.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Los Algoritmos Greedy son Juegos de Niños

30 de septiembre de 2024

Benjamin Castellano Bogdan	Joaquin Eduardo Embon	Lionel Gabriel Maydana Gonzalez
111519	111292	106512

1. Introduccion

El propósito de este trabajo práctico es consolidar nuestros conocimientos sobre los algoritmos Greedy, tema que hemos abordado en las primeras clases de la asignatura. Nuestra tarea consiste en primero analizar el problema brindado en el enunciado y luego conseguir su solución más óptima con un algoritmo Greedy.

Luego, nos vamos a enfocar en poder entender en profundidad el funcionamiento de este algoritmo, analizando su complejidad temporal, y estudiando también la optimalidad del mismo. Para ello vamos a proponer demostraciones, mostrar ejemplos de prueba del algoritmo, y gráficos que sean enriquecedores y terminen por ayudarnos a entender su comportamiento

1.1. ¿Qué es un Algoritmo Greedy?

Son algoritmos que iterativamente aplican una regla sencilla que nos permite obtener en cada paso el óptimo local a nuestro problema actual. Esta sucesión de óptimos locales, nos permite llegar al óptimo global de este mismo problema.

2. Análisis del problema

Disponemos de un juego con n monedas en fila, el cual consiste por turno, ir sacando una moneda. Pero no se puede elegir cualquiera, sino, 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. Los jugadores serán Sophia y Mateo, y jugarán hasta que no quede ninguna moneda.

2.1. Consideraciones

- Sophia elegirá la moneda tanto para ella, como para mateo y buscará ganar a toda costa.
- Sophia siempre comienza el juego (para sí misma).
- Deberemos buscar el algorítmo Greedy que obtenga siempre la solucion óptima, desestimando el caso de una cantidad par de monedas del mismo valor, en cuyo caso siempre sería empate más allá de la estrategia de Sophia.

3. Algoritmo

3.1. Codigo

```
def juego_sophia_mateo(monedas):
    monedas_sophia = []
    monedas_mateo = []
    es_turno_de_sophia = True
    ini = 0
    fin = len(monedas) - 1
    while ini <= fin:
        monedas_jugador = monedas_sophia
        posicion_moneda_a_agregar = posicion_moneda(monedas, ini, fin, maximo=
    es_turno_de_sophia)
        if not es_turno_de_sophia:
            monedas_jugador = monedas_mateo
        monedas_jugador.append(posicion_moneda_a_agregar)
        if posicion_moneda_a_agregar == fin:
            fin -= 1
        else:
            ini += 1
        es_turno_de_sophia = not es_turno_de_sophia

    return monedas_sophia, monedas_mateo
```

La funcion 'juego_sophia_mateo' recibe por parámetro un arreglo n de monedas. Se realiza un ciclo, en el que por cada iteración tomamos decisiones en función de quién es el turno. Si es el turno de Sophia, seleccionamos la moneda de mayor valor entre la moneda de la posición 'ini' y la de la posición 'fin', las cuales representan las monedas del principio y el fin del arreglo para esa iteración. Si el turno es de Mateo, tenemos en cuenta la misma cantidad de opciones, pero en este caso eligiendo la moneda de menor valor. Las monedas que van siendo elegidas, van dejando de ser tenidas en cuenta, hasta que ya no nos queden monedas por elegir.

```
def posicion_moneda(monedas, ini, fin, maximo):
    if maximo:
        if monedas[ini] >= monedas[fin]:
            return ini
        return fin
    else:
        if monedas[ini] < monedas[fin]:
            return ini
        return fin
```

Esta función 'posicion_moneda', es la que se encarga de, dependiendo de quien sea el turno, de buscar o bien, la posición de la moneda de mayor valor (si es el turno de Sophia), o la posición de la moneda de menor valor (si es el turno de Mateo). La función sabe qué criterio tiene que seguir dependiendo del valor del parametro booleano 'maximo'.

3.2. Optimalidad

Vamos a llamar a una iteración del Algoritmo, al par de momentos, Turno de Sophia – Turno de Mateo.

- Consideramos A como la moneda del principio y B a la moneda del final. El primer turno es el de Sophia. Si $A > B$, elijo A. Consideraremos ahora como A' es la moneda que ocupa el lugar de A. Ahora es el turno de Mateo. Si pensamos en nuestro objetivo, no nos afecta si A' es mayor que A, ya que para Mateo, vamos a elegir la moneda con el menor valor entre A' y B, y como sabemos desde un principio que $B < A$, necesariamente Mateo va a sumar una cantidad menor que la que Sophia sumo anteriormente. Esto quiere decir que, para cada iteración, Sophia siempre va a sumar una cantidad mayor que su hermano.
- En el caso de que la cantidad de monedas sea par, como la cantidad de turnos de Sophia sera

igual a la cantidad de turnos de Mateo, siguiendo la lógica anterior, Sophia siempre ganaría el juego.

- Para el caso en el que la cantidad de monedas es impar, la diferencia es que Sophia tendría un tiro extra, lo cual solo ayudará a que Sophia acumule una sumatoria de cantidades mayor.

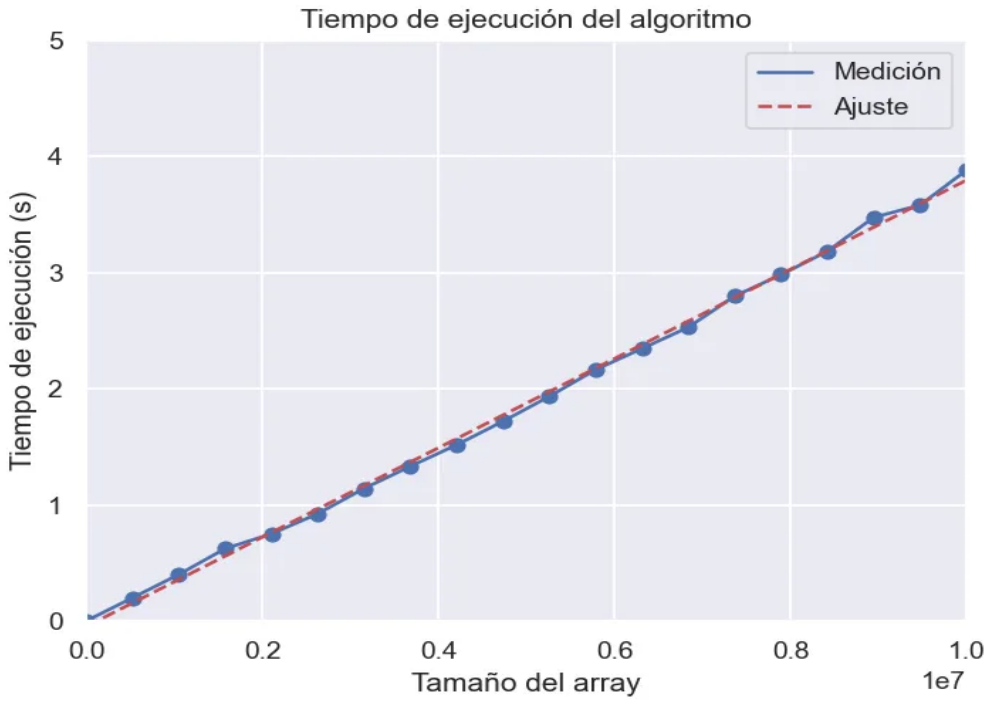
De esta manera, nos aseguramos que el algoritmo siempre da una solución óptima.

Es interesante remarcar lo que fue mencionado anteriormente, y es que la optimalidad de este algoritmo no se ve afectada por los diferentes valores que las monedas pueden tomar, el criterio que se utiliza, funciona de la misma manera en todos los casos posibles.

3.3. Complejidad

Para analizar la complejidad temporal del algoritmo propuesto, hay que observar las iteraciones realizadas, puesto que en el algoritmo 'juego_sophia_mateo' solo hay un bucle while, el cual va avanzando o retrocediendo una posición por ciclo según la moneda que se eligió, este bucle nos proporcionara n iteraciones. Ahora bien, adentrándonos en el código dentro del bucle, vemos que no se realizan operaciones que conlleven una complejidad mayor que O(1), lo que termina dando que dentro de cada una de las n iteraciones, se realizan finitas operaciones de complejidad O(1), por lo tanto, el algoritmo propuesto posee una complejidad de O(n). Para demostrar que el análisis es correcto y demostrar que los cálculos teóricos coinciden estrechamente con la realidad, se presentaran gráficos empíricos obtenidos a partir de casos de uso reales.

Se realizaron cálculos para 20(veinte) cantidades distintas de monedas, cantidades que oscilan entre 100(cien) y 10.000.000(diez millones), de manera creciente para cada cantidad, con valores aleatorios para cada moneda.



En el gráfico de arriba, se muestra en azul, el tiempo consumido por el algoritmo en función de la cantidad de monedas que poseía el arreglo del juego, donde los puntos azules representan un caso verdadero y mientras tanto, las líneas azules, representan la unión de estos puntos, esperando así generar el gráfico de una función muy similar a una recta. En el gráfico también se muestra

una línea punteada roja, la cual hace alusión a la función lineal esperada que se llegara a graficar con los datos de los casos reales. Como se ve en el gráfico, ambas funciones son muy cercanas, lo que lleva a una idea de aproximación acertada.

Errores

Al mirar el gráfico conseguido contra el grafico que debería de conseguirse en la teoría, se puede apreciar que, a pesar de ser bastante similares entre sí, no hay una concordancia absoluta entre ellos. Por esto último, surge la pregunta ¿cuál es el *error* que se comete entre el gráfico obtenido a partir de la experiencia y el gráfico que teóricamente debería conseguirse? Para dar una idea aproximada del *error*, se puede ver el siguiente gráfico.

En el gráfico anterior se observa la diferencia que hay entre el gráfico teórico y el gráfico obtenido en función del largo del problema. Al verlo, se denota que esta diferencia no llega a los 0,1s (segundos) de diferencia, ni siquiera para largos de 10.000.000 (diez millones). Por lo tanto, se puede considerar como un *error* muy acotado que una vez mas resalta la cercanía de los valores teóricos con los valores prácticos.

Relación Complejidad/valor

Para analizar la relación de la complejidad del algoritmo con respecto a los valores de las monedas en el array, se realizaron simulaciones del juego, con valores de monedas que oscilaban de manera aleatoria entre dos valores, para la misma cantidad de monedas. Repitiendo este test para intervalos crecientes que no se superponían entre sí, se consiguió el primer gráfico, que representa el tiempo que tarda el algoritmo en función del promedio de los valores de las monedas en cada

prueba. Mientras que el segundo gráfico corresponde a mismas cantidades de monedas, con valores de monedas iguales, salvo la última moneda del array, la cual posee solo un poco más de valor que las demás (uno más), para evitar el empate. En este caso, el gráfico representa el tiempo que tarda el algoritmo en función del valor que aparece mayoritariamente en el array (que, para el largo del array, se traduce prácticamente en el promedio). Para cada caso, para conseguir el tiempo graficado, se realizó un promedio entre 10 tiempos diferentes que correspondían al mismo array de monedas.

En el primer gráfico se observa que hay una pequeña pendiente en la recta de ajuste, y que los valores medidos oscilan alrededor de esta, estas oscilaciones se pueden deber a trabajos externos que pudo estar realizando el procesador en el momento de correrse la prueba y a la dificultad de predecir el resultado de los condicionales que tiene la computadora al estar tratando con valores aleatorios. Sin embargo, es una aproximación bastante buena para una recta del tipo y = k, con ɛ"siendo el tiempo medido al realizar la tarea y "Küna constante (para el caso, k entre [6, 6.5]). Mientras que, para el segundo caso, se muestra una recta de ajuste con una pendiente muy tendiente a 0, o de 0 literalmente, con tiempos medidos muy cercanos a la recta (es decir, a la constante K), lo que se traduce en una mejor aproximación a una complejidad temporal constante. Notar que el tiempo al tratar con valores iguales, en cada caso, se tradujo en un tiempo mucho menor del algoritmo, lo que decanta a pensar que la sospecha de que "se volvió mucho más difícil para la computadora resolver los condicionales con valores aleatorios", era probablemente acertada.
Como conclusión, al verse que, en ambos casos, la recta de ajuste se acerca a una recta del tipo ɛ = k", se ve que los valores de las monedas no afectan prácticamente al algoritmo, pero si lo hace la aleatoriedad de cada valor en relación con los otros valores en el array. Es altamente probable que esta aleatoriedad sea la causante de las oscilaciones en el primer gráfico.

4. Conclusiones

Nuestro objetivo desde un principio fue encontrar una solución óptima para el problema dado, específicamente mediante un Algoritmo Greedy.

Luego de una evaluación y estudio del problema, propusimos una solución que termino por satisfacer nuestras expectativas, para luego pasar a realizar un exhaustivo análisis del mismo.

Este análisis que realizado nos ayudo a entender mejor qué es lo que sucedía con nuestro algoritmo, con respecto a su complejidad y optimalidad.

Nos parece muy interesante como es que los gráficos mostrados en las secciones anteriores dan una clara, y a su vez, una forma simple de entender el comportamiento del algoritmo en la práctica, dandonos una visión que se correspondió con la teoría matemática calculada anteriormente.

5. Correcciones

5.1. Optimalidad

¿El algoritmo es Greedy?

El algoritmo que nostros proponemos en la sección del codigo, y es solución del problema es greedy, ya que sigue una regla que busca obtener un óptimo local para cada iteración del algoritmo (En este caso elegir la mayor moneda para Sophia y la menor para Mateo), y nos termina llevando a el óptimo global (Sophia gana el juego).

Demostración Dado que Sophia está maximizando sus ganacias y minimizando las de Mateo, en cada paso está garantizando el mayor margen de diferencia. Esto se puede demostrar formalmente usando una estategia de inducción:

Paso base (n = 2 monedas):
Si hay dos monedas, m_1 y m_n, Sophia tiene dos opciones:

- Si $m_1 >= m_n$, Sophia toma m_1 y le deja m_n a Mateo.
- Si $m_1 < m_n$, Sophia toma m_n y le deja m_1 a Mateo.

En este caso, claramente Sophia maximiza su ganancia porque elige la moneda de mayor valor y deja a Mateo la de menos valor.

Paso inductivo: Supongamos que el algoritmo es óptimo para cualquier secuencia de k monedas, y probemos que sigue siendo óptimo para $k + 2$ monedas.

Para $k + 2$ monedas (una fila de monedas), Sophia puede elegir la primera moneda m_1 o la última moneda m_{k+2}:

- Si elige m_1, entonces queda con las monedasm_2,...,m_{k+2}, y aplicando el algoritmo greedy a esta secuencia más corta, Sophia sigue eligiendo de manera óptima tanto para ella como para Mateo.
- Si elige m_{k+2}, entonces queda con las monedas m_1,...,m_{k+1}, y de nuevo aplica el algoritmo greedy a esta secuencia.

En ambos casos, Sophia está garantizando que maximiza su ganancia local y, dado que las decisiones futuras también siguen la misma estrategia greedy, maximiza el resultado global.

Consideramos:

- A = $\sum_{i=1}^{n} a_i$ como la sumatoria de las monedas de Sophia
- B = $\sum_{i=1}^{n} b_i$ como la sumatoria de monedas de Mateo

Queremos demostrar que si $\forall(a_i, b_i), a_i > b_i$, entonces $A > B$. Vamos a considerar la diferencia $D = A - B$.

$$D = A - B$$

$$D = \sum_{i=1}^{n} a_i \text{ - } \sum_{i=1}^{n} b_i$$

$$D = \sum_{i=1}^{n} (a_i - b_i)$$

Como sabemos que $a_i > b_i$, es decir, $a_i - b_i > 0$. Entonces podemos decir que:

$$D = \sum_{i=1}^{n} (a_i - b_i) > 0$$

Lo anterior es lo mismo que decir que $A - B = 0$. Lo que nos lleva a concluir, que la sumatoria A, siguiendo nuestra regla greedy, siempre será mayor que la sumatoria B, lo que equivale a decir que Sophia siempre gana el juego.

La demostración anterior vale para el caso en el que Sophia y Mateo agarren la misma cantidad de monedas(la cantidad de monedas dadas al principio es par). Si la cantidad de monedas fuera impar, sabemos que Sophia es la que se queda con una última moneda extra. Entonces sería:

$$D = \sum_{i=1}^{n+1} a_i \text{ - } \sum_{i=1}^{n} b_i$$

$$D = a_{n+1} + \sum_{i=1}^{n} a_i - \sum_{i=1}^{n} b_i$$

Este termino extra se puede despreciar, ya que solo aumenta el valor de la sumatoria de Sophia. Entonces quedamos en el mismo caso que antes.

Es interesante remarcar lo que fue mencionado anteriormente, y es que la optimalidad de este algoritmo no se ve afectada por los diferentes valores que las monedas pueden tomar, el criterio que se utiliza, funciona de la misma manera en todos los casos posibles.

5.2. Complejidad

Dado el arreglo de monedas [**1, 3, 2, 4**], la secuencia sería la siguiente:

Turno de Sophia: (Siempre arranca Sophia)

- Tiene para elegir entre el primer o último lugar (1 o 4).
- Elige el máximo, osea, el 4.
- Lo "toma" y lo guarda.
- El arreglo queda: [**1, 3, 2**]

Turno de Mateo:

- Tiene para elegir entre el primer o último lugar (1 o 2).
- Elige el mínimo, osea, el 1.
- Lo "toma" y lo guarda.
- El arreglo queda: [**3, 2**]

Turno de Sophia:

- Tiene para elegir entre el primer o último lugar (3 o 2).
- Elige el máximo, osea, el 3.
- Lo "toma" y lo guarda.
- El arreglo queda: [**2**]

Turno de Mateo:

- En ese caso ya no tiene que elegir, sólo le queda el 2.
- Lo "toma" y lo guarda.

Ahora que no hay más monedas en el arreglo, el juego finalizó, a lo que se devolverá dos arreglos. Un arreglo contiene las monedas que eligió Sophia y otro con las que eligió Mateo. En nuestro main implementamos que, por medio de otras funciones, se devuelva cuales fueron las "jugadas" y la sumatoria de los puntos de Sophia y Mateo, para este caso, seria así:

Última moneda para Sophia; Primera moneda para Mateo; Primera moneda para Sophia; Primera moneda para Mateo;
Ganancia Sophia: 7
Ganancia Mateo: 3
¿Ganó Sophia?: True