Programación y Estructuras de Datos Avanzadas
Segunda práctica

Carlos Caride Santeiro

15/12/2017

1. Datos personales

- Nombre y código de la asignatura: **Programación y Estructuras de Datos Avanzadas (71902019)**
- Título de la práctica: **Problema de la mochila con objetos no fraccionables**
- Nombre y Apellidos: **Carlos Caride Santeiro**
- DNI: **44446239G**
- Centro asociado: **Ourense**

2. ENUNCIADO DE LA PRÁCTICA: Problema de la mochila con objetos no fraccionables

Se tiene una mochila con una capacidad máxima V y n objetos con volúmenes $v = (v_1, v_2, \ldots, v_n)$ y beneficios $b = (b_1, b_2, \ldots, b_n)$. Los valores de los volúmenes son enteros. El objetivo de este problema es encontrar una selección de objetos cuya suma de volúmenes no supere la capacidad máxima de la mochila, y de forma que la suma de beneficios sea máxima. Por lo tanto es un problema de optimización con restricciones.

En esta versión del problema los objetos son indivisibles, con lo que sólo tenemos la opción de incluirlos o excluirlos. Este hecho hace que no se pueda aplicar un algoritmo voraz. Se pide que se desarrolle el programa que resuelva este problema mediante el **esquema de programación dinámica**, tal y como está planteado en el apartado 5.5 del texto base.

3. Descripción del esquema algorítmico utilizado y como se aplica al problema

Se tiene una mochila con una capacidad máxima de V y n objetos con volúmenes $v = (v_1, v_2, \ldots, v_n)$ y beneficios $b = (b_1, b_2, \ldots, b_n)$. Los valores de los volúmenes son enteros. El objetivo de este problema es encontrar una selección de objetos cuya suma de volúmenes no supere la capacidad máxima de la mochila, y de forma que la suma de beneficios sea máxima. Este es un problema de optimización con restricciones que se puede plantear de la siguiente forma:

$$maximizar \sum_{i=0}^{n} x_i b_i \text{ cumpliendo } \sum_{i=0}^{n} x_i v_i \leq V$$

donde x_i toma el valor 0 ó 1, 0 para indicar que el objeto i no se incluye en la mochila y 1 para indicar lo contrario.

Para la resolución de este problema se utiliza el esquema de programación dinámica.

3.1. Ecuaciones recursivas del problema

Se formula el problema de forma incremental, planteando las ecuaciones recursivas para una función $mochila(i, W)$ que da el máximo beneficio para un volumen W que queda libre en la mochila considerando los objetos entre 1 e i, siendo $i \leq n$. Cuando se pasa a considerar el objeto i se tienen dos posibilidades: que el objeto exceda de la capacidad de la mochila o quepa en ella. El primer caso se prueba con el resto de objetos. En el segundo caso se tienen también dos opciones: o bien se incluye, con lo que el beneficio b_i se añade al valor de la función, y el volumen v_i se resta del espacio libre, o bien no se incluye, con lo que se tiene que resolver el problema considerando la serie de objetos entre 1 y $i-1$. Se queda con el que maximice el beneficio total. Los casos base del problema se presentan cuando la capacidad de la mochila llega a 0, o cuando no queda ningún objeto. En estos casos, el beneficio es 0. También se pueden considerar casos para configuraciones no válidas a las que se puede llegar si se excede la capacidad de la mochila. Este caso se designa con el valor especial "$-\infty$", que es superado por el beneficio de cualquier configuración válida. Por tanto, las ecuaciones de concurrencia del problema son:

$$mochila(i,W) = \begin{cases} 0 & \text{si } i = 0 \text{ y } W \geq 0 \\ -\infty & \text{si } W < 0 \\ mochila(i-1,W) & \text{si } i > 0 \text{ y } v_i > W \\ max\{mochila(i-1,W), b_i + mochila(i-1,W-v_i)\} & \text{si } i > 0 \text{ y } v_i \leq W \end{cases}$$

3.2. Aplicación del esquema algorítmico

Para resolver el problema se construye una tabla `M[n,V]` con tantas filas como objetos y tantas columnas como indique el volumen V. Para calcular la posición `M[i,j]` se necesita haber calculado dos de las posiciones de la linea anterior. Por tanto se construye la tabla por filar y el valor `M[n,V]` de la última fila da la solución al problema.

El algoritmo que resuelve el problema se puede escribir de la siguiente forma:

```
tipo Tabla = matriz[0..n,0..V] de entero
tipo Vector = matriz[0..n] de entero
fun MochilaEntera(vol:Vector, ben:Vector, n:entero, V:entero, M:Tabla)
    var
        i,j: entero
    fvar
    para i ← 1 hasta n hacer
        M[i,0] ← 0
    fpara
    para j ← 1 hasta V hacer
        M[0,j] ← 0
    fpara
    para i ← 1 hasta n hacer
        para j ← 1 hasta V hacer
            si vol[i] > j entonces
                M[i,j] ← M[i-1,j]
            sino
                M[i,j] ← max(M[i-1,j], M[i-1, j-vol[i]] + ben [i]})
```

```
            fsi
        fpara
    fpara
ffun
```

Para que el algoritmo indique qué objetos hay que seleccionar para obtener el beneficio máximo, se puede llamar a una función que tenga como datos de entrada la tabla construida y los volúmenes de los objetos, y de como salida un vector objetos de ceros y unos, en el que un uno significa que hay que incluir el objeto para obtener el beneficio máximo. Esta función, empezando por la ultima casilla de la tabla, va comprobando a cada paso si el valor de la casilla coincide con el de la casilla de la fila superior, lo que indica que no se ha incluido el objeto i. Si no coinciden se sabe que el objeto se ha incluido y se pasa a comprobar la casilla correspondiente a la reducción de volumen que ha supuesto incluir el objeto.

```
fun ObjetosMochila(vol:Vector, M:Tabla, n:entero, V:entero, objetos:Vector)
    var
        i,W: entero
    fvar
    W ← V
    para i ← n hasta 1 incremento −1 hacer
        si M[i,W] = M[i−,W] entonces
            objetos[i] ← 0
        sino
            objetos[i] ← 1
            W ← W − vol[i]
        fsi
    fpara
ffun
```

4. Coste computacional del algoritmo utilizado

El coste computacional de este algoritmo está en $O(nV)$, por los dos bucles anidados para la construcción de la tabla. Asimismo, el método de obtención de los objetos contenidos en la mochila tiene un coste de $O(n)$ por el bucle de recorrer todos los los objetos.

5. Alternativas al esquema utilizado

La utilización de un algoritmo voraz consiste en introducir en la mochila según orden decreciente de utilidad (beneficio) los diversos objetos. En una primera etapa, se adicionarán unidades enteras hasta que, por motivo de capacidad, no sea posible seguir introduciendo enteros y haya que añadir la porción que quepa del siguiente objeto. Al ser objetos que no se pueden dividir en porciones, el uso de un algoritmo voraz no es aplicable a este problema.

6. Datos de prueba

Para la comprobación del algoritmo se realizan 5 pruebas.

6.1. Prueba 1

6.1.1. Mochila

Tamaño de la mochila: 8

Objetos:

	Volumen	Beneficio
1	1	2
2	3	5
3	4	10
4	5	14
5	7	15

6.1.2. Resultado obtenido

```
Limite volumen       |   0   1   2   3   4   5   6   7   8
---------------------|-----------------------------------
posicion 0           |   0   0   0   0   0   0   0   0   0
v1 =   1  b1 =   2   |   0   2   2   2   2   2   2   2   2
v2 =   3  b2 =   5   |   0   2   2   5   7   7   7   7   7
v3 =   4  b3 =  10   |   0   2   2   5  10  12  12  15  17
v4 =   5  b4 =  14   |   0   2   2   5  10  14  16  16  19
v5 =   7  b5 =  15   |   0   2   2   5  10  14  16  16  19

Objetos de volumen: 5, 3
Beneficio: 19
```

6.2. Prueba 2

6.2.1. Mochila

Tamaño de la mochila: 8

Objetos:

	Volumen	Beneficio
1	1	2
2	5	14
3	7	15
4	4	10
5	3	5

6.2.2. Resultado obtenido

```
Limite volumen       |   0   1   2   3   4   5   6   7   8
---------------------|-----------------------------------
posicion 0           |   0   0   0   0   0   0   0   0   0
v1 =   1  b1 =   2   |   0   2   2   2   2   2   2   2   2
v2 =   5  b2 =  14   |   0   2   2   2   2  14  16  16  16
```

```
v3 =   7  b3 =  15   |   0   2   2   2   2  14  16  16  17
v4 =   4  b4 =  10   |   0   2   2   2  10  14  16  16  17
v5 =   3  b5 =   5   |   0   2   2   5  10  14  16  16  19

Objetos de volumen: 3, 5
Beneficio: 19
```

6.3. Prueba 3

6.3.1. Mochila

Tamaño de la mochila: 10

Objetos:

	Volumen	Beneficio
1	4	42
2	8	91
3	1	18
4	8	96
5	6	87
6	10	38
7	6	38
8	2	45
9	10	18
10	10	22

6.3.2. Resultado obtenido

```
Limite volumen       |   0   1   2   3   4   5   6   7   8   9  10
---------------------|-----------------------------------------------
posicion 0           |   0   0   0   0   0   0   0   0   0   0   0
v1 =   4  b1 =  42   |   0   0   0   0  42  42  42  42  42  42  42
v2 =   8  b2 =  91   |   0   0   0   0  42  42  42  42  91  91  91
v3 =   1  b3 =  18   |   0  18  18  18  42  60  60  60  91 109 109
v4 =   8  b4 =  96   |   0  18  18  18  42  60  60  60  96 114 114
v5 =   6  b5 =  87   |   0  18  18  18  42  60  87 105 105 114 129
v6 =  10  b6 =  38   |   0  18  18  18  42  60  87 105 105 114 129
v7 =   6  b7 =  38   |   0  18  18  18  42  60  87 105 105 114 129
v8 =   2  b8 =  45   |   0  18  45  63  63  63  87 105 132 150 150
v9 =  10  b9 =  18   |   0  18  45  63  63  63  87 105 132 150 150
v10 =  10  b10 =  22 |   0  18  45  63  63  63  87 105 132 150 150

Objetos de volumen: 2, 6, 1
Beneficio: 150
```

6.4. Prueba 4

6.4.1. Mochila

Tamaño de la mochila: 20

Objetos:

	Volumen	Beneficio
1	10	19
2	9	42
3	3	35
4	3	85
5	10	9
6	3	54
7	8	13
8	8	73
9	8	14
10	6	8

6.4.2. Resultado obtenido

```
Limite volumen       |   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
---------------------|-----------------------------------------------------------------------------------
posicion 0           |   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
v1 =  10  b1 =  19   |   0   0   0   0   0   0   0   0   0   0  19  19  19  19  19  19  19  19  19  19  19
v2 =   9  b2 =  42   |   0   0   0   0   0   0   0   0   0  42  42  42  42  42  42  42  42  42  42  61  61
v3 =   3  b3 =  35   |   0   0   0  35  35  35  35  35  35  42  42  42  77  77  77  77  77  77  77  77  77
v4 =   3  b4 =  85   |   0   0   0  85  85  85 120 120 120 120 120 120 127 127 127 162 162 162 162 162 162
v5 =  10  b5 =   9   |   0   0   0  85  85  85 120 120 120 120 120 120 127 127 127 162 162 162 162 162 162
v6 =   3  b6 =  54   |   0   0   0  85  85  85 139 139 139 174 174 174 174 174 174 181 181 181 216 216 216
v7 =   8  b7 =  13   |   0   0   0  85  85  85 139 139 139 174 174 174 174 174 174 181 181 187 216 216 216
v8 =   8  b8 =  73   |   0   0   0  85  85  85 139 139 139 174 174 174 174 174 212 212 212 247 247 247 247
v9 =   8  b9 =  14   |   0   0   0  85  85  85 139 139 139 174 174 174 174 174 212 212 212 247 247 247 247
v10 =   6  b10 =   8 |   0   0   0  85  85  85 139 139 139 174 174 174 174 174 212 212 212 247 247 247 247

Objetos de volumen: 8, 3, 3, 3
Beneficio: 247
```

6.5. Prueba 5

6.5.1. Mochila

Tamaño de la mochila: 20

Objetos:

	Volumen	Beneficio
1	4	62
2	7	66
3	4	53
4	8	57
5	3	39
6	10	88
7	8	86
8	2	21
9	4	76
10	3	55
11	5	27
12	9	90
13	4	8
14	4	1
15	7	92
16	5	81
17	8	78
18	10	46
19	6	83
20	10	4

6.5.2. Resultado obtenido

Limite volumen	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
posicion 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v1 = 4 b1 = 62	0	0	0	0	62	62	62	62	62	62	62	62	62	62	62	62	62	62	62	62	62
v2 = 7 b2 = 66	0	0	0	0	62	62	62	66	66	66	66	128	128	128	128	128	128	128	128	128	128
v3 = 4 b3 = 53	0	0	0	0	62	62	62	66	115	115	115	128	128	128	128	181	181	181	181	181	181
v4 = 8 b4 = 57	0	0	0	0	62	62	62	66	115	115	115	128	128	128	128	181	181	181	181	185	185
v5 = 3 b5 = 39	0	0	0	39	62	62	62	101	115	115	115	154	154	154	167	181	181	181	220	220	220
v6 = 10 b6 = 88	0	0	0	39	62	62	62	101	115	115	115	154	154	154	167	181	181	189	220	220	220
v7 = 8 b7 = 86	0	0	0	39	62	62	62	101	115	115	115	154	154	154	167	187	201	201	220	240	240
v8 = 2 b8 = 21	0	0	21	39	62	62	83	101	115	122	136	154	154	175	175	187	201	208	222	240	241
v9 = 4 b9 = 76	0	0	21	39	76	76	97	115	138	138	159	177	191	198	212	230	230	251	251	263	277
v10 = 3 b10 = 55	0	0	21	55	76	76	97	131	138	152	170	193	193	214	232	246	253	267	285	285	306
v11 = 5 b11 = 27	0	0	21	55	76	76	97	131	138	152	170	193	193	214	232	246	253	267	285	285	306
v12 = 9 b12 = 90	0	0	21	55	76	76	97	131	138	152	170	193	193	214	232	246	253	267	285	285	306
v13 = 4 b13 = 8	0	0	21	55	76	76	97	131	138	152	170	193	193	214	232	246	253	267	285	285	306
v14 = 4 b14 = 1	0	0	21	55	76	76	97	131	138	152	170	193	193	214	232	246	253	267	285	285	306
v15 = 7 b15 = 92	0	0	21	55	76	76	97	131	138	152	170	193	193	214	232	246	253	267	285	285	306
v16 = 5 b16 = 81	0	0	21	55	76	81	97	131	138	157	170	193	212	219	233	251	274	274	295	313	327
v17 = 8 b17 = 78	0	0	21	55	76	81	97	131	138	157	170	193	212	219	233	251	274	274	295	313	327
v18 = 10 b18 = 46	0	0	21	55	76	81	97	131	138	157	170	193	212	219	233	251	274	274	295	313	327
v19 = 6 b19 = 83	0	0	21	55	76	81	97	131	138	157	170	193	212	219	233	251	274	276	295	313	327
v20 = 10 b20 = 4	0	0	21	55	76	81	97	131	138	157	170	193	212	219	233	251	274	276	295	313	327

```
Objetos de volumen: 5, 3, 4, 4, 4
Beneficio: 327
```

6.6. Resultado las pruebas

A la vista de los resultados obtenidos por el algoritmo, se estima que este es correcto. En todas las selecciones de objetos, el algoritmo ha escogido las que resulten con mayor beneficio.

7. ANEXO: Código fuente

Listado de archivos:

7.1. mochila.dinamica.MochilaDinamica

```java
/*
 * Archivo: MochilaDinamica.java
 * Autor: Carlos Caride Santeiro
 * DNI: 44446239G
 * Email: ccaride5@alumno.uned.es
 * Fecha: 15/12/2017
 * Asignatura: Programación y Estructuras de Datos Avanzadas.
 * Trabajo: Práctica 2 (2017-2018)
 */

package mochila.dinamica;

import java.io.File;
import java.io.IOException;
import java.io.PrintStream;

/**
 * Clase principal de la aplicación
 */
public class MochilaDinamica {

    /**
     * Punto de entrada de la aplicación
     */
    public static void main(String[] args) throws IOException {
        boolean traza = false;
        String ficheroIn = "";
        String ficheroOut = "";
        int desfaseArgumentos = 0;

        if (args.length == 0) {
            System.out.println("ERROR: Sintaxis incorrecta");
            imprimirAyuda();
            return;
        }
        if (args[0].equalsIgnoreCase("-h")){
```

```
37              imprimirAyuda();
38              return;
39          }
40
41          if (args[0].equalsIgnoreCase("-t")){
42              traza = true;
43              desfaseArgumentos++;
44          }
45
46          if (args.length <= desfaseArgumentos) {
47              System.out.println("ERROR: Pocos argumentos");
48              imprimirAyuda();
49              return;
50          }
51
52          ficheroIn = args[desfaseArgumentos++];
53
54          if (args.length > desfaseArgumentos) {
55              ficheroOut = args[desfaseArgumentos];
56          }
57
58          PrintStream ps;
59          if (ficheroOut.isEmpty()) {
60              ps = System.out;
61          } else {
62              File f = new File(ficheroOut);
63              if (f.exists()) {
64                  f.delete();
65              }
66              f.createNewFile();
67              ps = new PrintStream(f);
68          }
69
70          Mochila mochila = new Mochila();
71
72          if (!mochila.leerDatos(ficheroIn)) {
73              return;
74          }
75
76          mochila.resolver();
77          mochila.imprimirContenido(traza, ps);
78
79          if (!ficheroOut.isEmpty()) {
80              ps.close();
81          }
82      }
83
84      /**
85       * Imprime la ayuda que se le mostrará al usuario
86       */
87      private static void imprimirAyuda() {
88          System.out.println("SINTAXIS:");
89          System.out.println("mochila-dinamica [-t] [-h] [fichero_entrada] [fichero_salida]");
90          System.out.println("-t               Traza la selección de clientes");
91          System.out.println("-h               Muestra esta ayuda");
92          System.out.println("fichero_entrada  Nombre del fichero de entrada");
```

```
93          System.out.println("fichero_salida   Nombre del fichero de salida");
94      }
95  }
```

7.2. mochila.dinamica.Mochila

```
1   /*
2    * Archivo: Mochila.java
3    * Autor: Carlos Caride Santeiro
4    * DNI: 44446239G
5    * Email: ccaride5@alumno.uned.es
6    * Fecha: 15/12/2017
7    * Asignatura: Programación y Estructuras de Datos Avanzadas.
8    * Trabajo: Práctica 2 (2017-2018)
9    */
10
11  package mochila.dinamica;
12
13  import java.io.BufferedReader;
14  import java.io.FileNotFoundException;
15  import java.io.FileReader;
16  import java.io.IOException;
17  import java.io.PrintStream;
18  import java.util.ArrayList;
19  import mochila.dinamica.algoritmos.MochilaEntera;
20
21  /**
22   * Representa la mochila en la que se introducen los objetos
23   */
24  public class Mochila {
25
26      private int capacidadMaxima;
27      private ArrayList<Objeto> objetosEnMochila;
28      private Objeto[] objetosOriginales;
29      MochilaEntera algoritmo;
30
31      /**
32       * Inicializa una nueva mochila vacía.
33       */
34      public Mochila() {
35          objetosEnMochila = new ArrayList<Objeto>();
36      }
37
38      /**
39       * Inicializa una nueva mochila indicando el volumen máximo
40       * @param capacidad Volumen máximo que puede contener la mochila
41       */
42      public Mochila(int capacidad) {
43          this();
44          capacidadMaxima = capacidad;
45      }
46
47      /**
48       * Obtiene la capacidad máxima de la mochila
49       * @return la capacidad máxima
```

```
50          */
51         public int getCapacidadMaxima() {
52             return capacidadMaxima;
53         }
54 
55         /**
56          * Establece la capacidad máxima de la mochila
57          * @param capacidadMaxima la capacidad máxima
58          */
59         public void setCapacidadMaxima(int capacidadMaxima) {
60             this.capacidadMaxima = capacidadMaxima;
61         }
62 
63         /**
64          * Lee la configuración de la mochila y los objetos a seleccionar
65          * @param ficheroEntrada Fichero de donde leer la configuración
66          * @return Verdadero en caso de no existir ningún error.
67          */
68         public boolean leerDatos(String ficheroEntrada) {
69             FileReader reader;
70             BufferedReader br;
71             int numeroObjetos;
72 
73 
74             if (ficheroEntrada.isEmpty()) {
75                 System.out.println("ERROR: No se indicó fichero de entrada");
76                 return false;
77             }
78 
79             try {
80                 reader = new FileReader(ficheroEntrada);
81                 br = new BufferedReader(reader);
82 
83                 numeroObjetos = Integer.parseInt(br.readLine());
84                 objetosOriginales = new Objeto[numeroObjetos + 1];
85                 for (int i = 1; i <= numeroObjetos; i++) {
86                     String[] cadena = br.readLine().split(" ");
87                     objetosOriginales[i] = new Objeto(Integer.parseInt(cadena[0]),
88                             Integer.parseInt(cadena[1]));
89                 }
90                 capacidadMaxima = Integer.parseInt(br.readLine());
91                 br.close();
92             } catch (FileNotFoundException ex) {
93                 System.out.println("ERROR: No se encontró el fichero de entrada");
94                 return false;
95             } catch (IOException | NumberFormatException |
96                     IndexOutOfBoundsException ex) {
97                 System.out.println("ERROR: El formato del fichero es erróneo");
98                 return false;
99             }
100 
101             return true;
102         }
103 
104         /**
105          * Resuelve la selección de objetos a contener en la mochila mediante
```

```
106      * selección por máximo beneficio
107      */
108     public void resolver() {
109         algoritmo = new MochilaEntera(objetosOriginales, capacidadMaxima);
110         algoritmo.objetosMochila(objetosEnMochila);
111     }
112 
113     /**
114      * Imprime el resultado de la selección del algoritmo
115      * @param traza Verdadero si se desea imprimir la tabla del algoritmo
116      * @param ps Salida de texto, fichero o consola
117      */
118     public void imprimirContenido(boolean traza, PrintStream ps) {
119         int beneficio = 0;
120         Objeto actual;
121 
122         if (traza) {
123             algoritmo.trazarResultado(ps);
124         }
125 
126         ps.print("Objetos de volumen: ");
127         for (int i = 0; i < objetosEnMochila.size(); i++) {
128             actual = objetosEnMochila.get(i);
129             if (i != 0) {
130                 ps.print(", ");
131             }
132             ps.print(actual.getVolumen());
133             beneficio += actual.getBeneficio();
134         }
135         ps.println();
136         ps.print("Beneficio: ");
137         ps.println(beneficio);
138     }
139 }
```

7.3. mochila.dinamica.Objeto

```
1  /*
2   * Archivo: Objeto.java
3   * Autor: Carlos Caride Santeiro
4   * DNI: 44446239G
5   * Email: ccaride5@alumno.uned.es
6   * Fecha: 15/12/2017
7   * Asignatura: Programación y Estructuras de Datos Avanzadas.
8   * Trabajo: Práctica 2 (2017-2018)
9   */
10 
11 package mochila.dinamica;
12 
13 /**
14  * Representa un objeto que puede ser introducido en la mochila
15  */
16 public class Objeto {
17 
18     private int volumen;
```

```
19      private int beneficio;
20
21      /**
22       * Constructor por defecto
23       */
24      public Objeto() {
25
26      }
27
28      /**
29       * Constructor indicando el volumen y el beneficio del objeto
30       * @param volumen el volumen que ocupa el objeto
31       * @param beneficio el beneficio que reporta el objeto
32       */
33      public Objeto(int volumen, int beneficio) {
34          this.volumen = volumen;
35          this.beneficio = beneficio;
36      }
37
38      /**
39       * Obtiene el volumen del objeto
40       * @return el volumen que ocupa el objeto
41       */
42      public int getVolumen() {
43          return volumen;
44      }
45
46      /**
47       * Establece el volumen del objeto
48       * @param volumen el volumen que ocupa el objeto
49       */
50      public void setVolumen(int volumen) {
51          this.volumen = volumen;
52      }
53
54      /**
55       * Obtiene el beneficio del objeto
56       * @return el beneficio que reporta el objeto
57       */
58      public int getBeneficio() {
59          return beneficio;
60      }
61
62      /**
63       * Establece el beneficio del objeto
64       * @param beneficio el beneficio que reporta el objeto
65       */
66      public void setBeneficio(int beneficio) {
67          this.beneficio = beneficio;
68      }
69
70      /**
71       * Obtiene la relación entre beneficio y volumen (beneficio/volumen)
72       * @return la relación entre beneficio y volumen
73       */
74      public float getRelacion() {
```

```
75          return (float)beneficio/(float)volumen;
76      }
77  }
```

7.4. mochila.dinamica..algoritmos.MochilaEntera

```
1   /*
2    * Archivo: MochilaEntera.java
3    * Autor: Carlos Caride Santeiro
4    * DNI: 44446239G
5    * Email: ccaride5@alumno.uned.es
6    * Fecha: 15/12/2017
7    * Asignatura: Programación y Estructuras de Datos Avanzadas.
8    * Trabajo: Práctica 2 (2017-2018)
9    */
10
11  package mochila.dinamica.algoritmos;
12
13  import java.io.PrintStream;
14  import java.util.ArrayList;
15  import mochila.dinamica.Objeto;
16
17  /**
18   * Algoritmo para resolver el problema de la mochila entera mediante
19   * programación dinámica
20   */
21  public class MochilaEntera {
22      int[][] tabla;
23      Objeto[] objetosOriginales;
24      int volumenMaximo;
25
26      /**
27       * Inicializa la mochila y crea la tabla de beneficio máximo
28       * @param objetos Vector con los objetos a seleccionar
29       * @param volumen Volumen máximo de la mochila
30       */
31      public MochilaEntera(Objeto[] objetos, int volumen) {
32          objetosOriginales = objetos;
33          volumenMaximo = volumen;
34
35          tabla = new int[objetos.length][volumenMaximo + 1];
36
37          for (int i = 0; i < objetos.length; i++) {
38              tabla[i][0] = 0;
39          }
40
41          for (int j = 0; j <= volumenMaximo; j++) {
42              tabla[0][j] = 0;
43          }
44
45          for (int i = 1; i < objetos.length; i++) {
46              for (int j = 1; j <= volumenMaximo; j++) {
47                  Objeto obj = objetosOriginales[i];
48                  if(obj.getVolumen() > j) {
49                      tabla[i][j] = tabla[i-1][j];
```

```
                } else {
                    tabla[i][j] = Integer.max(tabla[i-1][j],
                            tabla[i-1][j-obj.getVolumen()] + obj.getBeneficio());
                }
            }
        }
    }

    /**
     * Devuelve la lista de objetos seleccionados por el algoritmo
     * @param salida Los objetos seleccionados
     */
    public void objetosMochila(ArrayList<Objeto> salida) {
        int w = volumenMaximo;

        for (int i = objetosOriginales.length - 1; i > 0; i--) {
            if (tabla[i][w] != tabla[i-1][w]) {
                salida.add(objetosOriginales[i]);
                w -= objetosOriginales[i].getVolumen();
            }
        }
    }

    /**
     * Traza la tabla del algoritmo
     * @param salida Recurso de salida
     */
    public void trazarResultado(PrintStream salida) {
        salida.print("Límite volumen       |");
        for (int i = 0; i <= volumenMaximo; i++) {
            salida.print(String.format("%4s", Integer.toString(i)));
        }
        salida.println();
        salida.print("----------------------|");
        for (int i = 0; i <= volumenMaximo; i++) {
            salida.print("----");
        }
        salida.println();

        for (int i = 0; i < objetosOriginales.length; i++) {
            if (i == 0) {
                salida.print("posición 0           |");
            } else {
                salida.print(String.format("v%1$d = %2$3s  b%1$d = %3$3s", i,
                        Integer.toString(objetosOriginales[i].getVolumen()),
                        Integer.toString(objetosOriginales[i].getBeneficio())));
                if (i < 10) {
                    salida.print("   |");
                } else {
                    salida.print(" |");
                }
            }
            for (int j = 0; j <= volumenMaximo; j++) {
                salida.print(String.format("%4s", Integer.toString(tabla[i][j])));
            }
            salida.println();
```

```
106             }
107
108             salida.println();
109         }
110     }
```