UNIVERSIDAD DE MÁLAGA

Análisis y Diseño de Algoritmos
Parcial 3 (Octubre 2025)

Nombre: **Grupo:**

1.(10 puntos) Parte teórica.

Una tienda de juguetes ofrece a sus clientes un bono de compra por valor de ***B*** euros. La tienda tiene ***n*** tipos de juguetes distintos, cada uno con un precio específico p_i y una cantidad limitada en stock q_i. Los clientes pueden elegir cualquier combinación de juguetes, incluso repitiendo juguetes del mismo tipo, siempre que:

- El **precio total** de los juguetes seleccionados **no exceda** el valor del bono ***B***.
- No se exceda la **cantidad disponible** de cada juguete.

Nuestro objetivo es diseñar un algoritmo de programación dinámica que **determine la combinación de juguetes** que se puede comprar con el bono ***B***, de tal manera que se minimice la cantidad de ***B*** que quedaría sin usar, es decir, que **maximice el precio total de los objetos comprados**.

Ejemplo:
Si *B = 15*, *p = [3,2,8]* y *q = [4,0,3]*, el precio total de objetos que podemos comprar para aprovechar el bono al máximo sería *14, que* se conseguiría comprando *2* objetos con precio *3* y *1* de valor *8*.

a) (6 ptos.) Definir la función recursiva que calcula el precio total de la combinación de juguetes que aprovecha mejor el bono de compra, indicando claramente el significado de cada uno de sus parámetros de entrada.
b) (3 ptos.) Describir la estructura de datos auxiliar necesaria, indicando claramente dónde estarían los casos base, la celda solución del problema y el sentido en que se rellenaría si se empleara el enfoque Bottom-up.
c) (1 pto.) Indicar el coste espacial que tendría el algoritmo.

La **solución** al problema será una lista $S = [s_0, s_1, \ldots, s_{n-1}]$, donde s_i indica la cantidad de objetos de tipo *i* que hemos comprado.

La **función objetivo** calculará el precio total de los objetos seleccionados, $f(S) = \sum_{i=0}^{n-1} s_i \cdot p_i$. Queremos maximizar dicho valor.

Secuencia de decisiones: vamos a ir decidiendo (de derecha a izquierda sobre la lista de precios de los objetos), la cantidad de objetos de tipo *i* que vamos incluyendo.

En la **primera decisión** de nuestra secuencia se considerará el objeto de tipo *n-1*. Las opciones sobre qué hacer con los juguetes de este tipo son:

- Incluir 0 unidades. En este caso el precio total será 0 + el precio total de la combinación de objetos restantes (0 a n-2) que pueda comprar con B.

- Incluir 1 unidad. En este caso el precio total será p_{n-1} + el precio total de la combinación de objetos restantes (0 a n-2) que pueda comprar con B - p_{n-1}.

-Incluir 2 unidades. En este caso el precio total será $2 \cdot p_{n-1}$ + el precio total de la combinación de objetos restantes (0 a n-2) que pueda comprar con $B - 2 \cdot p_{n-1}$.

….

-Incluir el número máximo de unidades que se pueda. Este número está limitado por dos datos. Por un lado, el número de unidades que hay en stock de ese tipo de juguete: no podemos superar q_{n-1}. Por otro lado, el valor del bono *B* también es un límite, ya que no podemos comprar objetos cuyo precio acumulado sea mayor que el del bono. Es decir, el número máximo de objetos de tipo *n-1* que se pueden comprar será B/p_{n-1}.

Por ello, en este último caso el precio total será $\min \{B/p_{n-1}, q_{n-1}\} \cdot p_{n-1}$ + el precio total de la combinación de objetos restantes (0 a n-2) que pueda comprar con $B - \min \{B/p_{n-1}, q_{n-1}\} \cdot p_{n-1}$.

Caracterización de los subproblemas: *P(i,j)* va a ser el problema de encontrar una combinación de juguetes de tipo *0* a *i* que se puedan pagar con la cantidad *j*.

Para los datos de entrada, estaremos interesados en resolver P(n-1, B).

Definición de la ecuación recursiva que resuelve el problema de forma óptima (**ecuación de Bellman**):

Bono(i,j) = precio total de la combinación de juguetes de tipo *0* a *i* que aproveche de la mejor manera la cantidad *j* (cuyo precio total sea lo más cercano posible a j sin pasarse).

$$Bono(i,j) = \begin{cases} 0 & j = 0 \\ \min\{q_i, \frac{j}{p_i}\} & j > 0, i = 0 \\ \max\limits_{0 \le k \le \min\{q_i, \frac{j}{p_i}\}} \{k \cdot p_i + Bono(i-1, j - k \cdot p_i)\} & j > 0, i > 0 \end{cases}$$

Estructura de datos auxiliar: Para resolver el problema *P(n-1,B)* vamos a necesitar una tabla de *n* filas (de *0* a *n-1* tipos de juguetes) y *B+1* columnas. La celda que resuelve el problema será (n-1, B), que es la celda de la esquina inferior derecha. Los casos base están en la primera columna (j= 0) y en la primera fila (i= 0). Para rellenar una celda *(i,j)*, necesitaremos que las celdas *(i-1, j- k·p_i)* ya estén calculadas. Es decir, celdas en la fila anterior y en columnas que están desde j hasta 0. Por ello, rellenaremos la tabla de arriba abajo y de izquierda a derecha.

Análisis de la complejidad: El coste espacial viene determinado por las variables extra que se usarán en la implementación (O(1)) y por la tabla auxiliar que utilizamos, que tiene un número O(n) de filas y un número O(B) de columnas. Por tanto, $E(n, B) \in O(n \cdot B)$.

UNIVERSIDAD DE MÁLAGA

2. (*10 puntos*) Parte práctica.

Queremos realizar una aplicación que nos ayude a preparar carreras campo a través (cross running). Para ello, disponemos de una matriz (mapa) ***M*** de ***n*** filas y ***m*** columnas, que indica en cada tramo de carrera el coste energético de cruzar dicho tramo (véase dibujo). Hay tramos que es imposible cruzar y están marcados con ∞ en el mapa. La salida siempre está en un tramo de la primera columna ***(xs,0)*** y la meta en uno de la última columna ***(xf, m-1)***. Suponiendo que los posibles movimientos del corredor son ↗, → ó ↘, nuestro objetivo es calcular el gasto energético mínimo para llegar a la meta.

M =

1	1	∞	1	1
1	1	∞	2	2
2	4	5	5	1
3	3	4	∞	∞
2	3	2	7	8

Salida = (0,0) Meta = (4,4)

Tras enfocar el algoritmo mediante Programación Dinámica hemos obtenido la siguiente definición de la función ***C(i,j),*** que encuentra el gasto energético mínimo para ir de la casilla ***(i,j)*** a la casilla de meta ***(xf,yf)***.

$$C(i,j) = \begin{cases} m_{ij} & j = m-1 \wedge i = xf \\ \infty & j = m-1 \wedge i \neq xf \\ m_{ij} + \min\{C(i,j+1), C(i+1,j+1)\} & j < m-1 \wedge i = 0 \\ m_{ij} + \min\{C(i-1,j+1), C(i,j+1)\} & j < m-1 \wedge i = n-1 \\ m_{ij} + \min\{C(i-1,j+1), C(i,j+1), C(i+1,j+1)\} & j < m-1 \wedge 0 < i < n-1 \end{cases}$$

Implementar un algoritmo de programación dinámica **Bottom-Up** que se base en la ecuación anterior para calcular el gasto energético mínimo.

Una posible implementación sería la siguiente:

```
//Precondición: (filaInicio, colInicio) y (filaMeta,colMeta) son
posiciones correctas del mapa M. Es
    //decir, 0<= filaInicio < m, 0<=colInicio<n, 0<=filaMeta<m,
0<=colMeta<n
     public static int gastoEnergia(int[][] M, int filaInicio, int
colInicio, int filaMeta, int colMeta) {
        int n = M.length;
        int m = M[0].length;

        int[][] gasto = new int[n][m];
        for(int j = m-1; j >=0; j--) {//de derecha a izquierda
           for(int i = 0; i< n; i++) {//de arriba a abajo
                if (j==m-1 && filaMeta == i) {
                     gasto[i][j] = M[i][j];
                }else if (j == m-1) {//No hay solución
                     gasto[i][j] = INF;
                }else if (i == 0) {//j < m-1
```

```
                    gasto[i][j] = sumaInf(M[i][j],
                                min2Inf(gasto[i][j+1], gasto[i+1][j+1]));
               }else if (i == n-1) {
                    gasto[i][j] = sumaInf(M[i][j],
                                min2Inf(gasto[i-1][j+1], gasto[i][j+1]));
               }else {
                    gasto[i][j] = sumaInf(M[i][j],
              min3Inf(gasto[i-1][j+1], gasto[i][j+1], gasto[i+1][j+1]));
               }//if
          }//for i
        }//for j
      return gasto[filaInicio][colInicio];
}
```