**562 Dividing coins**

1. Explique brevemente la estrategia de la solución y como hace uso de la estrategia de Memoización.

   Para solucionar el problema, se razonó que en el caso ideal la suma de los valores de todas las monedas sería par y se podría dividir en dos partes iguales. Por lo tanto, se concluyó que el valor_total // 2 sería una cota superior para la suma de las monedas de una persona en la repartición más justa. De este modo, el problema se reduce a encontrar el valor máximo que se puede obtener con la suma de las monedas de una persona sin exceder la cota superior, debido a que, en ese caso, la diferencia entre la suma de las monedas de una persona y la otra sería la menor posible, es decir, la más cercana al caso ideal. Además, se nota que una vez se encuentra el valor máximo que se puede obtener con la suma de las monedas de una persona, la suma de las monedas de la otra persona es el valor total menos dicho valor máximo, razón por la cual se tiene información suficiente para calcular la diferencia entre las sumas de las monedas de ambas personas tras encontrar el máximo mencionado.

   Consecuentemente, la parte principal de la solución está en la función menor_diferencia_auxiliar, la cual utiliza para memoización un arreglo de tamaño cota_superior + 1 (el +1 para un manejo más fácil de índices en los bucles), donde cada posición "subtotal" del arreglo almacena el valor máximo que se puede obtener con la suma de una cierta cantidad de monedas sin exceder "subtotal". Entonces, para solucionar el problema, por cada moneda se recorre el arreglo de derecha a izquierda y se actualiza el valor de la posición "subtotal" haciendo una comparación entre si es más beneficioso tomar la moneda actual o no dada una capacidad máxima i. Esto corresponde a la línea arreglo[subtotal] = max(arreglo[subtotal], arreglo[subtotal-valor] + valor). Por ende, el valor máximo que se puede obtener con la suma de las monedas de una persona sin exceder la cota superior será el valor almacenado en la posición cota_superior del arreglo una vez se hayan recorrido todas las monedas. Nótese que el esquema de los bucles en la función comienza resolviendo el subproblema de menor tamaño y luego va resolviendo progresivamente subproblemas de mayor tamaño. Esto se observa porque la primera iteración resuelve el problema de determinar cuál es el máximo valor que se puede obtener con una moneda sin exceder el subtotal, y la iteración i-ésima resuelve el problema para determinar cuál es el máximo valor que se puede obtener con i monedas sin exceder el subtotal. Además, se nota que la resolución del subproblema i-ésimo utiliza los valores de los subproblemas menores al empezar a llenar el arreglo de derecha a izquierda y utilizando arreglo[subtotal] = max(arreglo[subtotal], arreglo[subtotal-valor] + valor), lo cual refleja la estrategia de memoización.

Finalmente, al tener ya el valor máximo que se puede obtener con la suma de las monedas de una persona sin exceder la cota superior (valor_total // 2), el programa procede a calcular la diferencia entre la suma de las monedas de una persona y la otra.

2. Su solución es BU o TP.
   A partir de la explicación de la pregunta 1, es evidente que la solución es bottom-up (BU) debido a que es una solución iterativa que utiliza memoización para resolver el problema, empezando desde los casos más pequeños de 1 moneda, 2 monedas, etc., hasta llegar al caso de n monedas, lo cual es característico de un algoritmo BU.
3. ¿Cuál es la complejidad espacial y temporal, con una breve explicación?
   Dado que la cantidad de monedas está acotada, y tiene que ser menor que 100, se concluye que la complejidad espacial, para un valor total "n" de las monedas, es $O(n)$ debido a que el arreglo de memoización tiene tamaño = $(n \,//\, 2) + 1$.
   Adicionalmente, se observa que la complejidad temporal del algoritmo, otra vez para un valor total "n" de las monedas, es $O(n)$ debido a que los bucles anidados en la función menor_diferencia_auxiliar recorren el arreglo de memoización una vez por cada moneda, pero como la cantidad de monedas está acotada, se puede considerar como constante, resultando en que la complejidad sea $O(\text{cantidad\_monedas} * (\text{n}//2)) = O(n)$.