EJERCICIOS TEMA 4

Algoritmia y Complejidad

Realizado por:

Gabriel López Cuenca
Miguel Ángel Losada Fernández
Sergio Sanz Sacristán
Álvaro Zamorano Ortega

Ejercicio 3

Se tiene un sistema de billetes de distintos valores y ordenados de menor a mayor (por ejemplo 1, 2, 5, 10, 20, 50 y 100 euros), que se representan mediante los valores v_i, con i $\in \{1,\cdots, N\}$ (en el caso anterior, N=7) de manera que de cada billete se tiene una cantidad finita, mayor o igual a cero, que se guarda en c_i (siguiendo con el ejemplo, c_3=6 representaría que hay 6 billetes de 5 euros).

Se quiere pagar exactamente una cierta cantidad de dinero D, utilizando para ello la menor cantidad de billetes posible. Se sabe que D<= $\sum^{n}_{i=1}$ civi, pero puede que la cantidad exacta D no sea obtenible mediante los billetes disponibles.

Diseñar un algoritmo con la metodología de Programación Dinámica que determine, teniendo como datos los valores c_i, v_i y D:

- **Si la cantidad D puede devolverse exactamente o no, y**
- **en caso afirmativo, cuantos billetes de cada tipo forman la descomposición óptima.**

D = 12

V = {1, 3, 5}

C = {13, 2, 1}

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	1	2	3	4	5	6	7	8	9	10	11
3	0	1	2	1	2	3	2	3	4	5	6	7
5	0	1	2	1	2	1	2	3	2	3	4	3

Hay tres tipos distintos para calcular el valor a devolver en la tabla:

1. Si el valor a devolver es menor que la moneda actual en la que estamos calculando entonces el número de monedas será igual al de la anterior moneda. Por ejemplo, en (5, 4) el 4<5 por lo tanto el numero de monedas será el de la anterior moneda (3, 4)=2.
2. Si el valor a devolver es mayor o igual que la moneda actual.
    a. Si hay monedas suficientes a devolver ese valor, el numero de monedas necesarias será el mínimo entre el numero de monedas de la moneda anterior y el numero de monedas que se necesitan si añadimos la moneda actual. Por ejemplo, en la casilla (3, 7) será el mínimo de (7, 1+casilla(3,4))=3.
    b. Si se han agotado las monedas de un cierto valor, el numero de monedas necesarias será el mínimo entre la casilla superior y la casilla de la fila anterior restándole a la columna el valor de la moneda por la cantidad de monedas que hay mas el número de monedas que hay de ese cierto valor. Por ejemplo, para calcular la casilla (5, 11) es el mínimo entre la casilla (3, 11) y la casilla (3, 6)+el numero de monedas del 5 (1). Por lo tanto, es el mínimo entre 7 y 2+1.

```java
public class TEMA4_EJ3 {

    //Atributos para indicar las monedas que tenemos
    //y la cantidad de cada una de ellas
    private static final int[] monedas = {1, 3, 5};
    private static final int[] cantidad = {13, 2, 1};

    public static void main(String[] args) {
        int devolver = 11;

        int mon = cambio(devolver);

        if (mon != 0) {
            System.out.println("Cantidad monedas necesarias para devolver "
                    + devolver + ": " + mon);
        } else {
            System.out.println(devolver + " no se puede devolver con las"
                    + " monedas disponibles");
        }
    }

    /**
     * Método para rellenar la matriz de cambios
     *
     * @param precio
     * @return
     */
    private static int cambio(int precio) {

        int[][] matrizCambio = new int[monedas.length + 1][precio + 1];

        //Inicializar primeras filas y columnas
        for (int i = 0; i < monedas.length; i++) {
            matrizCambio[i][0] = 0;
        }

        for (int j = 1; j <= precio; j++) {
            matrizCambio[0][j] = 99;
        }

        //Algoritmo para rellenar la matriz
        for (int i = 1; i <= monedas.length; i++) {
            for (int j = 1; j <= precio; j++) {
                //El minimo en cada paso comienza siendo la casila superior a la
                //que estamos mirando (en caso de no coger la nueva moneda)
                int minimo = matrizCambio[i - 1][j];
                if (j < monedas[i - 1]) {
                    //Si la columna es menor que el valor de la nueva moneda,
                    //nos quedamos con la superior
                    matrizCambio[i][j] = minimo;
                } else {
                    if (hayMoneda(monedas[i - 1], cantidad[i - 1], j)) {
                        //Si hay suficientes monedas del nuevo valor para
                        //devolver el cambio
                        minimo = Math.min(minimo, matrizCambio[i][j
                                - monedas[i - 1]] + 1);
                        //Minomo entre la casilla superior y la casilla de la
                        //misma columna haciendo hueco a la nueva moneda y le
                        //sumamos 1 (la estamos usando)
                    } else {
                        //Si no hay monedas suficientes para devolver ese
                        //cierto valor
```

```
                        minimo = Math.min(minimo, matrizCambio[i - 1][j
                                - (monedas[i - 1] * cantidad[i - 1])]
                                + cantidad[i - 1]);
                        //Minimo entre la casilla superior y de la fila anterior
                        //la columna correspondiente a quitar la cantidad que
                        //podemos devolver usando las monedas disponibles más
                        //ese número de monedas que usamos
                    }
                    matrizCambio[i][j] = minimo;
                }
                //Reestablecer el mínimo
                minimo = 0;
            }
        }

        imprimirMatriz(monedas.length, precio, matrizCambio);

        //Si la cantidad de monedas es distinta de 99 (infinito), se puede
        //devolver esa cantidad con las monedas que disponemos
        int cant = 0;
        if (matrizCambio[monedas.length][precio] != 99) {
            cant = matrizCambio[monedas.length][precio];
        }
        return cant;
    }

    /**
     * Método para imprimir la matriz
     *
     * @param m
     * @param c
     * @param matriz
     */
    private static void imprimirMatriz(int m, int c, int[][] matriz) {
        System.out.println("**** MATRIZ DE CAMBIOS ****");
        for (int i = 1; i <= m; i++) {
            for (int j = 0; j <= c; j++) {
                System.out.print(matriz[i][j]);
                if (j < c) {
                    System.out.print(", ");
                }
            }
            System.out.println("\n");
        }
    }

    /**
     * Método para comprobar si hay monedas suficientes de una cierta moneda
     * para devolver una cierta cantidad
     *
     * @param valor
     * @param cantidad
     * @param precio
     * @return
     */
    private static boolean hayMoneda(int valor, int cantidad, int precio) {
        return ((valor * cantidad) >= precio);
    }
}
```

Ejercicio 6

¡Llega el torneo de EscobaBall, y más salvaje que nunca! Este año, en el Colegio de Magia y Hechicería han decidido que los cuatro equipos (Grifos, Serpientes, Cuervos y Tejones) jueguen en cada partido todos contra todos, y como siempre que ningún partido termine en empate. El torneo acabará cuando un mismo equipo haya ganado un total de N partidos (no necesariamente consecutivos).

El aprendiz de mago Javi Potter quiere apostar algo de dinero por su equipo, los Grifos, así que se dirige a la casa de apuestas de los gnomos para ver cuánto le darían si gana su equipo: sus ganancias serían iguales a la cantidad de dinero apostado dividido por la probabilidad de que el equipo gane el campeonato (por ejemplo, si los Grifos tuviesen un 50% de ganar el campeonato y Javi apuesta 10 monedas de oro, sus posibles ganancias serían 10/0'5 = 20 monedas de oro; si tuviesen un 20% de ganar, el beneficio posible sería de 10/0'2 = 50 monedas de oro, mayor recompensa al ser más difícil de conseguir).

Para obtener esta probabilidad, la casa de apuestas tiene un Valor de Calidad asignado a cada equipo (que mide la habilidad de los jugadores, su motivación, etc. y que es un valor fijo para el equipo e independiente del partido que esté jugando) de manera que cuanto mayor es el VC de un equipo, más probabilidades tiene de ganar un partido. Por ejemplo, si los cuatro equipos tuviesen igual VC todos tendrían un 25% de ganar un partido. Si tres equipos tuviesen el mismo VC y el cuarto equipo tuviese el doble de esa cantidad, los primeros tendrían un 20% y el último un 40%. Como los partidos no pueden terminar en empate, la suma de las probabilidades siempre es el 100%.

Teniendo como datos los Valores de Calidad de los equipos, la cantidad de partidos N que debe ganar un equipo para conseguir ganar el torneo, y el dinero D apostado por Javi Potter, obtener cuáles serían las ganancias si ganasen los Grifos.

P(0,j,k,l) = 1 → la probabilidad de que gane el primer equipo cuando le queden 0 partidos por ganar.

P(i,0,k,l) = 0 → probabilidad que le quedan a Javi Potter cuando gana el segundo equipo.

P(i,j,0,l) = 0 → probabilidad que le quedan a Javi Potter cuando gana el tercer equipo.

P(i,j,k,l) = 0 → probabilidad que le quedan a Javi Potter cuando gana el cuarto equipo.

P(i,j,k,l) → probabilidad de que gane el primer equipo cuando le quedan i partidor por ganar, al segundo equipo le faltan j, al tercero le faltan k y al cuarto equipo le faltan l partidos. Se calcula como:

- P(i,j,k,l) = probEquipo[1] * P(i-1,I,k,l) + probEquipo[2] * P(i,I-1,k,l) + probEquipo[3] * P(i,I,k-1,l) + probEquipo[4] * P(i,I,k,l-1) → cada equipo gana su partido.

Ganancias → Dinero apostado por Javi Potter / P(n,n,n,n)

```java
public class TEMA4_EJ6 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int[] VC = {35, 10, 5, 50};
        int numPartidos = 5;
        System.out.println("JaviPotter apuesta 10€ en un torneo de "
                + numPartidos + " partidos con una probabilidad para su equipo"
                + " del " + VC[0] + "%" + ", finalmente obtiene un total "
                + "de " + JaviPotter(VC, numPartidos, 10) + "€");
    }

    /**
     * Método que devuelve las ganancias de JaviPotter si ganasen los grifos en
     * función del número de partidos y probabilidad de cada equipo
     *
     * @param VC valores de la calidad de cada equipo (probabilidad como entero)
     * @param n número de partidos
     * @param dineroApostado dinero que apuesta JaviPotter
     * @return
     */
    public static float JaviPotter(int[] VC, int n, int dineroApostado) {
        //Sin sumarle 1 al número de partidos no se completan los bucles
        n++;
        //Incializamos la matriz de la probablididad (a todos los partidos le
        //faltan n partidos por ganar)
        float[][][][] P = new float[n][n][n][n];
        //Inicializamos el array de las probabilidades de cada equipo
        float[] probEquipos = new float[4];
        for (int i = 0; i < 4; i++) {
            //Obtenemos el porcentaje de la probabilidad para cada equipo
            probEquipos[i] = (float) VC[i] / 100;
        }

        //Apuesta por el primer equipo y si su probabilidad es 0 entonces no
        //gana nada
        if (probEquipos[0] == 0)
        {
            return 0;
        }
        for (int i = 1; i < n; i++) {
            for (int j = 1; j < n; j++) {
                for (int k = 1; k < n; k++) {
                    //Si el numero de partidos que le quedan por ganar a un
                    //equipo es 0 significa que la probabilidad de ganar es
                    //1 (ya ha ganado)
                    P[0][i][j][k] = 1;
                    P[i][0][j][k] = 0;
                    P[i][j][0][k] = 0;
                    P[i][j][k][0] = 0;
                }
            }
        }
```

```
        for (int i = 1; i < n; i++) {
            for (int j = 1; j < n; j++) {
                for (int k = 1; k < n; k++) {
                    for (int l = 1; l < n; l++) {
                        //Multiplicamos las probabilidades de cada equipo por
                        //la probabilidad cuando gana un partido más
                        P[i][j][k][l] = probEquipos[0] * P[i - 1][j][k][l]
                                + probEquipos[1] * P[i][j - 1][k][l]
                                + probEquipos[2] * P[i][j][k - 1][l]
                                + probEquipos[3] * P[i][j][k][l - 1];
                    }
                }
            }
        }
        //Dinero entre la probabilidad final
        return dineroApostado / P[n - 1][n - 1][n - 1][n - 1];
    }
}
```