

TEORÍA DE ALGORITMOS
(75.29) CURSO BUCHWALD - GENENDER

Trabajo Práctico 1

Los Algoritmos Greedy son Juegos de Niños

× ×

20 de septiembre de 2024

Benjamin Castellano
Bogdan
111519

Joaquin Eduardo
Embon
111292

Lionel Gabriel
Maydana Gonzalez
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 algoritmo Greedy que obtenga siempre la solución 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. Código

```
1 def juego_sophia_mateo(monedas):
2     monedas_sophia = []
3     monedas_mateo = []
4     es_turno_de_sophia = True
5     ini = 0
6     fin = len(monedas) - 1
7     while ini <= fin:
8         if es_turno_de_sophia:
9             posicion_moneda_a_agregar = posicion_moneda(monedas, ini,
10                 fin, maximo=True)
11             monedas_sophia.append(monedas[posicion_moneda_a_agregar])
12             if posicion_moneda_a_agregar == fin:
13                 fin -= 1
14             else:
15                 ini += 1
16         else:
17             posicion_moneda_a_agregar = posicion_moneda(monedas, ini,
18                 fin, maximo=False)
19             monedas_mateo.append(monedas[posicion_moneda_a_agregar])
20             if posicion_moneda_a_agregar == fin:
21                 fin -= 1
22             else:
23                 ini += 1
24         es_turno_de_sophia = not es_turno_de_sophia
25
26     sumatoria_sophia = sum(monedas_sophia)
27     sumatoria_mateo = sum(monedas_mateo)
28     return sumatoria_sophia > sumatoria_mateo
```

La función '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.

```
1 def posicion_moneda(monedas, ini, fin, maximo):
2     if maximo:
3         if monedas[ini] >= monedas[fin]:
4             return ini
5         return fin
6     else:
7         if monedas[ini] < monedas[fin]:
8             return ini
9         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.

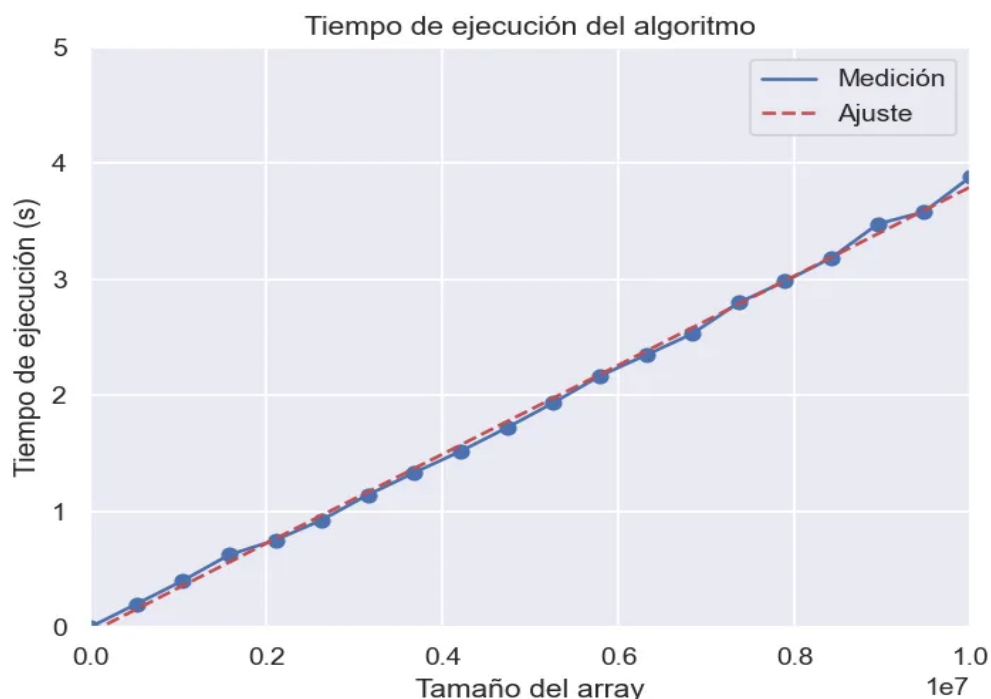
De esta manera, nos aseguramos que el algoritmo siempre da una solución óptima.

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 gráfico 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.

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.