Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico
Lineamientos sobre informes

02 de diciembre 2024

Franco de León 110223
Manuel Perez 108192
Julieta Sosa 109332
Julianna Sánchez 109621

Primera Parte: Introduccion y primeros años

Definición

Un algoritmo greedy es un algoritmo que toma decisiones localmente óptimas en cada paso con la esperanza de que estas decisiones conduzcan a una solución global óptima. Es decir, en cada etapa del algoritmo, se elige la opción que parece ser la mejor en ese momento, sin considerar las decisiones futuras.A diferencia de otros enfoques, como la programación dinámica o la búsqueda de backtracking, que pueden realizar un análisis más profundo, los algoritmos greedy simplemente hacen la elección más çodiciosa"(la que maximiza la ganancia en el corto plazo).Aunque un algoritmo greedy hace elecciones locales, se espera que la serie de decisiones conduzca a una solución global óptima o, en algunos casos, subóptima. Sin embargo, en muchos problemas, las soluciones greedy pueden no ser óptimas globalmente, pero para ciertos problemas bien definidos, es posible garantizar una solución óptima.

Análisis

El problema que se plantea en el enunciado consiste en la toma de decisiones de los jugadores(Sofia y Mateo) en cada turno. En cada turno se eligen monedas de una fila, y se debe elegir entre la primera o la última de la fila . Sin embargo Sofia juega por Mateo; con lo cual objetivo es que Sofia, por algoritmo greedy, tome las decisiones para ambos jugadores de manera óptima para garantizar su victoria. La solución consiste en que en los turnos de Sofia se tome la moneda de mayor valor, mientras que para el turno de Mateo, se tomara la moneda de menor valor. De esta forma, el algoritmo se asegura que Sofia maximice su ganancia en cada turno, mientras que el turno de Mateo consistir en tomar la decisión menos optima.

Algoritmo Planteado

Complejidad Algoritmica:

- Archivo readTxt: Dado que la lectura del archivo y el procesamiento de cada moneda ocurre una vez, la complejidad temporal de esta función es O(n), donde n es el número de monedas en el archivo.

```
1
2  def readTxt(txt):
3
4      #Input coins txt
5      coins = open(txt)
6      #Skip first line
7      coins.readline()
8      #Build array of coin values
9      coinsArray = []
10     tempNum = ""
11
12     for i in coins.read():
13         if i != ";":
14             tempNum += i
15         else:
16             coinsArray.append(int(tempNum))
17             tempNum = ""
18     coinsArray.append(int(tempNum))
19
20     return coinsArray
```

- SumArray: Recorre toda la lista de monedas seleccionadas para calcular la suma. Por lo tanto, la complejidad temporal es O(m), donde m es el número de elementos en la lista que se pasa a la función (en este caso, las monedas seleccionadas por cada jugador).

```
    def sumArray(array):
    result = 0
    for i in array:
        result += i
    return result
```

- printResults(sofia, mateo): Esta función imprime los resultados del juego, incluyendo la suma total de las monedas de cada jugador y quién ganó. Las operaciones de impresión son O(1).

```
def printResults(sofia, mateo):

    print("Sofia: ")
    #print(sofia)
    valueSofia = sumArray(sofia)
    print(valueSofia)
    print("Cantidad: " + str(len(sofia)))
    print("Mateo: ")
    #print(mateo)
    valueMateo = sumArray(mateo)
    print(valueMateo)
    print("Cantidad: " + str(len(mateo)))

    if valueSofia > valueMateo:
        print("Gano Sofia !!!!!!!!!!!!!")
    elif valueSofia < valueMateo:
        print("Gano Mateo !!!!!!!!!!!!!")
    else:
        print("Empate ????")
```

- SofiaGreedy: Recorre la lista de monedas, eligiendo monedas y eliminándolas de la fila hasta que no quede ninguna. En cada paso, se realiza una comparación entre la primera y la última moneda y se elimina la moneda seleccionada. Esto lleva O(1) para la comparación y O(n) para la eliminación, ya que coins.remove() tiene una complejidad temporal de O(n).

```

def sofiaGreedy(coins):
    #Turn positive=sofia negative=mateo
    turn = 1
    sofia = []
    mateo = []

    while coins:
        if turn > 0:
            first = coins[0]
            last = coins[-1]
            best = max(first, last)

            sofia.append(best)
            coins.remove(best)
        else:
            first = coins[0]
            last = coins[-1]
            worst = min(first, last)

            mateo.append(worst)
            coins.remove(worst)
        turn *= -1

    printResults(sofia, mateo)
```

En total, el algoritmo realiza n iteraciones y en cada una realiza una eliminación de un elemento de la lista, por lo que la complejidad temporal total de la función es O(n2).

La variabilidad de los valores de las monedas no afecta la complejidad temporal del algoritmo, ya que sigue siendo O(n2);ya que la variabilidad de los valores no cambia la cantidad de operaciones que el algoritmo necesita realizar. La complejidad temporal se basa en el número de operaciones necesarias para procesar la cantidad de datos que entren, no en los valores de esos elementos.

Optimalidad del algoritmo:

- Alta variabilidad en el valor de las monedas: el algoritmo greedy es óptimo, ya que elegir el valor más grande en cada turno garantiza una solución globalmente óptima.
- Baja variabilidad (valores similares entre sí): el algoritmo sigue funcionando a nivel local, pero el algoritmo puede no llevar a la mejor solución global.
- Valores idénticos: el algoritmo sigue siendo técnicamente óptimo, pero no tiene impacto real en la solución porque todas las decisiones son equivalentes.

  De igual forma, la optimalidad del algoritmo no se ve afectada por la variabilidad de los valores. La estrategia greedy de Sophia sigue siendo óptima porque siempre elige la moneda de mayor valor disponible, sin importar las monedas restantes.

Segunda Parte: Mateo empieza a jugar

Definición

La programación dinámica es un método de optimización basado en el principio de optimalidad, donde una política óptima consta de subpolíticas óptimas. Es una técnica poderosa que resuelve problemas descomponiéndolos en subproblemas más pequeños y combinando sus soluciones para obtener subproblemas más grandes. Este método es particularmente útil para procesos de toma de decisiones que involucran procesos secuenciales, como la gestión de la fabricación, la gestión de existencias, la estrategia de inversión y la teoría de la computación.

Análisis

El problema que se plantea en el enunciado consiste en la toma de decisiones de los jugadores(Sofía y Mateo) en cada turno. En cada turno se eligen monedas de una fila, y se debe elegir entre la primera o la última de la fila . En este caso Mateo aprendió sobre algoritmos greedy, y lo comenzó a aplicar. Esto hace que ahora quién gane dependa más de quién comience y un tanto de suerte. Ahora Sofia, por algoritmo de programación dinámica, buscara maximizar su ganancia eligiendo las monedas de manera estratégica. La solución consiste en que Sofía en su turno tome la moneda más conveniente teniendo en cuenta que Mateo siempre elegirá la moneda de mayor valor en la mesa, para eso Sofía tendrá que pensar de manera estratégica sobre las monedas que quedarían disponibles en el siguiente turno de acuerdo a la que ella elija. De esta forma, el algoritmo se asegura de ver los caminos de acuerdo a cada elección de moneda que haga, maximizando su ganancia y asegurando la victoria para Sofía.

Ecuación de Recurrencia

$$dp(i) = \text{máx}(dp(i-1), v[i] + dp(i-2))$$

- dp(i): Representa el valor máximo que puede obtener Sophia si juega con las primeras i monedas restantes.
- (i): Es el valor de la moneda en la posición i de la secuencia.
- v(i-1): Si Sophia decide no tomar la moneda y deja que Mateo elija entre las monedas restantes, el valor máximo que puede obtener Sophia será el valor máximo obtenido por el subproblema con i-1 monedas.
- v(i)+dp(i-2): Si Sophia decide tomar la moneda i, entonces el valor máximo que puede obtener es v(i) mas el valor máximo obtenido de las i-2 monedas restantes.