PRÁCTICA DE LA MOCHILA MÚLTIPLE

J. Javier Quirante Pérez

Análisis y Diseño de Algoritmos

ÍNDICE

1. Solución por fuerza bruta **3**

2. Solución mediante programación dinámica **4**

3. Solución mediante algoritmos voraces **5**

1. Solución por fuerza bruta

Para llevar a cabo este método de resolución mediante fuerza bruta se han tomado todas las combinaciones posibles que cumplan el límite de peso y cuyo valor sea máximo.

En primer lugar, se crean 3 arrays los cuales van a almacenar los pesos, valores y las unidades. Además, se ha creado una variable "permutaciones", ya que en el bucle "for" se van a calcular el número de permutaciones que se van a realizar.

```
public SolucionMochila resolver(ProblemaMochila pm) {
    SolucionMochila sm=null;
    int[] peso        = pm.getPesos();
    int[] valor       = pm.getValores();
    int[] items       = pm.getUnidades();
    int permutaciones = 1;

    for(int i=0; i<pm.size();i++){
        permutaciones *= pm.getUnidad(i)+1;
    }
```

A continuación, según el número de permutaciones que hemos calculado anteriormente, mediante dos bucles "for", se prueban todas las combinaciones para buscar la combinación con el valor máximo.

Tenemos una variable actual que se usará para evitar modificar la i, junto con el peso y el valor inicializados a 0.

El segundo bucle "for" se va a encargar de crear la combinación con la que vamos a trabajar. En esas 4 líneas se utiliza la aritmética modular para obtener el valor actual junto con el peso y valores de tal forma que, una vez finalizamos el bucle, comprobamos en una sentencia "if" que la combinación sea la mayor sin superar el peso de la mochila.

```
for(int i=0; i<permutaciones;i++){
    int actual = i, pActual = 0, vActual = 0;
    int[] a = new int[items.length];

    for(int j=0; j<items.length;j++){
        a[j] = actual % (items[j] + 1);
        pActual += peso[j] * a[j];
        vActual += valor[j] * a[j];
        actual /= items[j]+1;

    }

    if(pm.getPesoMaximo() >= pActual){
        if(sm == null || sm.getSumaValores() < vActual){
            sm = new SolucionMochila(a,pActual,vActual);
        }
    }
}
```

2. Solución mediante programación dinámica

Para llevar a cabo este método de resolución mediante programación dinámica se ha tenido en cuenta la ecuación de Bellman que se proporciona en el enunciado de la práctica.

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

En primer lugar se crea una matriz, a la cual le he llamado F, con las dimensiones correspondientes.

```
public SolucionMochila resolver(ProblemaMochila pm) {
    SolucionMochila sm=null;
    SolucionMochila[][] F = new SolucionMochila[pm.getItems().size()+1][pm.getPesoMaximo()+1];
```

Ahora, con los bucles for iremos rellenando la tabla con las condiciones establecidas en la ecuación. En primer lugar, el caso base, cuando i=0 o j=0 la posición en la tabla tomará el valor 0.

```
for(int i=0;i<=pm.size();i++){
    for(int j=0;j<=pm.getPesoMaximo();j++){

        if(i==0 || j==0){
            F[i][j] = new SolucionMochila(new int[pm.getItems().size()], sumaPesos: 0, sumaValores: 0);
```

Después contemplamos el segundo caso de la ecuación, donde devolvemos la suma de los pesos y los valores de la posición i-1, j

```
}else if(j < pm.getPeso( i: i-1)){
    SolucionMochila s = F[i-1][j];
    F[i][j] = new SolucionMochila(
            (ArrayList<Integer>) s.getSolucion().clone(),
            s.getSumaPesos(),
            s.getSumaValores());
```

En otro caso, se elige el elemento que cumpla la última condición de la ecuación, que sería el máximo entre F[i-1][j] y, con un k iremos iterando entre los valores F[i-1][j-k*pi]. Esto se hace con un bucle for y dos sentencias if que controlan que se cumplan los requisitos y actualizar el valor máximo.

```
}else{
    SolucionMochila s = F[i-1][j];
    int max = 0;

    for(int k=1;k<=pm.getUnidad( i: i-1);k++){

        if(j >= k * pm.getPeso( i: i-1)){
            SolucionMochila aux = F[i-1][j-k*pm.getPeso( i: i-1)];

            if(s.getSumaValores() < aux.getSumaValores() + k*pm.getValor( i: i-1)){
                s = aux;
                max = k;
            }
        }
    }
```

Finalmente guardamos en un ArrayList la solución al problema de la mochila.

```
                ArrayList<Integer> sol = (ArrayList<Integer>) s.getSolucion().clone();
                sol.set(i-1,max);
                F[i][j] = new SolucionMochila(
                        sol,
                        sumaPesos: s.getSumaPesos() + max * pm.getPeso( i: i-1),
                        sumaValores: s.getSumaValores() + max * pm.getValor( i: i-1));

            }
        }
    }

    sm = F[pm.getItems().size()][pm.getPesoMaximo()];
    return sm;
}
```

3. Solución mediante algoritmos voraces

La estrategia que hemos elegido para la resolución del problema es evaluar la "densidad" de los elementos para después ordenar la lista e introducir en la mochila los elementos una vez ordenada.
Para ello implementamos el método para comparar la densidad de los ítem o1 y o2, que devolverá -1, 1 ó 0 en función de si es mayor, menor o igual un objeto u otro.

```
pm.items.sort(new Comparator<Item>() {
    new *
    @Override
    public int compare(Item o1, Item o2) {
        double densidad1 = ((double)o1.valor) / ((double)o1.peso);
        double densidad2 = ((double)o2.valor) / ((double)o2.peso);

        if(densidad1 > densidad2){
            return -1;

        }else if(densidad1 < densidad2){
            return 1;

        }else{
            return 0;
        }
    }
});
```

Ahora, con iterator vamos a recorrer los items y los vamos a meter en “sol” mientras que cumpla la condición, que está controlado por el segundo bucle “while”.

```
Iterator<Item> it = pm.items.iterator();
int p = 0, v = 0;
int[] sol = new int[pm.items.size()];

while(it.hasNext()){
    Item item = it.next();

    while((sol[item.index] < item.unidades) && (pm.getPesoMaximo() - p - item.peso >= 0)){
        sol[item.index]++;
        p += item.peso;
        v+= item.valor;
    }
}

return (new SolucionMochila(sol,p,v));
```