

PROBLEMA DEL CAMBIO

SOLUCIÓN CON PROGRAMACIÓN DINÁMICA AL PROBLEMA DEL CAMBIO DE MONEDAS TENEMOS UNA CANTIDAD ILIMITADA DE MONEDAS DE DIFERENTES VALORES Y QUEREMOS PAGAR DE FORMA EXACTA UNA CANTIDAD UTILIZANDO EL MENOR NÚMERO DE MONEDAS



UNIVERSIDAD
COMPLUTENSE
MADRID

ALBERTO VERDEJO

Problema del cambio de monedas

- ▶ Conjunto finito $M = \{ m_1, m_2, \dots, m_n \}$ de **tipos** de monedas, donde cada m_i es un número natural.
- ▶ Existe una **cantidad ilimitada** de monedas de cada **valor**.
- ▶ Se quiere pagar una cantidad $C > 0$ utilizando el **menor** número posible de monedas.

Moneda de 20, 50 , 1 euro...



Para algunos sistemas monetarios existe una estrategia voraz que resuelve de forma óptima el problema. Pero la estrategia voraz no funciona siempre para cualquier sistema monetario

Problema del cambio de monedas

- ▶ Al no funcionar una estrategia voraz, tenemos que considerar diferentes alternativas hasta encontrar la mejor.

Cuantas menos considere mejor, siempre que no perdamos la mejor solución.

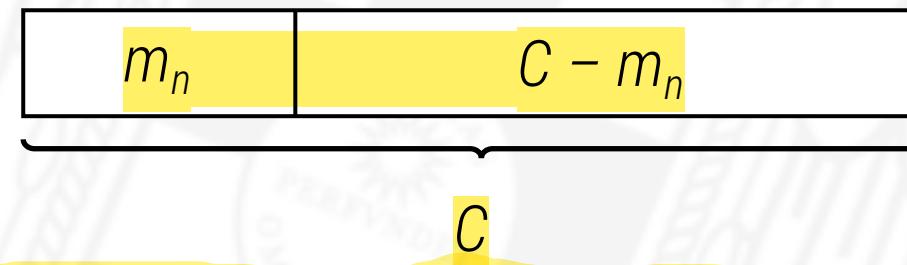
- ▶ Las soluciones son multiconjuntos de monedas.

Pensar que el orden en que se cogen las monedas es indiferente.

- ▶ Podemos fijar el orden en el que vamos considerando los tipos de monedas, sin que eso afecte al resultado final.

- ▶ ¿Qué hacemos con las monedas de tipo n ?

Si las monedas de tipo n superan a C , entonces podemos descartarlas y pasar al siguiente tipo de moneda.



Utilizamos la moneda m_n para pagar C , nos quedará por pagar $C - m_n$

Problema del cambio de monedas

$\text{monedas}(i, j)$

=

número *mínimo* de monedas para pagar la cantidad j
considerando los tipos de monedas del 1 al i

Con programación dinámica hallamos el valor de la solución óptima, y no la solucción óptima en si.

- **Principio de optimalidad de Bellman:** para conseguir una solución óptima basta con considerar subsoluciones óptimas.

Siempre tenemos que coger la mejor solución para los subproblemas.

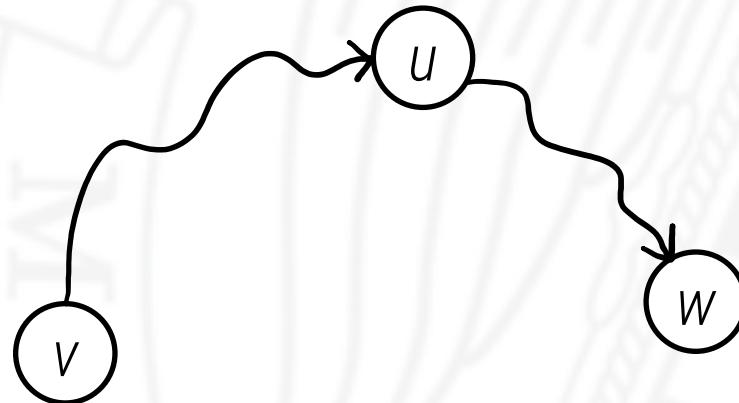
no se cumple siempre

Principio de optimalidad de Bellman

NO SE CUMPLE PARA TODOS LOS PROBLEMAS, ES MUY IMPORTANTE SABER SI SE CUMPLE O NO PARA TODOS

- El principio de optimalidad de Bellman se cumple en un problema si una solución óptima a una instancia del problema siempre contiene soluciones óptimas a todas sus subinstancias.

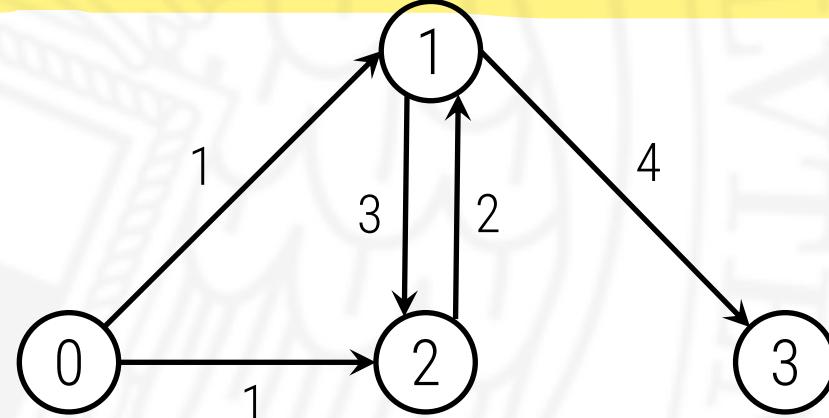
Camino mínimo



Aquí si se cumpliría el principio de Bellman.

Camino simple más largo

No se cumple porque el camino más largo es $0 \rightarrow 2 \rightarrow 1 \rightarrow 3$, pero el camino más largo a 2 es $0 \rightarrow 1 \rightarrow 2$, y no $0 \rightarrow 2$ por tanto NO SE CUMPLE para los subproblemas.



Aquí NO se cumple el principio de optimalidad de Bellman.

Definición recursiva

Lo primero, siempre lo mejor es ANTES DE HACER LA TABLA. DEFINIR LA RECURSIÓN.

► Casos recursivos:

Función recursivo compuesto por dos casos. Si nos queda algo por pagar j y nos quedan algunos tipos de monedas, o se pueden coger monedas de tipo i o no.

$$\text{monedas}(i,j) = \begin{cases} \text{monedas}(i-1,j) & \text{Si no podemos, tenemos que pagar con el resto de monedas} \\ \min(\text{monedas}(i-1,j), \text{monedas}(i,j-m_i) + 1) & \text{si } m_i \leq j \end{cases}$$

donde $1 \leq i \leq n$ y $1 \leq j \leq C$

Nos queda cantidad j por pagar y lo tenemos que hacer con las monedas del 1 al $i-1$. En el segundo caso nos falta por pagar $j - m_i$, lo hacemos de forma óptima con los tipos del 1 al i . De las dos opciones nos quedamos con la que use menos monedas.

► Casos básicos:

$$\text{monedas}(i,0) = 0 \quad 0 \leq i \leq n$$

El precio que nos queda por pagar es 0

$$\text{monedas}(0,j) = +\infty \quad 1 \leq j \leq C$$

Caso en el que nos quedemos sin poder utilizar ningún tipo de moneda y aún nos quede por pagar alguna cantidad mayor que 0. En ese caso no hay solución.

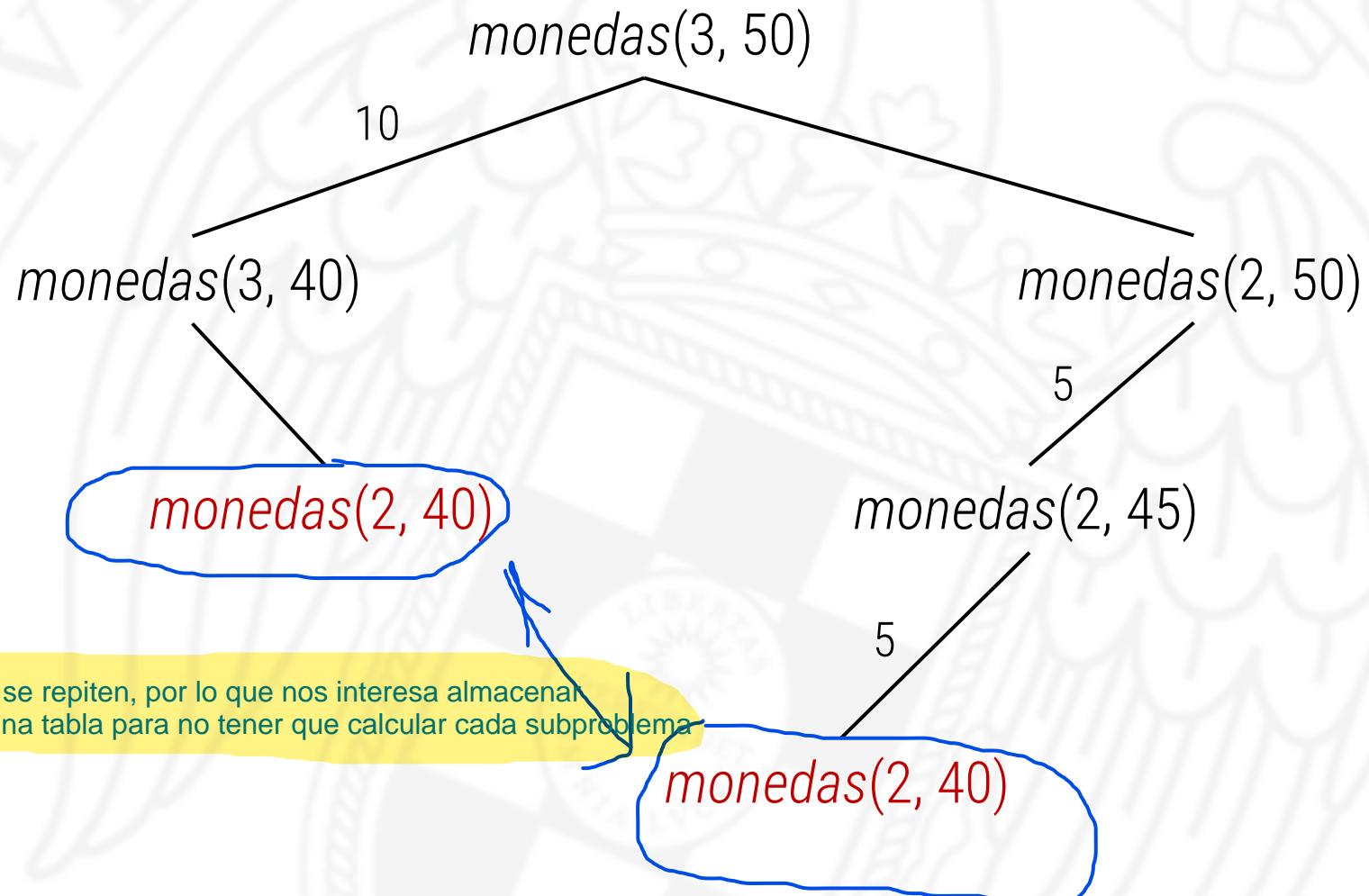
► Llamada inicial: $\text{monedas}(n, C)$

Si no hay solución devuelve más infinito

Usamos monedas del 1 al n y el precio a pagar es C .

Subproblemas repetidos

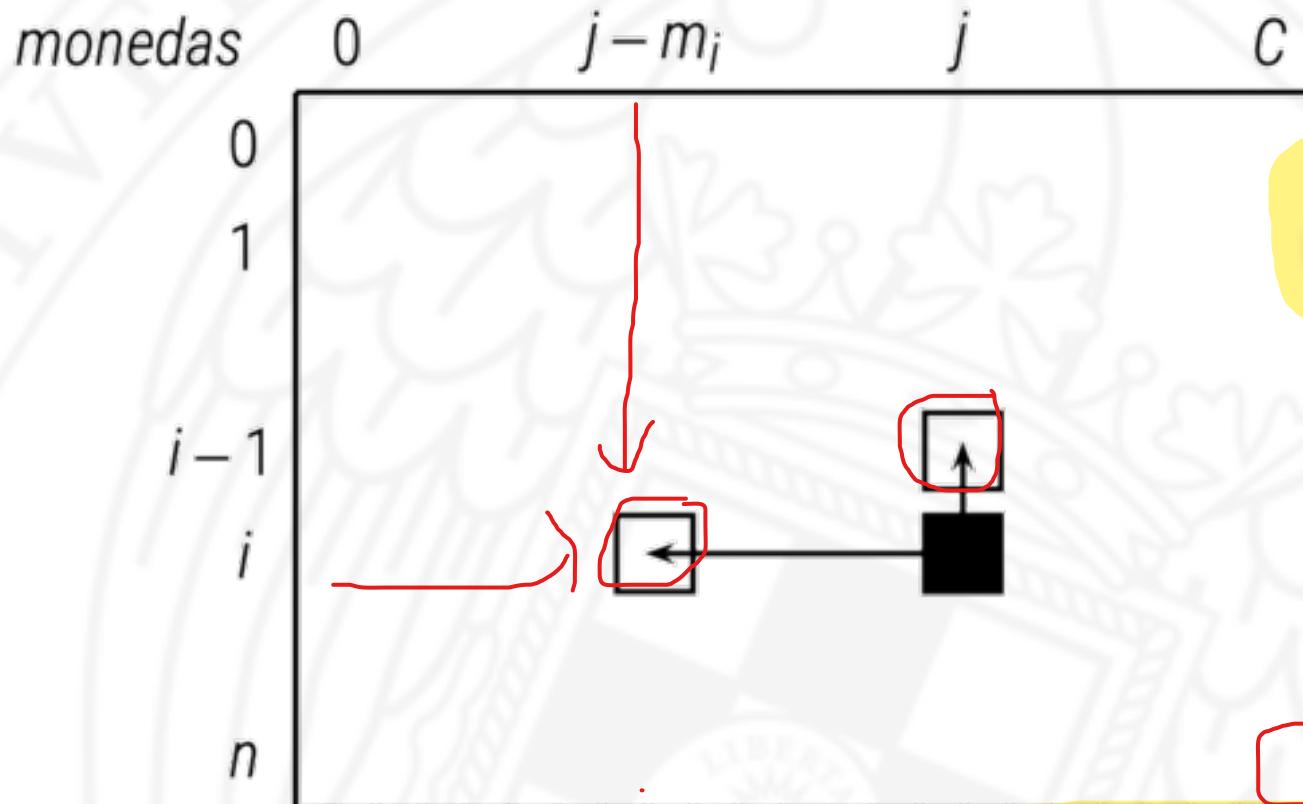
- $m_3 = 10, m_2 = 5$ (entre otras) Valores de nuestras monedas.



Los subproblemas se repiten, por lo que nos interesa almacenar los resultados en una tabla para no tener que calcular cada subproblema

Tabla

Como la función moneda tiene 2 argumentos la tabla es de 2×2 . Las filas van desde n hasta 0 y las columnas van desde C hasta 0.



Método ascendente, habría que ver en qué orden se recorren las celdas para ir de los subproblemas de tamaño más pequeño hasta ir a los subproblemas de tamaño más grande.

(n, C) contendría el valor de la solución óptima que buscamos.

Casos base son bien todos los de la primera fila, cuando ya no tenemos más tipos de monedas o la primera columna, cuando ya no queda nada más por pagar. El resto son casos recursivos.

Antes de llegar a $i j$, tenemos que haber llegado a $i-1 j$ y a $i j-m_i$
Podemos recorrer la matriz de izquierda a derecha desde arriba

Cómo tratar el infinito



EnterosInf.h

$$x + \infty = \infty \quad x < \infty$$

no todos los tipos numéricos tienen las propiedades del infinito. Las propiedades son las de la izquierda.

- ▶ `std::numeric_limits<T>` proporciona información sobre los tipos numéricos.
Los tipos enteros NO TIENEN INFINITO Y LOS REALES SI
- ▶ `has_infinity` dice si el tipo `T` tiene valor infinito o no. Y la función `infinity()` lo devuelve, si existe.
Nos dice si el tipo T en cuestión tiene un valor infinito.
- ▶ Podemos definir nuestra clase `EntInf` con la funcionalidad deseada. En particular, una constante `Infinito` que represente el infinito.

Como para los enteros NO HAY UN TIPO INFINITO lo podemos definir nosotros mismos

Implementación

$$monedas(i,j) = \begin{cases} monedas(i-1,j) & \text{si } m_i > j \\ \min(monedas(i-1,j), monedas(i,j - m_i) + 1) & \text{si } m_i \leq j \end{cases}$$

```
EntInf devolver_cambio(vector<int> const& M, int C) {
    int n = M.size();
    Matriz<EntInf> monedas(n+1, C+1, Infinito);
    monedas[0][0] = 0;
    for (int i = 1; i <= n; ++i) {
        monedas[i][0] = 0; 0 para la primera fila siempre
        for (int j = 1; j <= C; ++j)
            if (M[i-1] > j) M contiene el valor de las monedas, comienza en la posición 0 y los tipos de monedas está numerado desde 1, para acceder al valor correcto accedemos a i-1
                monedas[i][j] = monedas[i-1][j];
            else
                monedas[i][j] = min(monedas[i-1][j], monedas[i][j - M[i-1]] + 1);
    }
    return monedas[n][C];
}
```

$O(N \cdot C)$

m_i

Se da el valor infinito a todas las posiciones. Pero solamente hay que dárselo a la primera fila, porque en la primera fila residen los casos base para los cuales el número de monedas es 0.

La primera posición, aunque sea la primera fila se inicializa a 0 porque cuando el coste es 0, no tenemos que usar más monedas

Ejemplo

3 tipos de monedas y queremos pagar la cantidad 8

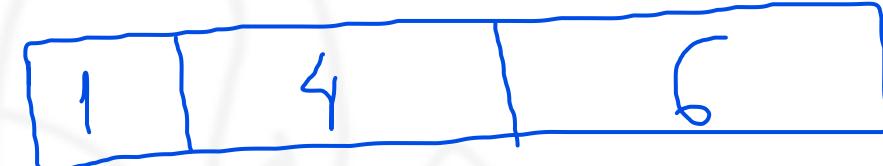
$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Aquí la solución voraz no funcionaría porque elegiría una moneda de 6 y dos de 1 (total 3 MONEDAS). Mientras que la solución óptima elegiría 2 MONEDAS DE CUATRO

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0								
2	0								
3	0								

Empezamos a llenar la primera fila



Ejemplo



Como $M[i-1] \leq j$ ya que $M[i-1] = 1$ habría que hacer lo siguiente:

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

`monedas[i][j] = min(moneda[i-1][j], moneda[i][j - M[i-1]] + 1);`

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞
2	0								
3	0								

Above the table, there is a red arrow pointing from the value `+∞` at $(1, 1)$ to the value `0` at $(1, 2)$, with the text "Nos quedamos con el menor" (We keep the minimum) written next to it.

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

Mismo procedimiento que antes

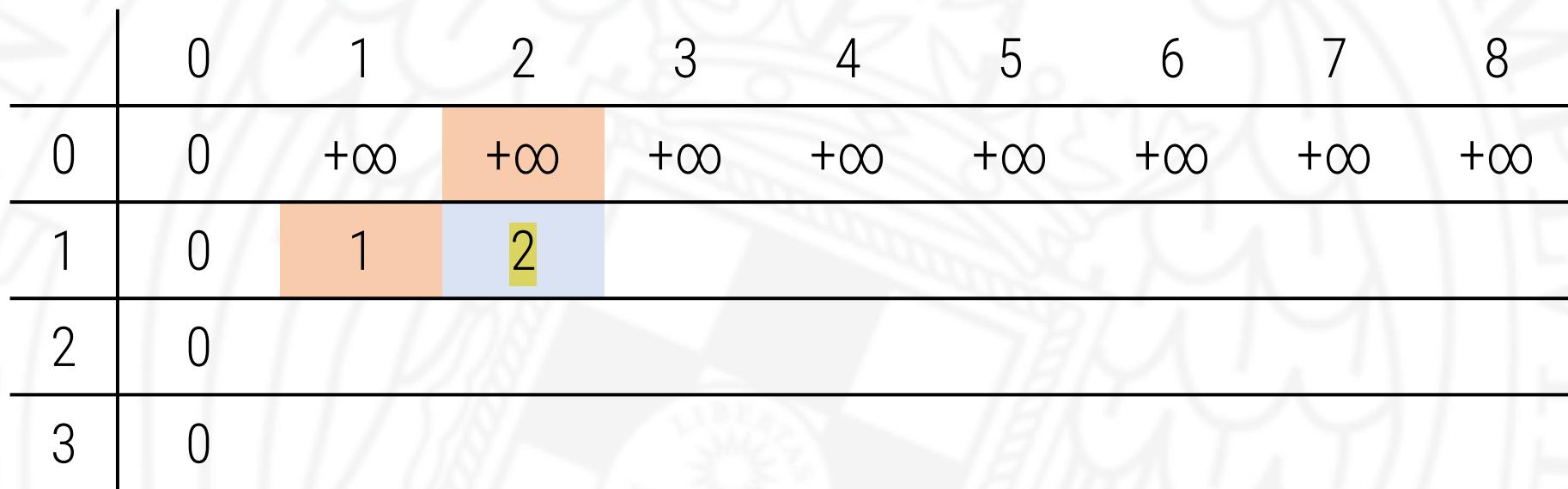
$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$



Ejemplo

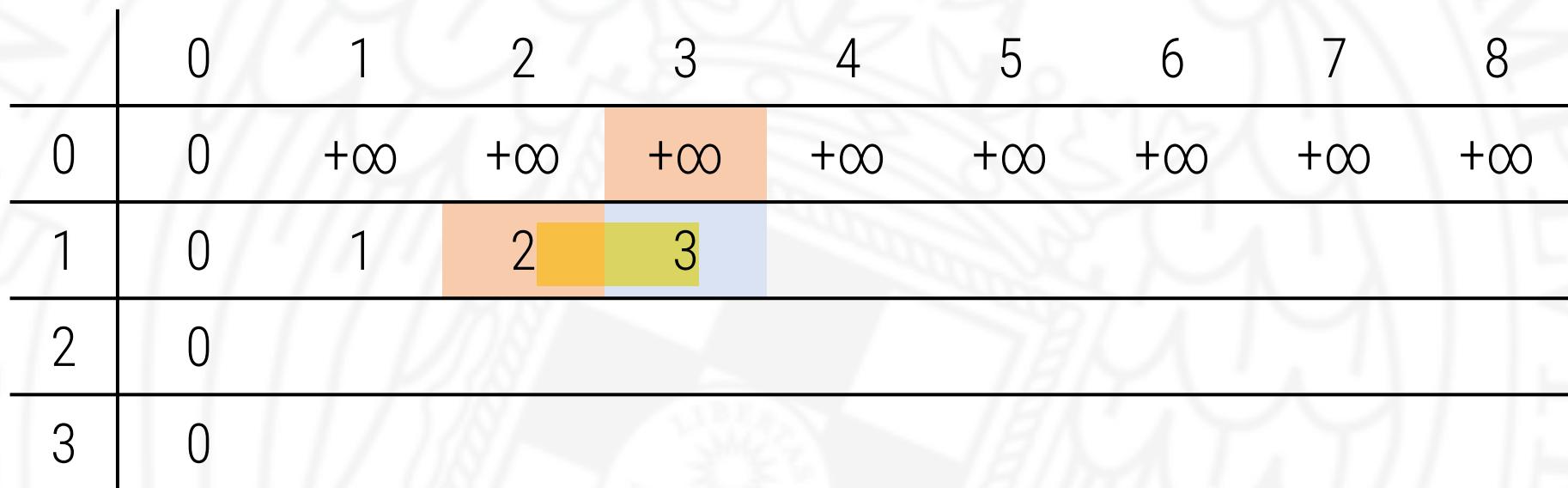
$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

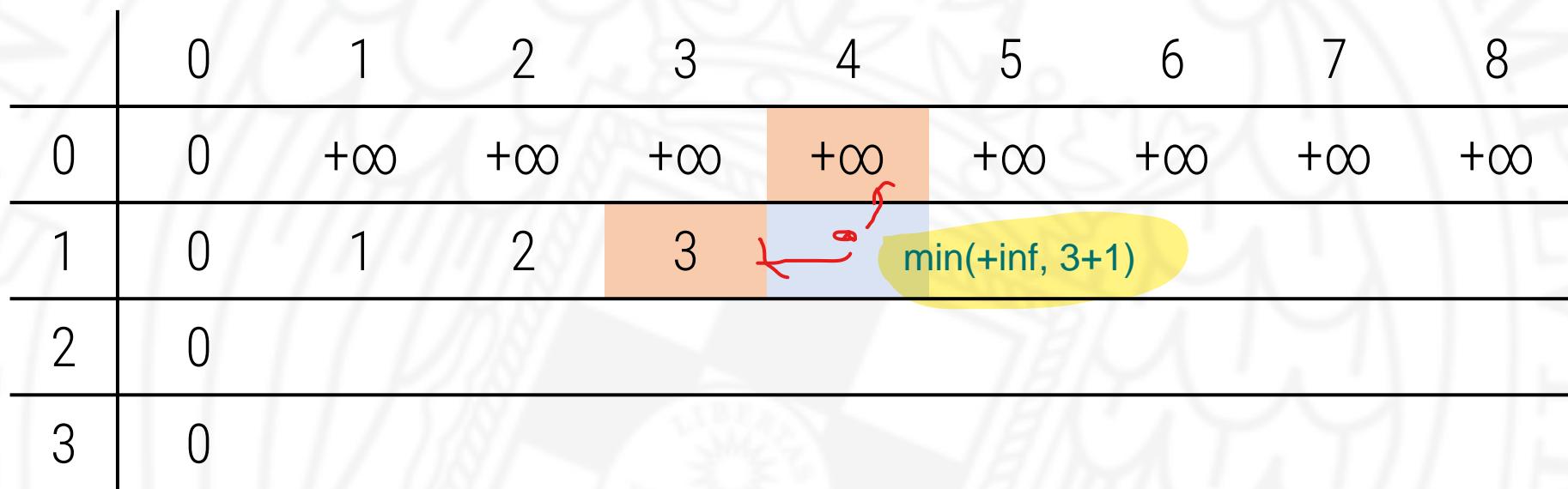
$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$



Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$



I Ejemplo

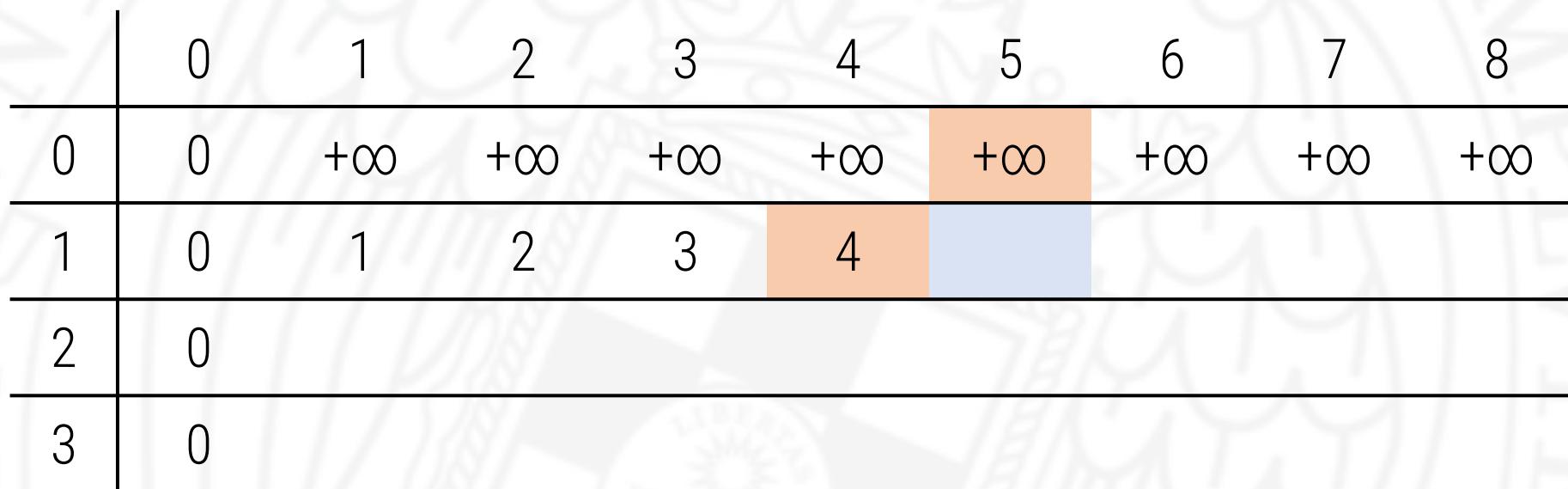
$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

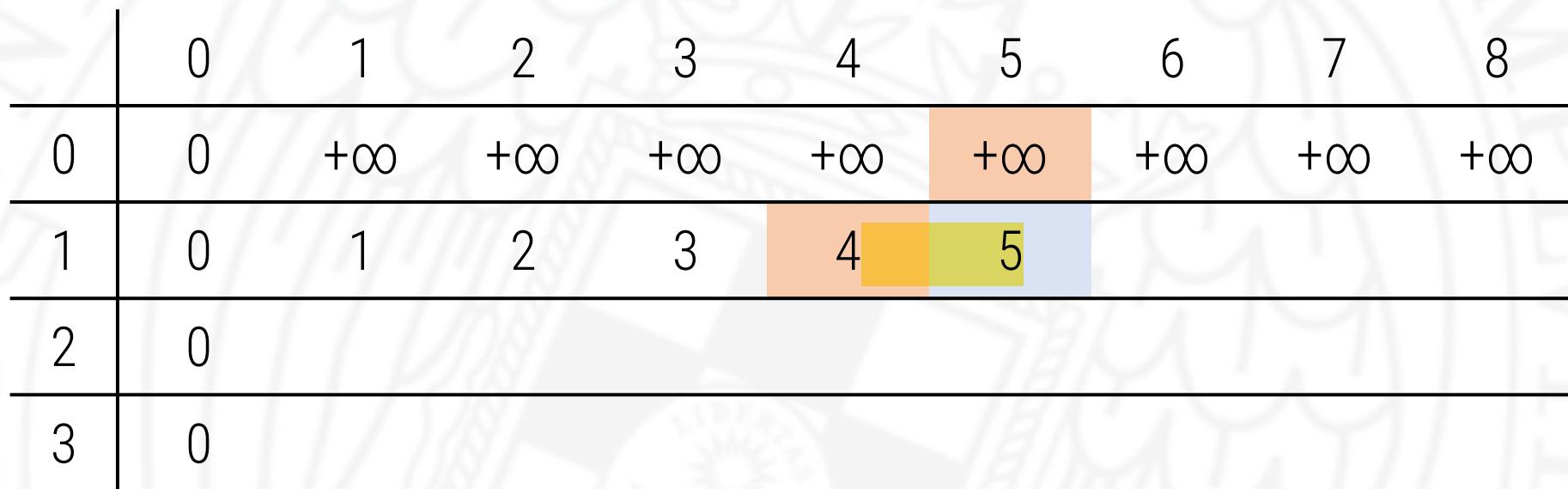
$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$



Ejemplo

$$C = 8, n = 3$$

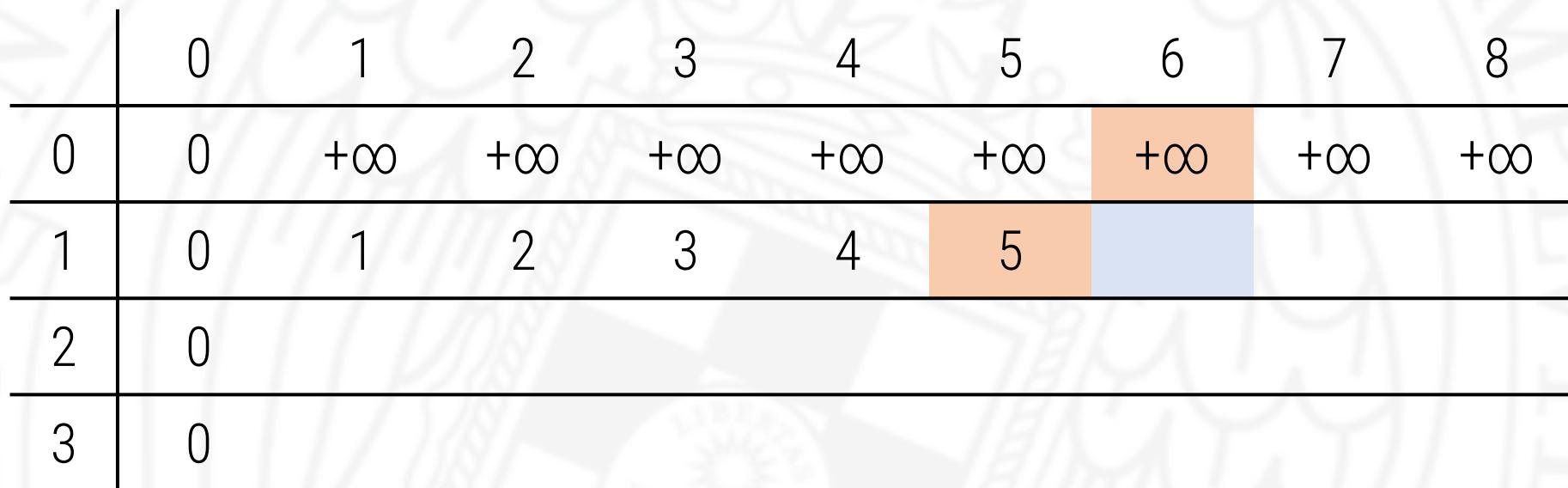
$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$



Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$



Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

8 monedas de 1 para
coste 8

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0								
3	0								

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Empezamos a considerar las monedas de valor 6, por lo que hasta la posición 6 tenemos que llenar con los valores de arriba.

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0								

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1							

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1							

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2						

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2						

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3					

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3					

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1				

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1				

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2			

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1		

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1		

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1		

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1	2	

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1	2	

Ahora gana que el mínimo es 2. Por tanto nos quedamos con ese valor

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1	2	2

Ejemplo

$$C = 8, n = 3$$

$$m_1 = 1, m_2 = 4 \text{ y } m_3 = 6$$

Sabemos cual es el valor óptimo, pero NO SABEMOS LA SOLUCIÓN ÓPTIMA.
LA SOLUCIÓN ÓPTIMA LA RECONSTRUIMOS A PARTIR DE LA TABLA FINAL.

	0	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1	2	2

Sabemos que son 2 el número de monedas mínimo para pagar 8 euros.

Para la solución óptima miramos su valor de arriba y su valor , $\text{monedas}[i][j] = \min(\text{monedas}[i-1][j], \text{monedas}[i][j - M[i-1]] + 1)$

Reconstrucción de la solución óptima

podría suceder que ambas devuelvan el mismo valor => varias soluciones óptimas.

$$\text{monedas}(i,j) = \min(\underbrace{\text{monedas}(i-1,j)}, \underbrace{\text{monedas}(i,j-m_i) + 1})$$

no cogemos
moneda m_i

sí cogemos
moneda m_i

- ▶ Sabemos que si $\text{monedas}(i,j) = \text{monedas}(i-1,j)$ es porque podemos no coger monedas de tipo i para pagar la cantidad j .
- ▶ Mientras que si $\text{monedas}(i,j) \neq \text{monedas}(i-1,j)$ debemos coger al menos una moneda de tipo i para pagar j .

Implementación

Solución con monedas que forman parte de la solución óptima.

```
vector<int> devolver_cambio(vector<int> const& M, int C) {  
    ... // rellenar la matriz monedas como antes  
    si no tiene solución -> n[C] = infinito. Se devuelve vacío
```

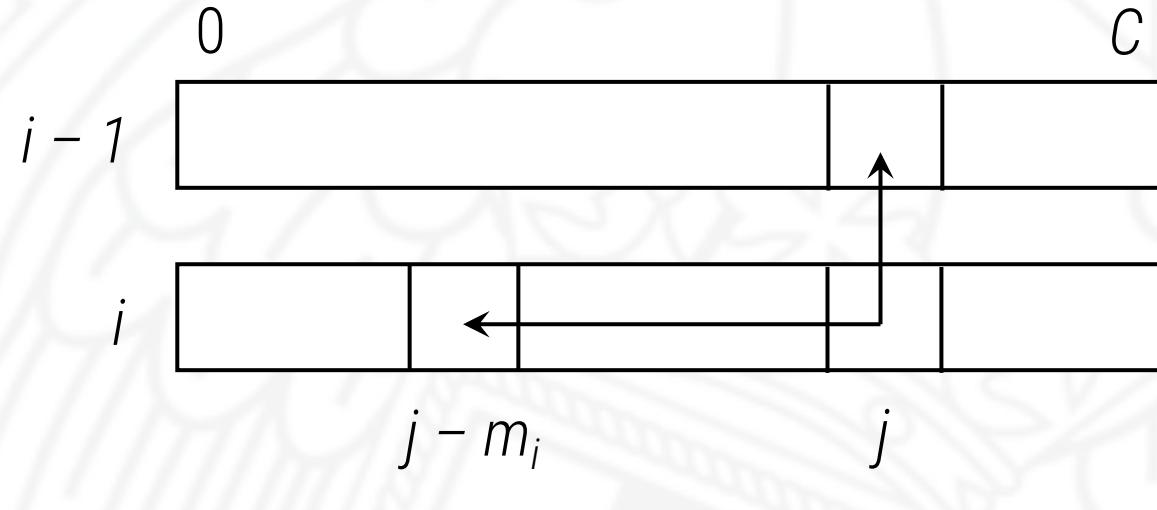
```
vector<int> sol;  
if (monedas[n][C] != Infinito) {  
    int i = n, j = C;  Mientras j>0  
    while (j > 0) { // no se ha pagado todo  
        if (M[i-1] <= j && monedas[i][j] != monedas[i-1][j]) {  
            // tomamos una moneda de tipo i  
            sol.push_back(M[i-1]); j = j - M[i-1];  
        } else // no tomamos más monedas de tipo i  
        --i;  Cogemos la de arriba  
    }  
}  
return sol;
```

$O(n \max\{n, c\})$

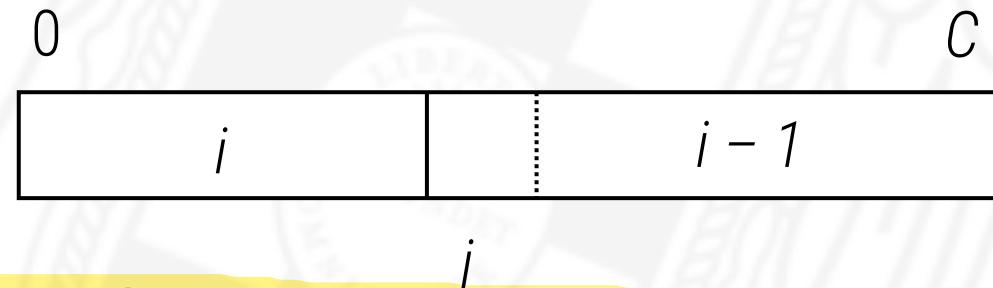
cogemos la de la izquierda

Mejorar espacio adicional

- ▶ ¿Podemos reducir la memoria necesaria?



Para llenar la fila i , solo necesitamos la fila $i-1$, por tanto podemos eliminar las demás y actualizando la fila a medida que avanzamos



En un momento el vector tendrá los valores de la fila i y a la derecha los de su fila de arriba para calcular la mejor opción. Y así todo el rato.

Mejorar espacio adicional

Pero... al mejorar el espacio adicional, podemos reconstruir la solución óptima igual que hacíamos antes partiendo de la matriz anterior? Aquí si

- ▶ ¿Y podemos seguir reconstruyendo la solución?
- ▶ La última fila contiene la información sobre el número de monedas mínimo necesario para cada cantidad, con el sistema monetario **completo**.

$$\text{monedas}(n,j) = \text{monedas}(n,j - m_i) + 1$$

- ▶ Al tener un número ilimitado de monedas de cada tipo, el sistema monetario **no cambia** al ir gastando monedas.

En otros problemas no.

Implementación

```
vector<int> devolver_cambio(vector<int> const& M, int C) {
    int n = M.size();          En el último valor se guarda el valor inmediatamente superior.
    vector<EntInf> monedas(C+1, Infinito);
    monedas[0] = 0;
    // calcular la matriz sobre el propio vector
    for (int i = 1; i <= n; ++i) { Recorre filas de la matriz
        for (int j = M[i-1]; j <= C; ++j) { Recorre sus posiciones de izquierda a derecha
            monedas[j] = min(monedas[j], monedas[j - M[i-1]] + 1);
        }
    }
}
```

Implementación

```
vector<int> sol;
if (monedas[C] != Infinito) {
    int i = n, j = C;
    while (j > 0) { // no se ha pagado todo
        if (M[i-1] <= j && monedas[j] == monedas[j - M[i-1]] + 1) {
            // tomamos una moneda de tipo i
            sol.push_back(M[i-1]);
            j = j - M[i-1];
        } else // no tomamos más monedas de tipo i
            --i;
    }
}
return sol;
}
```

O(nC) tiempo
O(C) en espacio(solo usamos un vector)