Semana 3 - Ejercicios de grupo

Métodos algorítmicos en resolución de problemas II
Facultad de Informática - UCM

	Nombres y apellidos de los componentes del grupo que participan	**ID Juez**
1	Daniela Alejandra Cordova	MAR220
2	Alejandro Corpas Calvo	MAR222
3	Erik Karlgren Domercq	MAR248
4	David Bugoi	MAR211

Instrucciones:

1. Para editar este documento, es necesario hacer una copia de él. Para ello:
    - Alguien del grupo inicia sesión con la cuenta de correo de la UCM (si no la ha iniciado ya) y accede a este documento.
    - Mediante la opción *Archivo* → *Hacer una copia*, hace una copia del documento en su propia unidad de *Google Drive*.
    - Abre esta copia y, mediante el botón *Compartir* (esquina superior derecha), introduce los correos de los demás miembros del grupo para que puedan participar en la edición de la copia.

2. La entrega se realiza a través del Campus Virtual. Para ello:
    - Alguien del grupo convierte este documento a PDF (dándole como nombre el número del grupo, 1.pdf, 2.pdf, etc…). Desde *Google Docs*, puede hacerse mediante la opción *Archivo* → *Descargar* → *Documento PDF*.
    - Esta misma persona sube el fichero PDF a la tarea correspondiente del *Campus Virtual*. Solo es necesario que uno de los componentes del grupo entregue el PDF.

Mejor no llevar muchas monedas

El objetivo de hoy es resolver el **problema 11 Mejor no llevar muchas monedas**, del juez automático. Ahí podéis encontrar el enunciado completo del problema.

Se trata de pagar de forma exacta una determinada cantidad con el menor número de monedas teniendo en cuenta que hay una cantidad limitada de cada tipo de monedas. Recordad que para ciertos sistemas monetarios y cuando la cantidad de monedas de cada tipo es ilimitada hay una estrategia voraz que resuelve el problema, pero no es el caso en un sistema monetario cualquiera o cuando el número de monedas de cada tipo es limitado.

Solución: (Plantead aquí la recurrencia que resuelve el problema y explicad las razones por las que es mejor resolver el problema utilizando programación dinámica que una implementación recursiva sin memoria.)
Definición:

- monedas(i, j) = mínimo número de monedas de los tipos 1 al i necesarias para pagar la cantidad j
- Rango de i: 0 <= i <= N
- Rango de j: 0 <= j <= precio

Llamada Inicial:

- monedas(N, precio)

Casos Base:

- monedas(i, 0) = 0 → el precio a pagar es 0
- monedas(0, j) = +∞ → no tengo monedas para pagar j (excepción: j==0 - Ver *Nota*)

Nota: monedas(0, 0) = 0

Casos Recursivos:

Siendo i un tipo de moneda en específico en el vector de monedas, Ci la cantidad de monedas que se tienen de ese tipo y Vi el valor de la moneda de ese tipo estando en la posición i de la matriz

- monedas(i, j) = monedas(i-1, j) → Ci <0 || Vi>j
- monedas(i, j) = min(monedas(i-1, j), monedas(i, j-Vi) + 1) → otro caso

Usamos programación dinámica ya que consiste en un problema de subproblemas repetidos. Al ir recorriendo los tipos de monedas y el precio a pagar y todas las posibles combinaciones posibles de usar las monedas que tenemos, se necesitará varias veces el mismo subproblema; por lo que es necesario programación dinámica.

Solución: (Escribid aquí las explicaciones necesarias para contar de manera comprensible la implementación del algoritmo de programación dinámica. En particular, explicad si es posible reducir el consumo de memoria y cómo hacerlo. Incluid el código y el coste justificado de las funciones que resuelven el problema. Extended el espacio al que haga falta.)

```
int monedas(vector<Datos> const& v, int n, int p) {
        Matriz <int> M(n + 1, p + 1, INFINITO);

        M[0][0] = 0;
        for (int i = 1; i <= n; ++i) {// recorre tipo monedas
```

```
                M[i][0] = 0;
                for (int j = 1; j <= p; ++j) { // recorrer precio a pagar

                        if (j >= v[i - 1].valor) {
                                int temp = M[i - 1][j];
                                for (int k = 1; k <= v[i - 1].cant && j - k * v[i - 1].valor >= 0 &&
 j - k * v[i - 1].valor <= p; ++k) //todos los casos que se pueden hacer con las cantidades de monedas posibles
                                        temp = min(M[i - 1][j - k * v[i - 1].valor] + k, temp);
                                M[i][j] = temp;
                        }
                        else
                                M[i][j] = M[i - 1][j];
                }
        }
        return M[n][p];
}
```

En el problema recorremos el vector de los tipos de monedas de derecha a izquierda, utilizándolas o no en función de si nos conviene para llegar a pagar el precio requerido (j), sin pasarnos y sin usar monedas de las que ya no nos quedan unidades. Utilizamos programación dinámica ascendente para guardar los resultados y así ahorrarnos el tener que calcularlos de nuevo en caso de que se repitan.

Coste en tiempo

El coste en tiempo de nuestra solución a este problema es de O(N*precio*cantidad total de monedas). n*p se ve de manera clara ya que tenemos 2 bucles for que se recorren esa cantidad de veces y el añadido de la cantidad total de monedas es debido a que tenemos otro bucle for interno que se recorre una cantidad de veces variable en función de la cantidad de monedas que tengamos de cada tipo, es decir, se recorrerá el sumatorio de cantidades de los tipos de monedas veces o lo que es lo mismo la cantidad de monedas total.

Coste en espacio

El coste en espacio de nuestra solución a este problema es del O(N*precio) siendo N el número de tipos de monedas y precio la cantidad a pagar. Esto es debido a que utilizamos una matriz de tamaño N+1 * precio+1.

Resolved el problema completo del juez, que ahora debe ser ya muy sencillo.

Número de envío: s32273