EJERCICIO DEL CAMBIO DE MONEDAS

Problema

Tenemos un suministro ilimitado de monedas de n valores distintos, d_i, con $1 \leq i \leq n$. Debemos pagar cierta cantidad M empleando el <u>mínimo</u> número de monedas. Se pide:

a) **Demostrar que el problema exhibe la propiedad de subestructura óptima.**
b) **Encontrar una expresión recurrente para el número óptimo de monedas (ecuación de Bellman).**
c) **Implementar dos algoritmos de programación dinámica que resuelvan el problema para un conjunto de denominaciones d y una cantidad a devolver M:**
    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 (número de monedas de cada tipo) a partir de los valores de la estructura de soluciones óptimas.**

Solución

Apartado a)

1. Sea D el conjunto de las denominaciones d_i.
2. Sea Sol el vector <u>solución óptima</u> a la instancia del problema $MONEDAS\langle D, M\rangle$, donde Sol_i es el número de monedas de la denominación d_i que se escogen.
3. Sea $S = \sum_{i=1}^{n} Sol_i$ el valor de la solución óptima Sol.
4. Sea $S' = \sum_{i=1, i\neq k}^{n} Sol_i$el valor de una solución a la instancia $MONEDAS\langle D - \{d_k\}, M - (d_k Sol_k)\rangle$.
5. ¿Puede existir $S'' < S'$? No, porque en ese caso $S'' + Sol_k$ sería el valor de una solución <u>mejor que Sol</u> para $MONEDAS\langle D, M\rangle$ y eso <u>contradice el enunciado 2</u>.

Apartado b)

Este apartado se resolvió en clase. La ecuación obtenida es la que se muestra a continuación, considerando los índices de las componentes de A y d comenzando en cero (para concordar con las implementaciones propuestas en java). Para simplificar la expresión, <u>estamos suponiendo que la denominación más pequeña tiene valor unitario</u>:

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

Apartado c)

Apartado c.1)

```java
private static int cambioMonedas(int[] d, int m) {
    int[][] a = new int[n][m+1];
    rellenarTablaCambioMonedas(a, d);
    return a[n-1][m];
}
```

```java
private static void rellenarTablaCambioMonedas(int[][] a, int[] d) {
    // j = 0
    for (int i = 0; i < n; i++)
        a[i][0] = 0;
    // i = 0, j > 0
    for (int j = 1; j <= M; j++)
        a[0][j] = 1 + a[0][j - d[0]];
    for (int i = 1; i < n; i++)
        for (int j = 1; j <= M; j++) {
            a[i][j] = (d[i] > j) ?
                    a[i - 1][j] : // d[i] > j, i > 0, j > 0
                    Math.min(a[i - 1][j], 1 + a[i][j - d[i]]); // En otro caso
        }
}
```

Apartado c.2)

```java
private static int cambioMonedasRec(int[] d, int m) {
    int[][] a = new int[n][m+1];
    rellenarTablaCambioMonedasRec(d, a, n-1, m);
    return a[n-1][m];
}

private static void rellenarTablaCambioMonedasRec(int[] d, int[][] a, int i, int j) {
    if (j == 0)
        a[i][j] = 0;
    else if (i == 0) { // i = 0, j > 0
        if (a[i][j - d[i]] == noCalculado)
            rellenarTablaCambioMonedasRec(d, a, i, j - d[i]);
        a[i][j] = 1 + a[i][j - d[i]];
    } else {
        if (a[i - 1][j] == noCalculado)
            rellenarTablaCambioMonedasRec(d, a, i - 1, j);
        if (!(d[i] > j) && a[i][j - d[i]] == noCalculado)
            rellenarTablaCambioMonedasRec(d, a, i, j - d[i]);
        a[i][j] = (d[i] > j) ? a[i - 1][j] : // d[i] > j, i > 0, j > 0
                Math.min(a[i - 1][j], 1 + a[i][j - d[i]]); // En otro caso
        verTabla(a);
    }
}
```

Apartado d)

```java
private static void reconstruirSolucionCambioMonedas(int[][] a, int[] d, int[] sol) {
    int j = M, i = n-1;
    while (i >= 0 && j >= 0) {
        if (!(d[i] > j) && a[i][j-d[i]] + 1 == a[i][j]) { // Se usa moneda d[i]
            sol[i]++;
            j = j-d[i];
        }
        else {// No se usa moneda d[i]
            i--;
        }
    }
}
```