Teoría de Algoritmos

(75.29) Curso Buchwald - Genender

Trabajo Práctico Integrador

28 de diciembre de 2024

Marcos Tomás Weng Xu
109153

Leandro Elias Brizuela
109842

José Rafael Patty Morales
109843

Lucas Ezequiel Araujo
109867

Matias Rea
99770

Índice

1. Greedy

1.1. Objetivo

El objetivo de esta primera parte de la práctica es brindarle ayuda a Sophia, para que pueda salir victoriosa en todas las ocasiones en que juegue contra su hermano Mateo. Para esto se recurrirá a la técnica de diseño de resolución de problemas conocida como Greedy. A continuación estudiaremos cómo se comportará el algoritmo propuesto, aplicando dicha técnica, diseñando diferentes escenarios, con la finalidad de poder medir su eficiencia, analizando el tiempo de resolución, haciendo variaciones tanto en el largo del problema como en sus respectivos valores.

1.2. ¿Qué es un algoritmo de Greedy?

Un algoritmo Greedy, se basa en seguir una regla sencilla que nos permita obtener un óptimo local, esta se va a aplicar sucesivamente hasta llegar a un óptimo global (la mejor solución). Los beneficios que tiene utilizar esta técnica es que son intuitivos de pensar, fáciles de entender y pueden servir como aproximaciones a problemas complejos. Sin embargo, tiene sus desventajas:

- No siempre obtendremos el resultado óptimo ya que pueden existir casos en los cuales la regla a seguir no garantiza que lleguemos a un óptimo global. Pero sí una solución aproximada
- La demostración de que siempre dan el resultado óptimo puede ser difícil de hallar o explicar.

1.3. Análisis del problema y propuesta de regla Greedy

Como punto de partida para nuestro análisis, asumiremos que Sophia siempre inicia la partida.

En este problema tenemos que resolver el juego de la fila de monedas el cual consiste en:

- Se dispone una fila de n monedas con diferente valor.
- Se juega por turnos donde, en cada uno de estos, el jugador debe eligir entre la primera o última moneda de la fila.
- La moneda elegida por el jugador de turno se retirará de la fila de monedas y el valor de esta se sumara a su puntaje personal.
- Aquel jugador que obtenga el mayor puntaje ganará la ronda.

Lo que buscamos es que Sophia gane siempre (óptimo global). Esto se logra si se cumple que:

$$\sum_{i=1}^{n_s} s_i > \sum_{i=1}^{n_m} m_i$$

Donde:

- n_s representa la cantidad de monedas selecinados por Sophia.
- s_i es la moneda elegida por Sophia en el turno i.
- n_m representa la cantidad de monedas selecinados por Mateo.
- m_i es la moneda elegida por Mateo en el turno i.

Por esta razón, se propone implementar la siguiente regla Greedy:

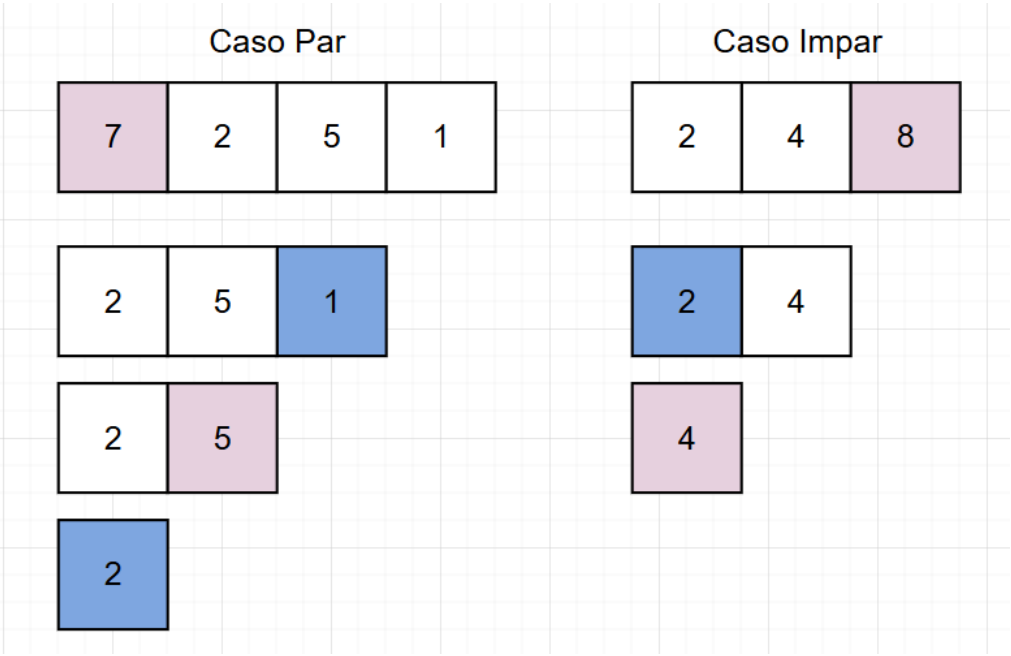
"Si es el turno de Sophia agarrar la moneda de mayor valor disponible. Si el turno es de Mateo seleccionar la moneda de menor valor disponible."

1.4. ¿La regla seleccionada, es Greedy?

Podemos afirmar que esta forma de resolución es Greedy porque en cada turno, Sophia elige la moneda de mayor valor disponible, que es la mejor decisión para ese momento, sin considerar consecuencias a futuro. Haciendo esto, estamos obteniendo el óptimo local para cada turno de Sophia porque estamos maximizando su ganancia y para el turno de mateo eligiendo la moneda de menor valor, estamos minimizando su ganancia.

2. Demostración de algoritmo Greedy óptimo

Este algoritmo Greedy siempre encuentra la solución óptima ya que, estamos haciendo que los valores de mayor valor posible siempre sean elegidos por Sophia y que Mateo ceda su mejor opción ya que siempre va a agarrar el valor de moneda más bajo. A continuación se mostrarán dos ejemplos, uno de largo par y otro de impar, para mostrar esto de forma visual.



Rosa: Sophia - Azul: Mateo

Como se puede observar en los ejemplos, en los turnos de Sophia, ella siempre obtendrá la de mayor valor, dejándole su peor opción a su hermano. Que es lo contrario a lo que sucede con Mateo, porque este siempre elige su peor opción, dejándole su mejor elección a su hermana. Teniendo esto en cuenta, Sophia siempre va a acumular mayor cantidad de puntos, lo que la hará la vencedora en todas las partidas que jueguen. En consecuencia, no encontramos un contraejemplo a dicho algoritmo. Esto tiene sentido ya que, no existe manera en la cual Mateo gane. Por la simple razón de que no tiene decisión respecto al juego, al ser Sophia la que elige las monedas, la misma tiene el control de cómo se desenlazará la partida, donde solo puede haber una opción y es donde ella gana. Entonces, con todo lo dicho, se corrobora que el algoritmo es óptimo.

2.1. ¿Afecta la variación de las monedas a la optimalidad del algoritmo?

La variabilidad de los valores de las monedas no afecta la optimalidad del algoritmo ya que, este hace que Sophia se quede con las monedas de mayor valor y Mateo con las de peor valor. Esto se genera debido a que Sophia siempre le deja las peores opciones a Mateo y este las mejores opciones a su hermana. Como vemos, la variabilidad de los valores no tiene ningún impacto en que el algoritmo sea óptimo. El único caso que podría afectar la optimabilidad es el caso en que todas las monedas tengan el mismo valor, en el cual se ocasionaría un empate, sin embargo, este no es tomado en cuenta para la realización del informe.

3. Implementación en código funcional del algoritmo

El código fue desarrollado en el lenguaje Python ya que, gracias a su simplicidad y su lenguaje de alto nivel permitió una rápida resolución al problema.

Como se verá a continuación en la función *"greedy strategy"*, inicializa dos listas, una para Sophia (*"sophia coins"*) y otra para Mateo (*"mateo coins"*). Posteriormente, se hará uso de un bucle while que recorrerá todas las monedas de la fila de monedas *coins* con el objetivo de simular los turnos. La primera en empezar sera Sophia y luego seguirá su hermano Mateo. En cada iteración se irán sumando las monedas que hayan elegido en ese turno. Una vez finalizado el juego, se devolverán dos listas, con las monedas que eligió cada uno.

```
def greedy_strategy(coins):
    sophia_coins = []
    mateo_coins = []
    while coins:
        sophia_play_taking_higher_coin(coins, sophia_coins)
        mateo_play_taking_lowest_coin(coins, mateo_coins)
    return sophia_coins, mateo_coins
```

Lo que se busca en la próxima función es agregar el máximo valor entre los extremos de la monedas, para poder garantizar la victoria de Sophia.

```
def sophia_play_taking_higher_coin(coins, sophia_coins):
    if coins[0]>=coins[-1]:
        sophia_coins.append((coins.pop(0), PRIMERA))
    else:
        sophia_coins.append((coins.pop(), ULTIMA))
```

En la siguiente función, se procederá agregar el menor valor de los extremos de la moneda, ocacionado asi que Mateo siempre eliga la peor elección posible en ese momento.

```
def mateo_play_taking_lowest_coin(coins, mateo_coins):
    if coins:
        if coins[0]>=coins[-1]:
            mateo_coins.append((coins.pop(), ULTIMA))
        else:
            mateo_coins.append((coins.pop(0), PRIMERA)
```

4. Análisis de complejidad del algoritmo

La complejidad del algoritmo planteado en este ejercicio es lineal, o sea **O(n)** , ya que:

- Utilizamos un ciclo while que va a iterar el tamaño del largo n de la cola de monedas.

```
    def greedy_strategy(coins):
    sophia_coins = []  #O(1)
    mateo_coins = []   #O(1)
    while coins:   #O(n)
        sophia_play_taking_higher_coin(coins, sophia_coins)  #O(1)
        mateo_play_taking_lowest_coin(coins, mateo_coins)  #O(1)
    return sophia_coins, mateo_coins   #O(1)

```

- Las comparaciones, asignaciones y sumas usadas tienen un costo de **O(1)**.

```
    def sophia_play_taking_higher_coin(coins, sophia_coins):
    if coins[0]>=coins[-1]:  #O(1)
        sophia_coins.append((coins.pop(0), PRIMERA))  #O(1)
    else:
        sophia_coins.append((coins.pop(), ULTIMA))  #O(1)

def mateo_play_taking_lowest_coin(coins, mateo_coins):
    if coins:  #O(1)
        if coins[0]>=coins[-1]:  #O(1)
            mateo_coins.append((coins.pop(), ULTIMA))  #O(1)
        else:
            mateo_coins.append((coins.pop(0), PRIMERA))  #O(1)

```

- Como se puede ver, se hace uso del método **.pop - .pop(0)**, que nos permite eliminar un elemento de la cola doblemente terminada (deque) que se encuentre ya sea al principio o al final de la misma. El costo de realizar este tipo de operación es **O(1)**. Ademas se realizan operaciones **.append**, las cuales también tienen un costo **O(1)**.

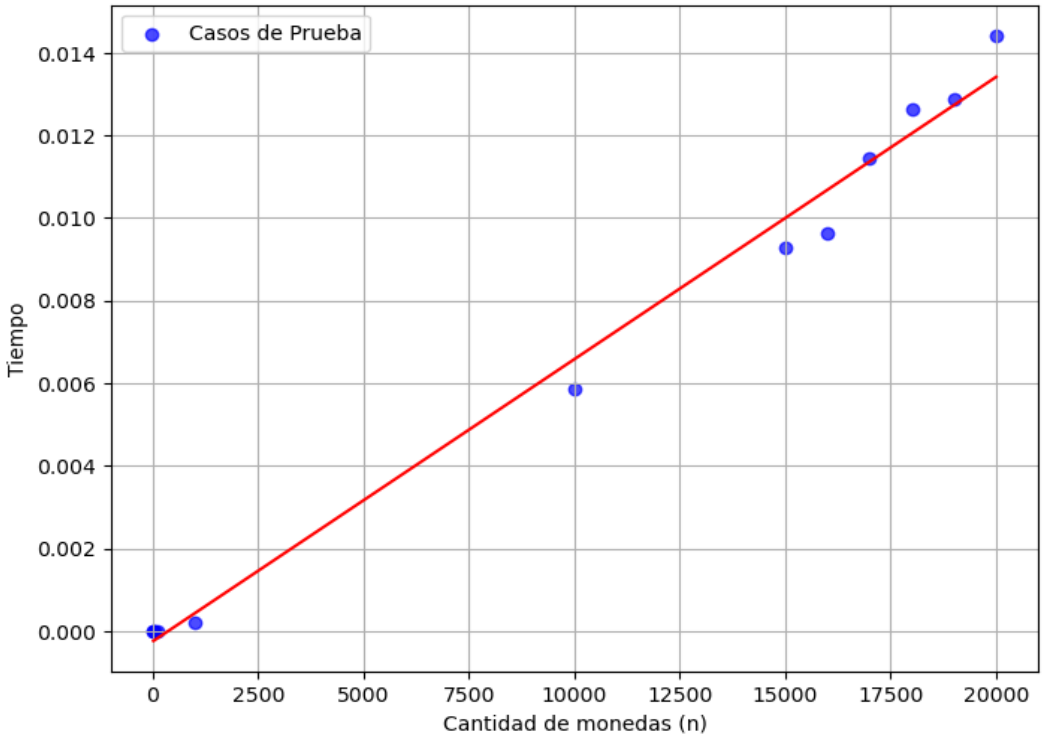
Entonces, se puede corroborar que el algoritmo planteado presenta una complejidad lineal, O(n), debido a que el ciclo principal itera sobre el tamaño de la cola de monedas, realizando operaciones constantes en cada iteración. Así, el tiempo de ejecución total del algoritmo está directamente relacionado con el número de elementos en la cola, lo que confirma que la complejidad es lineal.

5. Mediciones

En el siguiente apartado, se mostrarán los gráficos realizados para corroborar la complejidad teórica indicada anteriormente.

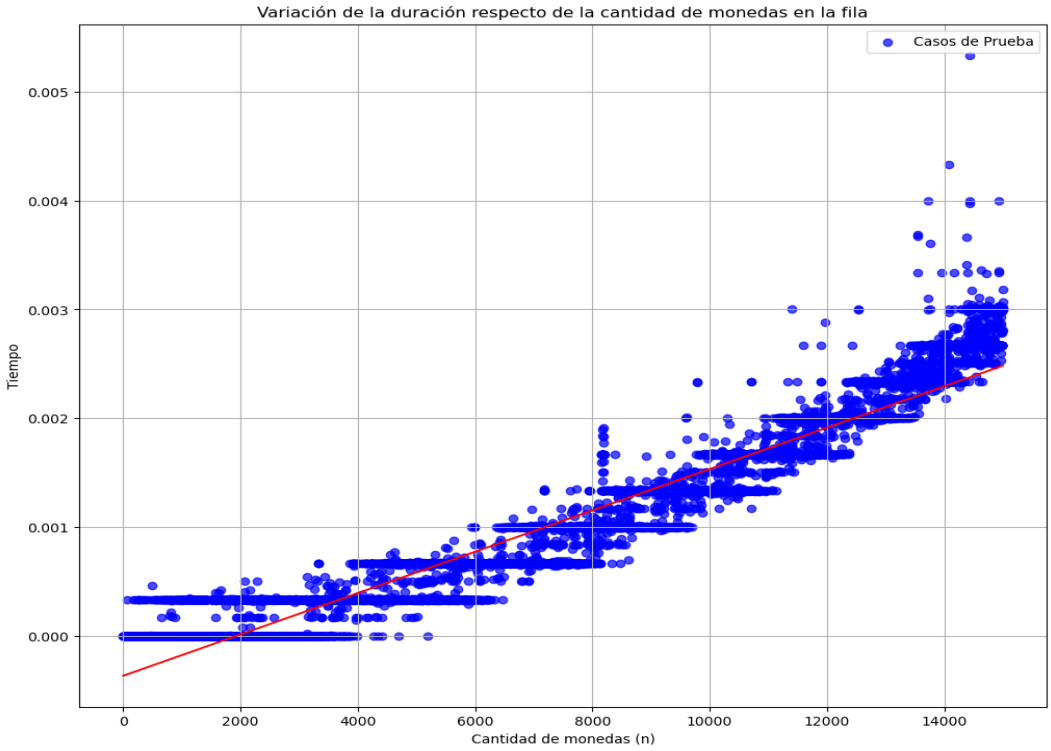
Para el primero de estos gráficos, se utilizaron los sets de datos dados por la cátedra y otros generados aleatoriamente (arreglos de distintos largos y valores generados con la librería **random** de Python) los cuales son:

Caso de prueba	Cantidad de monedas
20.txt	20
25.txt	25
50.txt	50
100.txt	100
1000.txt	1000
10000.txt	10000
15000.txt	15000
16000.txt	16000
17000.txt	17000
18000.txt	18000
19000.txt	19000
20000.txt	20000



En este gráfico se puede ver que, si bien son pocos casos de prueba, hay cierta tendencia lineal para los distintos tamaños de arreglos.

Para tener una mayor certeza de que nuestro algoritmo propuesto cumple con una complejdad lineal, también se lo evaluó de la siguiente manera: creamos arreglos de monedas con un largo incremental de 1 a 15000, donde cada posición tiene un valor aleatorio.



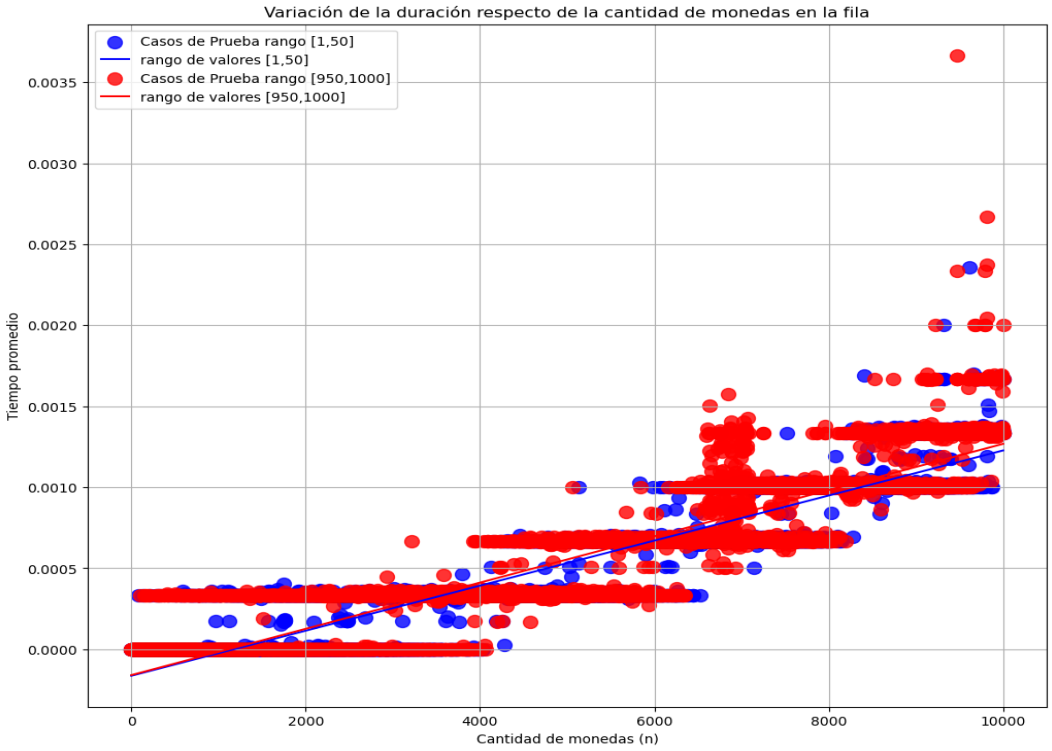
Basándonos en lo que se puede observar en el gráfico, podemos concluir que el algoritmo evaluado exhibe un comportamiento de complejidad temporal lineal. Esto significa que el tiempo de ejecución del algoritmo crece proporcionalmente al tamaño de entrada. Si bien lo que se observa es una dispersión alrededor de la línea esto se debe tanto a factores internos como costes en las operaciones del algoritmo como a factores externos como podría ser el rendimiento de la computadora en donde se corren los datos experimentales, aun así al ser aproximación podemos afirmar que es lineal.

5.1. Variablildad de los valores de las monedas

A continuación se estudiará cómo es que los distintos valores que puede tener las monedas afectan al rendimiento temporal del algoritmo. Para esto se decidió probar con casos de prueba generados aleatoriamente de la siguiente manera:

- Se usaron 10000 arreglos distintos tamaños (de 1 a 10000) los cuales contienen monedas con bajo valor, en un rango de [1 ; 50]
- Se usaron 10000 arreglos distintos tamaños (de 1 a 10000) los cuales contienen monedas con alto valor, en un rango de [950 ; 1000]

Para visualizar cómo impactan en el redimiento temporal, se realizó el siguiente gráfico con las pruebas mencionadas anteriormente:



Observando el gráfico, se puede ver que estos son bastante parecidos en cuanto a tiempos de ejecución. Esto comprueba que tener monedas con distintos valores no genera ningún impacto significativo en el performance temporal de nuestro algoritmo.

6. Conclusión

Una vez finalizada esta parte del trabajo práctico integrador, llegamos a la conclusión de que utilizar la técnica de diseño de problemas Greedy resulta ser muy eficiente para el problema de Sophia, siendo nuestro algoritmo fácil de hacer y de entender.

Para este caso, al momento de analizar la problemática en cuestión, la regla Greedy salió bastante rápido, siendo óptima y fácil de entender. Es óptima debido a lo mencionado anteriormente y porque no hay una variable que altere la optimalidad de nuestro algoritmo. Siempre va a hacer que Sopia obtenga la mejor opción posible y que Mateo obtenga la peor posible (óptimo local). Lo que generará que Sophia siempre gane ya que esta acumulará las monedas con los valores más altos (óptimo global).

Para poder estudiar en mayor profundidad la complejidad de nuestro algoritmo hicimos mediciones para poder graficarlas y ver que este tiene una cierta tendencia lineal, no es exactamente igual, pero esto es esperable ya que son datos experimentales y aproximados.