EJERCICIO DE LA MOCHILA 0-1

Problema

Tenemos n objetos distintos de pesos conocidos p_i, con valores v_i con $1 \leq i \leq n$ y una mochila capaz de almacenar el peso total W. Queremos maximizar el valor de los objetos introducidos en la mochila, teniendo en cuenta que la suma de sus pesos no puede exceder W. Se pide:

- **a) Demostrar que el problema exhibe la propiedad de subestructura óptima.**
- **b) Encontrar una expresión recurrente para el valor óptimo (ecuación de Bellman).**
- **c) Implementar dos algoritmos de programación dinámica que resuelvan el problema para un conjunto de objetos n, con valores v_i, pesos p_i y un peso máximo W:**
  - **c.1) El primero debe obtener el valor óptimo siguiendo un esquema iterativo.**
  - **c.2) El segundo debe obtener el valor óptimo siguiendo un esquema recursivo con memoria (*memoizado*).**
- **d) Implementar un algoritmo que reconstruya la solución (decisión tomada para cada objeto) a partir de los valores de la estructura de soluciones óptimas.**

Solución

Apartado a)

Este apartado se resolvió en clase:

1. Sea N el conjunto de los objetos n_i disponibles, con $1 \leq i \leq n$.
2. Sean p_i y v_i , respectivamente, el peso y el valor del objeto n_i.
3. Sea Sol el vector solución óptima a la instancia del problema $MOCHILA01\langle N, W\rangle$, donde Sol_i tiene el valor 1 si el objeto n_i se introduce en la mochila y 0 en caso contrario.
4. Sea $S = \sum_{i=1}^{n} v_i Sol_i$ el valor de la solución óptima Sol.
5. Sea $S' = S - v_k Sol_k$ el valor de una solución a la instancia $MOCHILA01\langle N - \{n_k\}, W - (p_k Sol_k)\rangle$.
6. ¿Puede existir $S'' < S'$? No, porque en ese caso $S'' + v_k Sol_k$ sería el valor de una solución mejor que Sol para $MOCHILA01\langle N, W\rangle$ y eso contradice el enunciado 3.

Apartado b)

Este apartado se resolvió en clase. La ecuación obtenida es la que se muestra a continuación:

$$A_{i,j} = \begin{cases} 0 & si\ (j = 0) \vee (i = 0) \\ A_{i-1,j} & si\ (p_i > j) \wedge (i > 0) \wedge (j > 0) \\ \max\left(A_{i-1,j}, v_i + A_{i-1,j-p_i}\right) & en\ otro\ caso \end{cases}$$

Apartado c)

Para hacer coincidir los índices de p, v y sol con los valores de las filas de la matriz a, añadimos a estos vectores una primera posición adicional.

Apartado c.1)

```
private static int mochila01(int[] p, int[] v, int w) {
    int[][] a = new int[n + 1][w + 1];
    rellenarTablaMochila01(a, p, v);
    return a[n][w];
}
```

```java
private static void rellenarTablaMochila01(int[][] a, int[] p, int[] v) {
    for (int i = 0; i <= n; i++) // j = 0
        a[i][0] = 0;
    for (int j = 1; j <= W; j++) // i = 0
        a[0][j] = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= W; j++) {
            a[i][j] = (p[i] > j) ?
                    a[i - 1][j] : // p[i] > j, i > 0, j > 0
                    Math.max(a[i - 1][j], v[i] + a[i - 1][j - p[i]]); // En otro caso
        }
}
```

Apartado c.2)

```java
private static int mochila01Rec(int[] p, int[] v, int w) {
    int[][] a = new int[n+1][w+1];
    rellenarTablaMochila01Rec(a, p, v, n, w);
    return a[n][w];
}

private static void rellenarTablaMochila01Rec(int[][] a, int[] p, int[] v, int i, int j) {
    if (i == 0 || j == 0)
        a[i][j] = 0;
    else {
        if (a[i - 1][j] == noCalculado)
            rellenarTablaMochila01Rec(a, p, v, i - 1, j);
        if (!(p[i] > j) && a[i - 1][j - p[i]] == noCalculado)
            rellenarTablaMochila01Rec(a, p, v, i - 1, j - p[i]);
        a[i][j] = (p[i] > j) ?
                a[i - 1][j] : // p[i] > j, i > 0, j > 0
                Math.max(a[i - 1][j], v[i] + a[i - 1][j - p[i]]); // En otro caso
    }
}
```

Apartado d)

```java
private static void reconstruirSolucionMochila01(int[][] a, int[] p, int[] v, int[] sol) {
    int j = W, i = n;
    while (i >= 1 && j >= 0) {
        if (!(p[i] > j) && a[i - 1][j - p[i]] + v[i] == a[i][j]) {
            sol[i] = 1; // Se usa el objeto i
            j = j - p[i];
        } else
            sol[i] = 0; // No se usa el objeto i
        i--;
    }
}
```