Examen de Algorítmica y Complejidad (Plan 2014)

22 de enero de 2021

Nº matrícula: _____ **Nombre:** _____

Apellidos: _____

Problema. Cuenta Herodoto, que el gran rey persa, Darío I, un día cazaba y se torció el tobillo. Aunque todos los médicos de la corte eran reputados egipcios, ninguno de ellos podía aliviar el dolor de su rey y sus consecuentes chillidos. Exhausto y harto de ellos, Darío recurrió a Democedes de Crotona, el más sabio médico griego, quien al poco tiempo de llegar, calma su dolor y cura su torcedura. Darío le colma de oro y de innumerables riquezas, pero, por mantener sus servicios le prohíbe regresar a su hogar como era su máximo deseo. Democedes, ante semejante situación, no tarda en urdir un ingenioso plan. Para llevarlo a cabo necesitará llevar, sin superar un cierto peso, la máxima cantidad de dinero posible para hacer uso de él en sobornos y alojamientos. Por cada tipo de moneda del sistema monetario persa, Democedes cuenta con **ilimitada** cantidad de monedas debido a la recompensa recibida. Le gustaría saber la cantidad de monedas de cada tipo que debe llevar para conseguir el máximo valor sin que se supere un peso máximo que él ha establecido.

Para ayudar a Democedes nos planteamos un algoritmo que resuelva el problema: dado los valores y pesos de cada tipo de moneda en un sistema monetario y un peso máximo en monedas, encontrar el número de monedas de cada tipo que podemos llevar sin que se supere el peso máximo dado.

Ejemplo: Consideremos que el peso máximo que podemos llevar es 26 y disponemos de los siguientes tipos de monedas con los pesos y valores que aparecen a continuación:

	0	1	2	3	4
Pesos	2	3	4	8	15
Valores	1	2	3	4	11

La solución sería recoger: **2** monedas de tipo 1 (con peso 3 y valor 2); y **5** moneda de tipo 2 (con peso 4 y valor 3):

	0	1	2	3	4
Sol:	0	2	5	0	0

Para resolver este problema se pide implementar un algoritmo en Java basado en **programación dinámica** con complejidad $O(N \cdot K)$ en tiempo donde N es el número de monedas y K el peso máximo que se puede llevar:

```
int[] mejorSeleccion(int[] pesos, int[] valores, int maxPeso)
```

donde:

- `int[] pesos` representa el peso de cada tipo de moneda. Todas las monedas pesan un valor mayor que 0.
- `int[] valores` representa el valor de cada tipo de moneda.
- `int maxPeso` representa el peso máximo en monedas que podemos llevar.
- El algoritmo debe devolver un vector de enteros indicando el número de monedas de cada tipo que se recogen (tal como aparece expresado en el ejemplo).

a) Define la **entrada**, la **salida** y la **semántica** de la función sobre la que estará basado el algoritmo de programación dinámica.

maxValor(i,p): Valor máximo que se puede conseguir utilizando las monedas 0…i sin que sobrepase el peso p.
Entrada: i>=0, p>=0. El valor i=0 indica que sólo se dispone de monedas de tipo 0. El valor p=0 indica que el peso máximo que se puede llevar es 0.
Salida: valor entero (valor máximo que se puede conseguir)

b) Expresa recursivamente la función anterior.

Si i=0 y pesos[i]>p maxValor(0, p) =0
Si i=0 y pesos[i]<=p maxValor(0, p) = valores[0]+ maxValor(0,p-pesos[0])

Si i>0 y pesos[i]>p
  maxValor(i, p) = maxValor(i-1,p)
Si i>0 y pesos[i] <=p
  maxValor(i, p) = max{ valores[i] + maxValor(i,p-pesos[i]), maxValor(i-1,p)}

c) Basándote en el anterior apartado implementa un algoritmo en Java basado en **programación dinámica** que tenga complejidad[1] en tiempo O(*NK*).

```
int[] mejorSeleccion (int[] pesos, int[] valores, int maxPeso) {
    int[][] maxValor = new int[2][maxPeso + 1];
    boolean[][] aux = new boolean[valores.length][maxPeso + 1];
    for (int p = 0; p <= maxPeso; p++) {
        aux[0][p]=(pesos[0]<=p);
        if (pesos[0] <= p)
            maxValor[0][p] = valores[0] + maxValor[0][p - pesos[0]];
        else
            maxValor[0][p]=0;
    }
    for (int i = 1; i < valores.length; i++) {
        for (int p = 0; p <= maxPeso; p++) {
            if (pesos[i] > p)
                maxValor[i % 2][p] = maxValor[(i - 1) % 2][p];
            else
                maxValor[i % 2][p] = Math.max(maxValor[(i - 1) % 2][p],
                        valores[i] + maxValor[i % 2][p - pesos[i]]);
            aux[i][p] = !(maxValor[i % 2][p] == maxValor[(i - 1) % 2][p]);
        }
    }
    return monedasSeleccionadas(aux, pesos,valores);
}

int[] monedasSeleccionadas (boolean[][] aux, int[] pesos, int[] valores){
    int i=valores.length-1; int p=aux[0].length-1;
    int[] decision = new int[valores.length];
    decision[i]=0;
    while (i>=0) {
        if (aux[i][p] == true) {
            p = p - pesos[i]; decision[i]++;
        } else {
            i--;
            if (i>=0) decision[i]=0;
        }
    }
    return decision;
}
```

[1] No cumplir con los requisitos pedidos conlleva **una puntuación de 0** en este apartado.