Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmo Greedy

21 de septiembre de 2024

Jonathan Dominguez
110057

Manuel Pato
110640

Mateo Godoy Serrano
110912

1. Introducción

Cuando Mateo nació, Sophia estaba muy contenta. Finalmente tendría un hermano con quien jugar. Sophia 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 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

1. 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 sí misma y para Mateo, de tal forma que se asegure de ganar siempre. Considerar que Sophia siempre comienza (para sí misma).

2. 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).

3. 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.

4. Analizar si (y cómo) afecta la variabilidad de los valores de las diferentes monedas a la optimalidad del algoritmo planteado.

5. 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.

6. 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.

7. Agregar cualquier conclusión que les parezca relevante.

2. Análisis del Problema

Para comenzar, tenemos que tener una idea de qué es un algoritmo greedy. Un algoritmo greedy es aquel que frente a un problema aplica una regla o restricción sencilla de forma constante con tal de encontrar una solución local al mismo. Una vez teniendo eso, espera aplicar esta solución por cada estado óptimo local con la esperanza de llegar a una solución global. Entonces, ¿qué es lo que buscamos hacer para este caso?

Partiendo de la base de que todas las monedas son positivas y distintas, se puede comenzar con la solución y no existe la posibilidad de empate ya que, sea una cantidad de monedas par o impar, Sophia siempre comienza el juego y agarra la más grande.

Sabemos que la condición para que gane Sophia es:

$$\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} S_m[i] > \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} M_m[i]$$

Donde:

- n: La cantidad de monedas totales.
- S_m: El arreglo de monedas de Sophia que tiene m elementos. $S_m[i]$ es el elemento i de dicho arreglo.
- M_m: El arreglo de monedas de Mateo.

No está de más mencionar que si la cantidad de monedas es impar, Sophia va a tener $\lfloor \frac{n}{2} \rfloor + 1$ monedas, mientras que Mateo va a seguir con $\lfloor \frac{n}{2} \rfloor$.

3. Implementación y descripción del algoritmo

```python
def juego_de_la_moneda(monedas):
    arr_sophia, arr_mateo = [], []
    i, j = [0], [len(monedas) -1]
    es_turno_de_sophia = True

    while i[0] <= j[0]:
        if es_turno_de_sophia:
            elige_sophia(i, j, arr_sophia, monedas)
        else:
            elige_mateo(i, j, arr_mateo, monedas)
        es_turno_de_sophia = not es_turno_de_sophia

    return arr_sophia, arr_mateo
```

Este fragmento de código es el encargado de realizar los turnos. Donde cada iteración es un turno distinto. El bucle define el tiempo de ejecución de nuestro algoritmo dándonos así un tiempo lineal. Cada iteración vamos reduciendo en 1 nuestra longitud del array monedas sea por izquierda o derecha.

```python
def elige_sophia(i, j, arr_sophia, monedas):
    if monedas[i[0]] >= monedas[j[0]]:
        arr_sophia.append(monedas[i[0]])
        i[0] += 1
    else:
        arr_sophia.append(monedas[j[0]])
        j[0] -= 1
```

```python
def elige_mateo(i, j, arr_mateo, monedas):
    if monedas[i[0]] <= monedas[j[0]]:
        arr_mateo.append(monedas[i[0]])
        i[0] += 1
    else:
        arr_mateo.append(monedas[j[0]])
        j[0] -= 1
```

Indistitanmente agarre Sophia o Mateo, cada uno tiene su condición para agarrar una u otra moneda. Y la guarda en su respectivo arreglo.

4. Análisis de la solución

4.1. Análisis de la complejidad

Naturalmente, dada la estructura del problema, donde se cuenta con un arreglo de tamaño n, lo mínimo que podemos esperar es que la distribución del total de las monedas entre Sophia y Mateo se implemente en un algoritmo de complejidad $O(n)$, ya que se necesita recorrer todos los elementos del arreglo al menos una vez. En este caso, logramos acotar la complejidad temporal por, justamente, $O(n)$, haciéndolo lo más óptimo posible en ese aspecto.

4.2. Análisis de la influencia de la variabilidad de los argumentos

El tamaño del arreglo moneda (denotado por n) afecta de manera lineal al tiempo de ejecución del algoritmo. La variabilidad en los valores de las monedas no influye en el resultado del juego ni en su funcionamiento. Es importante señalar que los valores negativos, nulos (cero) y las monedas duplicadas no están permitidos. Pero aún así considerando el caso de que existan monedas negativas o alguna con valor nulo, el algoritmo va a funcionar de todas maneras. El único caso que sería contraejemplo es cuando todas son iguales y n es un valor par, ya que terminaría en un empate.

En definitiva, independientemente de los valores específicos en monedas, siempre y cuando se respeten las restricciones mencionadas, la ganadora será siempre Sophia, y el efecto que tiene el aumento del tamaño del arreglo de monedas será lineal, teniendo el algoritmo una complejidad temporal $O(n)$.

4.3. Calificación como algoritmo greedy

Cabe preguntarse si se trata de un algoritmo greedy. Recordando lo que esto implica, supóngase que tanto Sophia como Mateo tienen 0 monedas cada uno (es decir, el estado inicial del problema). Si se demuestra que en este estado, el algoritmo toma la alternativa óptima al problema, se contará con un caso base; luego, dado que existe una cantidad m de monedas que tienen tanto Sophia como Mateo para el cual el algoritmo toma la alternativa óptima, basta con probar que para la cantidad $m + 1$ también lo hará.

Retomando, en el caso de $m = 0$, el algoritmo realizará la primera iteración, siendo el turno de Sophia; según el algoritmo, se toma la moneda de mayor valor. Seguidamente, en la segunda iteración es el turno de Mateo, y Sophia elige la moneda de menor valor para él, garantizando así que la suma total de sus monedas es mayor a la de su hermano (ya que las monedas deben ser necesariamente distintas por la hipótesis del problema). Luego, existe una cantidad de monedas m para la que el algoritmo toma la alternativa óptima. Después de esa iteración, ambos hermanos tienen $m + 1$ monedas; como el algoritmo no cambia su comportamiento según la cantidad de monedas de cada hermano, el proceso es el mismo: primero, en la iteración $2(m+1)$, Sophia toma la moneda de mayor valor, y en la iteración $2(m+1)+1$, toma la de menor valor para Mateo. Estas elecciones constituyen la alternativa óptima del estado actual. Así, independientemente de si esto conduce al óptimo global, se asegura de tomar siempre el óptimo local, permitiendo categorizar al algoritmo como un algoritmo greedy sin lugar a duda.

4.4. Optimalidad del algoritmo

Similar a la idea anterior, se propone el par de elecciones de monedas $[m_0, m_0 + 1]$ como caso base. Por inducción, demostraremos que, dado que este caso base cumple con la optimalidad global parcial del algoritmo, entonces existe un par de elecciones $[m, m + 1]$ que cumplen con que el algoritmo alcanza un óptimo global en ese par, y usaremos esta hipótesis para demostrar que esto implica que el par de elecciones $[m + 2, m + 3]$ también constituye un óptimo global parcial, implicando así que el algoritmo es totalmente óptimo global.

Asumamos que la cantidad de monedas, primero, sea par y mayor a 2. Sea el par de elecciones de monedas $[1, 1+1] = [1, 2]$, que corresponden a la primera y segunda iteración respectivamente. En este caso, como se trató en la verificación de la condición greedy del algoritmo, Sophia elige la "mejor"moneda (la de más valor) para sí y la "peor"moneda (la de menor valor) para Mateo; notar que, pese a que la elección de Sophia modifica potencialmente la elección para Mateo, ya que los extremos del arreglo se actualizan, si la nueva moneda que queda descubierta tras la elección de Sophia es mejor que la que ella tomó, necesariamente tomará la moneda del otro extremo para Mateo, garantizando la optimalidad global parcial en las iteraciones 1 y 2. Luego, existe un m para el que el par de elecciones $[m, m+1]$ haga que el algoritmo alcance un estado óptimo global parcial. Basta con probar que esto implica que el par de elecciones $[m+2, m+3]$ también alcanza un óptimo global parcial. Sea $t = 1 + \frac{m}{2}$ la cantidad de elementos de ambos arreglos para el par de elecciones $[m, m+1]$, que el algoritmo alcance el óptimo global parcial quiere decir que, dados S_t, M_t los arreglos de Sophia y Mateo con t monedas cada uno, respectivamente, entonces:

$$\sum_{i=1}^{t} S_t[i] > \sum_{i=1}^{t} M_t[i]$$

Para la cantidad t, esta desigualdad se cumple por hipótesis. Luego, en la elección $m+2$, Sophia toma la moneda de mayor valor entre los extremos (siendo su valor x), y, en la elección $m+3$ tomará la de menor valor para Mateo (siendo su valor y). Luego, para el par de elecciones $[m+2, m+3]$, se tiene que:

$$\sum_{i=1}^{t} S_t[i] + x > \sum_{i=1}^{t} M_t[i] + y$$

Cumpliendo la desigualdad, y garantizando el estado óptimo global parcial. Luego, dada esta relación de implicancia, se obtiene indefectiblemente que el algoritmo necesariamente llega al óptimo global total para una cantidad de monedas par y mayor a 2. El caso de cantidad impar es más sencillo, porque entonces debería existir un m que sea el número de la última elección de Sophia; habiendo garantizado la optimalidad del algoritmo hasta que quede más de una sola moneda, Sophia tomará la última, aún prevaleciendo el estado de optimalidad final. Para dos monedas, Sophia tomará la mayor moneda para ella y la menor para Mateo; para una moneda, Sophia se queda con la única moneda. Esto cubre la totalidad de casos, probando que el algoritmo es totalmente óptimo.

5. Mediciones

Para verificar la complejidad deducida de nuestro algoritmo realizamos una serie de mediciones empíricas, donde testeamos el algoritmo con distintos tamaños de arreglos y, luego, verificamos mediante el método de **cuadrados mínimos** que la complejidad es la indicada en las secciones anteriores. Si bien no es de interés particular detallar el mecanismo exacto por el cual aproximamos la complejidad a través de las mediciones, sí lo es observar cuán acertada fue la deducción; para ello, preparamos unos gráficos que ilustran los resultados de las mediciones reales, nuestra aproximación por la complejidad $O(n)$, y el error de cada aproximación dado el tamaño de entrada del arreglo.

5.1. Medición de nuestro algoritmo

Figura 1: Mediciones

Para este caso usamos una muestra inicial de 5 mil elementos, hasta 100 mil, escalando el tamaño del arreglo de a 5 mil elementos. A su vez, se le asignó en el otro eje su tiempo de ejecución en milisegundos.

5.2. Comparación con el ajuste

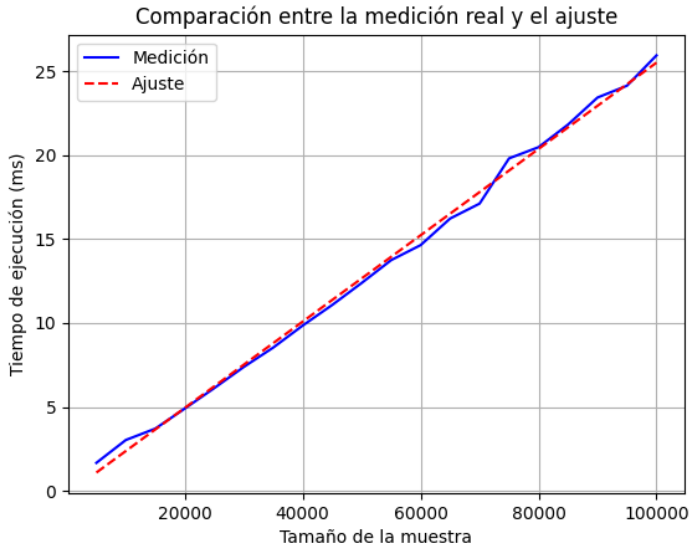

Figura 2: Comparación

Como se puede apreciar, la recta ajusta óptimamente a las mediciones realizadas, indicando que el algoritmo es, efectivamente, de complejidad $O(n)$.

5.3. Errores de la comparación

Figura 3: Errores y ECT

Finalmente, el **error cuadrático total** que proporciona la función elegida (la función que coincide con la complejidad el algoritmo, $f(n) = n$) para nuestra aproximación es, con estas mediciones, 3,02449...ms^2.

5.4. Aclaraciones sobre las mediciones

Unas aclaraciones que queremos dar antes de cerrar, es que, los gráficos que se usaron para demostrar el funcionamiento de nuestro algoritmo es uno de tantos que estuvimos creando en tres computadoras totalmente distintas. Esto se debe a que el algoritmo siempre va a depender del equipo/sistema en el que se trabaje, como también los componentes que lleve. Esos factores pueden generar que sea más o menos rápido frente a una muestra. Si bien no se notó un cambio en la complejidad del mismo, si hubo cotas más o menos grandes sobre el ECT.

6. Conclusiones

Habiendo desglosado el problema y la solución, pudimos observar, finalmente, que el problema se resumía en elaborar un criterio de elección sobre un arreglo de números, que satisface condiciones de unicidad y positividad de cada número, que permita separar al arreglo original en dos, garantizando que la suma de los elementos de uno sea mayor a la del otro. Alcanzamos este objetivo a través de un algoritmo greedy implementado en Python, aplicando una regla sumamente sencilla, y sin asunciones adicionales a las del algoritmo, garantizando la complejidad temporal mínima del algoritmo, que es $O(n)$, aunque sí usando memoria adicional.

Una consideración adicional del algoritmo es que demostramos su optimalidad a través de una demostración inductiva, con lo que podemos afirmar que, dada nuestra implementación, Sophia siempre vencerá a Mateo en el juego de las monedas.

7. Anexo 1

Se realizaron cambios en la implementación original del algoritmo del juego de la moneda: ahora se lleva registro de las elecciones que realiza Sophia tanto para sí como para su hermano. A continuación, se ilustra el código:

```
def juego_de_la_moneda(monedas):
    arr_sophia, arr_mateo, pasos = [], [], []
    i, j = [0], [len(monedas) -1]
    es_turno_de_sophia = True

    while i[0] <= j[0]:
        if es_turno_de_sophia:
            elige_sophia(i, j, arr_sophia, monedas, pasos)
        else:
            elige_mateo(i, j, arr_mateo, monedas, pasos)
        es_turno_de_sophia = not es_turno_de_sophia

    return (arr_sophia, arr_mateo, pasos)

def elige_sophia(i, j, arr_sophia, monedas, pasos):
    if monedas[i[0]] >= monedas[j[0]]:
        arr_sophia.append(monedas[i[0]])
        pasos.append('Primera moneda para Sophia')
        i[0] += 1
    else:
        pasos.append('Ultima moneda para Sophia')
        arr_sophia.append(monedas[j[0]])
        j[0] -= 1

def elige_mateo(i, j, arr_mateo, monedas, pasos):
    if monedas[i[0]] <= monedas[j[0]]:
        arr_mateo.append(monedas[i[0]])
        pasos.append('Primera moneda para Mateo')
        i[0] += 1
    else:
        arr_mateo.append(monedas[j[0]])
        pasos.append('Ultima moneda para Mateo')
        j[0] -= 1
```