Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2: Programación Dinámica For The Win

23 de diciembre 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 que, haciendo uso de programación dinámica, 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 sino que 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. Ambos siguen agarrando monedas en sus respectivos turnos 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 llamada Sophia la cual estará jugando con su hermano menor llamado Mateo. Considerando que Mateo siempre tomará la moneda de mayor valor en sus turnos, ayudaremos a Sophia a obtener el máximo valor acumulado posible en cada juego, teniendo en cuenta que esto no necesariamente le asegurará a ganar. Finalmente también debemos tener en cuenta que la niña siempre comienza.

2. Análisis del problema y demostración

2.1. Análisis

Debemos tener en cuenta que Sophia siempre va a tener el primer turno en todos los juegos y además que Mateo siempre va a elegir la moneda de mayor valor. Teniendo en cuenta que n equivale a la cantidad de monedas en la fila, pensemos la fila de monedas de la siguiente forma:

$$\mathrm{M} = [M_1, M_2, M_3, \ldots, M_n]$$

Consideremos el caso donde Sophia toma la primer moneda de la fila, Mateo tendrá para elegir entre M_2 y M_n, es decir, en el turno de Mateo, la fila de monedas dispone del siguiente estado:

$$\mathrm{M} = [M_2, M_3, \ldots, M_n]$$

De manera análoga debemos considerar el caso donde Sophia elige la última moneda de la fila, sabiendo que Mateo va a elegir la de mayor valor entre M_1 y M_{n-1}, dado que la fila va a disponer del siguiente estado:

$$\mathrm{M} = [M_1, M_2, M_3, \ldots, M_{n-1}]$$

Como hemos mencionado anteriormente, lo que buscamos es maximizar el valor acumulado de Sophia aprovechando que conocemos la estrategia empleada por Mateo. Teniendo esta idea en mente y que tenemos dos elecciones posibles en cada turno vamos a considerar que:

La moneda M_i representa la primer moneda de la fila y M_j la última. Si Sophia elige M_i, entonces Mateo va a tomar la moneda de mayor valor entre M_{i+1} y M_j, caso contrario Sophia elige M_j dando a Mateo como opciones las monedas M_i y M_{j-1}. Es por esto que debemos pensar en que la elección del turno actual va a influir en la elección de nuestro siguiente turno y por ende en el valor acumulado que tengamos tras finalizar el mismo, con lo cual buscamos que en ese mismo turno tengamos el mayor valor acumulado posible, mientras minimizamos el valor acumulado de Mateo.

Ahora bien, comencemos con la idea de construir una solución "bottom up"para ello vamos a plantear en un principio la situación donde no hay monedas para elegir, con lo cual el mayor valor posible por acumular es igual a cero. Posteriormente pensemos en que hay solo una moneda disponible en la fila, por lo tanto la primera y la última moneda son la misma y finalmente debo tomarla para incrementar mi valor acumulado, básicamente la única elección posible de la cual disponemos en la fila es la mejor. Vayamos un poco más allá, con dos monedas en la fila nuestra mejor opción es aquella de mayor valor, ¿verdad? ¿y qué sucede si tenemos tres monedas en la fila?

Aquí el problema comienza a tener forma, sin embargo, si es nuestro turno y hay tres monedas en la fila, sabemos que tras nuestra elección viene la del rival y finalmente tendremos una nueva posibilidad de incrementar nuestro valor acumulado, teniendo entonces la posibilidad de sumar en dos oportunidades mientras que nuestro rival sumará solamente una. Nuestra estrategia en este caso será tomar primero la de mayor valor entre la primer y la última moneda, sin importar el valor de la moneda del medio ya que no seremos capaces de tomarla nunca, nuestro rival será quien podrá tener acceso a la misma y es por esto que debemos priorizar la de mayor valor así al finalizar el juego contaremos con el mayor valor acumulado posible.

Yendo un poco más allá tenemos el caso de una fila de cuatro monedas, en esta situación tanto el rival como nosotros vamos a tener la posibilidad de incrementar nuestro valor acumulado en dos oportunidades.

Recordemos que vamos a considerar M_i como la primer moneda de la fila y a M_j como la última.

Si elegimos M_i el rival va a tomar la de mayor valor entre M_{i+1} y M_j. En caso de que el rival elija M_{i+1}, en nuestro próximo turno vamos a poder elegir entre M_{i+2} y M_j, por otro lado, en

caso de que el rival elija M_j, en nuestro próximo turno vamos a poder elegir entre M_{i+1} y M_{j-1}. Pensemos entonces que tendremos la posibilidad de incrementar nuestro valor acumulado en M_i sumado al valor del subproblema

$$\mathrm{M} = [M_{i+2}, \ldots, M_j]$$

por un lado y por el otro a

$$\mathrm{M} = [M_{i+1}, \ldots, M_{j-1}]$$

y la pregunta es ¿Cuál de las alternativas es la que consideraremos?

Vamos a querer tomar la alternativa que sea máxima para Sophia en su siguiente turno. De esta forma siempre vamos a conseguir el mayor puntaje posible para todas las rondas, podemos asumir cual será el estado del tablero posterior dado que conocemos la estrategia que utiliza Mateo en todo momento.

Resta mencionar que la estrategia si elegimos M_j será análoga ya que en este caso nuestro rival tendrá para elegir entre M_i y M_{j-1}, dando la posibilidad de que tengamos que considerar los subproblemas

$$\mathrm{M} = [M_i, \ldots, M_{j-2}]$$

por un lado, mientras que por el otro a

$$\mathrm{M} = [M_{i+1}, \ldots, M_{j-1}]$$

y volviendo al razonamiento previamente mencionado, consideraremos el subproblema que no incluye la moneda tomada por Mateo.

Vamos a considerar S_{pm} y S_{um} como los subproblemas a considerar tras la elección de la primera o de la última moneda respectivamente.

Finalmente nuestra ecuación de recurrencia es:

$$S_{pm} = \begin{cases} P(h-2, i+2) : monedas[i+1] > monedas[i+h-1] \\ P(h-2, i+1) : e.o.c. \end{cases}$$

$$S_{um} = \begin{cases} P(h-2, i+1) : monedas[i] > monedas[i+h-2] \\ P(h-2, i) : e.o.c. \end{cases}$$

$$P(h,i) = \begin{cases} 0 : h = 0 \\ monedas[i] : h = 1 \\ \text{máx}(monedas[i], monedas[i+1]) : h = 2 \\ \text{máx}(monedas[i] + S_{pm}, monedas[i+h-1] + S_{um}) : e.o.c. \end{cases}$$

Donde h es la cantidad de monedas e i es el índice de la primer moneda a considerar en el arreglo de monedas.

2.2. Demostración de la ecuación de recurrencia

Primero plantemos los casos base de la función de recurrencia.

$P_{h,i}$ representa la solución óptima del problema para el juego de monedas donde h es la cantidad de monedas del tablero e i es la i-ésima ventana de monedas con este tamaño tal que $i \in [0, n-h+1]$

donde n es la cantidad inicial de monedas del tablero. i también puede pensarse como el índice de la primer moneda a considerar en el arreglo de monedas.

Donde para un juego de 0 monedas

$$P(0, i) = 0$$

Para el de 1 moneda

$$P(1, i) = monedas[i]$$

Y el de tamaño 2

$$P(2, i) = \text{máx}(monedas[i], monedas[i + 1])$$

Todos estos casos maximizan el valor acumulado para Sophia, esto es importante porque son la base que sostiene nuestra hipotesis inductiva.

Para el caso general obtenemos las siguientes expresiones.

$$S_{pm} = \begin{cases} P(h-2, i+2) : monedas[i+1] > monedas[i+h-1] \\ P(h-2, i+1) : e.o.c. \end{cases}$$

S_{pm} es el valor óptimo para el próximo turno de Sophia si tomara la primer moneda. Notar que se toma en cuenta que moneda es mayor entre los extremos restantes, siendo estos el de la segunda y última monedas. Esto determina que moneda tomaría Mateo dado cualquier juego de monedas para luego saber que tablero le tocará jugar a Sophia en su próximo turno.

$$S_{um} = \begin{cases} P(h-2, i+1) : monedas[i] > monedas[i+h-2] \\ P(h-2, i) : e.o.c. \end{cases}$$

S_{um} es el valor óptimo para el próximo turno de Sophia si tomara la última moneda. En este caso comparamos los valores de la primer y anteúltima monedas.

$$P(h, i) = \text{máx}(monedas[i] + S_{pm}, monedas[i+h-1] + S_{um})$$

El problema de h monedas para cualquier desfazaje i se resuelve como el valor máximo entre tomar la primer y última moneda sumado al valor óptimo asociado al único posible juego siguiente habiendo tomado esa moneda.

Entonces $P(h, i)$ depende de dos posibles estados futuros del juego. Si Sophia tomase la primer moneda entonces sabría cual es el tablero dentro de dos turnos, estos siendo $P(h-2, i+2)$ o $P(h-2, i+1)$. Lo mismo pasaría si ella tomara la última moneda, Mateo guiaría el juego hacia $P(h-2, i)$ o $P(h-2, i+1)$.

Pareciera que en realidad Sophia tuviera que saber el estado del juego para estos cuatro casos pero no es así, Mateo siempre toma la moneda de mayor valor, por lo que ella puede predecir cual será la decisión de Mateo dado cualquier tablero de forma inmediata. Lo que acota las posibilidades a que el siguiente tablero sea único, sea cual sea su decisión en cualquier turno.

Por lo que simplificamos a

$$P(h, i) = \text{máx}(monedas[i] + P(h-2, i+1+j), monedas[i+h-1] + P(h-2, i+k))$$

Donde j y k valen 1 sii Mateo tomaría la primer moneda luego de Sophia tomar la primera o última moneda. $j = 1$ si la segunda moneda es mayor que la última, y $k = 1$ si la primer moneda

es mayor que la anteúltima. En otro caso j y k valen 0. Es muy importante que dado un tablero de monedas los valores de j y k son conocidos, pues, la estrategia de Mateo nunca cambia.

Para $h = 3$ obtenemos la siguiente expresión.

$$P(3, i) = \text{máx}(monedas[i] + P(1, i + 1 + j), monedas[i + 2] + P(1, i + k))$$

Notar que en ambos casos caemos en algún $h = 1$ dado que vamos a buscar la solución óptima al mismo problema dentro de dos turnos. Sabiendo que podemos resolver estos problemas de forma óptima (es uno de los casos base), resolver $P(3, i)$ es tomar una nueva decisión a partir de dos soluciones particulares de $P(1, i + r)$ con $r \in [0, 2]$.

Expandiendo tenemos que

$$P(3, i) = \text{máx}(monedas[i] + monedas[i + 1 + j], monedas[i + 2] + monedas[i + k])$$

Tomando el mismo j y k de antes, podemos calcular $P_{3,i}$ como el máximo entre tomar alguna moneda sumado a la moneda que Mateo vaya a dejar. De esta forma Sophia obtiene el valor óptimo del juego con tres monedas sin importar el valor de i.

Para $h = 4$ obtenemos la siguiente expresión.

$$P(4, i) = \text{máx}(monedas[i] + P(2, i + 1 + j), monedas[i + 3] + P(2, i + k))$$

Análogamente como con $P(3, i)$, los subproblemas son de dos monedas menos. Para resolver $P(4, i)$ recurrimos a las soluciones óptimas de $P(2, i + 2)$ si ambos jugadores toman la primer moneda, $P(2, i+1)$ si alternan su decisión, y a $P(2, i)$ si ambos toman la última. Como mostramos más arriba, ya sabemos resolver $P(2, i)$ de forma óptima para cualquier i, por lo que podemos garantizar que en el próximo turno Sophia va a llegar al mejor tablero posible. Para resolver $P(4, i)$ tomamos en cuenta los dos posibles futuros tableros y la moneda que tomamos para llegar a él, al final nos quedamos con la que maximiza el puntaje.

Expandiendo tenemos que

$$P(4, i) = \text{máx}\Big(monedas[i] + \text{máx}\big(monedas[i + 1 + j], monedas[i + 1 + j + 1]\big), \\ monedas[i + 3] + \text{máx}\big(monedas[i + k], monedas[i + k + 1]\big)\Big) \quad (1)$$

El óptimo para Sophia es el resultado de tomar alguna de las dos monedas sumado a la máxima moneda que nos deje Mateo después de su turno. De esta forma queda demostrado que para $h = 4$ Sophia obtiene el mejor resultado posible.

Ahora tomamos el paso inductivo de calcular $P(h + 2)$, pues $P(h + 1)$ fué último turno de Mateo.

$$P(h+2, i) = \text{máx}(monedas[i]+P((h+2)-2, i+1+j), monedas[i+(h+2)-1]+P((h+2)-2, i+k))$$

Que es equivalente a:

$$P(h + 2, i) = \text{máx}(monedas[i] + P(h, i + 1 + j), monedas[i + h + 1] + P(h, i + k))$$

Entonces $P(h + 2, i)$ se reduce a resolver $P(h, i + 1 + j)$ y $P(h, i + k)$ de los cuales dependen de juegos de dos monedas más chicos y eventualmente caen sobre los casos base que definimos anteriormente.

Tomando el máximo de estos dos casos junto con la moneda tomada para llegar a ellos es como aseguramos quedarnos con el valor máximo acumulado del puntaje de Sophia para todo estado de tablero futuro.

Por ende queda demostrado que para todo $h \in \mathbb{N}^0$ la solución obtenida por el algoritmo es óptima y maximiza el puntaje de Sophia teniendo en cuenta que Mateo siempre toma la moneda de mayor valor en todos sus turnos.

3. Algoritmo Planteado

3.1. Implementación

Se tiene una matriz de tabulación para guardar los resultados óptimos de los subproblemas. Se llenan los casos base, estos siendo de 0, 1 y 2 monedas.

Luego por cada cantidad de monedas y cada ventana posible se calcula el óptimo de forma bottom-up hasta alcanzar nuestro caso de n monedas.

Para la parte de reconstrucción de la solución iteramos de forma inversa la matriz empezando por el último calculado y tomando las monedas dependiendo de si el óptimo de un lado es mejor que el otro. Se toma en cuenta la decisión de Mateo para saber cual será el estado de las monedas dentro de 2 turnos.

De esta forma se arma un arreglo de acciones a tomar para que Sophia obtenga la solución óptima al problema. De paso también calculamos el valor acumulado de Mateo.

```
1
2  def jugar(monedas):
3      """
4      f(0, s) = 0
5      f(1, s) = m[s]
6      f(2, s) = max(m[s], m[s + 1])
7      f(n, s):
8          pm = m[s] + f(n - 2, s + 1 + (m[s + 1] > m[s + n - 1]))
9          um = m[s + n - 1] + f(n - 2, s + (m[s] > m[s + i - 2]))
10         return max(pm, um)
11
12     La complejidad del algoritmo es O(n^2)
13     """
14     def jugar_pd(n, m):
15         # O(n^2)
16         mem = [[0] * (n - i + 1) for i in range(n + 1)]
17         mem[1] = m
18
19         # O(n)
20         for s in range(n - 1):
21             mem[2][s] = max(m[s], m[s + 1])
22
23         # O(n^2)
24         for i in range(3, n + 1):
25             for s in range(n - i + 1):
26                 pm = m[s] + mem[i - 2][s + 1 + (m[s + 1] > m[s + i - 1])]
27                 um = m[s + i - 1] + mem[i - 2][s + (m[s] > m[s + i - 2])]
28                 mem[i][s] = max(pm, um)
29
30         return mem
31
32     def construir_solucion(n, f, m):
33         mateo = 0
34         sol = []
35
36         s = 0  # O(n)
37         for i in range(n, 0, -2):
38             # Sophia
39             if i >= 2:
40                 pm = m[s] + f[i - 2][s + 1 + (m[s + 1] > m[s + i - 1])]
41                 um = m[s + i - 1] + f[i - 2][s + (m[s] > m[s + i - 2])]
42             else:
43                 pm = um = m[s]
44
45             if pm > um:
46                 sol.append(f"Sophia debe agarrar la primera ({m[s]})")
47                 s += 1
48             else:
49                 sol.append(f"Sophia debe agarrar la ultima ({m[s + i - 1]})")
50
51             if i == 1:
```

```
52                  break
53
54              # Mateo
55              pm = m[s]
56              um = m[s + (i - 1) - 1]
57              if pm > um:
58                  sol.append(f"Mateo agarra la primera ({pm})")
59                  s += 1
60              else:
61                  sol.append(f"Mateo agarra la ultima ({um})")
62
63              mateo += max(pm, um)
64
65          return f[n][0], mateo, sol
66
67      n = len(monedas)
68      f = jugar_pd(n, monedas)
69      return construir_solucion(n, f, monedas)
```

3.2. Complejidades

En cuanto a la complejidad espacial, en ambas funciones, es decir, tanto en la de construcción de la matriz de tabulación como en la de reconstrucción de la solución, es $O(n^2)$. Esto es así ya que se utiliza una matriz triangular superior, entonces tiene un tamaño de $((n^2)/2) + (n+1)$, esto es la mitad de elementos de una matriz cuadrada más los elementos de la diagonal principal. En notación O, esto es $O(n^2)$, ya que las constantes no importan y el término dominante es n^2.

Con respecto a la complejidad temporal, haciendo el mismo desarrollo previo, obtenemos que la contrucción de la matriz es $O(n^2)$. Por otro lado, dado que para realizar la reconstrucción de la solución se recorre la matriz de dos en dos, la complejidad temporal es $O(n/2)$ lo que equivale a $O(n)$. Entonces, la complejidad temporal total del algoritmo, teniendo en cuenta la contrucción de la matriz y reconstrucción de la solución, es $O(n^2)$.

3.3. 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 solo le interesa saber si debe o no tomar la primer o última moneda usando operaciones de tiempo constante que tienen en cuenta que número es mayor que otro.

3.4. 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 acumula el mayor valor posible, aunque esto no necesariamente le asegura ganar.

4. Ejemplos

A continuación presentamos una serie de ejemplos del algoritmo propuesto.

4.1. Ejemplo 1: Sophia gana

Para el primer ejemplo, consideraremos el siguiente arreglo de monedas:

$$M = [96, 594, 437, 674]$$

4.1.1. Construcción de la matriz

En primer lugar se inicializa la matriz que irá almacenando los resultados en distintas etapas del algoritmo. Esta matriz será triangular superior, con una cantidad de filas igual a la cantidad de monedas más uno y la primera tendrá una cantidad de columnas igual a la cantidad de monedas más uno.

Inicio del arreglo →

Cantidad de monedas ↓

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	
2	0	0	0		
3	0	0			
4	0				

A medida que nos movemos hacia la derecha en la matriz, se desplaza el inicio del arreglo. A medida que nos movemos hacia abajo, se incrementa la cantidad de monedas consideradas.

Fila 0

La fila 0 de la matriz se mantendrá con todos sus valores en 0 ya que representa los casos en el que se cuenta con 0 monedas, por lo tanto Sophia no acumularía nada.

Fila 1

La fila 1 representa los casos en los que se cuenta con 1 moneda. En este caso, Sophia acumularía el valor de la única moneda que tiene para tomar, por lo que esta fila es equivalente al arreglo de monedas inicial.

	0	1	2	3	4
0	0	0	0	0	0
1	96	594	437	674	
2	0	0	0		
3	0	0			
4	0				

Fila 2

La fila 2 representa los casos en los que se cuenta con 2 monedas. En este caso, Sophia tomaría la moneda con mayor valor entre las dos disponibles.

Veamos en detalle los cálculos para cada posición de la fila:

$$mem[2][0] = max\{96, 594\} = 594$$

$$mem[2][1] = max\{594, 437\} = 594$$

$$mem[2][2] = max\{437, 674\} = 674$$

Resultando parcialemte la matriz:

	0	1	2	3	4
0	0	0	0	0	0
1	96	594	437	674	
2	594	594	674		
3	0	0			
4	0				

Fila 3

La fila 3 representa los casos en los que se cuenta con 3 monedas. Veamos cada posicion de la fila y sus casos posibles:

- $mem[3][0]$: se consideran las monedas $[96, 594, 437]$ y hay dos escenarios a considerar:

1. Sophia toma la primera moneda (96)

   Mateo tendría que elegir entre las monedas $[594, 437]$, y dado que siempre elige la de mayor valor toma la 594. Luego Sophia tomaría la moneda 437, este escenario está almacenado en $mem[1][2]$, es decir se cuenta con 1 moneda y el arrelgo inicia en la posición 2. Por lo tanto:

   $$pm = 96 + mem[1][2](437) = 533$$

2. Sophia toma la última moneda (437)

   Mateo tendría que elegir entre las monedas $[96, 594]$, en esta caso tomaría la moneda 594. Luego Sophia tomaría la de 96, este escenario está almacenado en $mem[1][0]$, es decir se cuenta con 1 moneda y el arrelgo inicia en la posición 0. Por lo tanto:

   $$um = 437 + mem[1][0](96) = 533$$

   El máximo a acumular en este caso es el máximo entre los dos escenarios, es decir:

$$mem[3][0] = max\{533, 533\} = 533$$

- $mem[3][1]$: se consideran las monedas $[594, 437, 674]$ y se cosideran los escenarios:

1. Sophia toma la primer moneda (594)

   Mateo tomará la 674. Luego Sophia tomaría la única moneda disponible, la 437, este escenario está almacenado en $mem[1][2]$, entonces:

   $$pm = 594 + mem[1][2](437) = 1031$$

2. Sophia toma la última moneda (674)

   Mateo tomará la 594. Luego Sophia tomaría la 437, este escenario está almacenado en $mem[1][2]$, entonces:

   $$um = 674 + mem[1][2](437) = 1111$$

   El máximo a acumular en este caso es el máximo entre los dos escenarios, es decir:

$$mem[3][1] = max\{1031, 1111\} = 1111$$

La matriz parcial resultante es:

	0	1	2	3	4
0	0	0	0	0	0
1	96	594	437	674	
2	594	594	674		
3	533	1111			
4	0				

Fila 4

Para la fila 4 se consideran todas las monedas $[96, 594, 437, 674]$ y se cosideran los escenarios:

1. Sophia toma la primer moneda (96)

   Mateo tomará la 674. Luego Sophia se encontraría con las monedas $[594, 437]$, este escenario ya lo resolvimos y está almacenado en $mem[2][1]$, entonces:

   $$pm = 96 + mem[2][1](594) = 690$$

2. Sophia toma la última moneda (674)

   Mateo tomará la 437. Luego Sophia se encontraría con las monedas $[96, 594]$, este escenario ya lo resolvimos y está almacenado en $mem[2][0]$, entonces:

   $$um = 674 + mem[2][0](594) = 1268$$

El máximo a acumular en este caso es el máximo entre los dos escenarios, es decir:

$$mem[4][0] = max\{690, 1268\} = 1268$$

La matriz resultante es:

	0	1	2	3	4
0	0	0	0	0	0
1	96	594	437	674	
2	594	594	674		
3	533	1111			
4	1268				

Sophia habrá acumulado un total de 1268 mientras que Mateo habrá acumulado 533.

4.1.2. Reconstrucción de la solución

Para reconstruir la solución recorreremos la matriz desde el último elemento en esta. Empezamos entonces por la última fila, es decir, la primera elección de Sophia. Veamos los casos posibles teniendo en cuenta que su acumulado es de 1268:

- Sophia toma la primer moneda (96):

  Se debe cumplir que:

  $$mem[4][0] = 1268 \stackrel{?}{=} 96 + mem[2][1](594)$$

  $$mem[4][0] = 1268 \neq 690$$

  Esto no se cumple por lo que Sophia no toma la primer moneda.

- Sophia toma la última moneda (674):

  Se debe cumplir que:

$$mem[4][0] = 1268 \stackrel{?}{=} 674 + mem[2][0](594)$$

$$mem[4][0] = 1268 = 1268$$

  Esto se cumple por lo que Sophia toma la última moneda.

Luego Mateo tomará la moneda con mayor valor de los extremos de $[96, 594, 437]$, es decir, la 437. Por lo que Sophia deberá resolver el problema con las monedas $[96, 594]$. Veamos los casos posibles:

- Sophia toma la primer moneda (96)

  Se debe cumplir que:

$$mem[2][1] = 594 \stackrel{?}{=} 96 + mem[0][1](0)$$

$$mem[2][1] = 594 \neq 96$$

  Esto no se cumple por lo que Sophia no toma la primer moneda.

- Sophia toma la última moneda (594)

  Se debe cumplir que:

$$mem[2][1] = 594 \stackrel{?}{=} 594 + mem[0][0](0)$$

$$mem[2][1] = 594 = 594$$

  Esto se cumple por lo que Sophia toma la última moneda.

Mateo toma la única moneda restante, la 96.

4.1.3. Ejecución del ejemplo

Para ejecutar el ejemplo, se debe correr el siguiente comando:

```
python3 main.py ejemplos/ejemplo_1.txt
```

Y se obtendrá la siguiente salida:

```
Sophia debe agarrar la ultima (674); Mateo agarra la ultima (437); Sophia debe agarrar la ultima (594); Mateo agarra la ultima (96)
Ganancia Sophia: 1268
Ganancia Mateo: 533
```

La cual coincide con la solución obtenida a partir del seguimiento realizado.

4.2. Ejemplo 2: Empatan

Para el segundo ejemplo, consideraremos el siguiente arreglo de monedas:

$$\mathrm{M} = [1, 100, 100, 1]$$

4.2.1. Construcción de la matriz

Fila 0, 1 y 2

Al igual que en el ejemplo anterior, se inicializa la matriz. La fila 0 se mantiene en 0, la fila 1 es igual al arreglo de monedas inicial y la fila 2 es el máximo entre las dos monedas disponibles. Resultando la matriz parcial:

	0	1	2	3
0	0	0	0	0
1	1	100	100	1
2	100	100	100	
3	0	0		
4	0			

Fila 3

Realizando la misma operación que en el ejemplo anterior, se obtiene que:

- $mem[3][0]$: se considera las monedas $[1, 100, 100]$ y se cosideran los escenarios:

  1. Sophia toma la primer moneda (1)
     Mateo tomará la última (100). Luego Sophia tomará la única moneda disponible, la 100, este escenario está almacenado en $mem[1][1]$, entonces:

     $$pm = 1 + mem[1][1](100) = 101$$

  2. Sophia toma la última moneda (100)
     Mateo tomará la última (100). Luego Sophia tomará la 1, este escenario está almacenado en $mem[1][0]$, entonces:

     $$um = 100 + mem[1][0](1) = 101$$

  El máximo a acumular en este caso es el máximo entre los dos escenarios, es decir:

  $$mem[3][0] = max\{101, 101\} = 101$$

- $mem[3][1]$: se considera las monedas $[100, 100, 1]$ y se cosideran los escenarios:

  1. Sophia toma la primer moneda (100)
     entonces:

     $$pm = 100 + mem[1][3](1) = 101$$

  2. Sophia toma la última moneda (1)
     entonces:

     $$um = 1 + mem[1][1](100) = 101$$

  El máximo a acumular en este caso es el máximo entre los dos escenarios, es decir:

  $$mem[3][1] = max\{101, 101\} = 101$$

La matriz parcial resultante es:

	0	1	2	3
0	0	0	0	0
1	1	100	100	1
2	100	100	100	
3	101	101		
4	0			

Fila 4

Para la fila 4 se consideran todas las monedas $[1, 100, 100, 1]$ y se cosideran los escenarios:

1. Sophia toma la primer moneda (1)

   entonces:

$$pm = 1 + mem[2][2](100) = 101$$

2. Sophia toma la última moneda (1)

   entonces:

$$um = 1 + mem[2][0](100) = 101$$

Entonces:

$$mem[4][0] = max\{101, 101\} = 101$$

y la matriz resultante:

	0	1	2	3
0	0	0	0	0
1	1	100	100	1
2	100	100	100	
3	101	101		
4	101			

Por lo que Sophia habrá acumulado un total de 101.

4.2.2. Reconstrucción de la solución

Nuevamente recorremos la matriz desde el último elemento. Resultando en que Sohia toma la última (1), Mateo toma la última (100), Sophia toma la última (100) y Mateo toma la última disponible (1).

Se obtiene entonces que Sophia acumula 101 y Mateo acumula 101, por lo que se da un empate. Lo cual era esperado ya que, al tratarse una cantidad de moneda par toman la misma cantidad de monedas y por como se distribuyen las monedas, ambos acumulan la misma cantidad.

4.2.3. Ejecución del ejemplo

Para ejecutar el ejemplo, se debe correr el siguiente comando:

```
1   python3 main.py ejemplos/ejemplo_2.txt
```

4.3. Ejemplo 3: Sophia pierde

Para el tercer ejemplo, consideraremos el siguiente arreglo de monedas:

$$M = [1, 100, 1]$$

4.3.1. Construcción de la matriz

Fila 0, 1 y 2

Nuevamente, la fila 0 se mantiene en 0, la 1 es igual al arreglo de monedas inicial y la 2 es el máximo entre las dos monedas disponibles. Resultando la matriz parcial:

	0	1	2
0	0	0	0
1	1	100	1
2	100	100	
3	0		

Fila 3

Teniendo en cuenta que se cuenta con las monedas $[1, 100, 1]$, y se considera toma el máximo entre los casos posibles, sacar la primer moneda o la última, la matriz resultante es:

	0	1	2
0	0	0	0
1	1	100	1
2	100	100	
3	2		

Por lo que Sophia habrá acumulado un total de 2.

4.3.2. Reconstrucción de la solución

Recorriendo la matriz desde el último elemento, se obtiene que Sophia toma la última (1), Mateo toma la última (100) y Sophia toma la última disponible (1).

En este caso, debido a los valores y distribución de las monedas, Sophia acumula 2 y Mateo acumula 100, por lo que Sophia pierde.

4.3.3. Ejecución del ejemplo

Para ejecutar el ejemplo, se debe correr el siguiente comando:

```
python3 main.py ejemplos/ejemplo_3.txt
```

4.4. Conclusiones

En los ejemplos presentados se pueden observan los tres casos posibles. El algoritmo siempre va a enconrtar el máximo valor que puede acumular Sophia, y podría resultar en victoria, empate o derrota.

5. Mediciones

Se realizaron mediciones del algoritmo tomando distintas cantidades de monedas para demostrar empíricamente su complejidad teórica de $O(n^2)$. Se tomaron 500 muestras separadas uniformemente de 1 a 1000 elementos.

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

El código utilizado se encuentra dentro de la carpeta *codigo* en el archivo jupyter *mediciones.ipynb*. Utilizamos la función *time_algorithm* otorgada por la cátedra para realizar las mediciones junto con los distintos métodos de crear juegos de monedas explicados más abajo.

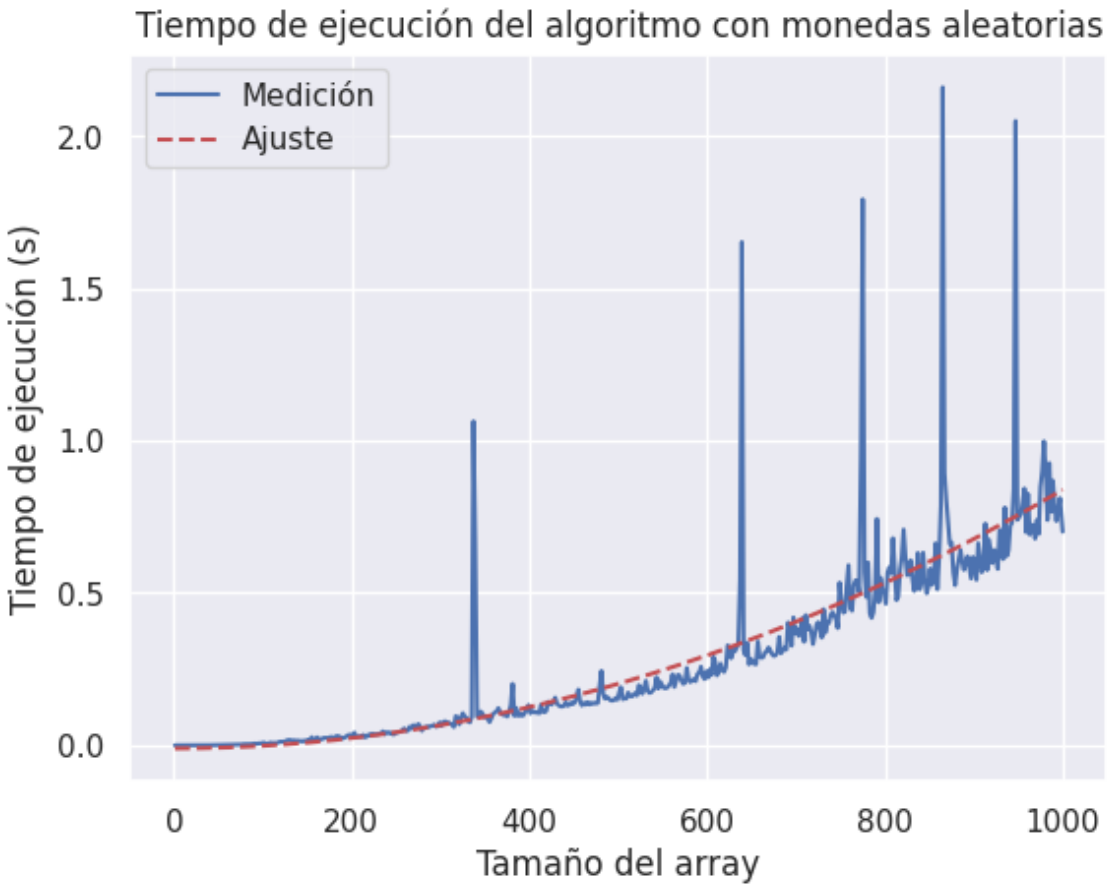
El archivo está dividido en secciones para cada una de las distintas mediciones, en caso de querer medir el algoritmo con un set de monedas en particular, se debe definir una nueva función generadora de monedas que devuelva un arreglo de monedas dado un cierto tamaño. $graficar$ es la función que recibe una función generadora de juegos de monedas, un tope de monedas (siendo la máxima cantidad de monedas a medir) y un título para el gráfico.

También implementamos el cálculo de cuadrados mínimos como fué recomendado en el enunciado del trabajo práctico.

Para todo esto utilizamos módulos de python como: *matplotlib.pyplot*, *seaborn*, *scipy* y *numpy*.

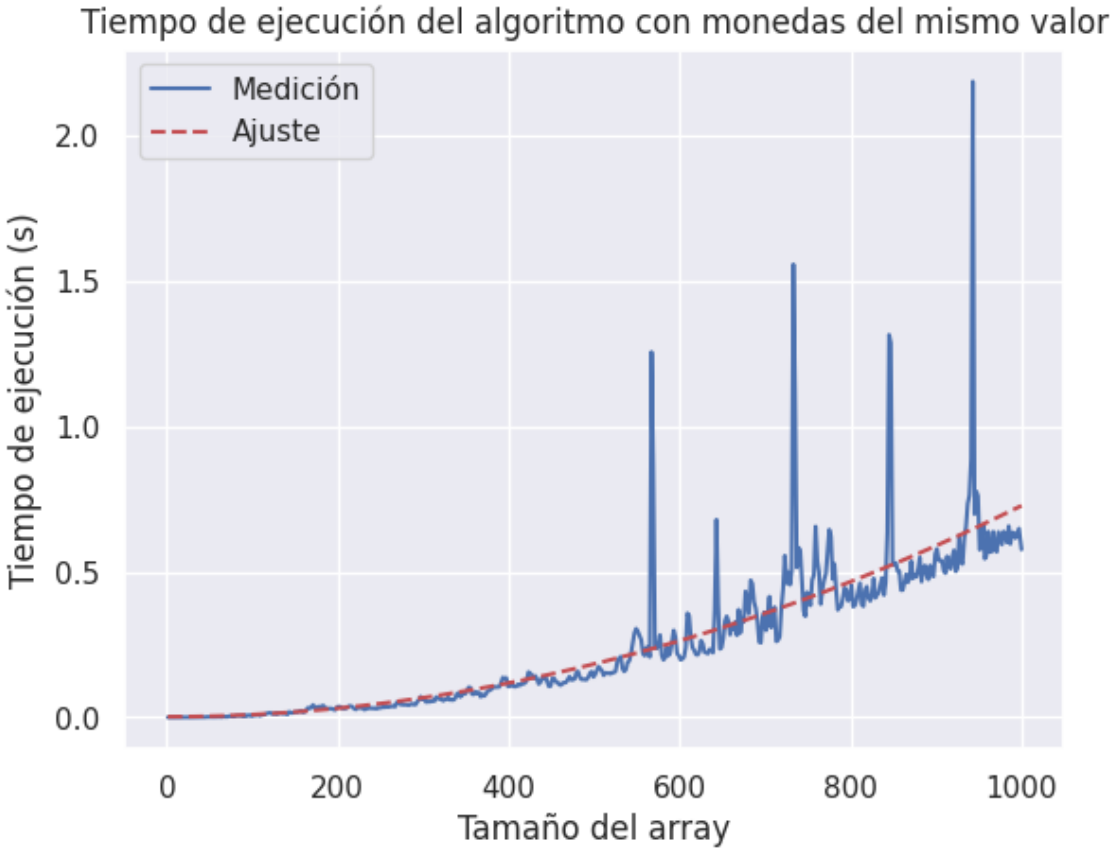
5.1. Monedas aleatorias

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



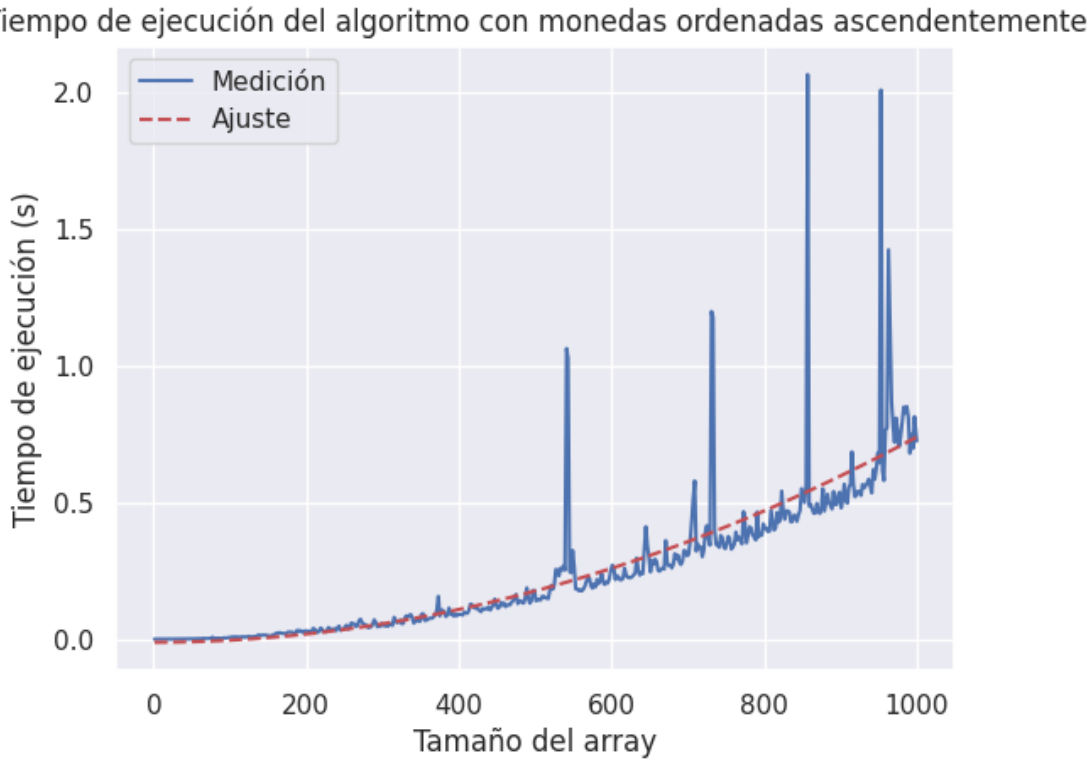
Como se puede observar, la complejidad temporal del algoritmo es cuadrática, tal como se esperaba.

5.2. Monedas del mismo valor



Nuevamente, podemos observar que la complejidad temporal del algoritmo es cuadrática, sin importar que los valores de las monedas sea el mismo.

5.3. Monedas ordenadas ascendentemente



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

Es importante notar que la constante que acompaña al termino cuadrático es lo suficientemente similar como para determinar que la variabilidad de las monedas no afecta el tiempo de ejecución del algoritmo.

6. Conclusiones

A partir del desarrollo realizado podemos afirmar que el algoritmo de programación dinámica propuesto para resolver el problema en cuestión siempre encuentra la solución que maximiza el valor acumulado por Sophia. Esto no siempre garantiza que Sophia gane el juego, pero siempre obtendrá la mejor solución posible. Por otro lado, se pudo comprobar, a través de las mediciones realizadas que la complejidad temporal del algoritmo es $O(n^2)$, como se esperaba. Además, se pudo observar que la variabilidad de los valores de las monedas no afecta los tiempos de este.