Semana 0 - 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	Pablo Martínez	**MAR258**
4	Roberto Plaza Hermira	**MAR267**

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.

La cuerda de la cometa

Se dispone de varios cordeles con los que un niño desea, a base de anudarlos entre sí, formar una cuerda de una determinada longitud para volar una cometa. Vuestra misión es ayudar al niño a decidir si se puede conseguir la longitud deseada con los cordeles de los que dispone. Los tres adultos que en ese momento cuidan del niño desean además saber otras cuestiones relacionadas con este problema: el número de formas posibles de conseguir la cuerda, el número mínimo de cordeles que se han de utilizar para conseguirla y el coste mínimo de los cordeles utilizados si conocemos el coste de cada uno de ellos. También debéis ayudarles a resolver esos problemas.

Podéis encontrar el enunciado completo en el **Problema 1** del juez automático.

Solución: (Plantead aquí por separado cada una de las cuatro recurrencia que resuelven los problemas planteados y explicad las razones por las que es mejor resolver los problemas utilizando programación dinámica que implementaciones recursivas sin memoria.)

Recurrencia 1: Posibilidad de construir la cuerda.

Definición:
posible(i, j) = variable booleana que indica si es posible construir una cuerda de longitud j utilizando las primeras i cuerdas.

Casos base:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

posible(0, j) = false 1 <= j <= R → No es posible construir una cuerda de tamaño distinto de 0 sin utilizar ninguna cuerda.

posible(i, 0) = true 0 <= i <= N → Siempre se puede construir una cuerda de tamaño 0 independientemente de las cuerdas que utilicemos (no escoger ninguna).

Casos recursivos:
Sea Li la longitud de la cuerda i y j la longitud de la cuerda deseada.

- Si j < Li → posible(i - 1, j)
- Si j >= Li → posible(i - 1, j) || posible(i - 1, j - Li)

Llamada inicial:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

posible(N, R)

Recurrencia 2: Cuantas formas de construir la cuerda.

Definición:
cuantas(i, j) = variable entera que indica de cuántas formas es posible construir una cuerda de longitud j utilizando las primeras i cuerdas.

Casos base:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

cuantas(0, j) = 0 1 <= j <= R → No se puede construir de ninguna manera una cuerda de tamaño distinto de 0 sin utilizar ninguna cuerda.

cuantas(i, 0) = 1 0 <= i <= N → Solo hay una forma de construir una cuerda de tamaño 0 con las primeras i cuerdas, no escoger ninguna.

Casos recursivos:
Sea Li la longitud de la cuerda i y j la longitud de la cuerda deseada.

- Si j < Li → cuantas(i - 1, j)
- Si j >= Li → cuantas(i - 1, j) + cuantas(i - 1, j - Li)

Llamada inicial:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

cuantas(N, R)

Recurrencia 3: Mínima cantidad de cuerdas para construir la cuerda.

Definición:
minCuerdas(i, j) = variable entera que indica el mínimo número de cuerdas a utilizar para construir una cuerda de longitud j utilizando las primeras i cuerdas.

Casos base:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

minCuerdas(0, j) = ∞ 1 <= j <= R → No se puede construir de ninguna manera una cuerda de tamaño distinto de 0 sin utilizar ninguna cuerda. Al estar minimizando, utilizamos el valor ∞ para las comparaciones.

minCuerdas(i, 0) = 0 0 <= i <= N → Se necesitan 0 cuerdas para construir una cuerda de tamaño 0 con las primeras i cuerdas.

Casos recursivos:
Sea Li la longitud de la cuerda i y j la longitud de la cuerda deseada.

- Si j < Li ⟶ minCuerdas(i - 1, j)
- Si j >= Li ⟶ min(minCuerdas(i - 1, j), minCuerdas(i - 1, j - Li) + 1)

Llamada inicial:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

minCuerdas(N, R)

Recurrencia 4: Mínimo coste para construir la cuerda.

Definición:
minCoste(i, j) = variable entera que indica el mínimo coste necesario para construir una cuerda de longitud j utilizando las primeras i cuerdas.

Casos base:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

minCoste(0, j) = ∞ 1 <= j <= R ⟶ No se puede construir de ninguna manera una cuerda de tamaño distinto de 0 sin utilizar ninguna cuerda. Al estar minimizando, utilizamos el valor ∞ para las comparaciones.

minCoste(i, 0) = 0 0 <= i <= N ⟶ Se necesitan 0 cuerdas para construir una cuerda de tamaño 0 con las primeras i cuerdas, por tanto, el coste es 0.

Casos recursivos:
Sea Li la longitud de la cuerda i, Ci el coste de la cuerda i y j la longitud de la cuerda deseada.

- Si j < Li ⟶ minCoste(i - 1, j)
- Si j >= Li ⟶ min(minCoste(i - 1, j), minCoste(i - 1, j - Li) + Ci)

Llamada inicial:
Sea N el número de cuerdas disponibles y R el tamaño de la cuerda objetivo:

minCoste(N, R)

El principal objetivo de la programación dinámica está en evitar el cálculo repetido de subproblemas. En este caso, si lo hacemos de forma recursiva repetiremos algunos subproblemas y además podemos provocar un fallo de pila si esta se llena. Es por ello que utilizamos memoria para tener guardados los resultados de los subproblemas para que cuando se repitan, no tengamos que calcularlos sino simplemente consultar su valor y además nos quitamos las llamadas recursivas que pueden llegar a llenar la pila.

Solución: (Escribid aquí las explicaciones necesarias para contar de manera comprensible la implementación de cada uno de los cuatro algoritmos 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 los problemas. Extended el espacio al que haga falta.)

Todas las funciones se realizan mediante programación dinámica ascendente.

Función 1 : posibilidad de construir la cuerda.

Para la resolución de este problema, usaremos una matriz que consta de :

- Filas : Cada fila representa un subconjunto de cuerdas.Tendremos tantas como número de cuerdas + 1. Así, si estamos en la fila i, consideramos el conjunto de las i primeras cuerdas.
- Columnas: Cada columna representa un tamaño de cuerda objetivo. Tendremos tantas columnas como indique la cuerda objetivo + 1 (cuerda de longitud 0).

Así, el elemento tabla[i][j] = true si podemos construir una cuerda de tamaño j teniendo en cuenta las i primeras cuerdas, o false en caso contrario.

Sabemos que este problema cumple el principio de optimalidad de Bellman, por lo que para resolver el problema final de manera óptima basta con considerar las sub soluciones óptimas.

En nuestro caso, cada fila que bajamos en la matriz o cada columna que nos desplazamos a la derecha complica el problema pues o bien pasamos a considerar una cuerda más o la cuerda objetivo crece en uno su tamaño, por lo que rellenaremos la matriz de arriba abajo y de izquierda a derecha.

A la hora de rellenar la tabla, longitudes[i - 1] nos indicará la cuerda que estamos considerando en ese momento y la j el tamaño de la cuerda objetivo. Por tanto, nos podemos encontrar en dos situaciones:

- longitudes[i - 1] > j : Esto significa que la cuerda que estamos considerando ahora mismo es mayor a la cuerda objetivo, por lo que no podemos usar este trozo de cuerda y solo tenemos comprobar si ya era posible construir la cuerda final sin tenerlo en cuenta.

- longitudes[i - 1] <= j : Esto significa que la cuerda que estamos considerando ahora mismo es menor o igual que la que tenemos que construir y por tanto se puede utilizar para conseguir la cuerda objetivo. En este caso solo tenemos que comprobar si ya era posible construir la cuerda final sin tener en cuenta el trozo de cuerda actual o si ya podíamos construir una cuerda de tamaño deseado menos la longitud del trozo de cuerda que estamos considerando actualmente (j - longitudes[i - 1]), pues al añadir este trozo, obtendríamos una cuerda del tamaño objetivo.

Por tanto, cuando terminemos de rellenar la matriz, habremos solucionado todos los subproblemas de nuestro problema global, incluyendolo. De modo que podremos encontrar la solución en la última posición de la matriz (tabla[N][R]).

Versión 1:

```
bool esPosible(int longitudes[], int numCuerdas, int tamCuerda) {
    bool tabla[MAX + 1][MAX + 1];

    //Si el tamaño de la cuerda es 0 se puede conseguir con cualquier
    //subconjunto de las cuerdas (no escoger ninguna).
    for (int i = 0; i <= numCuerdas; i++) tabla[i][0] = true;

    //Si el tamaño de la cuerda es distinto de 0 y no podemos
    //usar ninguna cuerda no se puede conseguir.
    for (int i = 1; i <= tamCuerda; i++) tabla[0][i] = false;

    //Rellenamos los datos
    for (int i = 1; i <= numCuerdas; i++) {
        for (int j = 1; j <= tamCuerda; j++) {
            if (j < longitudes[i - 1]) tabla[i][j] = tabla[i - 1][j];
            else tabla[i][j] = tabla[i - 1][j] || tabla[i - 1][j - longitudes[i - 1]];
        }
        if (tabla[i][tamCuerda]) return true;
    }

    //Si llego aquí significa que no he encontrado solución.
    return false;
}
```

Siendo N el número de cuerdas y R el tamaño de la cuerda a conseguir:

COMPLEJIDAD TEMPORAL	COMPLEJIDAD ESPACIAL
O(N*R)	O(N*R)

Al tener dos bucles anidados, uno que recorre N cuerdas y otro que recorre los R tamaños, habrá que recorrer la matriz de tamaño N*R entera calculando su valor. Además, guardamos en memoria la matriz de N*R.

Versión 2:

Con el objetivo de reducir la complejidad espacial hemos comprobado qué datos de los subproblemas anteriores necesitamos para resolver el problema actual. Los datos necesarios para resolver el subproblema tabla[i][j] son: tabla[i - 1][j] y tabla[i - 1][j - Li]. Como ambos datos pertenecen a la fila anterior, podemos ir olvidando el resto de filas de la matriz, por lo que solo necesitamos una matriz de tamaño 2*R en la que una fila representa el subproblema actual y la otra fila el subproblema anterior a este.

```
bool esPosible(int longitudes[], int numCuerdas, int tamCuerda) {
    bool tabla[2][MAX + 1];

    //Si el tamaño de la cuerda es 0 se puede conseguir con cualquier
    //subconjunto de las cuerdas (no escoger ninguna).
    tabla[0][0] = true;
    tabla[1][0] = true;

    //Si el tamaño de la cuerda es distinto de 0 y no podemos usar
    //ninguna cuerda no se puede conseguir.
    for (int i = 1; i <= tamCuerda; i++) tabla[0][i] = false;

    //Rellenamos los datos
    for (int i = 1; i <= numCuerdas; i++) {
        for (int j = 1; j <= tamCuerda; j++) {
            if (j < longitudes[i - 1]) tabla[i % 2][j] = tabla[(i - 1) % 2][j];
            else tabla[i % 2][j] = tabla[(i - 1) % 2][j] || tabla[(i - 1) % 2][j - longitudes[i - 1]];
        }
        if (tabla[i % 2][tamCuerda]) return true;
    }

    return tabla[numCuerdas % 2][tamCuerda];
}
```

Siendo N el número de cuerdas y R el tamaño de la cuerda a conseguir:

COMPLEJIDAD TEMPORAL	COMPLEJIDAD ESPACIAL
O(N*R)	O(R)

Al tener dos bucles anidados, uno que recorre N cuerdas y otro que recorre los R tamaños, habrá que recorrer el vector de tamaño R, N veces. Además, guardamos en memoria la matriz de dimensión 2*R.

Versión 3 :

Podemos reducir aún más la complejidad espacial de manera que sólo se usará un array de R elementos, pero para ello, tenemos que rellenar el vector de derecha a izquierda para no sobreescribir los subproblemas que necesitaremos más adelante.

Para esto, el array se dividirá en 2 partes:

- La parte derecha a j serán los subproblemas resueltos hasta la etapa i.
- La parte izquierda a j (incluida) serán los subproblemas resueltos hasta la etapa i - 1.

De este modo, cuando queramos calcular el subproblema tabla[j], siempre tendremos los valores que dijimos que eran necesarios [i - 1][j] y [i - 1][j - Li] por lo que podremos acceder a estos valores sin que hayan sido previamente cambiados.


```
bool esPosible(int longitudes[], int numCuerdas, int tamCuerda) {

    bool tabla[MAX + 1];

    //Si el tamaño de la cuerda es 0 se puede conseguir sin coger
    //ninguna cuerda, por tanto, es posible.
    tabla[0] = true;

    //Si el tamaño de la cuerda es distinto de 0 y no podemos
    //usar ninguna cuerda no se puede conseguir.
    for (int i = 1; i <= tamCuerda; i++) tabla[i] = false;

    //Rellenamos los datos
    for (int i = 0; i < numCuerdas; i++) {
        for (int j = tamCuerda; j >= 1; j--) {
            if (j < longitudes[i]) tabla[j] = tabla[j];
            else tabla[j] = tabla[j] || tabla[j - longitudes[i]];
            if (tabla[tamCuerda]) return true;
        }
    }

    return tabla[tamCuerda];
}
```

Siendo N el número de cuerdas y R el tamaño de la cuerda a conseguir:

COMPLEJIDAD TEMPORAL	COMPLEJIDAD ESPACIAL
O(N*R)	O(R)

Al tener dos bucles anidados, uno que recorre N cuerdas y otro que recorre los R tamaños, habrá que recorrer el vector de tamaño R, N veces. Además, guardamos en memoria el array de R posiciones.

Función 2 : Cuántas formas de construir la cuerda.

Este problema se podría hacer de las tres mismas maneras que para la función anterior. Se puede reducir el uso de memoria exactamente igual que antes, por lo que no volveremos a explicarlo.

- Cada posición del array representa un tamaño de cuerda objetivo. Tendremos tantas posiciones como indique la cuerda objetivo + 1 (cuerda de longitud 0).

Así, el elemento tabla[j] contendrá el número de maneras de conseguir una cuerda de tamaño j teniendo en cuenta las i primeras cuerdas.

Sabemos que este problema cumple el principio de optimalidad de Bellman, por lo que para resolver el problema final de manera óptima basta con considerar las sub soluciones óptimas.

En nuestro caso, cada vez que aumentamos en 1 la i , pasamos a tener en cuenta una cuerda más, por lo que estaremos ante un subproblema mayor. De igual modo, cuando la j crece en 1, el objetivo es una cuerda más grande, por lo que también complica el subproblema. Es por ello que iremos aumentando la i hasta alcanzar el valor de N rellenando el vector tabla[] de derecha a izquierda para poder reducir la memoria.

A la hora de rellenar el vector tabla[], longitudes[i - 1] nos indicará la cuerda que estamos considerando en ese momento y la j el tamaño de la cuerda objetivo. Por tanto, nos podemos encontrar en dos situaciones:

- longitudes[i - 1] > j : Esto significa que la cuerda que estamos considerando ahora mismo es mayor a la cuerda objetivo, por lo que no podemos usar este trozo de cuerda y no modificaremos el número de maneras posibles.

- longitudes[i - 1] <= j : Esto significa que la cuerda que estamos considerando ahora mismo es menor o igual que la que tenemos que construir y por tanto se puede utilizar para conseguir la cuerda objetivo. En este caso, tenemos que sumar las maneras posibles que teníamos de construir la cuerda de tamaño j sin tener en cuenta la cuerda i-ésima (tabla[j]) más las formas que teníamos de construir la cuerda de tamaño ji menos el tamaño de la cuerda actual, pues al añadirle este trozo, alcanzaremos la cuerda de tamaño j (tabla[j - Li]).

Por tanto, cuando la i alcance el valor N, es decir, que hemos tenido en cuenta todas las cuerdas, habremos solucionado todos los subproblemas de nuestro problema global, incluyendolo. De modo que podremos encontrar la solución en la última posición del vector (tabla[R]).

```
long long int cuantasFormas(int longitudes[], int numCuerdas, int tamCuerda) {

    long long int tabla[MAX + 1];

    //Si el tamaño de la cuerda es 0 se puede conseguir de una sola forma,
    //no escoger ninguna.
    tabla[0] = 1;

    //Si el tamaño de la cuerda es distinto de 0 y no podemos usar ninguna
    //cuerda no se puede conseguir.
    for (int i = 1; i <= tamCuerda; i++) tabla[i] = 0;

    //Rellenamos los datos
    for (int i = 0; i < numCuerdas; i++) {
        for (int j = tamCuerda; j >= 1; j--) {
            if (j < longitudes[i]) tabla[j] = tabla[j];
            else tabla[j] = tabla[j] + tabla[j - longitudes[i]];
        }
    }

    return tabla[tamCuerda];
}
```

Siendo N el número de cuerdas y R el tamaño de la cuerda a conseguir:

COMPLEJIDAD TEMPORAL	COMPLEJIDAD ESPACIAL
O(N*R)	O(R)

Al tener dos bucles anidados, uno que recorre N cuerdas y otro que recorre los R tamaños, habrá que recorrer el vector de tamaño R, N veces. Además, guardamos en memoria el array de R posiciones.

Función 3 : Mínima cantidad de cuerdas para construir la cuerda.

Esta función es muy similar a la anterior, la diferencia es que como estamos ante un problema de optimización y queremos minimizar la cantidad de cuerdas, aquellas que no pueden ser construidas toman un valor de infinito (caso base) que nos será útil para las posteriores comparaciones con la función min.

Al igual que en los casos anteriores, rellenamos el vector de derecha a izquierda y nos encontramos con dos casos:

- longitudes[i - 1] > j : Esto significa que la cuerda que estamos considerando ahora mismo es mayor a la cuerda objetivo, por lo que no podemos usar este trozo de cuerda y no modificaremos el mínimo número de cuerdas necesarias.

- longitudes[i - 1] <= j : Esto significa que la cuerda que estamos considerando ahora mismo es menor o igual que la que tenemos que construir y por tanto se puede utilizar para conseguir la cuerda objetivo. En este caso, el subproblema tendrá como resultado el valor mínimo entre: el número mínimo de cuerdas que eran necesarias para construir la cuerda de tamaño j sin tener en cuenta la cuerda i-ésima y el mínimo número de cuerdas que hacían falta para construir una cuerda de tamaño j - Li más uno (la cuerda actual que estamos añadiendo).

```
long long int minCuerdas(int longitudes[], int numCuerdas, int tamCuerda) {

    long long int tabla[MAX + 1];

    //Si el tamaño de la cuerda es 0 se puede conseguir sin escoger ninguna,
    //0 cuerdas.
    tabla[0] = 0;

    //Si el tamaño de la cuerda es distinto de 0 y no podemos usar ninguna
    //cuerda no se puede conseguir.
    for (int i = 1; i <= tamCuerda; i++) tabla[i] = INF;

    //Rellenamos los datos
    for (int i = 0; i < numCuerdas; i++) {
        for (int j = tamCuerda; j >= 1; j--) {
            if (j < longitudes[i]) tabla[j] = tabla[j];
            else tabla[j] = min(tabla[j], tabla[j - longitudes[i]] + 1);
        }
    }

    return tabla[tamCuerda];
}
```

Siendo N el número de cuerdas y R el tamaño de la cuerda a conseguir:

COMPLEJIDAD TEMPORAL	COMPLEJIDAD ESPACIAL
O(N*R)	O(R)

Al tener dos bucles anidados, uno que recorre N cuerdas y otro que recorre los R tamaños, habrá que recorrer el vector de tamaño R, N veces. Además, guardamos en memoria el array de R posiciones.

Función 4 : Mínimo coste para construir la cuerda.

Esta función es muy similar a la anterior,estamos ante un problema de optimización y queremos minimizar el coste de la cuerda,, aquellas que no pueden ser construidas toman un valor de infinito (caso base) que nos será útil para las posteriores comparaciones con la función min.

Al igual que en los casos anteriores, rellenamos el vector de derecha a izquierda y nos encontramos con dos casos:

- longitudes[i - 1] > j : Esto significa que la cuerda que estamos considerando ahora mismo es mayor a la cuerda objetivo, por lo que no podemos usar este trozo de cuerda y no modificaremos el coste mínimo necesario.

- longitudes[i - 1] <= j : Esto significa que la cuerda que estamos considerando ahora mismo es menor o igual que la que tenemos que construir y por tanto se puede utilizar para conseguir la cuerda objetivo. En este caso, el subproblema tendrá como resultado el mínimo entre: el coste mínimo que era necesarias para construir la cuerda de tamaño j sin tener en cuenta la cuerda i-ésima y el coste mínimo que hacía falta para construir una cuerda de tamaño j - Li más Ci (coste de la cuerda actual que estamos añadiendo).

```
long long int minCoste(int longitudes[], int valores[], int numCuerdas, int tamCuerda) {

    long long int tabla[MAX + 1];

    //Si el tamaño de la cuerda es 0 se puede conseguir sin escoger ninguna a coste 0.
    tabla[0] = 0;

    //Si el tamaño de la cuerda es distinto de 0 y no podemos usar ninguna cuerda
    //no se puede conseguir y por tanto el coste es infinito.
    for (int i = 1; i <= tamCuerda; i++) tabla[i] = INF;

    //Rellenamos los datos
    for (int i = 0; i < numCuerdas; i++) {
        for (int j = tamCuerda; j >= 1; j--) {
            if (j < longitudes[i]) tabla[j] = tabla[j];
            else tabla[j] = min(tabla[j], tabla[j - longitudes[i]] + valores[i]);
        }
    }

    return tabla[tamCuerda];
}
```

Siendo N el número de cuerdas y R el tamaño de la cuerda a conseguir:

COMPLEJIDAD TEMPORAL	COMPLEJIDAD ESPACIAL
O(N*R)	O(R)

Al tener dos bucles anidados, uno que recorre N cuerdas y otro que recorre los R tamaños, habrá que recorrer el vector de tamaño R, N veces. Además, guardamos en memoria el array de R posiciones.

Utilizad el esquema de resolución de problemas del juez con casos ilimitados, recordad que están disponibles en el Campus Virtual. Los casos de prueba del juez son exclusivamente los del enunciado pero posteriormente añadiré el resto de casos para reevaluar el ejercicio.

Número de envío: s31760