Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

12 de septiembre de 2024

Valentín Schneider 107964
Ian von der Heyde 107638
Juan Martín de la Cruz 109588

Índice

1. Introducción y primeros años

Cuando Mateo nació, Sophia estaba muy contenta. Finalmente tendría un hermano con quien jugar. Sophi tenía 3 años cuando Mateo nació. Ya desde muy chicos, ella jugaba mucho con su hermano.

Pasaron los años, y fueron cambiando los juegos. Cuando Mateo cumplió 4 años, el padre de ambos le explicó un juego a Sophia: 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).

El problema es que Mateo es aún pequeño para entender cómo funciona esto, por lo que Sophia debe elegir las monedas por él. Digamos, Mateo está "jugando". Aquí surge otro problema: Sophia es muy competitiva. Será buena hermana, pero no se va a dejar ganar (consideremos que tiene 7 nada más). Todo lo contrario. En la primaria aprendió algunas cosas sobre algoritmos greedy, y lo va a aplicar.

1.1. Consigna

- Hacer un análisis del problema, y proponer un algoritmo greedy que obtenga **la solución óptima** al problema planteado: Dados los n valores de todas las monedas, indicar qué monedas debe ir eligiendo Sophia para si misma y para Mateo, de tal forma que se asegure de **ganar siempre**. Considerar que Sophia siempre comienza (para sí misma).
- Demostrar que el algoritmo planteado obtiene siempre la solución óptima (desestimando el caso de una cantidad par de monedas de mismo valor, en cuyo caso siempre sería empate más allá de la estrategia de Sophia).
- Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta la variabilidad de los valores de las diferentes monedas a los tiempos del algoritmo planteado.
- Analizar si (y cómo) afecta la variabilidad de los valores de las diferentes monedas a la optimalidad del algoritmo planteado.
- Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares que deben cumplirse su optimalidad también.
- Hacer mediciones de tiempos para corroborar la complejidad teórica indicada. Agregar los casos de prueba necesarios para dicha corroboración. Esta corroboración empírica debe realizarse confeccionando gráficos correspondientes, y utilizando la técnica de cuadrados mínimos. Para esto, proveemos una explicación detallada, en conjunto de ejemplos.
- Agregar cualquier conclusión que les parezca relevante.

1.2. ¿Qué es un algoritmo Greedy?

Un algoritmo greedy es una estrategia de diseño algorítmico que toma decisiones paso a paso, eligiendo en cada paso la opción que parece ser la mejor o más óptima en ese momento, sin reconsiderar decisiones previas. Su objetivo es construir una solución óptima global tomando decisiones locales óptimas en cada paso. Sin embargo, plantear un algoritmo de este tipo es un proceso de análisis, prueba y error, y su éxito depende de que el problema cumpla con las propiedades necesarias para garantizar que las elecciones locales conduzcan a la solución óptima global.

2. Análisis del problema

El problema consiste en una fila de monedas de diferentes valores, y dos jugadores que deben alternarse en elegir monedas. La elección solo puede ser de la primera o la última moneda de la fila. El objetivo radica en maximizar la suma del valor de las monedas obtenidas al final del juego por parte de Sophia.

2.1. Ejemplo del juego

Para desarrollar una breve explicación sobre el juego planteado en el problema, daremos un ejemplo de nuestra protagonista Sophia jugando contra su amigo Lionel. Para ello, supongamos que se tiene una fila de monedas con los valores $[3, 9, 2, 7]$.

- **Turno 1:** Sophia elige la última moneda, de valor 7. Ahora la fila es $[3, 9, 2]$.
- **Turno 2:** Lionel elige la primera moneda, de valor 3. Ahora la fila es $[9, 2]$.
- **Turno 3:** Sophia elige la primera moneda, de valor 9. Ahora la fila es $[2]$.
- **Turno 4:** Lionel elige la única moneda restante, de valor 2.

Resultado: Sophia tiene un total de $7 + 9 = 16$, mientras que Lionel tiene $3 + 2 = 5$. Sophia gana el juego con un total de 16.

3. Análisis de posibles soluciones

3.1. Elegir siempre un mismo extremo

3.1.1. Descripción del algoritmo

Una estrategia que consiste en elegir siempre el mismo extremo de la fila de monedas (ya sea el primero o el último) no es óptima y no sigue los principios de la estrategia *greedy*. Aunque se aplica una regla simple de forma constante, este enfoque no maximiza el valor acumulado a lo largo del juego, ya que no toma en consideración las opciones disponibles en cada turno. Esto lleva a perder oportunidades valiosas de seleccionar monedas de mayor valor. Además, esta estrategia no contempla que Sophia puede influir en la elección de las monedas que tome Mateo, lo que le permitiría maximizar su ventaja.

3.1.2. Contraejemplo

A pesar que este algoritmo no implica seguir con la definición de uno *greedy*, planteamos un contraejemplo considerando el siguiente conjunto de monedas:

$$[5, 20, 4, 10]$$

Supongamos que Sophia sigue la estrategia de elegir siempre la última moneda disponible. El juego se desarrollaría de la siguiente manera:

- **Turno 1:** Sophia elige la última moneda, 10. La nueva fila es $[5, 20, 4]$.
- **Turno 2:** Mateo elige la primera moneda, 5. La nueva fila es $[20, 4]$.
- **Turno 3:** Sophia elige la última moneda, 4. La nueva fila es $[20]$.
- **Turno 4:** Mateo elige la moneda restante, 20.

Resultado: Sophia ha obtenido las monedas 10 y 4, con un valor total de 14. Mateo ha obtenido las monedas 5 y 20, con un valor total de 25.

En este caso, Sophia se lamenta, ya que la estrategia de elegir siempre la última moneda le impidió tomar monedas de mayor valor como la moneda 20, que fue posteriormente tomada por Mateo. Para obtener mejores resultados, es necesario evaluar ambas opciones (primera y última moneda) en cada turno y tomar decisiones más informadas.

La complejidad del algoritmo propuesto para encontrar el máximo es $\mathcal{O}(n)$, debido a que para cada elemento del arreglo se realizan operaciones $\mathcal{O}(1)$.

3.2. Algoritmo de sumas

3.2.1. Descripción del algoritmo

Otra solución posible sería recorrer el arreglo y calcular dos sumas: una para los valores en las posiciones pares y otra para los valores en las posiciones impares. Sophia, que juega primero, seleccionaría una moneda de una posición par si la suma de las posiciones pares es mayor que la suma de las posiciones impares, o viceversa. De esta manera, bloquearía todas las posiciones pares o impares para Mateo.

Al final del juego, Sophia tendría una mayor suma de monedas al haberse apropiado de todas las posiciones de mayor valor. Sin embargo, este enfoque presenta varios problemas. En primer lugar, solo funciona correctamente si el tamaño del arreglo es par, restricción que no está incluida en el enunciado. En caso contrario, no sería posible asignar de manera equitativa todas las monedas a posiciones pares o impares.

Veamos un ejemplo, dado el siguiente arreglo:

$$[1, 2, 70, 5]$$

Siguiendo este algoritmo, tenemos:

$$\text{Suma de valores de monedas en posiciones impares} = 1 + 70 = 71$$
$$\text{Suma de valores de monedas posiciones pares} = 2 + 5 = 7$$

Según la regla de este algoritmo, Sophia en su primer turno elegiría la moneda en la posición impar, es decir, el 1. Cabe mencionar que esta no es una decisión óptima localmente, ya que el 1 tiene un valor mucho menor que el 5, que también está disponible. No obstante, en el siguiente turno, Mateo solo podría elegir los pares y así desbloquear la moneda de máximo valor para Sophia.

3.2.2. Justificación del descarte

A pesar de su eficiencia en el caso particular donde el arreglo está constituido por n monedas y n sea par, este enfoque no puede considerarse un algoritmo *greedy* en sentido estricto, ya que no toma decisiones óptimas en cada turno basándose en el estado actual del juego. En lugar de ello, toma una decisión al inicio del juego y sigue repitiendo esa heurística sin importar cómo cambien las opciones disponibles. Esto lo convierte en una estrategia rígida que no se adapta a las circunstancias cambiantes y por ende, la descartamos como una posible solución al problema planteado en el enunciado.

3.3. Sophia el máximo, Mateo el mínimo

3.3.1. Descripción del algoritmo

La solución *greedy* que encontramos a este problema es que para cada mano, si juega Sophia para si misma, que elija la moneda mas grande de las dos. Por otro lado, si esta jugando para Mateo, que elija la menor, por lo cual desarrollamos la solución con esta regla.

4. Algoritmo para obtener óptimo

En esta sección, se presenta el análisis del algoritmo propuesto (3.3) que ha permitido obtener una solución óptima al problema planteado. A continuación, describimos de manera detallada los pasos del algoritmo y justificamos por qué podría garantizar el resultado más óptimo posible.

4.1. ¿El algoritmo elegido es Greedy?

Definimos como heurística la siguiente resolución.

Denotamos una fila de monedas $A = [a_1, a_2, \ldots, a_n]$, donde a_1 es la primera moneda y a_n es la última moneda, t al número de turnos jugados hasta el momento, y n como el número total de turnos posibles.

Para garantizar que Sophia siempre gane, debemos considerar cómo optimizar su estrategia en cada turno. Esto implica alternar entre tomar el valor máximo disponible en un turno y el valor mínimo en otro, dependiendo de lo que sea más beneficioso para ella en ese momento. Dado que Sophia tiene control sobre todas las decisiones de elección de monedas, puede aplicar ambas estrategias, ajustando sus decisiones según el estado actual del juego para maximizar su ventaja acumulada.

- Para los **turnos impares** (t impar, es decir, cuando juega Sophia), Sophia siempre toma el valor máximo entre los extremos de la fila:

$$\text{Si } t \text{ es impar}, \quad \text{Sophia elige } \max(a_1, a_n)$$

  Luego, remueve el elemento $\max(a_1, a_n)$ de la fila.

- Para los **turnos pares** (t par, es decir, cuando "juega"Mateo pero Sophia elige por él), Sophia elige el valor mínimo entre los extremos de la fila para Mateo:

$$\text{Si } t \text{ es par}, \quad \text{Sophia elige } \min(a_1, a_n)$$

  Luego, remueve el elemento $\min(a_1, a_n)$ de la fila.

En cada turno t, después de elegir el valor a_i, el nuevo arreglo de monedas A se actualiza a $A' = [a_1, \ldots, a_{n-1}]$, y el proceso continúa hasta que no queden más monedas.

Formalmente, si t es impar (Sophia juega), se tiene:

$$A_t = A_{t-1} \setminus \{\max(a_1, a_n)\}$$

Y si t es par (Sophia elige por Mateo), se tiene:

$$A_t = A_{t-1} \setminus \{\min(a_1, a_n)\}$$

Este proceso se repite hasta que la fila de monedas esté vacía.

Presentando esta regla sencilla, podemos decir que cumple con lo que llamamos regla *greedy*, ya que se irá repitiéndose iterativamente para conseguir óptimos locales buscando maximizar su ganancia inmediata y minimizar la de su contrincante hasta llegar al óptimo global y ganar el juego.

4.2. Demostración

Supongamos que tenemos el arreglo [A, C, B], en donde A>B y desconocemos el valor de C. Se completa un turno cuando tanto Sophia como Mateo hayan sacado una moneda.

[A, C, B]

Sophia comienza sacando siempre el máximo de entre los extremos (A).

[C, B]

Quedan solamente B y C. Mateo va a sacar el menor entre ambos por lo que tenemos tres casos posibles:

- **Caso 1: B > C:**
  Como B>C y A>B entonces, por transitividad, A>C.

  Y siempre se cumple que A>B porque ya lo eligió Sophia. Mateo entonces elige C, que es menor que lo que eligió Sophia.

- **Caso 2: C > B:**
  Ahora mateo entre C y B elige B por ser el menor. Si B hubiera sido mayor que A, Sophia lo hubiera elegido en el turno anterior.
  Entonces otra vez, lo que eligió Sophia es mayor a lo que eligió Mateo para un mismo turno.

- **Caso 3: B=C:**
  Al tener mismo valor mateo elige cualquiera de las 2. De todas maneras A>B y A>C.

  Un ejemplo numérico para entender mejor el caso 2:

  [75, 100, 50]
  Juega Sophia: max(75, 50) -> 75
  [100, 50]
  Juego Mateo: min(100,50) -> 50

  El 50 de la derecha necesariamente es menor a lo que tomo Sophia, porque es el que descarto. Entonces Mateo ahora va a elegir el menor entre lo que descarto Sophia (50), y lo que queda a izquierda (100), así que si o si va a elegir algo menor a lo que eligió Sophia en este turno.

- **Aclaración con arreglo par:** En el caso en el que se tenga un arreglo de largo par, y entonces el ultimo turno lo tenga Mateo, de igual manera se cumple lo anterior. Ya que por mas que Mateo en ese ultimo caso no este eligiendo lo menor posible, esta tomando lo que Sophia descarto, porque ella se quedo con la moneda mas grande que había para ese turno.

Por lo anterior queda demostrado que turno a turno Sophia elige una moneda de valor mayor o igual a la de Mateo. Por lo que la suma de todas las monedas de Sophia será necesariamente mayor suma de las de Mateo (o igual en el caso de que tengamos una cantidad par de monedas de igual valor)

4.2.1. Inversiones

Decimos que existe una inversión en la secuencia de decisiones si, al evaluar dos monedas en las posiciones i y j del arreglo, donde $i < j$, se elige la moneda de la posición j antes que la de la posición i, pero el valor de la moneda en j es menor que el valor de la moneda en i. En términos más sencillos, hay una inversión si un jugador elige primero una moneda de menor valor, dejando para más adelante una de mayor valor que ya estaba disponible en los extremos del arreglo.

Sucede que todos las posibles elecciones sin inversiones Sophia siempre obtiene el mismo valor en la suma final de sus monedas elegidas.

Podemos demostrar que un esquema de elecciones sin inversiones es siempre óptimo. Supongamos lo contrario, que existe un esquema de elecciones óptimo que tiene inversiones. Esto significaría que el jugador está, en algún punto, eligiendo una moneda de menor valor cuando tenía la oportunidad de tomar una de mayor valor, lo cual no tiene sentido en el contexto de maximizar la puntuación dada en la solución óptima.

Si se intercambian las elecciones de las monedas que generan la inversión (es decir, se elige la de mayor valor primero), la diferencia de puntuación entre los jugadores no empeora, y de hecho, podría mejorar. Esto significa que cualquier esquema de decisiones que tenga inversiones puede transformarse en otro sin inversiones, obteniendo como mínimo el mismo resultado (o incluso una mejor).

Dado que podemos eliminar las inversiones sin empeorar el resultado, podemos concluir que cualquier esquema óptimo no puede tener inversiones. Por lo tanto, la estrategia greedy que siempre elige la moneda de mayor valor disponible es una solución óptima para este problema.

5. Implementación del algoritmo óptimo

Presentamos la implementación del algoritmo propuesto utilizando el lenguaje de programación *Python*. El código ha sido estructurado de manera que siga una lógica clara y modular, facilitando su lectura, mantenimiento y futuras extensiones.

La elección de *Python* se debe a su legibilidad y a las bibliotecas avanzadas que ofrece para la manipulación de estructuras de datos y la optimización de procesos. El enfoque utilizado para la implementación refleja fielmente el análisis presentado anteriormente, asegurando que se mantengan las propiedades de optimalidad y eficiencia.

5.1. Código funcional

```python
def turno_s(lista, inicio, fin):
    return inicio if lista[inicio] >= lista[fin] else fin

def turno_m(lista, inicio, fin):
    return fin if lista[inicio] >= lista[fin] else inicio

def jugar(lista):
    turno = "s"
    suma_s = 0
    suma_m = 0
    inicio = 0
    res = []
    fin = len(lista)-1
    while fin >= inicio:
        if turno == "s":
            saco = turno_s(lista, inicio, fin)
            suma_s += lista[saco]
            if saco == inicio:
                res.append("Primera moneda para Sophia")
                inicio += 1
            else:
                res.append("Ultima moneda para Sophia")
                fin -= 1
            turno = "m"
        else:
            saco = turno_m(lista, inicio, fin)
            suma_m += lista[saco]
            if saco == inicio:
                res.append("Primera moneda para Mateo")
                inicio += 1
            else:
                res.append("Ultima moneda para Mateo")
                fin -= 1
            turno = "s"
    return res, suma_s, suma_m
```

El análisis paso a paso del mismo será reflejado en la explicación posterior de la *complejidad*.

5.2. Complejidad

La complejidad teórica de nuestra solución es lineal. Vamos a analizarlo más a detalle sección a sección:

La función `jugar()` comienza inicializando variables (tiempo constante)

```
7  def jugar(lista):
8      turno = "s"
9      suma_s = 0
10     suma_m = 0
11     inicio = 0
12     res = []
13     fin = len(lista)-1
```

Luego inicia un ciclo que recorrerá todo el arreglo (tiempo lineal) aplicando nuevamente operaciones constantes y llamando a las funciones `turno_s()` y `turno_m()` (para mayor simplicidad solo mostramos el turno de Sophia, el de Mateo es analogo).

```
14     while fin >= inicio:
15         if turno == "s":
16             saco = turno_s(lista, inicio, fin)
17             suma_s += lista[saco]
18             if saco == inicio:
19                 res.append("Primera moneda para Sophia")
20                 inicio += 1
21             else:
22                 res.append("Ultima moneda para Sophia")
23                 fin -= 1
24             turno = "m"
```

Las funciones `turno_s()` y `turno_m()` tambien realizan operaciones constantes:

```
1  def turno_s(lista, inicio, fin):
2      return inicio if lista[inicio] >= lista[fin] else fin
3
4  def turno_m(lista, inicio, fin):
5      return fin if lista[inicio] >= lista[fin] else inicio
```

En conclusión, la complejidad teórica de nuestra solución es lineal, es decir $\mathcal{O}(n)$ con n siendo la cantidad de monedas en el juego.

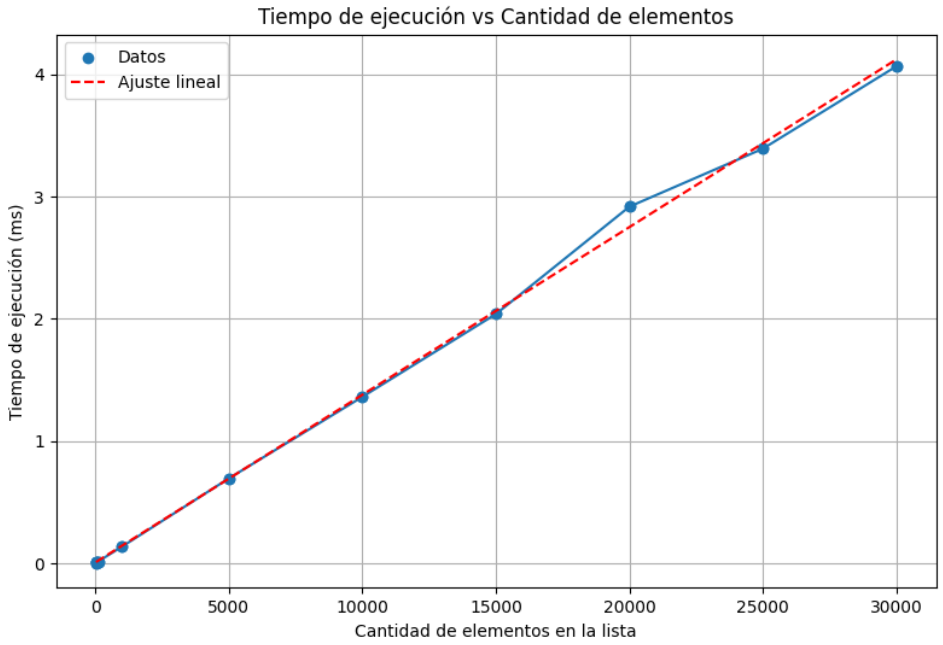
5.3. Impacto de la variabilidad de las monedas en la complejidad y optimalidad

Lo que podría cambiar con la variabilidad de los valores es **cómo se distribuyen las puntuaciones entre Sophia y Mateo**, pero no el resultado, el número de operaciones realizadas ni la complejidad temporal del algoritmo. En otras palabras, tanto si las monedas tienen valores pequeños o grandes, o si tienen gran variación entre ellas, el número de comparaciones realizadas y el resultado final serán los mismos.

6. Mediciones

Se realizaron mediciones en base a arreglos de diferentes cantidades de elementos (los elementos en cada caso fueron generados por valores aleatorios)

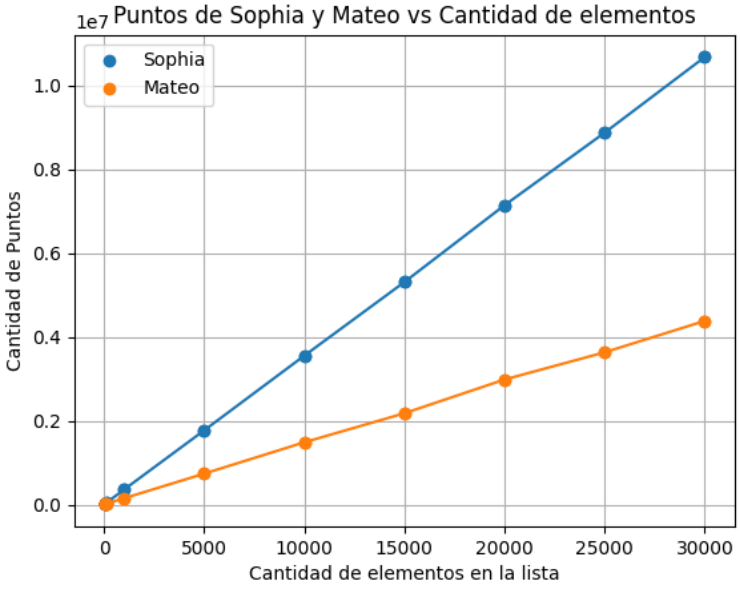
6.1. Tiempo de ejecución vs Cantidad de elementos



Como se puede apreciar, el algoritmo es de complejidad lineal, lo que coincide con lo analizado en el punto anterior.

6.2. Puntos vs Cantidad de elementos

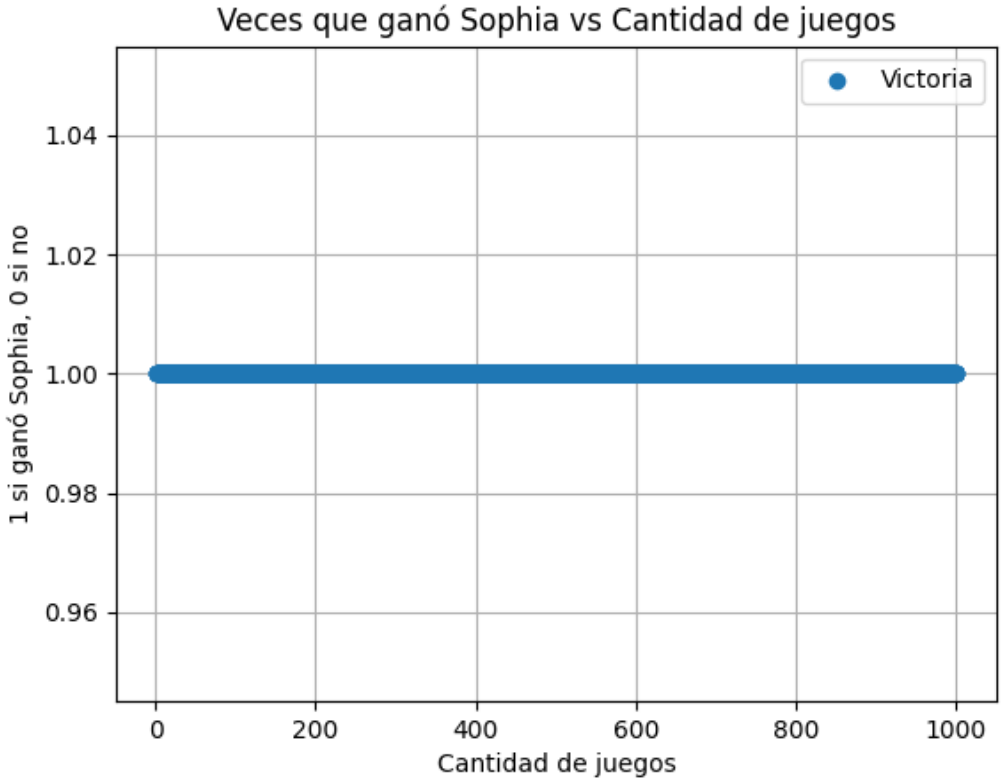
Para mostrar que Sophia gana en todas las partidas, también graficamos la cantidad de puntos ganados por cada jugador vs la cantidad de elementos que tenia la pila de monedas en cada partida.



Podemos observar una tendencia lineal entre la cantidad de puntos y la cantidad de elementos en la lista. Esta relación se debe principalmente a la distribución de las monedas que utilizamos, ya que (por simplicidad) generamos todos los arreglos limitados bajo los mismos rangos de cotas para las monedas (1-999). Como las monedas se distribuyen de manera similar, el crecimiento en la cantidad de puntos sigue un patrón predecible y consistente con el aumento en la cantidad de elementos en la lista.

6.3. Cantidad de victorias vs Cantidad de juegos

Por ultimo, se muestra una funcion que grafica 1 si gano Sophia, 0 si gano Mateo. Lo realizamos con mil simulaciones del juego con valores generados aleatoriamente.



Demostrando una vez mas que nuestro algoritmo es óptimo ya que siempre gana Sophia.

7. Conclusiones

El presente trabajo demuestra cómo un enfoque greedy bien diseñado puede resolver de manera eficiente el problema propuesto, garantizando una solución óptima. Destacamos que la estrategia greedy elegida no solo fue la más *natural*, sino también la más adecuada para maximizar el rendimiento y, a pesar de una posible obviedad, nos enfocamos en evaluar diversas estrategias de resolución. Sin embargo, tras un examen detallado, todas ellas fueron refutadas por no cumplir con los requisitos de eficiencia o por no garantizar una solución óptima. Algunas opciones presentaron enfoques más complejos o indirectos, pero tras una evaluación teórica, se determinó que no lograban mejorar el rendimiento ni garantizar un resultado óptimo de manera consistente.

Este proceso de refutación fue clave para confirmar que, a pesar de su simplicidad, el enfoque greedy no solo era el más directo, sino también el más sólido para este tipo de problema. Su

capacidad para optimizar localmente en cada turno conduce de manera natural a una solución globalmente óptima, lo que lo convierte en la mejor estrategia para resolver el problema con una complejidad temporal eficiente.

En resumen, Sophia elige siempre la moneda que optimiza su beneficio en cada turno, mientras minimiza el de Mateo. En el peor de los casos, si las monedas tienen el mismo valor, el resultado será un empate, pero nunca una derrota para Sophia.

La complejidad temporal del algoritmo es $\mathcal{O}(n)$, lo que lo convierte en una solución eficiente, incluso para filas largas de monedas. Esto se debe a que en cada turno solo es necesario comparar los valores en los extremos de la fila, una operación constante que se repite n veces.

La variabilidad en los valores de las monedas no afecta la complejidad del algoritmo, aunque sí influye en la distribución de los puntos entre Sophia y Mateo. Cuando los valores de las monedas difieren significativamente, la diferencia de puntuación tiende a ser mayor a favor de Sophia.

Los resultados empíricos obtenidos confirman que la implementación cumple con la complejidad teórica estimada. Los tiempos de ejecución observados siguen un comportamiento lineal, lo que valida la eficiencia del algoritmo en contextos de gran volumen de datos.

En conclusión, la simplicidad y eficiencia del enfoque greedy propuesto muestran que, en ocasiones, no es necesario recurrir a métodos más complejos para garantizar soluciones rápidas y óptimas.