.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1: Los Algoritmos Greedy son juegos de niños

18 de septiembre de 2024

Santiago Nahuel Ruiz Sugliani, 106768

Marcos Bianchi Fernandez, 108921

Ariana Magalí Salese D'Assaro, 105558

Índice

1. Introducción

Mediante el presente informe presentamos un algoritmo greedy el cual resuelve de manera óptima el siguiente problema:

Contamos con un juego donde disponemos de una fila de n monedas, de diferentes valores. En cada turno, un jugador debe elegir alguna moneda. Pero no puede elegir cualquiera: sólo puede elegir o bien la primera de la fila, o bien la última. Al elegirla, la remueve de la fila, y le toca luego al otro jugador, quien debe elegir otra moneda siguiendo la misma regla. Siguen agarrando monedas hasta que no quede ninguna. Quien gane será quien tenga el mayor valor acumulado (por sumatoria).

En el juego estaremos tomando el papel de una niña de siete años de edad llamada Sophia la cual estará jugando con su hermano menor de unos cuatro años llamado Mateo. Considerando que Sophia es sumamente competitiva (por lo cual perder no es una opción para ella) y dado que Mateo no comprende el juego debido a su edad, la niña deberá "jugar por su hermano". Finalimente también debemos tener en cuenta que la niña siempre comienza (si bien ella juega por su hermano, el primer turno siempre le pertenece).

2. Análisis del problema y demostración

Para obtener la solución óptima a este problema, proponemos que Sophia siempre elija la moneda de mayor valor en su turno (o cualquiera de las dos en caso de que ambas tengan el mismo) mientras que en el turno correspondiente a Mateo, elija la de menor valor (o cualquiera de las dos en caso de que ambas tengan el mismo). De esta forma es sencillo de observar que en los casos donde la fila tiene una cantidad impar de monedas Sophia siempre ganará, mientras que en el caso de una cantidad par de monedas a lo sumo habrá un empate (en caso de que todas tengan el mismo valor).

Podemos decir que la fila de monedas es un arreglo de longitud n (siendo n igual a la cantidad de monedas de la fila) de la forma:

$$M = \{M_1, M_2, M_3, M_4, M_5, M_6, \ldots, M_n\}$$

Vamos al caso inicial, Sophia estará eligiendo la moneda de mayor valor entre M_1 y M_n, suponiendo que M_1 es mayor o igual a M_n entonces Sophia elije dicha moneda y el arreglo de monedas ahora tiene una moneda menos, con lo cual pasa a tener el siguiente estado:

$$M = \{M_2, M_3, M_4, M_5, M_6, \ldots, M_n\}$$

Ahora estamos en el turno de Mateo, con lo cual Sophia elije la moneda de menor valor entre M_2 y M_n por lo tanto vamos a suponer que M_2 cuenta con un valor menor o igual a M_n y Sophia elije la moneda en cuestión para su hermano, luego el arreglo queda en el siguiente estado:

$$M = \{M_3, M_4, M_5, M_6, \ldots, M_n\}$$

Ya teniendo claro el funcionamiento del algoritmo, procedemos a demostrar que el mismo es óptimo por el método inductivo.

Sean $\sum_{i=0}^{m} S_i$ la sumatoria de los valores de las monedas elegidas por Sophia y $\sum_{j=0}^{m} M_j$ la sumatoria de los valores de las monedas elegidas por Mateo. Siendo que m puede tomar como máximo un valor igual a la mitad del largo original del arreglo de monedas. Tengamos en cuenta que esto sucede tanto para Sophia como para Mateo en caso de que dicho arreglo cuente con una cantidad par de monedas en un inicio, ya que en caso de una cantidad impar de monedas, Mateo siempre tendrá un turno menos que su hermana.

Veamos que en el caso base con $m = 1$, para Sophia $\sum_{i=1}^{1} S_i = S_1$ y para Mateo $\sum_{j=1}^{1} M_j = M_1$, donde $S_1 \geq M_1$.

Ahora veamos el caso inductivo. Supongamos que para $m = h$ se cumple que $\sum_{i=1}^{h} S_i \geq \sum_{j=1}^{h} M_j$. Queremos demostrar que esto es cierto para $m = h + 1$.

Para Sophia,

$$\sum_{i=1}^{h+1} S_i = \sum_{i=1}^{h} S_i + S_{h+1}$$

y para Mateo,

$$\sum_{j=1}^{h+1} M_j = \sum_{j=1}^{h} M_j + M_{h+1}$$

Dado que $S_{h+1} \geq M_{h+1}$, y por la hipótesis inductiva $\sum_{i=1}^{h} S_i \geq \sum_{j=1}^{h} M_j$, se cumple que:

$$\sum_{i=1}^{h+1} S_i = \sum_{i=1}^{h} S_i + S_{h+1} \geq \sum_{j=1}^{h} M_j + M_{h+1} = \sum_{j=1}^{h+1} M_j$$

Por lo tanto, por el principio de inducción matemática, hemos demostrado que $\sum_{i=0}^{m} S_i \geq \sum_{j=0}^{m} M_j$ para cualquier n.

Podemos observar que entonces el valor acumulado por Sophia será mayor o igual al de Mateo en cada turno, el arreglo de monedas es modificado teniendo una moneda menos que el turno anterior. Aplicando esta lógica turno a turno hasta que no hayan más monedas Sophia tendrá un valor acumulado mayor a su hermano y resultará ganadora.

3. Algoritmo Planteado

3.1. ¿Es el algoritmo propuesto Greedy?

La regla Greedy es que Sophia en su turno tome la moneda de mayor valor y en el turno de Mateo él se quede con la moneda de menor valor. De esta forma maximizamos el valor acumulado de Sophia y minimizamos el valor acumuldo de mateo en cada ronda.

Aplicando esta regla sencilla de manera iterativa conseguimos resolver cada ronda de forma óptima y al final de la partida alcanzamos la solución óptima general.

3.2. Implementación

Se tienen un par de variables que delimitan el arreglo de monedas para acceder solo a sus extremos. En cada ronda los jugadores toman una de estas dos monedas y actualizan estos valores acordemente, Sophia (quien arranca) toma la moneda de mayor valor y Mateo la de menor.

Guardamos el valor acumulado de cada jugador por separado y cuando nos quedamos sin monedas para contar entonces devolvemos los puntajes de cada jugador.

```
def elegir_moneda(monedas, cmp, a, b):
    if cmp(monedas[a], monedas[b]):
        moneda = monedas[a]
        a += 1
    else:
        moneda = monedas[b]
        b -= 1

    return moneda, a, b


def jugar(monedas):
    valor_sophia = 0
    valor_mateo = 0

    primera = 0
    ultima = len(monedas) - 1

    while primera <= ultima:
        moneda, primera, ultima = elegir_moneda(monedas, lambda x, y: x > y,
    primera, ultima)
        valor_sophia += moneda

        if primera > ultima:
            break

        moneda, primera, ultima = elegir_moneda(monedas, lambda x, y: x < y,
    primera, ultima)
        valor_mateo += moneda

    return valor_sophia, valor_mateo
```

3.3. Complejidades

Todas las operaciones del algoritmo son de complejidad constante, excepto por el ciclo principal que itera sobre todas las monedas. Por lo tanto, la complejidad temporal del algoritmo es $O(n)$, donde n es la cantidad de monedas del juego.

La complejidad espacial del algoritmo es $O(1)$, pues no importa cuantas monedas recibamos, siempre vamos a tener dos indices para delimitar el rango de monedas dentro del arreglo.

3.4. Variabilidad de los valores y efectos en los tiempos del algoritmo

La variabilidad de los valores de las monedas no tiene efectos en los tiempos del algoritmo, pues a Sophia y a Mateo solo les interesa cual del par de monedas es mayor o menor, que es una operación de complejidad constante. No hay un cómputo extra porque haya menor o mayor variabilidad entre estos valores.

3.5. Variabilidad de valores y efectos en la optimalidad del algoritmo

La variabilidad de los valores de las monedas no tienen efecto sobre la optimalidad del algoritmo. Con el algoritmo propuesto siempre se encuentra la solución optima. Sean cuales sean los valores de las monedas, Sophia siempre llega al mejor caso posible, ganar o empatar en el caso donde haya cantidad par de monedas y todas del mismo valor.

4. Ejemplos

A continuación presentamos algunos ejemplos del algoritmo propuesto.

4.1. Ejemplo 1: cantidad par de monedas

Tenemos un arreglo de monedas de la forma:

$$M = [72, 165, 794, 892]$$

Como se mencionó anteriormente, Sophia siempre comienza. Realizamos entonces a continuación el seguimiento de los turnos de Sophia y Mateo.

Turno 1: Sophia

Dado que Shopia siempre elige la moneda de mayor valor, en este caso tomará la moneda de 892. De esta forma el arreglo de monedas y la ganacia de cada uno hasta el momento son:

$$M = [72, 165, 794]$$
$$Ganancia\ Sophia = 892$$
$$Ganancia\ Mateo = 0$$

Turno 2: Mateo

Ahora es el turno de Mateo, para el cual Sophia siempre elige la moneda de menor valor, en este caso, esa es la moneda de 72. Por lo tanto:

$$M = [165, 794]$$
$$Ganancia\ Sophia = 892$$
$$Ganancia\ Mateo = 72$$

Turno 3: Sophia

Nuevamente toma la moneda de mayor valor, es deicr, 794, resultando:

$$M = [165]$$
$$Ganancia\ Sophia = 892 + 794 = 1686$$
$$Ganancia\ Mateo = 72$$

Turno 4: Mateo

Finalimente, Mateo toma la única moneda restante, la de 165, por lo tanto:

$$Ganancia\ Sophia = 892 + 794 = 1686$$
$$Ganancia\ Mateo = 72 + 165 = 237$$

Sophia resulta ganadora con una ganancia de 1686 contra 237 de Mateo.

Dado que la cantidad de monedas es par, toman la misma cantidad de monedas pero gracias al algoritmo utilizado, desde el primer turno Shopia tiene mayor ganancia que Mateo.

4.2. Ejemplo 2: cantidad impar de monedas

Tenemos un arreglo de monedas de la forma:

$$M = [72, 165, 400, 794, 892]$$

Este arreglo es el dado en el ejemplo anterior con la diferencia de la moneda de 400 en el centro. Por su posición, el proceso resultará igual que el anterior pero Sophia tendrá un turno más que Mateo en el que tomará la moneda de 400. Por lo mencionado:

$$Ganancia\ Sophia = 892 + 794 + 400 = 2086$$

$$Ganancia\ Mateo = 72 + 165 = 237$$

Sophia resulta ganadora con una ganancia de 2086 contra 237 de Mateo.

Dado que en este caso la cantidad de monedas es impar, Sophia toma una moneda más que Mateo y extiende asi la ventaja que ya tenía en el ejemplo anterior.

4.3. Ejemplo 3: todas las monedas con el mismo valor

Tenemos un arreglo de monedas de la forma:

$$M = [10, 10, 10, 10]$$

Como se menciono anteriormente, al tratarse de cantidad de monedas par ambos tomarán la misma cantidad de monedas. En este caso todas las monedas tienen el mismo valor, por lo tanto el resultado final será un empate.

$$Ganancia\ Sophia = 10 + 10 = 20$$

$$Ganancia\ Mateo = 10 + 10 = 20$$

Por otro lado si la cantidad de monedas fuera impar, Sophia tomaría una moneda más que Mateo y resultaría ganadora.

$$Ganancia\ Sophia = 10 + 10 + 10 = 30$$

$$Ganancia\ Mateo = 10 + 10 = 20$$

A partir de los ejemplos presentados podemos observar que el algoritmo propuesto encuentra la solución que le da la victoria a Sophia en los casos posibles, maximizando su ganancia y minimizando la de Mateo.

4.4. Ejemplos de la catedra

Para ejecutar los ejemplos de la catedra y así verificar las respectivas ganancias, que la ganadora sea Sophia y comparar con la solución propuesta se puede ejecutar el programa de la siguiente manera:

```
python3 main.py
```

Se debe tener en cuenta que la ganancia de Sophia podría no ser la misma que la de la catedra, pero la ganadora siempre será Sophia.

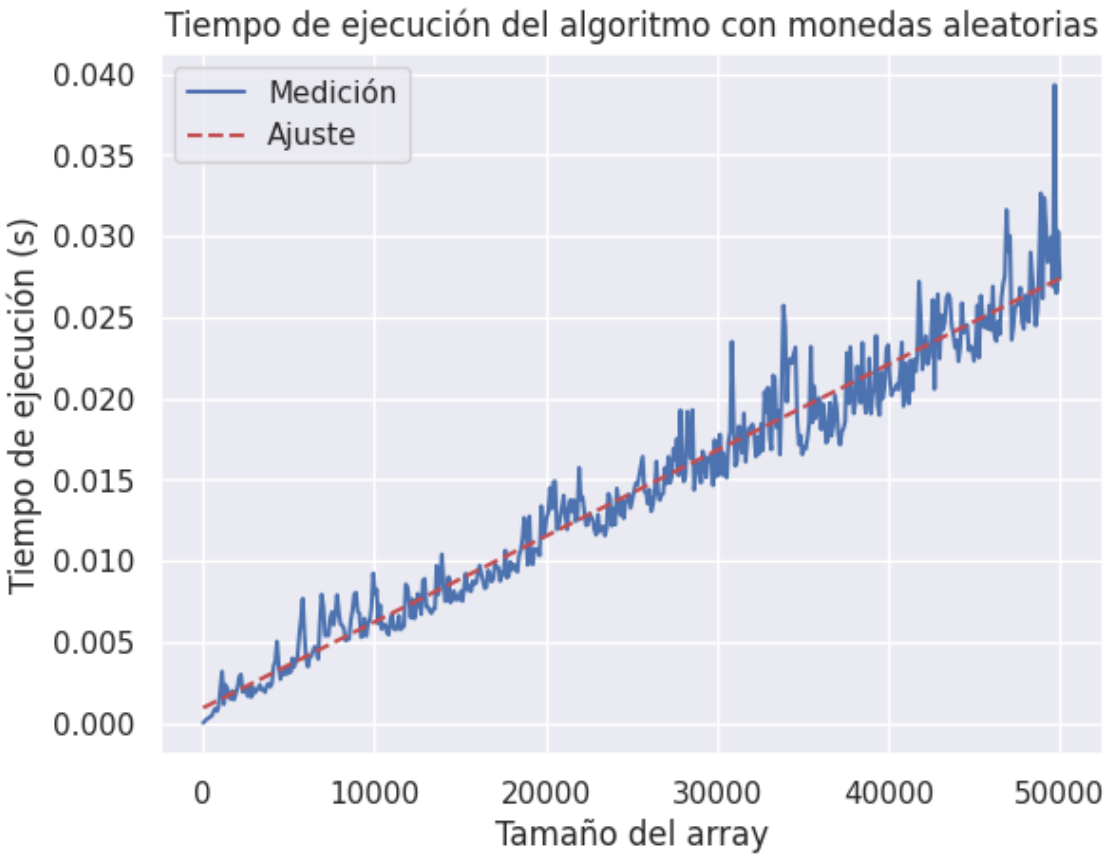
5. Mediciones

Se realizaron mediciones del algoritmo tomando distintas cantidades de monedas para demostrar empíricamente su complejidad teórica de $O(n)$. Se tomaron 500 muestras separadas uniformemente de 1 a 50.000 elementos.

Todos los gráficos muestran la recta que mejor se ajusta a los datos conseguida luego de aplicar el método de cuadrados mínimos.

5.1. Monedas aleatorias

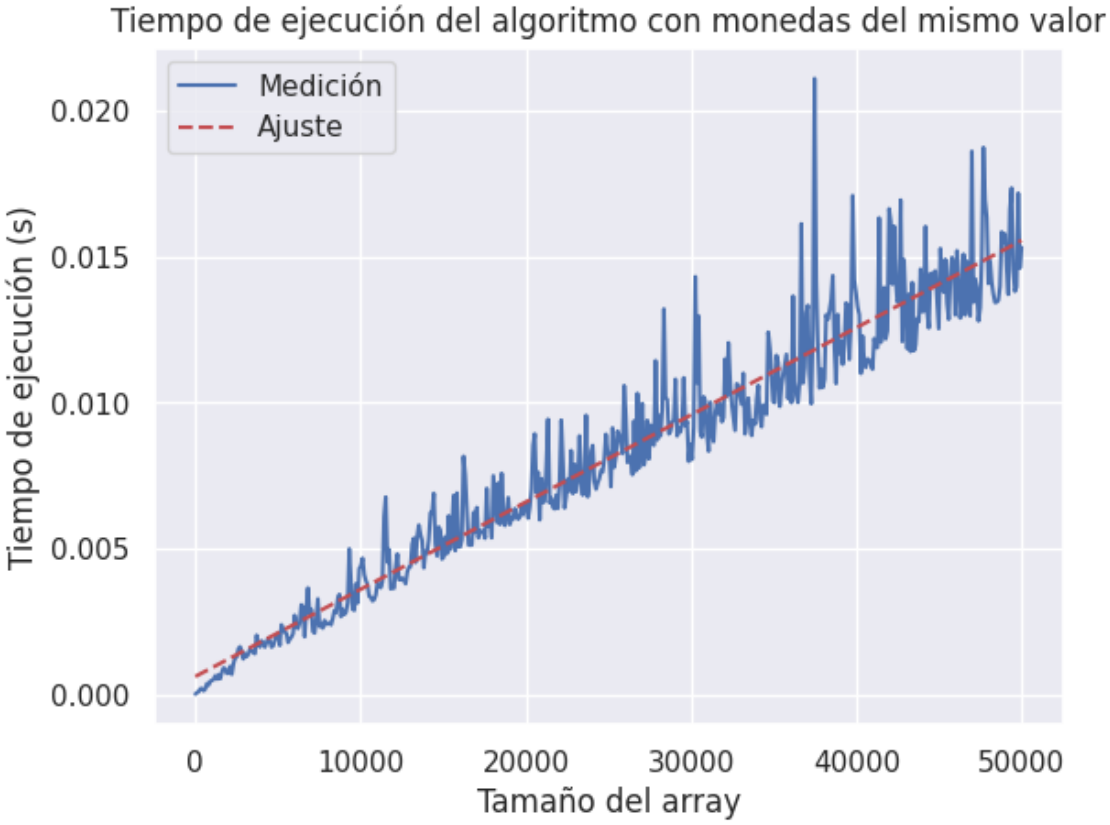
Esta medición se realizó utilizando el generador de números pseudo-aleatorios que provee la biblioteca `numpy` dentro de su módulo `random`



Como se puede observar, la complejidad temporal del algoritmo es lineal, tal como se esperaba.

5.2. Monedas del mismo valor

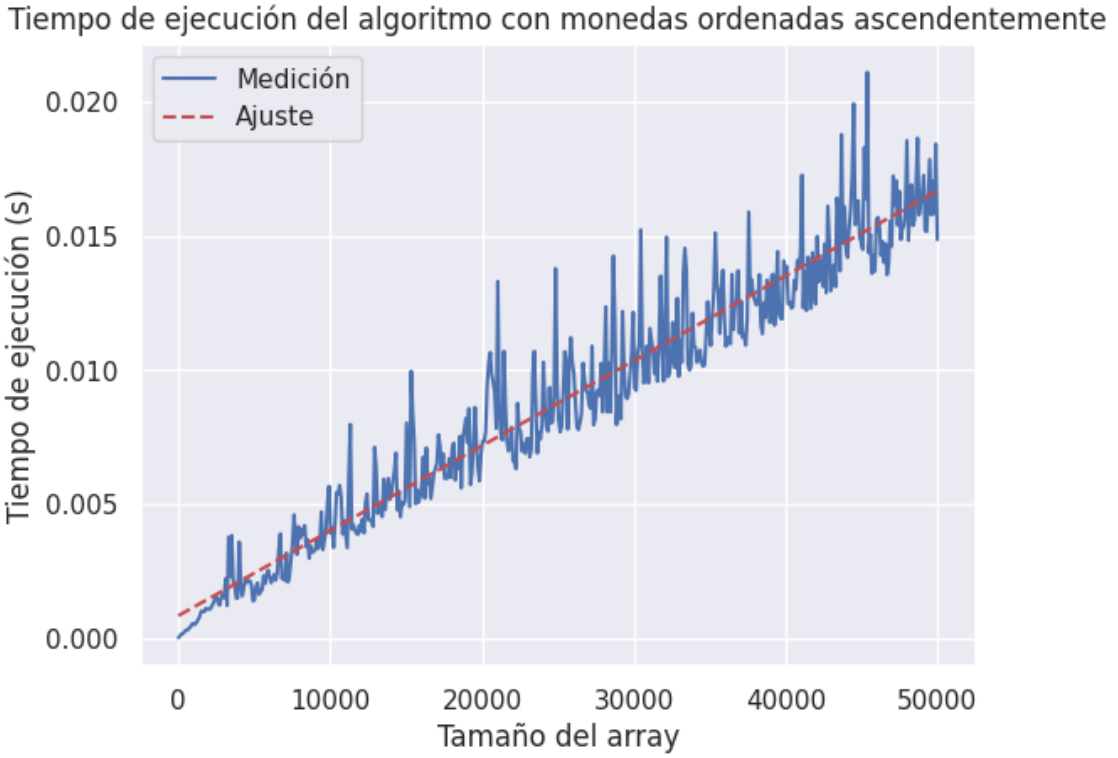
En el siguiente gráfico se plasma el tiempo de ejecución del algoritmo en función de la cantidad de monedas, todas con el mismo valor.



Nuevamente, podemos observar que la complejidad temporal del algoritmo es lineal, sin importar la igualdad de los valores de las monedas.

5.3. Monedas ordenadas ascendentemente

Por último, se midió el tiempo de ejecución del algoritmo en función de la cantidad de monedas, todas ordenadas de forma ascendente.



Así como en los casos anteriores, la complejidad temporal del algoritmo es lineal.

6. Conclusiones

Para resolver el problema planteado se propuso un algoritmo Greedy y se demostró que resuelve de forma óptima siempre. Además se realizaron mediciones para constatar su complejidad temporal lineal de forma empírica y se comprobó que la variabilidad de los valores no afecta su performance ni el resultado del algoritmo.