiento persistente y la organización modular del código para facilitar su mantenimiento. Para futuras mejoras, se podría optimizar el ordenamiento usando algoritmos más eficientes o implementar búsquedas más rápidas como la búsqueda binaria en listas ordenadas.

7. Bibliografía

3.13.4 Documentation

Apuntes de la clase + trabajos personales

8. Anexos

repositorio GitHub con código fuente disponible: DismalOmen/UTN-TUPaD-P1: Repositorio de ejemplo para los estudiantes de la Tecnicatuca Universitaria en Programación a distancia de la Universidad Tecnológica Nacional.

link al video: tp integrador - programación I - algoritmos de búsqueda y ordenamiento