

TEORÍA DE ALGORITMOS
(75.29) CURSO BUCHWALD - GENENDER

Trabajo Práctico 2

Programación Dinámica

For The Win

22 de octubre de 2024

Benjamin Castellano
Bogdan
111519

Joaquin Eduardo
Embon
111292

Lionel Gabriel
Maydana Gonzalez
106512

1. Introduccion

Sophia y Mateo son hermano que aún son niños pero ya tienen la madurez necesaria para querer competir entre ambos, quieren jugar un juego explicado por su padre: Se dispone 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).

Lo que buscamos en este trabajo práctico es encontrar un algoritmo por programación dinámica que obtenga la solución óptima al problema planteado: Dada la secuencia de monedas m_1, m_2, \dots, m_n , sabiendo que Sophia empieza el juego y que Mateo siempre elegirá la moneda más grande para sí entre la primera y la última moneda en sus respectivos turnos, definir qué monedas debe elegir Sophia para asegurarse obtener el **máximo valor acumulado posible**. Esto no necesariamente le asegurará a Sophia ganar, ya que puede ser que esto no sea obtenible, dado por cómo juega Mateo. Por ejemplo, para $[1, 10, 5]$, no importa lo que haga Sophia, Mateo ganará.

Primero que nada, para un algoritmo de programación dinámica, debemos buscar la **ecuación de recurrencia**. ¿Cómo podemos pensarla? Tenemos que hacer que gane Sophia a toda costa, o en caso contrario, que pueda tener el máximo valor acumulable posible porque hay que tener en cuenta que Mateo también quiere ganar y va a hacer todo lo posible siempre (elegir la moneda más grande entre la primera y la última cada vez que le toque jugar). Si sólo pensamos en elegir la moneda de mayor valor en el turno de Sophia, nos puede pasar, que la moneda que reemplace la misma sea de un valor muy alto, por ejemplo, Sophia eligió la moneda '7' teniendo el arreglo $[7, 100, 1, 5]$, en este caso le estamos dejando la moneda de mayor valor a Mateo.

Para cada turno de Sophia, la elección será entre la primer moneda o la última pero, para ambas, sumada la máxima sumatoria del subarreglo que las acompaña, es decir, el valor de la moneda + máximo del subarreglo. ¿Qué quiere decir esto? Que tendremos en cuenta las monedas que le dejamos a Mateo para elegir. Teniendo en cuenta cada arreglo con una posición *inicio* y *fin*, la ecuación de recurrencia quedaría:

opción_izquierda = Valor primer moneda + Máximo del subarreglo correspondiente

opción_derecha = Valor última moneda + Máximo del subarreglo correspondiente

$$\text{Opt}[\text{ini}][\text{fin}] = \max(\text{opción_izquierda}, \text{opción_derecha})$$

2. Algoritmo

2.1. Código

```
1 def juego_sophia_mateo(monedas):
2     cant_monedas = len(monedas)
3
4     matriz = [[0] * cant_monedas for _ in range(cant_monedas)]
5
6     # Caso base: cuando solo hay una moneda (Sophia toma la unica moneda)
7     for i in range(cant_monedas):
8         matriz[i][i] = monedas[i]
9
10    # Vamos llenando la tabla desde casos menores.
11    for longitud in range(2 + cant_monedas % 2, cant_monedas + 1, 2):
12        for ini in range(cant_monedas - longitud + 1):
13            fin = ini + longitud - 1
14
15            # Sophia toma la moneda en el extremo izquierdo
16            opcion_izquierda = monedas[ini]
17            # Sumamos valores dependiendo de la decision de Mateo
18            if monedas[ini + 1] > monedas[fin]:
19                opcion_izquierda += matriz[ini + 2][fin] if ini + 2 <= fin else 0
20            else:
21                opcion_izquierda += matriz[ini + 1][fin - 1] if ini + 1 <= fin - 1
22        else 0
23
24            # Mismo procedimiento pero para la moneda del extremo derecho
25            opcion_derecha = monedas[fin]
26
27            if monedas[ini] > monedas[fin - 1]:
28                opcion_derecha += matriz[ini + 1][fin - 1] if ini + 1 <= fin - 1
29            else 0
30            else:
31                opcion_derecha += matriz[ini][fin - 2] if ini <= fin - 2 else 0
32
33            # Guardar el valor maximo en matriz
34            matriz[ini][fin] = max(opcion_izquierda, opcion_derecha)
35
36    return rearmar_solucion(matriz, monedas)
```

Nuestro algoritmo de programación dinámica está planteado de manera "bottom up", guardando primero los resultados conseguidos de los subproblemas anteriores, para ir utilizandolos luego para construir el problema principal.

Para trabajar con las monedas, vamos a tomar un arreglo con estas monedas, en el orden con el que se nos las dan.

Para la parte de "memoization", característica de este tipo de algoritmo, las sumatorias máximas de monedas para cada subproblema, serán guardados en una matriz, creada en la línea 4 del código anterior. Esta decisión se da por las variables del problema, que en este problema identificamos dos. Por un lado "ini", como la posición relativa de la moneda del extremo izquierdo, y por el otro "fin", como la posición relativa de la moneda del extremo derecho.

Para llenar esta matriz con los valores que nos interesan, primero vamos al caso base, el cual sería un arreglo con una sola moneda ($ini == fin$), simplemente llenamos esas posiciones con la única moneda del arreglo. Notar que también está cubierto el caso de un arreglo vacío (la matriz tendría largo cero, y los siguientes pasos que recorren esta matriz no harían nada, el resultado sería 0).

Luego, vamos a ir llenando la matriz incrementando el tamaño de cada subproblema, cada vez que hayamos calculado su sumatoria máxima de monedas (Ciclo que comienza en línea 11). Estos subproblemas los vamos a resolver de acuerdo a la siguiente ecuación de recurrencia:

$$matriz[ini][fin] = \max(opcion_izquierda, opcion_derecha)$$

Donde:

- $matriz[ini][fin]$ es la sumatoria máxima que Sophia puede tener con las monedas desde la posición ini hasta la posición fin .
- $opcion_izquierda$ es el valor que obtiene Sophia si elige la moneda en el extremo izquierdo (posición ini). Este valor se calcula como: $opcion_izquierda = monedas[ini] + ganancia_segun_mateo$. Donde $ganancia_segun_mateo$ es: $matriz[ini + 2][fin]$ (si $ini + 2 \leq fin$) si mateo elige la moneda en $ini + 1$, o $matriz[ini + 1][fin - 1]$ (si $ini + 1 \leq fin - 1$) si mateo elige la moneda fin .
- $opcion_derecha$ es el valor que obtiene Sophia si elige la moneda en el extremo derecho (posición fin). Este valor se calcula como: $opcion_derecha = monedas[fin] + ganancia_segun_mateo$. Donde $ganancia_segun_mateo$ es: $matriz[ini][fin - 2]$ (si $fin - 2 \leq ini$) si mateo elige la moneda en ini , o $matriz[ini + 1][fin - 1]$ (si $fin - 1 \leq ini + 1$) si mateo elige la moneda fin .

Entonces, la ecuación de recurrencia completa sería:

$$matriz[ini][fin] = \max(monedas[ini] + \min(matriz[ini + 2][fin], matriz[ini + 1][fin - 1]), monedas[fin] + \min(matriz[ini][fin - 2], matriz[ini + 1][fin - 1]))$$

Finalmente, devolvemos la función `rearmar_solucion()` (El código aparece más abajo), que lo que hace es devolver dos arreglos, uno que tiene las posiciones de las monedas que Sophia tiene que agarrar, y otro que tiene las posiciones de las monedas que Mateo tiene que agarrar (las posiciones son respecto de el arreglo de monedas principal)

```
1 def rearmar_solucion(matriz, monedas):
2     solucion_sophia = []
3     solucion_mateo = []
4     cant_monedas = len(monedas)
5     ini = 0
6     fin = cant_monedas - 1
7     valor_maximo = matriz[ini][fin]
8     while valor_maximo > 0:
9         if ini == fin:
10             solucion_sophia.append(ini)
11             break
12         siguiente_ini, siguiente_fin = siguientes_indices_eligiendo_mateo(ini + 1,
13 fin, monedas)
14         if valor_maximo == monedas[ini] + matriz[siguiente_ini][siguiente_fin]:
15             solucion_sophia.append(ini)
16             valor_maximo -= monedas[ini]
17             if siguiente_ini == ini + 1:
18                 solucion_mateo.append(fin)
19             else:
20                 solucion_mateo.append(ini + 1)
21         else:
22             siguiente_ini, siguiente_fin = siguientes_indices_eligiendo_mateo(ini,
23 fin - 1, monedas)
24             solucion_sophia.append(fin)
25             valor_maximo -= monedas[fin]
26             if siguiente_fin == fin - 1:
27                 solucion_mateo.append(ini)
28             else:
29                 solucion_mateo.append(fin - 1)
30         ini = siguiente_ini
31         fin = siguiente_fin
32     return solucion_sophia, solucion_mateo
33
34 def siguientes_indices_eligiendo_mateo(ini, fin, monedas):
35     if monedas[fin] >= monedas[ini]:
36         return ini, fin - 1
37     return ini + 1, fin
```

Mediante un ciclo, y utilizando la ecuación de recurrencia a la inversa por cada iteración, podemos rearmar el camino y conocer bien cuales son los pasos que Sophia tiene que seguir para conseguir el máximo valor acumulado.

Nos parece importante aclarar explícitamente, que a partir de todas las características que este algoritmo presenta, y teniendo en cuenta la forma en la que está escrito, este algoritmo termina resolviendo el problema puramente por programación dinámica, cumpliendo con el requisito que la consigna nos exigía.

2.2. Optimalidad

Para demostrar que nuestro algoritmo es óptimo, primero vamos a hablar un poco sobre la forma de nuestro problema.

Un problema tiene una subestructura óptima, si la solución óptima del problema principal, se puede construir a partir de soluciones óptimas de los subproblemas que derivan del él. En este caso, el problema de las monedas tiene subestructura óptima. Esto es así ya que para maximizar la ganancia de Sophia al tomar monedas entre *ini* y *fin*, su elección se reduce a una subestructura óptima; una subsecuencia de monedas de menor tamaño. Esto significa que la solución óptima para un intervalo de monedas depende de las soluciones óptimas de las subsecuencias que se generan luego de las decisiones de Sophia y Mateo.

Teniendo en cuenta lo anterior, vamos a probar la optimalidad del algoritmo usando inducción:

Base de la Inducción

Cuando tenemos una sola moneda en el intervalo ($ini == fin$), Sophia siempre tomara esa moneda, ya que es la decisión que maximiza su valor acumulado de monedas. Este caso es óptimo porque no hay decisiones adicionales, ni consideraciones para optimizar. De esta manera, el algoritmo es óptimo para intervalos de longitud 1.

Hipótesis de la Inducción

Supongamos que el algoritmo es óptimo para cualquier intervalo de longitud $k \leq n - 1$, (con n siendo la cantidad de monedas), es decir, que $matriz[ini][fin]$ calcula correctamente la ganancia máxima de Sophia cuando el intervalo *ini* a *fin* tiene longitud k .

Paso Inductivo

Vamos a probar que el algoritmo es óptimo para un intervalo de longitud $k + 1$.

Para el intervalo *ini* a *fin*, Sophia tiene dos opciones:

- Tomar la moneda en *ini*, para que luego Mateo tome la moneda más grande entre *ini* + 1 y *fin*, dejando a Sophia con un subintervalo que es más pequeño que tiene la solución óptima almacenada en la matriz.
- Tomar la moneda en *fin*, para que luego Mateo tome la moneda más grande entre *ini* y *fin* - 1, dejando nuevamente a Sophia con un subintervalo más pequeño.

Las dos opciones se pueden expresar de acuerdo a la ecuación de recurrencia de nuestro problema, recordemos:

$$matriz[ini][fin] = \max(monedas[ini] + \min(matriz[ini + 2][fin], matriz[ini + 1][fin - 1]), \\ monedas[fin] + \min(matriz[ini + 1][fin - 1], matriz[ini][fin - 2]))$$

Por la hipótesis de la inducción, sabemos que las subsoluciones $matriz[ini + 1][j - 1]$, $matriz[i + 2][fin]$, $matriz[ini][fin - 2]$ son óptimas, ya que se resuelven de intervalos más pequeños ($k - 1$) correctamente.

Dado que Sophia elige la opción que maximiza su ganancia y que cada subsolución es óptima (hipótesis), podemos concluir que $matriz[ini][fin]$ es la solución óptima para el intervalo ini a fin .

Ejemplo de Ejecución

Para concluir esta explicación, vamos a hacer un seguimiento de este algoritmo para un arreglo de 5 elementos. Vamos a considerar el arreglo [5, 10, 2, 6, 8].

Crear la matriz

Primero creamos una matriz de 5x5, y la llenamos de ceros.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Caso Base

Para los subintervalos que son de longitud 1, es decir cuando $ini = fin$, los cuales corresponden a la diagonal de la matriz, simplemente agregamos el valor de la moneda de esa posición en la matriz. En este caso, si $ini = fin = 0$, se elige la moneda de la primera posición, el 5 en este caso, luego para $ini = fin = 1$ será el 10, luego el 2, y así hasta el final.

$$\begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Llenar la matriz incrementando el tamaño de los subintervalos

Comenzamos con los subintervalos de longitud 2. Siempre ini toma valores hasta $n - longitud + 1$ (en este caso el valor es 3, que es lo máximo que el inicio puede ir para que el subintervalo de longitud 2), y fin , por cada valor de ini toma el valor $ini + longitud - 1$. Los valores iniciales ahora entonces serán $ini = 0$, $fin = 0 + 2 - 1 = 1$. Con estos definidos, empezamos a aplicar la ecuación de recurrencia:

$$\begin{aligned} \text{opcion_izquierda} &= \text{monedas}[0] + \min(\text{matriz}[0 + 2][1], \text{matriz}[0 + 1][1 - 1]) \\ \text{opcion_izquierda} &= 5 + \min(0, 0) \quad (\text{son } 0 \text{ porque } ini + 2 > fin \text{ y } ini + 1 > fin - 1) \\ \text{opcion_izquierda} &= 5 \end{aligned}$$

Ahora para el otro extremo:

$$\begin{aligned} \text{opcion_derecha} &= \text{monedas}[1] + \min(\text{matriz}[0 + 1][1 - 1], \text{matriz}[0][1 - 2]) \\ \text{opcion_derecha} &= 10 + \min(0, 0) \\ \text{opcion_derecha} &= 10 \end{aligned}$$

Por último, guardamos el máximo valor en $matriz$:

$$\text{matriz}[\text{ini}][\text{fin}] = \max(\text{opcion_izquierda}, \text{opcion_derecha})$$

$$\text{matriz}[0][1] = \max(5, 10)$$

$$\text{matriz}[0][1] = 10$$

Entonces la matriz nos queda con un nuevo valor

$$\begin{bmatrix} 5 & 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Los proximos valores que tomaran *ini* y *fin* hasta cubrir todos los subintervalos de longitud 2 serán, 1 y 2, 2 y 3, 3 y 4 respectivamente. Siguiendo el mismo procedimiento que antes, pero de acuerdo a estos valores, llegaremos a que:

$$\text{matriz}[1][2] = 10$$

$$\text{matriz}[2][3] = 6$$

$$\text{matriz}[3][4] = 8$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Estas son las soluciones optimas para cada subintervalo de longitud 2, una vez terminado con estos, pasamos a los de longitud 3. Para este caso, *ini* va a comenzar en 0, y *fin* = 0 + 3 - 1 = 2. Entonces comenzamos utilizando la ecuación de recurrencia:

$$\text{opcion_izquierda} = \text{monedas}[0] + \min(\text{matriz}[0+2][2], \text{matriz}[0+1][2-1])$$

$$\text{opcion_izquierda} = 5 + \min(2, 10)$$

$$\text{opcion_izquierda} = 7$$

Ahora para el otro extremo:

$$\text{opcion_derecha} = \text{monedas}[2] + \min(\text{matriz}[0+1][2-1], \text{matriz}[0][2-2])$$

$$\text{opcion_derecha} = 2 + \min(10, 5)$$

$$\text{opcion_derecha} = 7$$

Por último, guardamos el maximo valor en matriz:

$$\text{matriz}[\text{ini}][\text{fin}] = \max(\text{opcion_izquierda}, \text{opcion_derecha})$$

$$\text{matriz}[0][2] = \max(7, 7)$$

$$\text{matriz}[0][2] = 7$$

Entonces la matriz nos queda con un nuevo valor

$$\begin{bmatrix} 5 & 10 & 7 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Los proximos valores que tomaran *ini* y *fin* hasta cubrir todos los subintervalos de longitud 3 serán, 1 y 3, 2 y 4 respectivamente. Siguiendo el mismo procedimiento que antes, pero de acuerdo a estos valores, llegaremos a que:

$$\text{matriz}[1][3] = 12$$

$$\text{matriz}[2][4] = 10$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 7 & 0 & 0 \\ 0 & 10 & 10 & 12 & 0 \\ 0 & 0 & 2 & 6 & 10 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Ya entendimos la dinámica de este algoritmo, sabemos que ahora vamos a resolver los subproblemas de longitud 4, para finalmente, conseguir resolver el problema principal, de longitud 5, utilizando los valores utilizados anteriormente. Para los subarreglos de longitud 4, los valores de *ini* y *fin* seran, 0 y 3, 1 y 4 respectivamente. Aplicando la ecuación de recurrencia, llegamos a los siguientes resultados:

$$\text{matriz}[0][3] = 16$$

$$\text{matriz}[1][4] = 16$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 7 & 16 & 0 \\ 0 & 10 & 10 & 12 & 16 \\ 0 & 0 & 2 & 6 & 10 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Finalmente, llegamos al paso más importante, tenemos todos los resultados necesarios para resolver el problema principal. Ahora longitud vale 5, y los valores de *ini* y *fin* van a ser 0 y 4, apliquemos la ecuacion de recurrencia:

$$\text{opcion_izquierda} = \text{monedas}[0] + \min(\text{matriz}[0+2][4], \text{matriz}[0+1][4-1])$$

$$\text{opcion_izquierda} = 5 + \min(10, 12)$$

$$\text{opcion_izquierda} = 15$$

Ahora para el otro extremo:

$$\text{opcion_derecha} = \text{monedas}[4] + \min(\text{matriz}[0+1][4-1], \text{matriz}[0][4-2])$$

$$\text{opcion_derecha} = 8 + \min(12, 7)$$

$$\text{opcion_derecha} = 15$$

Por último, guardamos el maximo valor en matriz:

$$\begin{aligned}\text{matriz}[\text{ini}][\text{fin}] &= \max(\text{opcion_izquierda}, \text{opcion_derecha}) \\ \text{matriz}[0][4] &= \max(15, 15) \\ \text{matriz}[0][4] &= 15\end{aligned}$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 7 & 16 & 15 \\ 0 & 10 & 10 & 12 & 16 \\ 0 & 0 & 2 & 6 & 10 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

La posición $[0][4]$ de la matriz es el valor máximo acumulado para Sophia en este problema, ahora con la matriz completa, vamos a poder rearmar la solución.

Para rearmar soluciones, inicializamos dos arreglos vacíos, uno para Sophia y otro para Mateo, inicializamos dos variables: ini , $\text{fin} = 0$, $\text{cantidad_total_monedas} - 1$, y por último nos guardamos el valor máximo acumulado en una variable *valor_maximo*. Para este caso $\text{ini} = 0$, $\text{fin} = 4$ y $\text{valor_maximo} = 15$. Vamos a seguir el algoritmo.

Como $\text{ini} \neq \text{fin}$ y $\text{valor_maximo} > 0$ vamos a descubrir de dónde venimos. Luego de comparar, vamos a ver que habíamos elegido la moneda *ini* (número 5). Entonces le agregamos la posición de esa moneda a Sophia, le restamos el valor de la moneda a *valor_maximo*, le agregamos la posición de la moneda correspondiente para Mateo (El 10 en este caso), y actualizamos los índices para la próxima iteración, y seguimos hasta llegar alguna condición de corte.

Así los arreglos van a ser $\text{Sophia} = [0, 4, 2]$, $\text{Mateo} = [1, 3]$.

Como podemos ver, en este ejemplo Sophia no va a ganar el juego, pero el objetivo del algoritmo era encontrar las monedas que aseguran que Sophia consiga el mayor valor acumulado, lo cual es correcto en este caso.

2.3. Complejidad

Análisis de la complejidad

Para analizar la complejidad del algoritmo, se debe tener en cuenta el largo de la entrada inicial, el cual guarda una relación con el largo del array de monedas, por lo cual, diremos que nuestra complejidad, depende de 'n', siendo 'n' la cantidad de monedas en el array; luego deberemos analizar las diferentes partes del código. Al principio del código se ven inicializaciones de variables, entre las que está incluida una matriz de n*n, esto conlleva una complejidad inicial de O(n*n). Al final del algoritmo, se reconstruye la solución, lo cual lleva a recorrer todas las monedas que escogió Sophia y las que escogió Mateo, lo que nos da una complejidad de O(n) para esta parte. Por último, tenemos un for que recorrerá desde 0 hasta n (sin incluir), rellenando la matriz inicial con los casos base, aportando una complejidad temporal de O(n); como parte central del código tenemos dos bucles for, el más externo de estos se mueve entre 2 o 3 (dependiendo de si la cantidad de monedas es par o impar) y de dos en dos, hasta n (incluido), este bucle nos determinará el valor de la variable "longitud", la cual representará el largo del subarray a considerar en el bucle. El for más interno va desde 0, de uno en uno, hasta n - longitud + 1, este for le dará el valor a la variable "ini", la cual representará el índice de comienzo del subarray a considerar con respecto al array de monedas original. Mirando dentro de estos for anidados, descubrimos que solo se realizarán finitas operaciones, con lo que, dentro de los for, la complejidad es de O(1). Ahora, si consideramos que el array original tiene una longitud par, entonces veremos que para la primera pasada del bucle más externo, la longitud es de 2, por lo que el bucle más interno se moverá entre 0 y n - 2 + 1, es decir, entre 0 y n - 1, con lo que, aportando a la complejidad un O(n-1) o, mejor analizado, O(n), para la segunda pasada hay que considerar que el for se mueve de dos en dos, con lo que tendremos que la variable "longitud", valdrá 4, con lo que el bucle interno se moverá entre 0 y n - 4 + 1, es decir, entre 0 y n - 3. Siguiendo analizando, obtenemos que la cantidad de operaciones será proporcional a:

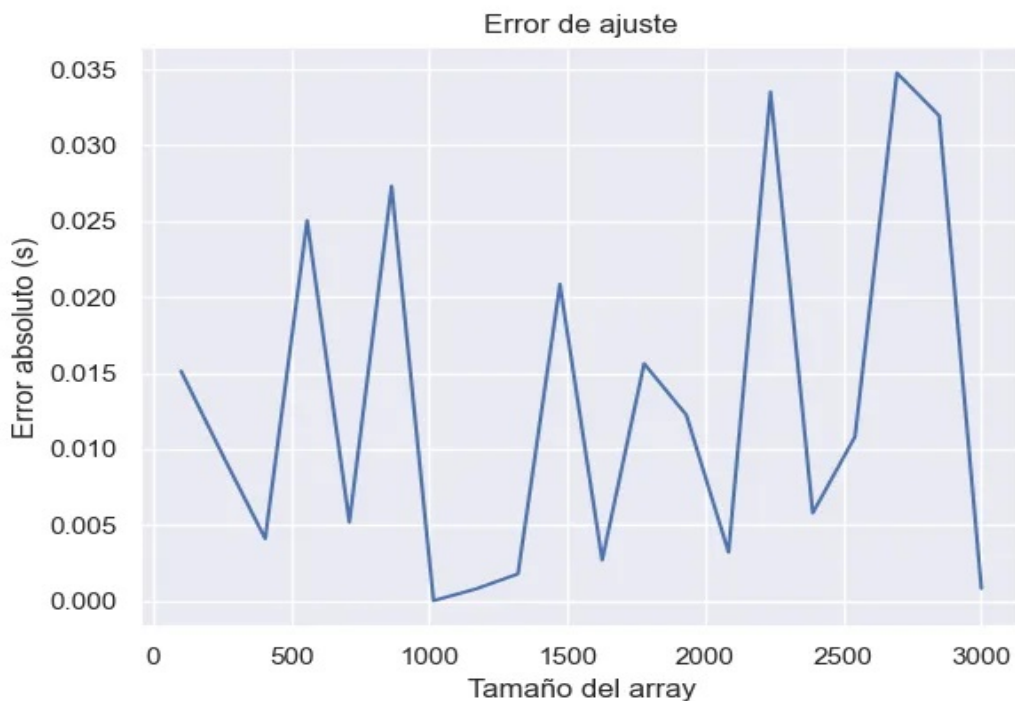
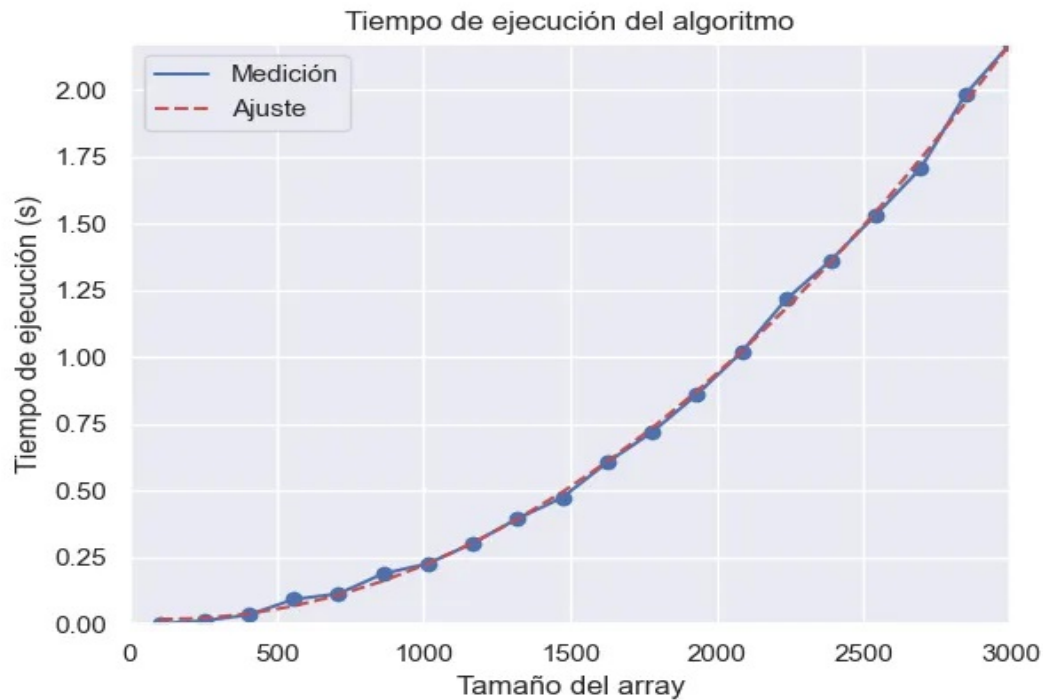
$$\blacksquare \sum_{i=1}^{(n-1)/2} n - (2i + 1) + 1 = n * \frac{n-1}{2} - 2 * \frac{n-1}{2} * \frac{\frac{(n-1)}{2} + 1}{2} + 2 * \frac{n-1}{2} = \frac{n^2}{2} - \frac{n}{2} - \frac{1}{4} * [n^2 - 2 * n + 1] - \frac{n-1}{2} + n - 1 = \frac{n^2}{4} - \frac{1}{4} - \frac{n}{2} + \frac{1}{2} + n - 1 = \frac{n^2}{4} + \frac{n}{2} - \frac{3}{4}$$

Con esto podemos deducir que la complejidad será O(n²) en los bucles anidados. Finalmente, tenemos que el algoritmo tendrá una complejidad de O(n²) + O(n) + O(n²) + O(n) = O(n²). Análogamente se puede analizar para el caso impar y se obtendrá la misma complejidad.

Complejidad teórica vs complejidad empírica

Por lo anterior, se intentará comparar los tiempos de ejecución con una función del tipo $y = a * x^2 + b * x + c$

Al hacerlo, se obtuvo lo siguiente:



En el primer gráfico se observa en forma de puntos azules los tiempos medidos del algoritmo para arrays de monedas con valores aleatorios y con la longitud especificada en el eje x. Mientras que en el segundo gráfico se observa la diferencia de tiempos que hay entre el tiempo obtenido para un determinado conjunto de monedas y el tiempo que se debería obtener según el ajuste propuesto como función que más se ajusta a los datos. Vemos que en el caso más lejano no se acerca siquiera

a una diferencia de 0,1, lo que sugiere que el modelo teórico ajusta con una exactitud muy alta a los resultados empíricos.

Relación valores/complejidad

Para analizar la relación de la complejidad con los valores de las monedas, se realizaron comparaciones con diferentes arrays de igual longitud, que poseían valores aleatorios, y comparaciones entre arrays de igual longitud, que poseían todos sus valores iguales salvo el ultimo. Mientras que en el eje y de ambos se refleja el tiempo que tardó el algoritmo en resolver el problema para los diversos arrays, en el eje x la situación es distinta para el primer caso y el segundo caso. En el caso de los valores aleatorios, se decidió que el eje x represente el promedio de los valores contenidos en los diferentes arrays. Mientras que en el segundo caso se usó, como valor del eje x, el valor que predominaba en el array. Luego de analizar los tiempos en relación a los valores en cada caso, se planteó una aproximación lineal como ajuste, de esta manera, si los valores de las monedas provocarían un cambio en el tiempo, se debería ver una clara pendiente en la recta de ajuste. De esta manera se obtuvo:





En la primera imagen se ve que se presenta una pequeña pendiente en el ajuste lineal, con una magnitud de 10^{-7} , esta pendiente se puede deber a las diferentes cargas que pudo poseer el procesador en el momento de ejecución y a la aleatoriedad de las monedas, aleatoriedad que forzó al branch predictor a una resolución cada vez más lenta. Sin embargo, se puede apreciar que es una pendiente cercana al "0", con lo cual es una aproximación cercana a un tiempo constante por cada valor promedio posible. Mientras que en el segundo gráfico se puede observar que el valor de la pendiente obtenida es incluso más bajo, acercándose a valores de incluso 10^{-8} , mostrando así una recta prácticamente paralela con el eje x, acentuándose el hecho de que el valor de las monedas no afecta al tiempo de ejecución, pero si lo puede hacer la aleatoriedad de los mismos.

3. Conclusiones

Teniendo en cuenta que el objetivo no es sólo que gane Sophia, sino también, que obtenga el máximo valor acumulado posible y que en consecuencia a eso pueda ganar o no, el algoritmo propuesto es óptimo, ya que resuelve el problema utilizando una estructura que garantiza soluciones óptimas a los subproblemas más pequeños, construyendo así la solución al problema completo. Además, la complejidad $O(n^2)$ es adecuada para resolver instancias de tamaño moderado del problema, como se evidencia en las mediciones realizadas.

4. Correcciones

Ecuación de Recurrencia

$$\text{matriz}[\text{ini}][\text{fin}] = \max(\text{opcion_izquierda}, \text{opcion_derecha})$$

Donde:

- $\text{matriz}[\text{ini}][\text{fin}]$ es la sumatoria máxima que Sophia puede tener con las monedas desde la posición ini hasta la posición fin .
- opcion_izquierda es el valor que obtiene Sophia si elige la moneda en el extremo izquierdo (ini). Este valor se calcula como: $\text{opcion_izquierda} = \text{monedas}[\text{ini}] + \text{ganancia_segun_mateo_izq}$. Donde $\text{ganancia_segun_mateo_izq}$ es: $\text{matriz}[\text{ini} + 2][\text{fin}]$ (si $\text{ini} + 2 \leq \text{fin}$) si mateo elige la moneda en $\text{ini} + 1$, o $\text{matriz}[\text{ini} + 1][\text{fin} - 1]$ (si $\text{ini} + 1 \leq \text{fin} - 1$) si mateo elige la moneda fin .
- opcion_derecha es el valor que obtiene Sophia si elige la moneda en el extremo derecho (fin). Este valor se calcula como: $\text{opcion_derecha} = \text{monedas}[\text{fin}] + \text{ganancia_segun_mateo_der}$. Donde $\text{ganancia_segun_mateo_der}$ es: $\text{matriz}[\text{ini} + 1][\text{fin} - 1]$ (si $\text{ini} + 1 \leq \text{fin} - 1$) si mateo elige la moneda en ini , o $\text{matriz}[\text{ini}][\text{fin} - 2]$ (si $\text{ini} \leq \text{fin} - 2$) si mateo elige la moneda fin .

Entonces, la ecuación de recurrencia completa sería:

$$\text{matriz}[\text{ini}][\text{fin}] =$$

- 0 si $\text{ini} > \text{fin}$
- $\text{monedas}[\text{ini}]$ si $\text{ini} = \text{fin}$
- $\max(\text{monedas}[\text{ini}] + \text{ganancia_segun_mateo_izq}, \text{monedas}[\text{fin}] + \text{ganancia_segun_mateo_der})$, para otros casos

Paso Inductivo

Corregimos, la ecuación de recurrencia que tenemos que seguir aquí es:

$$\text{matriz}[\text{ini}][\text{fin}] = \max(\text{monedas}[\text{ini}] + \text{ganancia_segun_mateo_izq}, \text{monedas}[\text{fin}] + \text{ganancia_segun_mateo_der})$$

Ejemplo de ejecución

Vamos a hacer un seguimiento del algoritmo para el arreglo [5, 10, 2, 6, 8], teniendo en cuenta la ecuación de recurrencia corregida.

Crear la matriz

Primero creamos una matriz de 5x5, y la llenamos de ceros.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Caso Base

Para los subintervalos que son de longitud 1, es decir cuando $ini = fin$, los cuales corresponden a la diagonal de la matriz, simplemente agregamos el valor de la moneda de esa posición en la matriz. En este caso, si $ini = fin = 0$, se elige la moneda de la primera posición, el 5 en este caso, luego para $ini = fin = 1$ será el 10, luego el 2, y así hasta el final.

$$\begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Llenar la matriz incrementando el tamaño de los subintervalos

Comenzamos con los subintervalos de longitud 2. Siempre ini toma valores hasta $n - longitud + 1$ (en este caso el valor es 3, que es lo máximo que el inicio puede ir para que el subintervalo de longitud 2), y fin , por cada valor de ini toma el valor $ini + longitud - 1$. Los valores iniciales ahora entonces serán $ini = 0$, $fin = 0 + 2 - 1 = 1$. Con estos definidos, empezamos a aplicar la ecuación de recurrencia:

$$\begin{aligned} \text{opcion_izquierda} &= \text{monedas}[0] + \text{matriz}[0+1][1-1] \text{ (Mateo va a tomar la moneda de la} \\ &\quad \text{posición 1, no le queda otra)} \\ \text{opcion_izquierda} &= 5 + 0 \text{ (es 0 porque } ini + 1 > fin - 1) \\ \text{opcion_izquierda} &= 5 \end{aligned}$$

Ahora para el otro extremo:

$$\begin{aligned} \text{opcion_derecha} &= \text{monedas}[1] + \text{matriz}[0+1][1-1] \text{ (de nuevo única opción para Mateo)} \\ \text{opcion_derecha} &= 10 + 0 \\ \text{opcion_derecha} &= 10 \end{aligned}$$

Por último, guardamos el máximo valor en matriz :

$$\begin{aligned} \text{matriz}[ini][fin] &= \max(\text{opcion_izquierda}, \text{opcion_derecha}) \\ \text{matriz}[0][1] &= \max(5, 10) \\ \text{matriz}[0][1] &= 10 \end{aligned}$$

Entonces la matriz nos queda con un nuevo valor

$$\begin{bmatrix} 5 & 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Los próximos valores que tomarán ini y fin hasta cubrir todos los subintervalos de longitud 2 serán, 1 y 2, 2 y 3, 3 y 4 respectivamente. Siguiendo el mismo procedimiento que antes, pero de acuerdo a estos valores, llegaremos a que:

$$\begin{aligned} \text{matriz}[1][2] &= 10 \\ \text{matriz}[2][3] &= 6 \\ \text{matriz}[3][4] &= 8 \end{aligned}$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Estas son las soluciones optimas para cada subintervalo de longitud 2, una vez terminado con estos, pasamos a los de longitud 3. Para este caso, *ini* va a comenzar en 0, y *fin* = 0 + 3 - 1 = 2. Entonces comenzamos utilizando la ecuación de recurrencia:

$\text{opcion_izquierda} = \text{monedas}[0] + \text{matriz}[0+2][2]$ (Mateo elige la moneda izquierda en su subarreglo)

$$\text{opcion_izquierda} = 5 + 2$$

$$\text{opcion_izquierda} = 7$$

Ahora para el otro extremo:

$\text{opcion_derecha} = \text{monedas}[2] + \text{matriz}[0][2-2]$ (Mateo elige la moneda de la derecha en su subarreglo)

$$\text{opcion_derecha} = 2 + 5$$

$$\text{opcion_derecha} = 7$$

Por último, guardamos el maximo valor en matriz:

$$\text{matriz}[\text{ini}][\text{fin}] = \max(\text{opcion_izquierda}, \text{opcion_derecha})$$

$$\text{matriz}[0][2] = \max(7, 7)$$

$$\text{matriz}[0][2] = 7$$

Entonces la matriz nos queda con un nuevo valor

$$\begin{bmatrix} 5 & 10 & 7 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Los proximos valores que tomaran *ini* y *fin* hasta cubrir todos los subintervalos de longitud 3 serán, 1 y 3, 2 y 4 respectivamente. Siguiendo el mismo procedimiento que antes, pero de acuerdo a estos valores, llegaremos a que:

$$\text{matriz}[1][3] = 12$$

$$\text{matriz}[2][4] = 10$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 7 & 0 & 0 \\ 0 & 10 & 10 & 12 & 0 \\ 0 & 0 & 2 & 6 & 10 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Ya entendimos la dinámica de este algoritmo, sabemos que ahora vamos a resolver los subproblemas de longitud 4, para finalmente, conseguir resolver el problema principal, de longitud 5, utilizando los valores utilizados anteriormente. Para los subarreglos de longitud 4, los valores de *ini* y *fin* serán, 0 y 3, 1 y 4 respectivamente. Aplicando la ecuación de recurrencia, llegamos a los siguientes resultados:

$$\text{matriz}[0][3] = 16$$

$$\text{matriz}[1][4] = 16$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 7 & 16 & 0 \\ 0 & 10 & 10 & 12 & 16 \\ 0 & 0 & 2 & 6 & 10 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Finalmente, llegamos al paso más importante, tenemos todos los resultados necesarios para resolver el problema principal. Ahora longitud vale 5, y los valores de *ini* y *fin* van a ser 0 y 4, apliquemos la ecuación de recurrencia:

$$\text{opcion_izquierda} = \text{monedas}[0] + \text{matriz}[0+2][4]$$

$$\text{opcion_izquierda} = 5 + 12$$

$$\text{opcion_izquierda} = 15$$

Ahora para el otro extremo:

$$\text{opcion_derecha} = \text{monedas}[4] + \text{matriz}[0][4-2]$$

$$\text{opcion_derecha} = 8 + 7$$

$$\text{opcion_derecha} = 15$$

Por último, guardamos el maximo valor en matriz:

$$\text{matriz}[\text{ini}][\text{fin}] = \max(\text{opcion_izquierda}, \text{opcion_derecha})$$

$$\text{matriz}[0][4] = \max(15, 15)$$

$$\text{matriz}[0][4] = 15$$

Quedandonos entonces la matriz:

$$\begin{bmatrix} 5 & 10 & 7 & 16 & 15 \\ 0 & 10 & 10 & 12 & 16 \\ 0 & 0 & 2 & 6 & 10 \\ 0 & 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

La posición $[0][4]$ de la matriz es el valor máximo acumulado para Sophia en este problema, ahora con la matriz completa, vamos a poder rearmar la solución.

Para rearmar soluciones, inicializamos dos arreglos vacíos, uno para Sophia y otro para Mateo e inicializamos dos variables: *ini*, *fin* = 0, *cantidad_total_monedas* - 1, y por último nos guardamos el valor máximo acumulado en una variable *valor_maximo*. Para este caso *ini* = 0, *fin* = 4 y *valor_maximo* = 15. Vamos a seguir el algoritmo.

Como $ini \neq fin$ y $valor_maximo > 0$ vamos a descubrir de donde venimos. Primero inicializamos las variables *siguiente_ini* y *siguiente_fin*, con los valores que corresponden a los índices del subarreglo que nos queda luego de la elección de Sophia (líneas 12 y 21 del código de *rearrear_solucion*, son 2 llamados porque uno es para ver si Sophia elige la moneda izquierda, y otro para la derecha).

Ahora nos preguntamos si la moneda que corresponde elegir en este caso es la izquierda o la derecha (línea 13 y 20 del código de *rearrear_solucion*), en este caso, como $valor_maximo == monedas[ini] + matriz[siguiente_ini][siguiente_fin]$, confirmamos que la moneda que elige Sophia es la de la izquierda. Agregamos la posición *ini* en este caso al arreglo de Sophia y restamos el valor de $monedas[ini]$ a $valor_maximo$ ($15 - 5 = 10$). Antes de pasar a la siguiente iteración, nos fijamos que moneda va a elegir Mateo comparando la variable *siguiente_ini* con los índices que Mateo puede elegir, y le agregamos el índice que corresponde a la moneda de mayor valor.

Por último, actualizamos el valor de *ini* y *fin* haciendo $ini, fin = siguiente_ini, siguiente_fin$ (líneas 28 y 29 del código de *rearrear_solucion*), para trabajar con el subarreglo correspondiente la próxima iteración.

Seguiremos aplicando el mismo razonamiento, hasta llegar a que si $ini = fin$, vamos a guardar en el arreglo de Sophia la posición *ini* y terminamos, o si $valor_maximo = 0$, simplemente terminamos y en ambos casos devolvemos los arreglos de Sophia y Mateo.

Así los arreglos van a ser $Sophia = [0, 4, 2]$, $Mateo = [1, 3]$.

Como podemos ver, en este ejemplo Sophia no va a ganar el juego, pero el objetivo del algoritmo era encontrar las monedas que aseguran que Sophia consiga el mayor valor acumulado, lo cual es correcto en este caso.