

PARTE 1:

PROBLEMA DEL CD:

RECONSTRUCCIÓN DE LA  
SOLUCIÓN

SOLUCIÓN BOTTOM- UP

## Problema del CD.

Enunciado: Dado un CD con  $K$  minutos y un conjunto  $W$  con  $n$  canciones de duración  $w_1, \dots, w_n$ . Determinar la máxima cantidad de minutos que podemos grabar sin correr ni repetir canciones.

→ Problema de OPTIMIZACIÓN

$$cd(W_i, K) = \max \left\{ \sum_{i \in S} w_i \mid S \in \mathcal{D}_i \right\} \quad \mathcal{D}_i = \left\{ S \subseteq \{1, \dots, i\} \mid \sum_{i \in S} w_i \leq K \right\}$$

(nosotros buscamos  $cd(W, K) = cd(W_n, K)$ )

Función recursiva:

$$f_w : \mathbb{N} \times \mathbb{Z} \rightarrow \mathbb{N} \cup \{-\infty\}$$

Semántica:

$$f_w(n, k) = \text{CD}(\{w_i\}, k)$$

es el máximo tiempo grabable con las canciones  $\{w_1, \dots, w_n\}$  en un CD de capacidad  $K$ .

$$f_w(i, j) = \begin{cases} -\infty & j < 0 \\ 0 & i = 0 \text{ y } j \geq 0 \\ \max\{f_w(i-1, j), w_i + f_w(i-1, j-w_i)\} & \text{c.c.} \end{cases}$$

→ Tiene superposición de subproblemas (seguro?)

# llamadas =  $\Omega(2^m)$  (potencialmente, contempla toda selección posible)

# Subinstancias =  $O(nK)$  ( $0 \leq i \leq m; 0 \leq j \leq k$ )

Hay que ver si  $K << \frac{2^m}{n}$

Supongamos que lo es. Me monto en una tabla de  $n.K$

# Algoritmo Top-Down

cd ( $\{w_1, \dots, w_m\}, K$ ) {

Inicializo  $M$  de  $n+1 \times K+1$  com  $\perp$   
retorno  $f(n, K)$  tq

$f(i, j) \{$

  S:  $j < 0$  retorna  $-\infty$

  S:  $i = 0$  retorna 0

  S:  $M[i, j] = \perp$

$M[i, j] = \max \{ f(i-1, j), w_i + f(i-1, j - w_i) \}$

  Retornar  $M[i, j]$

}

}

El algoritmo devuelve la máxima capacidad que puedo llenar.

¿Qué pasa si quiero saber qué termas solo me aseguran esa capacidad?

Idea I: agregarlo al cálculo de la función y guardarlo

$cd(\{w_1, \dots, w_m\}, K)$

Inicializo  $M$  de  $M+1 \times K+1$  con  $(\perp, \perp)$   
retorno  $f(m, k)$  tq

$f(i, j) \left\{ \begin{array}{l} \end{array} \right.$

$\begin{array}{ll} \text{si } j < 0 & \text{retorno } \{-\infty, \emptyset\} \\ \text{si } i = 0 & \text{retorno } \{0, \emptyset\} \\ \text{si } M[i, j] = \perp & \end{array}$

$Val_1, Conj_1 = f(i-1, j); Val_2, Conj_2 = w_i + f(i-1, j-w_i)$

if  $Val_1 > Val_2$

$M[i, j] = (Val_1, Conj_1)$

else  $M[i, j] = (Val_2, Conj_2 \cup \{i\})$

retornar  $M[i, j]$

3

Problema?

Es costoso en  
espacio (y en  
tiempo, s' copia  
los conjuntos  
cada vez)

Podemos mejorarla si en cada espacio de  $M$  ponemos no el conjunto completo sino un 'hint' de si en la elección dejó el elemento o no.

Pero si ya resolvimos el problema una vez (ie.  $M$  ya está poblado)  
Podemos deducirlo!

Supongamos que ya calculamos  $f(i, j)$ .

$$\text{Si } f(i, j) = \max \{ f(i-1, j), w_i + f(i-1, j - w_i) \}$$

Podemos pedir el valor de  $f(i-1, j)$  y  $f(i-1, j - w_i)$  (que también ya están calculados) y comparar si el valor es  $f(i-1, j)$  o  $w_i + f(i-1, j - w_i)$ .

Según este uno u otro, sé que de cierto tomé.

y si son iguales... da lo mismo, ¿no? (Si, pues implica que hay una manera de obtener ese valor tanto agregando como no agregando a  $w_i$ )

$\Rightarrow$  Calculo de resultados exactamente como antes y reconstruyo al final.

$cd(\{w_1, \dots, w_n\}, k)$

Inicializo  $M$  de  $n+1 \times k+1$  con  $\perp$   
Valor =  $f(m, k)$  tq

$f(i, j) \left\{ \begin{array}{l} \end{array} \right.$

Si  $j < 0$  retorno  $-\infty$

Si  $i = 0$  retorno  $0$

Si  $M[i, j] = \perp$

$M[i, j] = \max \{ f(i-1, j), w_i + f(i-1, j-w_i) \}$

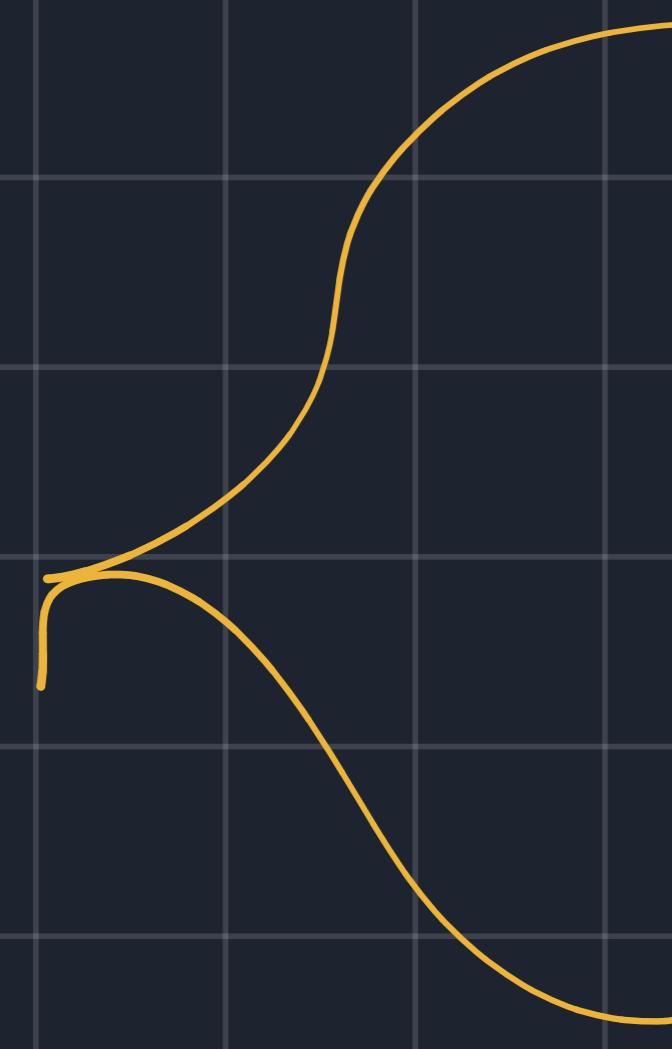
Retornar  $M[i, j]$

$\exists$

Solución = reconstruir( $m, k$ ) tq  
reconstruir( $i, j$ )  $\left\{ \begin{array}{l} \end{array} \right.$

costo:

$O(n)$



Si  $i = 0$  retorno  $[ ]$

Si  $f(i, j) == f(i-1, j-w_i) + w_i$ :

Retorno  $\text{reconstruir}(-1, j-w_i) + [i]$

Si NO Retorno  $\text{reconstruir}(i-1, j)$

Pregunta: ¿Qué pasa si queremos la lista de todas las soluciones que optimizan la función objetivo?

(Necesito ir guardando las soluciones óptimas en una estructura de datos. dependiendo del problema, podrían ser exponenciales)

Bottom-up:

Enfoque alternativo:

- Ya establecimos que para resolver  $f(m, k)$  vamos a requerir algunas soluciones de  $f(i, j)$ ,  $i \leq m$ ,  $j \leq k$ .
- Empezando desde los casos base, puedo construir todas las subinstancias de  $f$ .

EJEMPLO:  $K=5$

$$W = \begin{bmatrix} 2 & 3 \\ \omega_1 & \omega_2 \\ \cdot & \cdot \\ 2 & 1 \\ \omega_3 & \omega_4 \end{bmatrix}$$

$$f(m, k) \rightarrow f(4, 5) = \max \left( f(3, 5), 1 + f(3, 4) \right) =$$

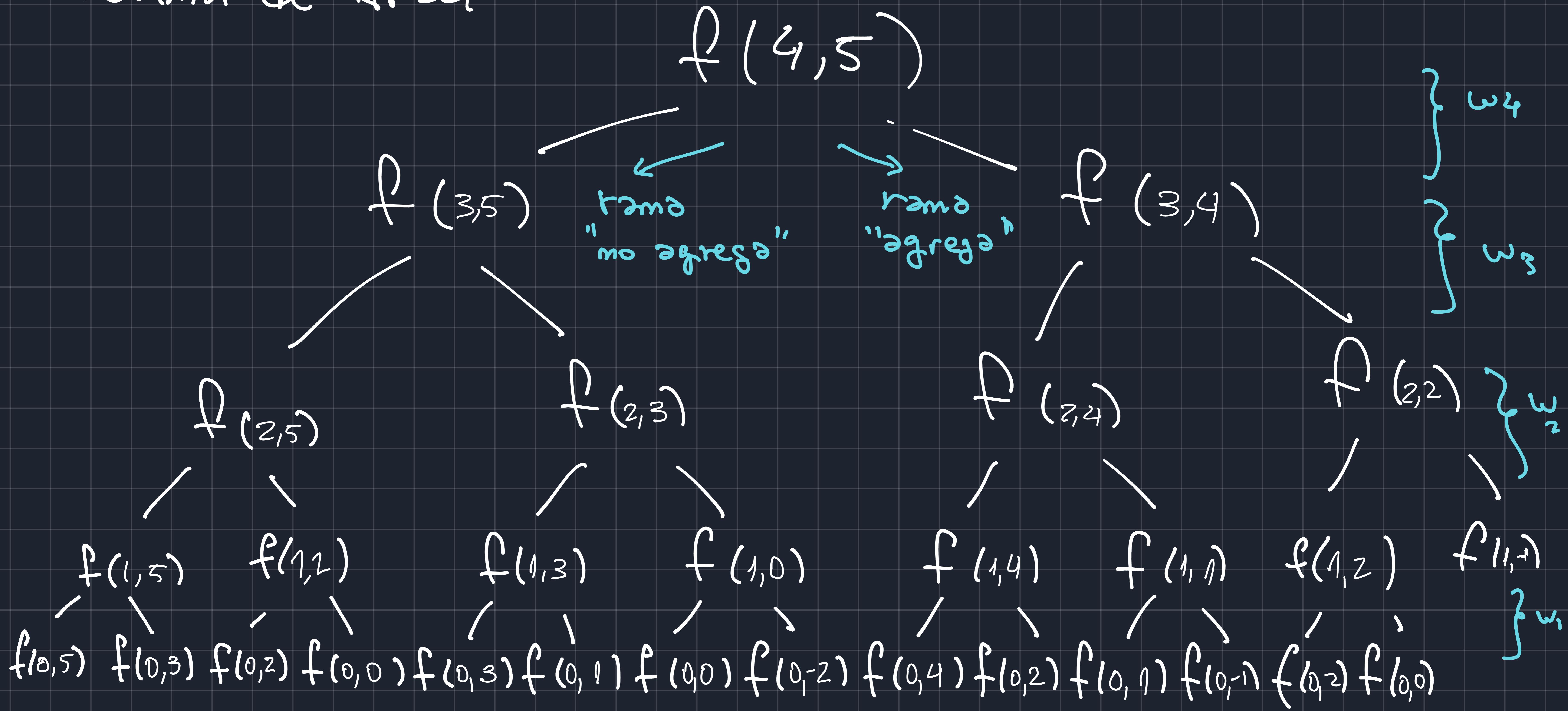
$$\max \left( \max \left( f(2, 5), 2 + f(2, 3) \right), \right.$$

$$1 + \max \left( f(2, 4), 2 + \max \left( f(2, 2) \right) \right)$$

= ...

# Forma de Árbol

decide  
sobre:



# Forma de Árbol

$$w_4 = 1$$

$$w_3 = 2$$

$$w_2 = 3$$

$$w_1 = 2$$

$$f(4,5) \overset{5}{\searrow}$$

$$f(3,5) \overset{5}{\searrow}$$

$$f(3,4) \overset{4}{\searrow}$$

$$f(2,5) \overset{5}{\searrow}$$

$$f(2,4) \overset{3}{\searrow}$$

$$f(2,2) \overset{2}{\searrow}$$

$$f(2,3) \overset{3}{\searrow}$$

$$f(1,1) \overset{0}{\searrow}$$

$$\begin{cases} f(1,2) \overset{2}{\searrow} \\ f(1,-1) \end{cases}$$

$$f(1,2) \overset{2}{\searrow}$$

$$f(1,3) \overset{2}{\searrow}$$

$$f(1,0) \overset{0}{\searrow}$$

$$f(1,4) \overset{2}{\searrow}$$

$$f(0,5) \overset{0}{\searrow}$$

$$f(0,3) \overset{0}{\searrow}$$

$$f(0,2) \overset{0}{\searrow}$$

$$f(0,0) \overset{0}{\searrow}$$

$$f(0,3) \overset{0}{\searrow}$$

$$f(0,1) \overset{0}{\searrow}$$

$$f(0,0) \overset{0}{\searrow}$$

$$f(0,1) \overset{0}{\searrow}$$

$$f(0,0) \overset{0}{\searrow}$$

$$f(0,-2) \overset{0}{\searrow}$$

$$f(0,4) \overset{0}{\searrow}$$

$$f(0,2) \overset{0}{\searrow}$$

$$f(0,1) \overset{0}{\searrow}$$

$$f(0,-1) \overset{0}{\searrow}$$

$$f(0,2) \overset{0}{\searrow}$$

$$f(0,0) \overset{0}{\searrow}$$

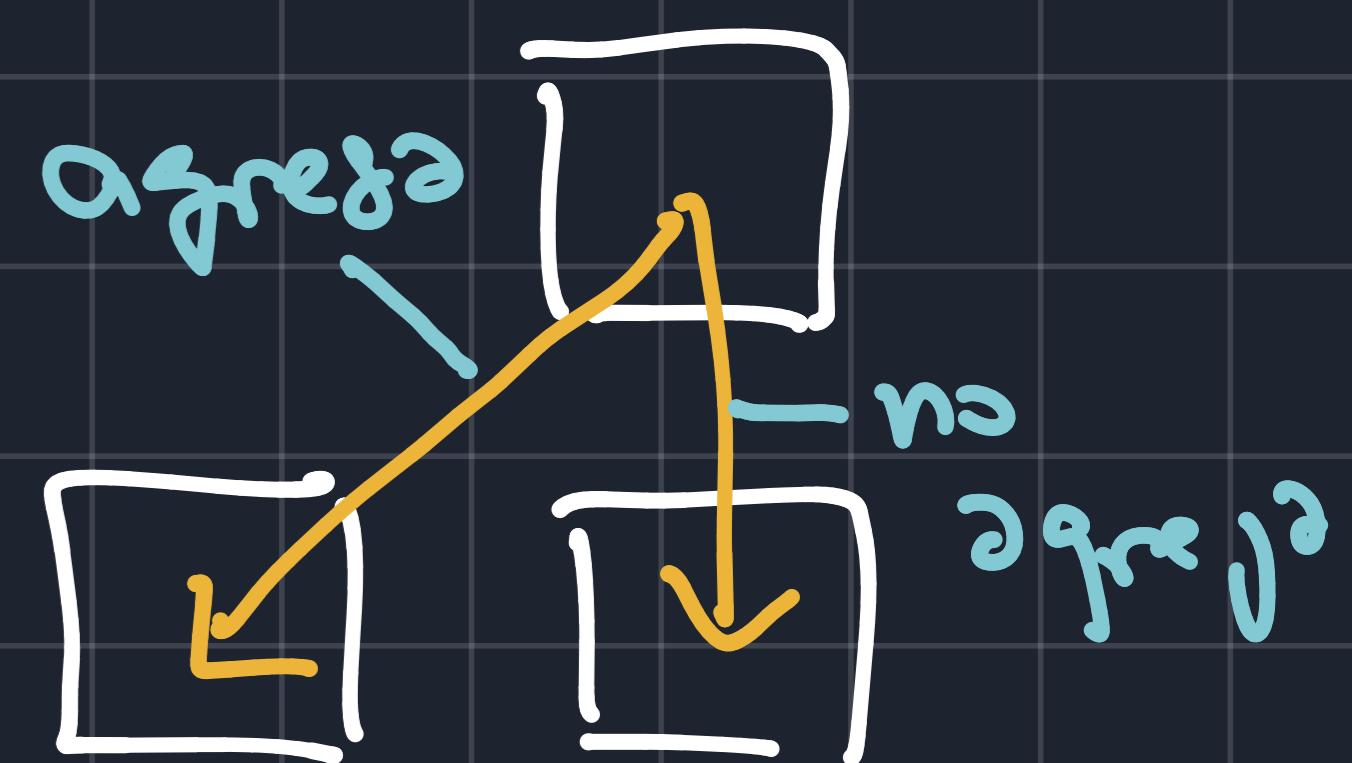
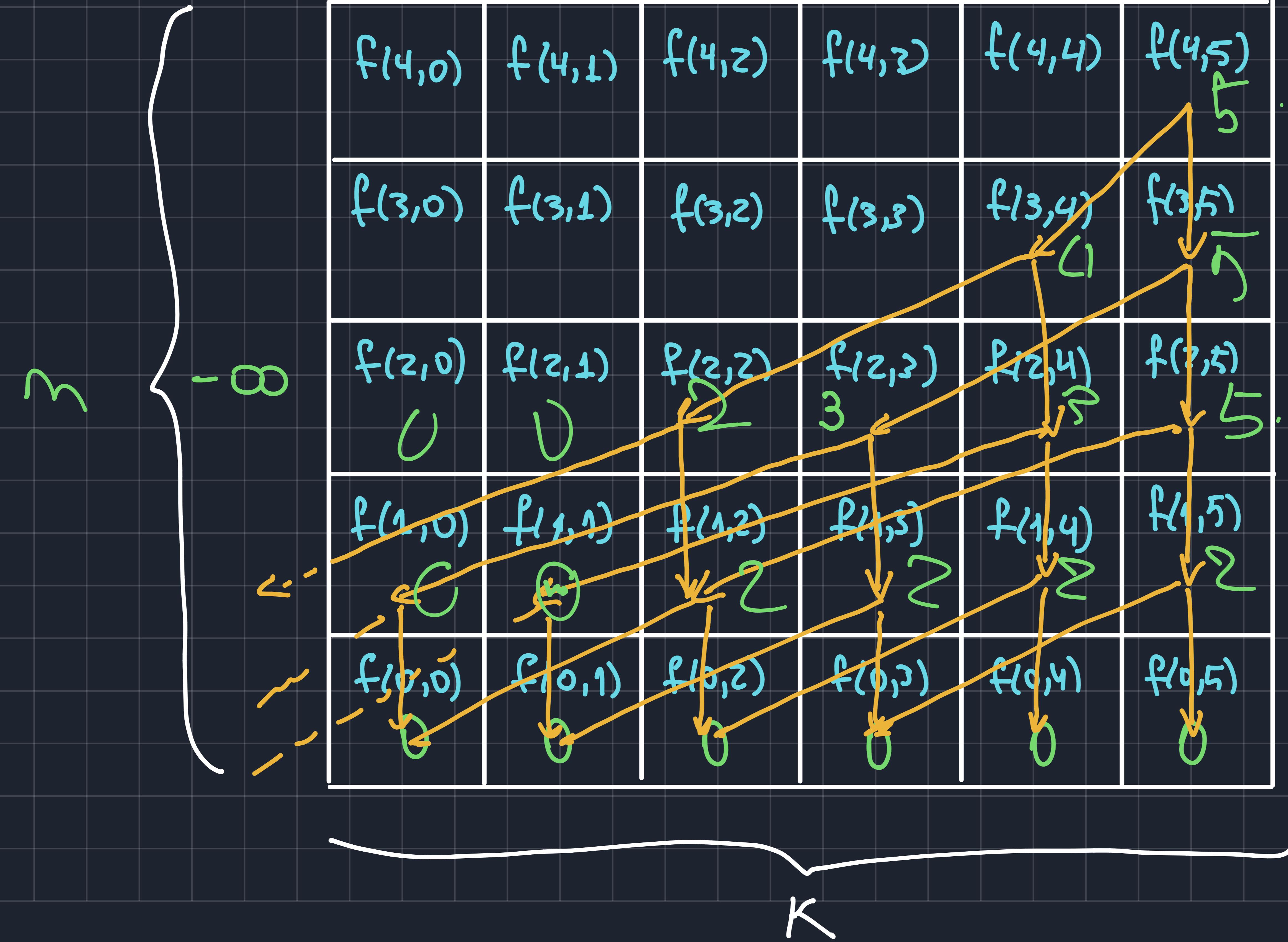
$$f(0,2) \overset{0}{\searrow}$$

$$f(0,0) \overset{0}{\searrow}$$

$$f(0,0) \overset{0}{\searrow}$$

el valor de arriba es el máximo entre el de los izquierdos y  $w_i + \text{el de la derecha}$

# Visto en la estructura de Memoria



$$w_4=1$$

$$w_3=2$$

$$w_2=3$$

$$w_1=2$$

A) termativamente, Prede ir Poblano la tabla desde la primera  $f'$  la

$i$

$-\infty$

$f(4,0)$	$f(4,1)$	$f(4,2)$	$f(4,3)$	$f(4,4)$	$f(4,5)$
0	1	2	3	4	5
$f(3,0)$	$f(3,1)$	$f(3,2)$	$f(3,3)$	$f(3,4)$	$f(3,5)$
0	0	2	2	4	5
$f(2,0)$	$f(2,1)$	$f(2,2)$	$f(2,3)$	$f(2,4)$	$f(2,5)$
0	0	2	3	3	5
$f(1,0)$	$f(1,1)$	$f(1,2)$	$f(1,3)$	$f(1,4)$	$f(1,5)$
0	0	2	2	2	2
$f(0,0)$	$f(0,1)$	$f(0,2)$	$f(0,3)$	$f(0,4)$	$f(0,5)$
0	0	0	0	0	0

$$w_4=1$$

$$w_3=2$$

$$w_2=3$$

$$w_1=2$$

Cada celda es el máximo de la que tiene inmediatamente abajo y la que esto  $w_i$  lugares a la izquierda, más  $w_i$ .

Es más, si no pienso reconstruir la solución, puedo tirar todas las filas menos la última!

$O(K)$  espacio

Todavía más, teniendo un poco de suerte puedes hacer todo en la misma fila! (aunque ya no se gana tanto)

$O(K)$  espacio

# Problema del CD (Solución Bottom-up)

CD ( $w_1, \dots, w_n$ , K)

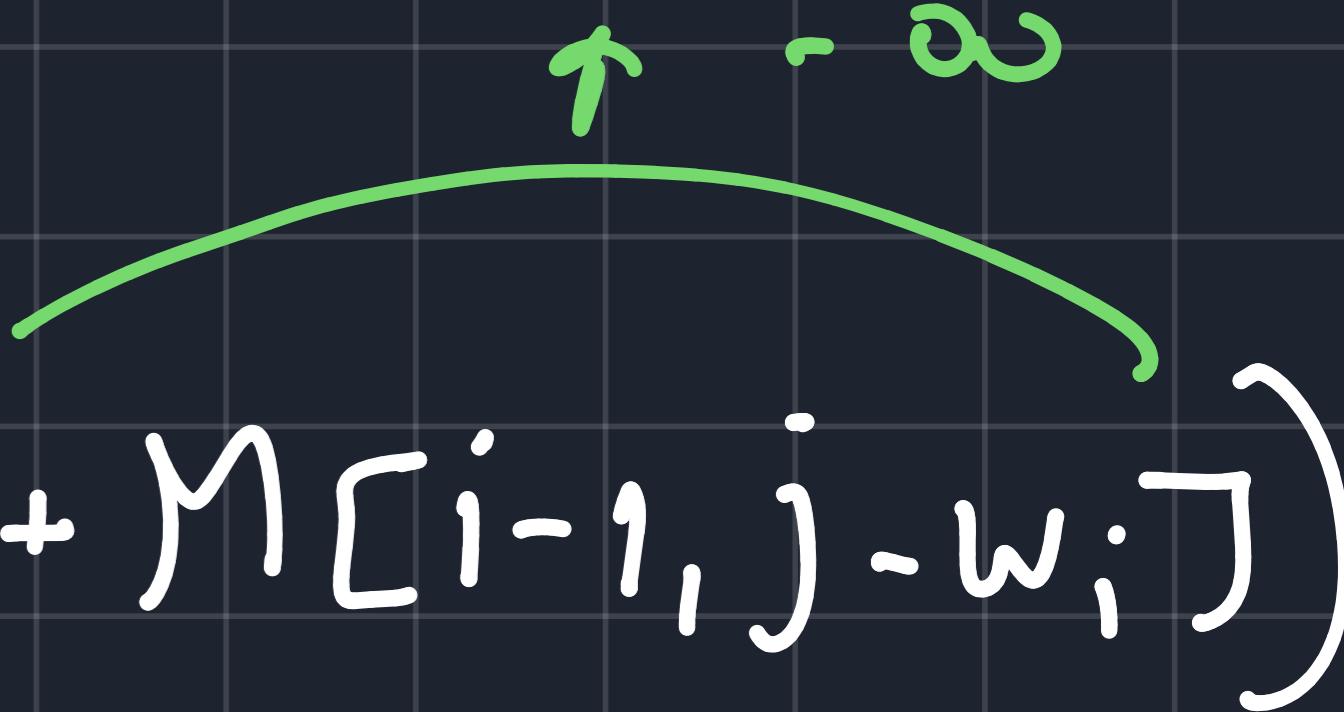
Iniciarizo M de  $M+1 \times K+1$  con 0 en la fila 0  
y  $\infty$  en el resto de las celdas

for ( $i = 1 \dots n$ )

    for ( $j = 0 \dots K$ )

$M[i, j] = \max(M[i-1, j], w_i + M[i-1, j-w_i])$

si  $j - w_i < 0$ ,



Costo:

return M[n, K]

$O(nK)$

# Ventajas y desventajas

TD

- Consiste únicamente en implementar la eliminación recursiva del problema (agregando memorización)
- Potencialmente no recorre todos los subproblemas (aunque no baje la complejidad en peor caso)

BU

- Ahorra espacio (sacrificando reconstrucción)
- Puede llegar a superar instancias más grandes si el árbol de retroceso es muy profundo

En principio, ninguno es inherentemente más eficiente.

PARTE 2:

PROBLEMA DEL FERRY

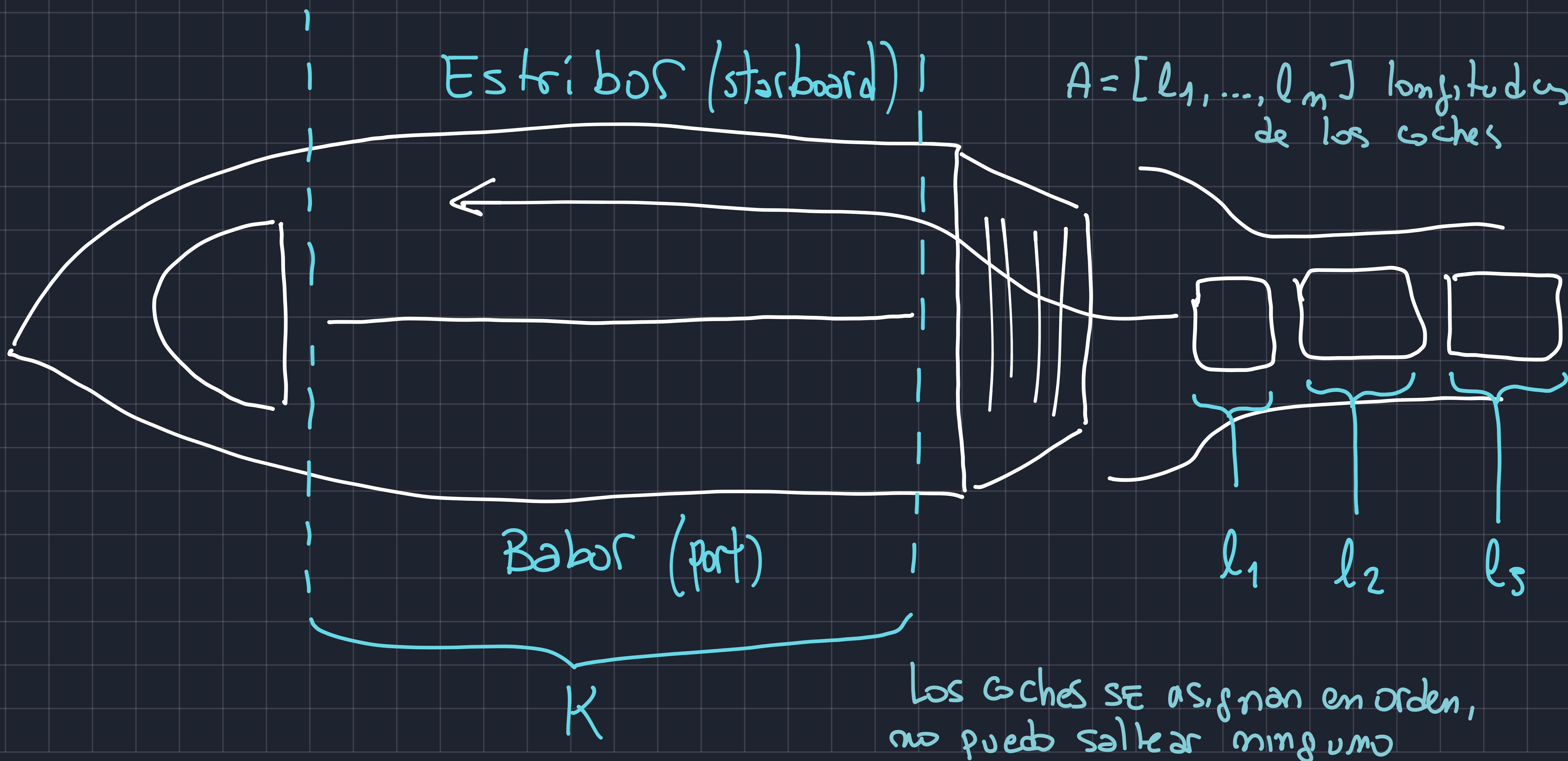
# Problema del Ferry (UVa 10261)

## 10261 Ferry Loading

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Two lanes of cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

The cars waiting to board the ferry form a single queue, and the operator directs each car in turn to drive onto the port (left) or starboard (right) lane of the ferry so as to balance the load. Each car in the queue has a different length, which the operator estimates by inspecting the queue. Based on this inspection, the operator decides which side of the ferry each car should board, and boards as many cars as possible from the queue, subject to the length limit of the ferry. Your job is to write a program that will tell the operator which car to load on which side so as to maximize the number of cars loaded.

# Problema del Ferry (UVa 10261)



# Espacio Combinatorio.

Quiero el  $i \leq m$  más grande tal que puedo cargar todos los coches hasta el  $i$ .

Problema de OPTIMIZACIÓN



$$\text{ferry}(A, K) = \max \{ i \mid F_i \neq \emptyset \}$$

$$F_i = \left\{ L \subseteq \{1, \dots, i\} \mid \sum_{j \in L} l_j \leq k \text{ y } \sum_{j \in \{1, \dots, i\} - L} l_j < k \right\}$$
$$i \leq |A|$$

"¿No basta con buscar el  $i$  tal que  $\sum_{j=1}^i l_j > 2K$ ?"  
Esa condición es necesaria pero no suficiente. Por ejemplo, si  $l_1 = 60, l_2 = 60, l_3 = 60$ ,  $K = 100$ ,  $l_1 + l_2 + l_3 = 180 < 200$ . Pero no es posible cargar los 3 coches.

Solución 1:

Solución:  $f(1, K_1, K_2)$ ,

$$f(i, R_1, R_2) = \begin{cases} -\infty & \\ 0 & \\ \max \left\{ 0, f(i+1, R_1 - l_i, R_2) + 1, \right. \\ \left. f(i+1, R_1, R_2 - l_i) + 1 \right\} & \end{cases}$$

Hago recursión sobre las bodegas que admiten autos. Si ambas están llenas (recursión de  $-\infty$ ), devuelvo 0.

SEMÁNTICA:

Con  $l_i$  dad. máx. # de autos en  $[i, M+1]$  que entran cuando las líneas tienen capacidades  $R_1, R_2$ .

Como hay m coches, el caso cuando ya no hay más coches es  $i = n+1$ .

# llamadas :  $\Omega(2^n)$  (Pólenchamente prueba todas las asignaciones)

# Sub problemas :  $O(MK^2)$  ( $1 \leq i \leq M+1$ ,  $0 \leq k_1 \leq K$ ,  $0 \leq k_2 \leq K$ )

... OK?

Se puede mejorar:  $k_1$  y  $k_2$  son interdependientes  
(exactamente todos los autos que no fueron a babor  
fueron a estribor)

Si en babor hay  $k_1$  espacio disponible, en estribor  
hay ocupado

$$\sum_{j=1}^i l_j - (K - k_1)$$

espaciado. Es decir, queda

$$q_{ik} = K - \left( \sum_{j=1}^i l_j - (K - k_1) \right) = 2K - \left( \sum_{j=1}^i (l_j + k_1) \right)$$

espacio

Llamo  $q_{ik}$  a  $k_2(i, k_1)$ , el espacio libre en estribor considerando el que hay a babor.

Versión mejorada:

$$f(i, R, \cancel{R_2}) = \begin{cases} -\infty \\ 0 \\ \max \left\{ 0, f(i+1, R - l_i, \cancel{R_2}) + 1, \right. \\ \left. f(i+1, R, \cancel{R_2 - l_i}) + 1 \right\} \end{cases}$$

$R < 0 \vee R_2 < 0$   
 $i = m+1, R_1 \geq 0, R_2 \geq 0$   
 $q_{ik} < 0$

SEMÁNTICA:

Con  $l_i$  capacidad máxima de autos en  $[i, m+1]$   
que entran cuando la fila de babor tiene  
capacidad  $R$  (pues podemos deducir la capacidad de la  
línea de estribor)

la cantidad de estados pasa de  $\Theta(nk^2)$  a  $\Theta(nk)$

Nota: Si calculo  $q_{ik}$  paso a paso, incremento el costo de la ejecución. Sin embargo, como en los casos que vimos en la primera clase, podemos obtener  $q_{ik}$  en  $O(1)$ . Si precalculamos un vector  $Q$  tal que  $Q[i] = \sum_{j=1}^i l_j$ .

Con esta definición de  $f$ , ya contamos con una solución para el problema (intentalo implementar).

Solución Alternativa:

Este Problema se Puede resolver  
como un caso de CD.

CD

$$cd(w_i, k) = \max \left\{ \sum_{i \in S} w_i \mid S \in D_i \right\}$$

$$D_i = \left\{ S \subseteq \{1, \dots, i\} \mid \sum_{i \in S} w_i \leq k \right\}$$

Ferry

$$\text{Ferry}(A, K) = \max \{ i \mid F_i \neq \emptyset \}$$

$$F_i = \left\{ L \subseteq \{1, \dots, i\} \mid \sum_{i \in L} l_i \leq k \right\}$$

$$\left. \begin{array}{l} \sum_{i \in \{1, \dots, i\} - L} l_i \leq k \\ \end{array} \right\}$$

Intuitivamente:

① Es claro que si  $\sum_{j=1}^i l_j > 2K$ , no puedo meter i coches en el Ferry.

② Si no es el caso, algo que puedo hacer es elegir quién coches poner a babor resolviendo el problema del CD. Si los coches que corresponden a temas que no puse en el CD tienen una longitud acumulada menor o igual a  $K$ , los mando todos a estribor y resolví el problema.

③ Si los coches correspondientes a Temas que no van en el CD tienen una longitud acumulada mayor a  $K$ , entonces para ese valor de i no hay asignación posible que los haga entrar todos al Ferry.

es decir, entran i coches en el Ferry si:

$$\sum_{j=1}^i l_j - cd(A_i, K) \leq K$$

AFIRMACIÓN :

$$\text{Sea } i \in \{1, \dots, n\}. F_i \neq \emptyset \Leftrightarrow \sum_{j=1}^i l_j - cd(A_i, k) \leq k$$

es más, si  $L \in D_i$  es tal que  $\sum_{j \in L} l_j = cd(A_i, k) \Rightarrow L \in F_i$ .

(es decir, si vale la condición, no sólo afirma que el problema del ferry tiene solución, sino que puede decir cuál es)

Demonstración :

$\Rightarrow$  Sea  $L \in F_i$ , i.e.  $L$  es un conjunto de coches que se puede poner a la barba de forma tal que el resto van a estribor y entran. Por definición,  $L \in D_i$ , así que

$$\sum_{j \in L} l_j \leq cd(A_i, k)$$

También por definición,

$$\sum_{j=1}^i l_j - \sum_{j \in L} l_j \leq k \Rightarrow \sum_{j=1}^i l_j - cd(A_i, k) \leq k$$



$\Leftarrow$  Si  $L \in D_i$  es tal que  $\sum_{j \in L} l_j = cd(A_i, k)$ , por  
definición  $\sum_{j \in L} l_j \leq k$  y por hipótesis

$$\sum_{j=1}^i l_j - cd(A_i, k) \leq k$$

$$\sum_{j=1}^i l_j - \sum_{j \in L} l_j \leq k \Rightarrow L \in F_i$$

■

Si quiero encontrar  $L \in \mathbb{F}_x$  con  $x = \max\{i \mid i \in \{1 \dots n\}, F_i \neq \emptyset\}$

Puedo hacer el siguiente algoritmo:

input:  $A = \{l_1, \dots, l_n\}$ ,  $K \in \mathbb{N}$ .

Preproceso:  $Q[0] = 0$ ,  $Q[i] = Q[i-1] + l_i \quad \forall i \in \{1 \dots n\}$

calculo  $cd(m, k)$

$X = \max\{i \mid i \in \{0, \dots, n\}, Q[i] - cd(A_i, K) \leq k\}$

reformar  $L \in \mathbb{D}_x$  tal que  $\sum_{j \in L} l_j = cd(A_i, K)$

Complejidad:

$O(nK)$

calcular el cd.