

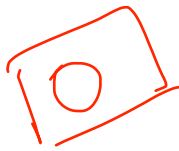
## Semana 3 - Ejercicios de grupo

Métodos algorítmicos en resolución de problemas II  
Facultad de Informática - UCM

	Nombres y apellidos de los componentes del grupo que participan	ID Juez
1	Jorge Arevalo Echevarria	MAR205
2	Miguel Verdaguer Velázquez	MAR285
3	Daniel Hernández Martínez	MAR246
4		

Instrucciones:

- Para editar este documento, es necesario hacer una copia de él. Para ello:
  - Alguien del grupo inicia sesión con la cuenta de correo de la UCM (si no la ha iniciado ya) y accede a este documento.
  - Mediante la opción *Archivo* → *Hacer una copia*, hace una copia del documento en su propia unidad de *Google Drive*.
  - Abre esta copia y, mediante el botón *Compartir* (esquina superior derecha), introduce los correos de los demás miembros del grupo para que puedan participar en la edición de la copia.
- La entrega se realiza a través del Campus Virtual. Para ello:
  - Alguien del grupo convierte este documento a PDF (dándole como nombre el número del grupo, 1.pdf, 2.pdf, etc...). Desde *Google Docs*, puede hacerse mediante la opción *Archivo* → *Descargar* → *Documento PDF*.
  - Esta misma persona sube el fichero PDF a la tarea correspondiente del *Campus Virtual*. Solo es necesario que uno de los componentes del grupo entregue el PDF.



## Mejor no llevar muchas monedas

El objetivo de hoy es resolver el **problema 11 Mejor no llevar muchas monedas**, del [juez automático](#). Ahí podéis encontrar el enunciado completo del problema.

Se trata de pagar de forma exacta una determinada cantidad con el menor número de monedas teniendo en cuenta que hay una cantidad limitada de cada tipo de monedas. Recordad que para ciertos sistemas monetarios y cuando la cantidad de monedas de cada tipo es ilimitada hay una estrategia voraz que resuelve el problema, pero no es el caso en un sistema monetario cualquiera o cuando el número de monedas de cada tipo es limitado.

**Solución:** (Plantead aquí la recurrencia que resuelve el problema y explicad las razones por las que es mejor resolver el problema utilizando programación dinámica que una implementación recursiva sin memoria.)

Ya que vamos a realizar muchos subproblemas iguales para ver las distintas soluciones, nos viene mucho mejor utilizar programación dinámica, ya que si tenemos guardada la solución del subproblema el coste se reducirá considerablemente, siendo el coste el acceso a la solución del subproblema.

Hacemos dos recorridos recursivos en una matriz de vectores (con el número de monedas de cada tipo).

En el primer recorrido empezamos poniendo  $\text{monedas}(N, \text{Coste})$  igual al vector inicial de número de monedas y vamos actualizando el resto de celdas recursivamente:

$\text{monedas}(i-1, j) = \text{monedas}(i, j)$

$\text{monedas}(i, j - v_i) = \text{monedas}(i, j)$  restando 1 al valor en  $[i]$

??

→ y esto?

Luego haríamos otra recursión empezando desde arriba (Llamada inicial:  $\text{monedas}(N, \text{Coste})$ ) para coger el vector mínimo con la cual acabaríamos teniendo en la posición  $\text{monedas}(N, \text{Coste})$  el vector mínimo de número de monedas. Si este es el vector de infinitos, entonces No hay solución, en otro caso si, y es la suma de los elementos del vector inicial de número de monedas menos la suma de elementos de ese vector:

si  $v_i > j$  o  $\text{monedas}(i-1, j)[i] = 0$ :

$\text{monedas}(i, j) = \text{monedas}(i-1, j)$

si  $v_i \leq j$  ó  $\text{monedas}(i-1, j)[i] > 0$ :

$\text{monedas}(i, j) = \text{monedas}(i, j - v_i)$  si  $\text{suma}(\text{monedas}(i-1, j - v_i)[i-1]) < \text{suma}(\text{monedas}(i-1, j))$

$\text{monedas}(i, j) = \text{monedas}(i-1, j)$  en otro caso

No entiendo


Casos Base

$\text{monedas}(0, j) = \text{Vector de Infinitos}$

$\text{monedas}(i, 0) = \text{su valor.}$

$\text{suma}()$ : la suma de los elementos del vector

**Solución:** (Escribid aquí las explicaciones necesarias para contar de manera comprensible la implementación del algoritmo de programación dinámica. En particular, explicad si es posible reducir el consumo de memoria y cómo hacerlo. Incluid el código y el coste justificado de las funciones que resuelven el problema. Extended el espacio al que haga falta.)



El coste en espacio es **cúbico**. Concretamente el número de monedas al cuadrado por el coste, dando lugar a una matriz de vectores.

El coste en tiempo depende de la función suma, así que sería cúbico, dos recorridos de la matriz por un recorrido de cada vector para sumarlos.

Resolved el problema completo del juez, que ahora debe ser ya muy sencillo.

**Número de envío:**