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	Borja Aday Guadalupe Luis	MAR241
2	Óscar Morujo Fernández	MAR262
3	Roberto Plaza Hermira	MAR267
4	Pablo Martínez	MAR258

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 : Sea N el número de monedas disponibles y C el precio que hay que pagar:
minMonedas(i,j) = mínimo número de monedas para pagar la cantidad j (0 <= j < C) teniendo en cuenta las monedas i (0 <= i <= N).

Casos base:

minMonedas(0,j) = Infinito 0 <= j <= C (No hay manera de pagar j con 0 monedas)

minMonedas(i,0) = 0 0 <= i <= N (Para pagar una cantidad de 0 necesitamos 0 monedas)

Casos recursivos:

minMonedas(i,j) = min(minMonedas(i,j),minMonedas(i-1, j * Vi),minMonedas(i-1, (j -1)* Vi,....,minMonedas(i-1, (j-k) * Vi)
Siendo Vi es valor de la moneda i
Siendo k el número de monedas del tipo i, con 0 <= k * Vi <= j

Llamada inicial :

minMonedas(N, C), donde 0 <= i < N y donde 0 <= j < C

Como en el resto de problemas que hemos realizado con programación dinámica, el objetivo está en evitar el cálculo repetido de subproblemas.

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.)

Versión 1: Matriz (N + 1) (C +1)

La complejidad temporal es O(N*C), ya que tenemos que recorrer la matriz entera.
La complejidad espacial es O(N*C).

```
bool minMonedas(vector<Moneda> moneda, int N, int C) {
    Matriz<EntInf> M(N + 1, C + 1);

    for (int i = 0; i < N + 1; ++i)
        M[i][0] = 0;

    for (int j = 1; j < C + 1; ++j)
        M[0][j] = Infinito;

    for (int i = 1; i < N + 1; ++i)
        for (int j = 1; j < C + 1; ++j) {
            EntInf mini = Infinito;
            for (int k = 0; k <= moneda[i - 1].cantidad && k * moneda[i - 1].valor <= j; ++k) {
                if (M[i - 1][j - k * moneda[i - 1].valor] != Infinito) {
                    mini = min(M[i - 1][j - k * moneda[i - 1].valor] + k, mini);
                }
            }
            M[i][j] = mini;
        }

    if (M[N][C] == Infinito)
        cout << "NO";
    else
        cout << "SI " << M[N][C];

    cout << '\n';

    return true;
}
```

Versión 2: matriz de 2 x C+1

La complejidad temporal es O(N*C), ya que tenemos que recorrer la matriz entera.
La complejidad espacial es O(2*N), es decir O(N).

```
bool minMonedas(vector<Moneda> moneda, int N, int C) {
    Matriz<EntInf> M(2, C + 1);

        M[0][0] = 0;
        M[1][0] = 0;

    for (int j = 1; j < C + 1; ++j)
        M[0][j] = Infinito;

    for (int i = 1; i < N + 1; ++i)
        for (int j = 1; j < C + 1; ++j) {
            EntInf mini = Infinito;
            for (int k = 0; k <= moneda[i - 1].cantidad && k * moneda[i - 1].valor <= j; ++k) {
                if (M[(i - 1) % 2][j - k * moneda[i - 1].valor] != Infinito) {
                    mini = min(M[(i - 1) % 2][j - k * moneda[i - 1].valor] + k, mini);
                }
            }
            M[i % 2][j] = mini;
        }

    if (M[N % 2][C] == Infinito)
        cout << "NO";
    else
        cout << "SI " << M[N % 2][C];

    cout << '\n';

    return true;
}
```

Versión 3 : vector de C+1

La complejidad temporal es O(N*C), ya que tenemos que recorrer el vector de tamaño C+1 N veces..
La complejidad espacial es O(N).

```
bool minMonedas(vector<Moneda> moneda, int N, int C) {
    vector<EntInf> M(C + 1);

    M[0] = 0;

    for (int j = 1; j < C + 1; ++j)
        M[j] = Infinito;

    for (int i = 1; i < N + 1; ++i)
        for (int j = C; j >= 1; --j) {
            EntInf mini = Infinito;
            for (int k = 0; k <= moneda[i - 1].cantidad && k * moneda[i - 1].valor <= j; ++k) {
                if (M[j - k * moneda[i - 1].valor] != Infinito) {
                    mini = min(M[j - k * moneda[i - 1].valor] + k, mini);
                }
            }
            M[j] = mini;
        }

    if (M[C] == Infinito)
        cout << "NO";
    else
        cout << "SI " << M[C];

    cout << '\n';

    return true;
}
```

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

Número de envío: s32271