PRACTICA 01

Análisis y Diseño de algoritmos

3CM1

Jorge Luis Vargas Hernández

15 de septiembre de 2023

Capítulo 1

Introducción

1.1. Objetivos

El objetivo principal de esta práctica es que los estudiantes adquieran una comprensión sólida y profunda de los conceptos clave relacionados con la complejidad computacional de los algoritmos. Los tres casos principales a analizar, mejor caso, peor caso y caso promedio, permiten una evaluación completa del rendimiento de un algoritmo bajo diferentes circunstancias.

1.2. Introducción a la práctica

En esta práctica se elaborará un código en Python que busque un número en un arreglo de tamaño n. Para ello, el usuario podrá elegir entre dos tipos de ordenamiento: el ordenamiento burbuja y el ordenamiento burbuja optimizada. El objetivo de la práctica es, mediante tres ejemplos por cada tipo de ordenamiento, clasificar, comparar y analizar según qué tipo de ordenamiento y qué entrada de datos es la notación Big O a la que pertenece ese algoritmo.

Capítulo 2

Desarrollo de la práctica

2.1. Implementación de los algoritmos

2.1.1. Se deben escribir en Python los siguientes métodos de ordenamiento: Burbuja y Burbuja Optimizada.

2.2. Análisis de Casos

2.2.1. Desarrollar un reporte con lo siguiente:

- **2.2.2. Calcular el mejor, peor y caso promedio para ambos algoritmos de ordenamiento, dando 3 ejemplos de datos de entrada para cada caso.**

- **2.2.3. Encontrar y justificar a cuál clase de las siguientes pertenece cada caso de cada algoritmo:**

  - $O(1)$
  - $O(\log n)$
  - $O(n)$
  - $O(n \log n)$
  - $O(n^2)$
  - $O(2^n)$

2.3. Ordenamiento burbuja

En el mejor caso tenemos una notación de O(n), este se produce cuando el arreglo ya viene ordenado, por lo que solo se hace una pasada que tarda en función del tamaño del arreglo.

Cuando se trata de un caso normal, tenemos un $O(2^n)$, y se produce porque el algoritmo realizará una cantidad de comparaciones e intercambios normalmente significativa, lo que resulta en una complejidad cuadrática.

```python
# Función para realizar el ordenamiento burbuja
def burbuja(bubbles):
    nbu = len(bubbles) #Función que sirve para obtener el numero de elementos de un arreglo
    flag = True
    while flag:
        flag = False
        for i in range(1, nbu): #Iteración de los elementos del arreglo desde 1 hasta el total de elementos
            if bubbles[i - 1] > bubbles[i]:
                bubbles[i - 1], bubbles[i] = bubbles[i], bubbles[i - 1]
                flag = True
        nbu -= 1
#El algoritmo de la función funciona de modo que la variable nbu (number_bubbles) se iguala al numero de variables que hay en el arreglo
#Luego el algoritmo declara una variable booleana llamada flag como verdadera. A continuación se hace un bucle mediante un while el cual no parara mientras la variable flag siga estando declarada como verdadera
#Dentro del while esta el ciclo for que se repetira los n elementos del arreglo (Desde 1 hasta el total de elemntos). Por cada iteración se pondra un if que pregunte si el elemento de la posición actual menos
# uno en el arreglo es mayor al elemento de la posición actual, en caso de que esta condición se cumpla, cambiara de posiciones los elementos, finalmente declarará flag como true. El ciclo continuará hasta que
#la condición if se deje de cumplir, en ese momento flag dejará de declararse como verdadera y el ciclo terminara, indicando que el ordenamiento a finalizado.

# Función para realizar el ordenamiento burbuja optimizada
def burbuja_op(bubbles):
    nbu = len(bubbles)
    for i in range(nbu):
        flag = False
        for j in range(0, nbu - i - 1):
            if bubbles[j] > bubbles[j + 1]:
                bubbles[j], bubbles[j + 1] = bubbles[j + 1], bubbles[j]
                flag = True
        if not flag:
            break
#Esta función tiene casi la misma lógica que el burbuja normal, tan solo es que este enfoque se evita hacer iteraciones innecesarias pues hace verificaciones para evitar hacer comparaciones en los arreglos que
#ya esten casi ordenados.

# Función para ingresar el tamaño del arreglo y los datos
def arreglo_dat_tam():
    tamaño = int(input("Ingrese el tamaño del arreglo: "))
    bubbles = []
    for i in range(tamaño):
        datos = int(input(f"Ingrese el elemento {i + 1}: "))
        bubbles.append(datos)
    return bubbles
#Esta función trata de asignar un input a una variable (tamaño del arreglo), esta variable definirá la cantidad de veces que se repetira la instrucción input("Ingrese un elemento"),
#cada elemento que se ingrese se concatenara en el arreblo bubbles con ayuda del comando .append

# Función principal
def main():
    print("Elige el tipo de ordenamiento:")
    print("1: Ordenamiento Burbuja Optimizada")
    print("2: Ordenamiento Burbuja")

    elec = int(input("Ingrese su elección (1 o 2): "))

    if elec == 1:
        bubbles = arreglo_dat_tam()
        burbuja_op(bubbles)
        print("Arreglo ordenado con ordenamiento burbuja:", bubbles)
```

Figura 2.1: Código de los algoritmos de ordenamiento.

El peor caso ocurre cuando el arreglo está ordenado en orden inverso, sin embargo, la complejidad se mantiene como en el caso normal, $O(2^n)$.

2.4. Ordenamiento burbuja optimizado

El mejor caso dentro del burbuja optimizado se da cuando el arreglo ya está ordenado O(n).

Aunque esta versión del algoritmo evita iteraciones innecesarias, en el caso normal, aún realiza una cantidad significativa de comparaciones e intercambios, lo que resulta en una complejidad cuadrática $O(2^n)$.

El peor caso sigue siendo cuando el arreglo está ordenado en orden inverso, lo que requiere el máximo número de comparaciones e intercambios $O(2^n)$.

2.4.1. Comparar el tiempo de ejecución de cada algoritmo de ordenamiento para el mejor, peor y caso promedio.¿Qué conclusiones puedes sacar de estos resultados?

Bueno podemos concluir que las notaciones en ambos sistemas de ordenamiento son iguales, para los dos el tiempo de ejecución incremento bajo las mismas condiciones, dependiendo del tamaño del arerglo y del ordenamiento inicial de sus elementos; el ordenamiento optimizado tenía la ventaja de que su estructura estaba diseñada para evitar dar menos vueltas innecesarias analizando de mejor manera las secciones del arreglo que ya se encontraban ordenadas. De

```python
#Esta función trata de asignar un input a una variable (tamaño del arreglo), esta variable definirá la cantid
#cada elemento que se ingrese se concatenara en el arreblo bubbles con ayuda del comando .append

# Función principal
def main():
    print("Elige el tipo de ordenamiento:")
    print("1: Ordenamiento Burbuja Optimizada")
    print("2: Ordenamiento Burbuja")

    elec = int(input("Ingrese su elección (1 o 2): "))

    if elec == 1:
        bubbles = arreglo_dat_tam()
        burbuja_op(bubbles)
        print("Arreglo ordenado con ordenamiento burbuja:", bubbles)
    elif elec == 2:
        bubbles = arreglo_dat_tam()
        burbuja(bubbles)
        print("Arreglo ordenado con burbuja optimizada:", bubbles)
    else:
        print("Esa opción no existe")

#Menú de siempre, hacemos un if para un menu de opciones, en cada opción la variable bubbles toma el tamaño d
# de la opción tomada en el menú y listo, la imprimimos.
if __name__ == "__main__":
    main()
```

Figura 2.2: Código de los algoritmos de ordenamiento.

```
Ingrese el tamaño del arreglo: 5
Ingrese el elemento 1: 5
Ingrese el elemento 2: 6
Ingrese el elemento 3: 7
Ingrese el elemento 4: 8
Ingrese el elemento 5: 9
Arreglo ordenado con ordenamiento burbuja: [5, 6, 7, 8, 9]
```

Figura 2.3: Gráfica del mejor caso de Burbuja.

```
1: Ordenamiento Burbuja Optimizada
2: Ordenamiento Burbuja
Ingrese su elección (1 o 2): 1
Ingrese el tamaño del arreglo: 6
Ingrese el elemento 1: 4
Ingrese el elemento 2: 8
Ingrese el elemento 3: 5
Ingrese el elemento 4: 9
Ingrese el elemento 5: 2
Ingrese el elemento 6: 0
Arreglo ordenado con ordenamiento burbuja: [0, 2, 4, 5, 8, 9]
PS C:\Users\s62j5\OneDrive\Documentos\Python Projects>
```

Figura 2.4: Gráfica del caso normal de Burbuja.

```
Ingrese su elección (1 o 2): 1
Ingrese el tamaño del arreglo: 8
Ingrese el elemento 1: 8
Ingrese el elemento 2: 7
Ingrese el elemento 3: 6
Ingrese el elemento 4: 5
Ingrese el elemento 5: 4
Ingrese el elemento 6: 3
Ingrese el elemento 7: 2
Ingrese el elemento 8: 1
Arreglo ordenado con ordenamiento burbuja: [1, 2, 3, 4, 5, 6, 7, 8]
PS C:\Users\s62j5\OneDrive\Documentos\Python Projects>
```

Figura 2.5: Gráfica del peor caso de Burbuja.

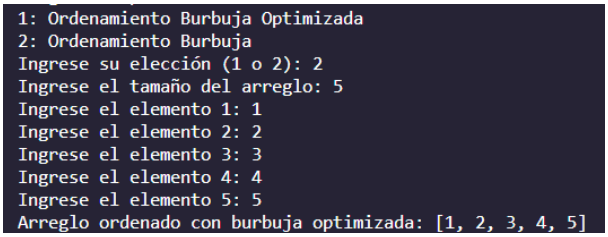

Figura 2.6: Gráfica del mejor caso de Burbuja Optimizada.

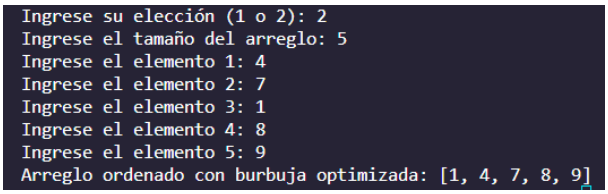

Figura 2.7: Gráfica del caso normal de Burbuja Optimizada.

todos modos, los tiempos de ejecución pueden llegar a ser no muy distintos, igual es bueno siempre usar la opción que nos ahorre la mayor cantidad de tiempo, no perdemos nada teniendo en cuenta que la estructura es muy similar entre ambas.

2.5. Conclusiones

Las comparaciones de esta práctica son interesantes para ir estudiando el como optimizar un algoritmo nos puede ahorrar tiempo y dinero, sin embargo, en este caso las diferencias entre optimizaciones no llegan a ser muy relevantes a mi parecer, aún más teniendo en cuenta que ambos métodos de ordenamiento se consideran como dos de los peores métodos que existen para hacer esta tarea de ordenar un arreglo. Sea como fuera, cuando tengamos que elegir entre un algoritmo y su versión optimizada, siempre hay que tomar la versión optimizada. Pda: Una disculpa por el ordenamiento de las paginas y de las imagenes sobre todo, batalle demasiado intentando hacer que se ordenaran pero no pude.

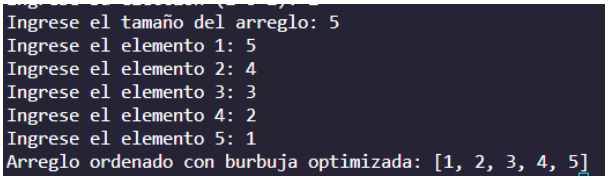

Figura 2.8: Gráfica del peor caso de Burbuja Optimizada.

2.6. Bibliografía

Amorin, D. (2023). Ordenamiento de burbuja en Python. Diego Amorin. https://diegoamorin.
burbuja-python/
Oscarfmdc. (2022). Big-O para principiantes. Aprender BIG DATA. https://aprenderbigdata.
o/