Problema de la mochila con múltiples ítems

Definición del problema[1]

El problema de la mochila, comúnmente abreviado por KP (del inglés *Knapsack problem*) es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema. Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de n ítems, cada uno con un peso y valor específicos. Cada uno de estos ítems, estará disponible un número q_i de veces. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo

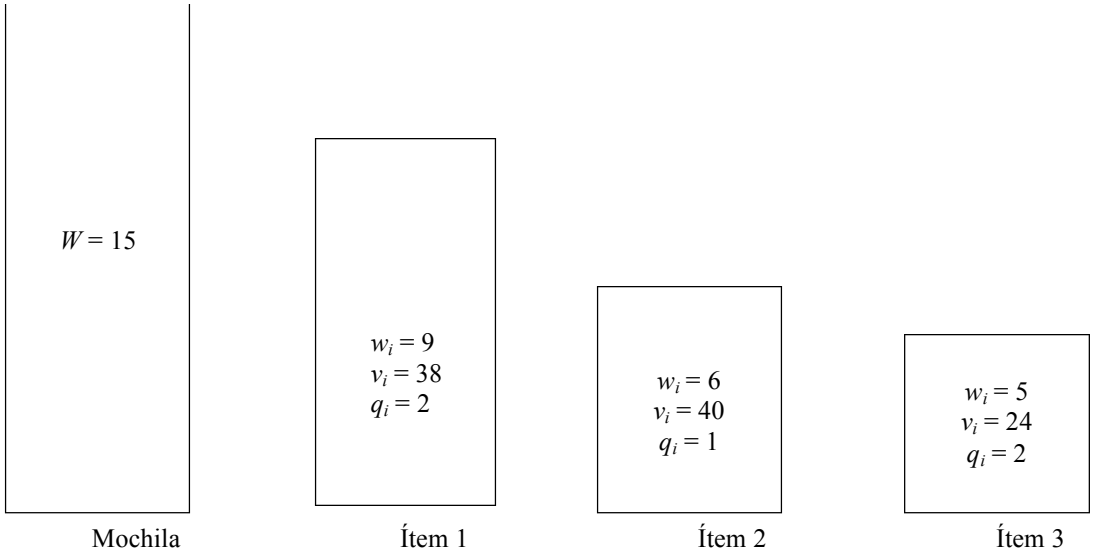

Figura 1. Ejemplo del problema de la mochila con múltiples tipos de objeto

Supongamos que tenemos n distintos tipos de ítems, que van del 1 al n. De cada tipo de ítem se tienen q_i ítems disponibles, donde q_i es un entero positivo que cumple $1 \leqslant q_i < \infty$.

Cada tipo de ítem i tiene un beneficio asociado dado por v_i y un peso (o volumen) w_i. Usualmente se asume que el beneficio y el peso no son negativos.

Por otro lado se tiene una mochila, donde se pueden introducir los ítems, que soporta un peso máximo (o volumen máximo) $W > 0$.

El problema consiste en meter en la mochila ítems de tal forma que se maximice el valor de los ítems que contiene y siempre que no se supere el peso (o volumen) máximo que puede soportar la misma. La solución al problema vendrá dado por la secuencia de variables $x_1, x_2, ..., x_n$ donde el valor entero de x_i indica cuantas copias se meterán en la mochila del tipo de ítem i.

Para el ejemplo de la figura 1, $W=15$, si suponemos que hay dos unidades del ítem 1, una sola unidad del ítem 2 y dos unidades del ítem 3, es decir $(q_1, q_2, q_3) = (2,1,2)$; con un vector de pesos $(w_1, w_2, w_3) = (9,6,5)$; y un vector de valores asociados de $(v_1, v_2, v_3) = (38,40,24)$, la solución óptima consiste en meter en la mochila una unidad del ítem 1 y otra unidad del ítem 2; es decir el resultado puede expresarse como $(x_1, x_2, x_3) = (1,1,0)$. Ver figura 2.

[1] Esta práctica esta adaptada del libro "Levitin , A. (2012): Introduction to The design and Analysis of Algorithms. Pearson Education Inc, USA"

Sin embargo, si modificamos algunos de estos valores la solución cambia. Por ejemplo, si hay dos ítems disponibles de cada tipo: (q_1, q_2, q_3) = (2,2,2), manteniendo los demás parámetros, la solución sería (x_1, x_2, x_3) = (0,2,0).

Si además de haber dos ítems de cada tipo, el tamaño de la mochila crece *W=16*, el resultado ahora sería ahora (x_1, x_2, x_3) = (0,1,2).

Solución	***Peso total***	***Valor total***
(0,0,0)	0	0
(1,0,0)	9	38
(2,0,0)	18(NO_VÁLIDO)	76
(0,1,0)	6	40
(1,1,0)	15	78
(2,1,0)	24(NO_VÁLIDO)	116
(0,0,1)	5	24
(1,0,1)	14	62
(2,0,1)	23(NO_VÁLIDO)	100
(0,1,1)	11	64
(1,1,1)	20(NO_VÁLIDO)	102
(2,1,1)	29(NO_VÁLIDO)	140
(0,0,2)	10	48
(1,0,2)	19(NO_VÁLIDO)	86
(2,0,2)	28(NO_VÁLIDO)	124
(0,1,2)	16(NO_VÁLIDO)	88
(1,1,2)	25(NO_VÁLIDO)	126
(2,1,2)	34(NO_VÁLIDO)	164

Figura 2. Búsqueda exhaustiva para W=15 y (q_1, q_2, q_3) = (2,1,2)

Solución por "fuerza bruta".

Para resolver este problema se puede diseñar una estrategia de "fuerza bruta", simplemente tomado todos las posibles combinaciones, comprobar que cumplen la limitación de peso total y de entre ellas elegir la de mayor valor.

Solución mediante "Programación dinámica".

Para plantear una solución por programación dinámica es necesario obtener una relación de recurrencia que exprese la solución a una instancia en función de las soluciones a instancias menores. Supongamos una instancia definida por los *i* primeros ítems $1 \leqslant i \leqslant$ n., con pesos (w_1, w_2, ... w_i), valores (v_1, v_2, ...v_i) y número de repeticiones (q_1, q_2, ... q_i), para una capacidad de la mochila j $\leqslant$ W. Sea *F(i,j)* una solución óptima para este problema, es decir, un subconjunto de los *i* primeros ítems cuya suma de pesos total es menor o igual que *j*. Se consideran dos tipos de subconjuntos del conjunto de los *i* primeros ítems: Los que contienen al ítem *i*, y los que no.

- Para los subconjuntos que no incluyen al ítem i-esimo, el valor de la solución optima es exactamente mismo que F(i-1, j).
- Para los que si incluyen al ítem i-esimo, (y que por tanto verifican que j-w_i >= 0), la solución óptima se puede obtener comprobando cuantas unidades de dicho ítem se pueden añadir a la solución óptima del subconjunto de los primeros (i-1*)* ítems para una mochila de capacidad j - *(*w_i k_i*)* , donde la variable k_i es un valor entre 0 y q_i , cuyo valor sería k_i v_i + F(i-*1*,j-*(*$k_i w_i$*)*))

La solución para el problema limitado a los i primeros ítems será la que de un mayor valor entre ambos casos. Ni que decir tiene que si el ítem i-ésimo hiciera que el peso de la mochila sobrepasara el límite, la solución para los i primeros ítems seria la misma que para los (*i-1*) primeros ítems. Con todo esto obtenemos la siguiente función de recurrencia:

$$F(i,j) = \begin{cases} 0, & si\ i = 0\ ó\ j = 0 \\ F(i-1,j), & si\ j < wi \\ \max\left(F(i-1,j), F\left(i-1, j-(ki*wi)\right)\right) + ki*vi, & si\ j \geq ki*wi\ \text{^}\ 0 < ki \leq qi \end{cases}$$

Una vez determinada la matriz, a partir de la información que contiene pueden identificarse los ítems que deben seleccionarse. Empezando por el último valor (n,W), y razonando de la siguiente manera: Un punto cualquiera de la matriz (i,j) es una solución al problema con los i primeros ítems y suponiendo una capacidad j. Por tanto si F(i-1,j) es menor que F(i,j), el ítem i está en la solución, y si no, no lo está. Si no lo está, se repite el mismo razonamiento para (i-1,j). Si lo está, es decir si F(i-1,j) < F(i,j) entonces descontando el peso del ítem i de la capacidad total, tendríamos una solución para un problema de menor tamaño, es decir, se vuelve a razonar a partir del ítem (i,j-w_i). Así hasta llegar a primer ítem.

Solución mediante "Algoritmos voraces".

Existen múltiples soluciones mediante algoritmos voraces a este problema. Una primera aproximación seria ordenar los ítems en orden creciente de pesos e ir metiéndolos en la mochila empezando por el menor, luego el siguiente menor, y si no probar con el tercero, etc. Otra solución seria ordenar los ítem por su valor de forma decreciente, e ir metiendo los mas valiosos primero, mientras quepan. Sin embargo una estrategia aun mejor consiste en evaluar su "densidad", es decir el valor por unidad de peso, ordenar la lista e ir metiéndolos en la mochila por este orden, es decir:

- Hallar la "densidad" $r_i = v_i / w_i$ (relación entre el valor y el peso) para todos los ítems.
- Ordenar la lista $r_1, r_2, \ldots r_n$ de forma decreciente.
- Repetir la siguiente operación para todos los ítems siguiendo el orden de la lista: Seleccionar un ítem, si cabe, meterlo en la lista, si no, pasar al siguiente.

Se pide:

Implementar varias soluciones al problema de la mochila:

```
MochilaFB.java
MochilaPD.java
MochilaAV.java
```

herederas de la clase denominada `Mochila.java` (ver ficheros auxiliares de la práctica).

```
public abstract class Mochila {
        public abstract SolucionMochila resolver(ProblemaMochila pm);
        ...
}
```

redefiniendo el método `resolver`. Este método tiene como entrada y salida sendos objetos de las clases auxiliares `ProblemaMochila` y `SolucionMochila`, que encapsulan los datos necesarios del problema y su solución respectivamente, y que proporcionan ya implementados los métodos de lectura y escritura.

Por ejemplo, la ejecución del programa correspondiente al ejemplo de la figura 1 dará la siguiente salida:

```
> java MochilaFB
```

Se acompaña un fichero de ejemplos ya resueltos que pueden probarse y que una vez que se hayan implementado las tres clases correctamente dará los siguientes resultados:

```
> java ProbarEjemplos
El metodo de fuerza bruta supera las pruebas de los ejemplos
El metodo de Programación Dinámica supera las pruebas de los ejemplos
El metodo de Algoritmos Voraces supera las pruebas de los ejemplos
```

Evaluación de la práctica

* No utilice paquetes de Java, coloque todas las clases en un mismo directorio.

* Para realizar este ejercicio NO ES NECESARIO UTILIZAR ECLIPSE, aunque se puede hacer si se quiere y se sabe como. Para compilar el código que se entrega solo es necesario poner los ficheros en una misma carpeta, abrir una terminal y utilizar las instrucciones:

```
>javac *.java
```

* En caso de utilizar Eclipse, el fichero `ejemplos.txt` debe colocarse en la carpeta raíz del proyecto.

* Para evaluar la practica se deben enviar solamente los tres ficheros que contienen las clases que implementan el método `resolver`.

* No basta con implementar los tres métodos iguales. Se comprobará que la complejidad de cada uno de las implementaciones corresponde a la complejidad del método y que las soluciones obtenidas son las esperadas en cada caso. Si la complejidad no corresponde a la esperada, no se dará por válida la implementación.

* Se valorara la eficiencia de la implementación en comparación con la implementación realizada por el profesor, y en caso de que el tiempo de ejecución sea muy superior al esperado no se alcanzará la máxima puntuación.