# Programación dinámica:

# El problema de los menús saludables.

Ángel Alberto Hamilton López
alu0100888102
alu0100888102@ull.edu.es

**Introducción.**

En esta práctica se nos pide crear un código capaz de resolver el problema de los menús saludables mediante programación dinámica con enfoque top-down y bottom-up y con un algoritmo puramente recursivo. Este problema no es más que una variante del problema de la mochila, en el que en lugar del valor de los objetos lo que intentamos maximizar es el valor nutricional y el factor limitante son los nutrientes poco deseables (como grasas o colesterol). Nuestro objetivo es buscar la combinación de los platos predeterminados que consigan el mayor valor nutricional posible sin exceder el valor dado de nutrientes nocivos.

**Estructura óptima de sub-problemas.**

Para afrontar este problema es bastante útil la recursividad. Para ver si tenemos que incluir un plato en el menú, tenemos que ver qué es mayor: el valor nutricional del menú sin este plato y el valor nutricional del menú con este plato, teniendo en cuenta que no puede excederse el valor de sustancias nocivas. Para la programación dinámica creamos una tabla n*m donde n es el numero de platos y m el valor del umbral. Al calcular el valor de cada plato lo colocamos en la tabla en las cordenadas (i,w) donde i es el identificador del plato y w es el valor de sustancias nocivas total incluidas hasta el momento.

Para calcular el valor del menú sin este plato, cogemos el plato que está antes que este en la lista (esto es por seguir un orden, no importa cómo estén ordenados los platos) y consultamos en la tabla el valor que tiene este nodo. Si resulta que el plato que estamos analizando es primero de la lista, no hay un plato anterior y el resultado de este paso será 0.

Para calcular el valor del menú con este plato, cogemos el valor del paso anterior y le sumamos el valor nutricional de este plato. Si la cantidad de sustancias nocivas de los platos anteriores más este supera el umbral propuesto, este paso devuelve menos infinito.

Por ultimo, calculamos el valor máximo entre los dos anteriores y lo escribimos en la tabla en las coordenadas (i,w) para este plato.

**Pseudocódigos de los algoritmos implementados**

**Algoritmo recursivo:**

```
recursiveSolution( platos[0,…,n], index, umbral)
        if (index == 0) && (umbral >= 0)
                return 0
        if (umbral < 0)
                return -inf
        v1 = recursiveSolution (platos, index - 1, umbral)
        v2 = recursiveSolution ( platos, index - 1, umbral – platos[index].nocivas) +
                platos[index].nutricion
        return max(v1, v2)
```

**Algoritmo Top-Down:**

```
topdownSolution( matriz[0...n,0...umbral], platos[0,…,n],  index, umbral)
        if (index == 0) && (umbral >= 0)
                return 0
        if (umbral < 0)
                return -inf
        if (matriz[index, umbral] != -1)
                return matriz[index, umbral]
        v1 = topdownSolution (matriz, platos, index - 1, umbral)
        v2 = topdownSolution (matriz platos, index - 1, umbral – platos[index].nocivas)
                + platos[index].nutricion
        matriz[index, umbral] = max(v1, v2)
        return max(v1, v2)
```

**Algoritmo Bottom-Up:**

```
recursiveSolution( matriz[0...n,0...umbral], platos[0,…,n],  n, umbral)
        for j = 0 to umbral
                matriz[0, j] = 0
        for i =  1 to n
                for j = 0 to umbral
                        if(platos[i].nocivo <= j)
                                v1 = recursiveSolution (platos, index - 1, umbral)
                                v2 = KNAP_RECURSIVE ( platos, index - 1, umbral –
                                        platos[index].nocivas) + platos[index].nutricion
                                matriz[index, umbral] = max(v1, v2)
                        else
                                matriz[index, umbral] = matriz[index – 1,  j]
```

**Análisis de complejidad de los algorimos.**

En el caso del algoritmo recursivo, para cada uno de los platos, comprueba como sería mejor, si tenerlo o no tenerlo. Si tenemos n platos y 2 estados posibles (tenerlo o no tenerlo). Como este algoritmo comprueba todas las combinaciones acabamos con un algoritmo de complejidad 2^n.

Para ambos algoritmos de programación dinámica el problema se simplifica. Creamos una matriz de tamaño n*m donde n son los platos y m el umbral. Solamente tenemos que resolver cada una de las casillas de la matriz una vez y luego podemos simplemente consultarla,por lo que acabamos con un problema de complejidad n*m.

**Ejemplos de soluciones.**

Pongamos el problema de ejemplo que viene en el enunciado de la página:

5
110
Ensalada 10 10
Arvejas 60 20
Carne cabra 180 50
Rapadura 220 60
Escaldón 280 70

Para este ejemplo,todos los algoritmos llegan a la misma conclusión: Cojemos la Carne cabra y las Arvejas para un total de 400 de nutrición y empleamos la totalidad de nuestros 110 nutrientes nocivos.

5
110
Ensalada 10 10
Arvejas 60 20
Helado 160 30
Carne Cabra 180 50
Rapadura 220 60
Escaldón 280 70

Sin embargo, para este otro ejemplo eligen Rapadura, Helado y Arvejas, para un total de 440.