EJEMPLOS DE PROBLEMAS RESUELTOS

UNIVERSIDAD DE MÁLAGA
E.T.S. INGENIERÍA INFORMÁTICA

Fco Javier Moreno Barea

1. Sean $A_1;\ldots;A_n$ un conjunto de matrices que se desean multiplicar en secuencia. Se supone que estas matrices son compatibles, esto es, la dimensión de A_i es $m_i \times n_i$ y se cumple que $n_i = m_{i+1}$.
Nótese que al multiplicar dos matrices de tamaños m×n y n×p se realizan m·n·p multiplicaciones.
El numero total de multiplicaciones escalares que se realizan dependiera del modo en que se asocien las matrices para la multiplicación. Por ejemplo, sea $A_1^{10\times100}$, $A_2^{100\times5}$ y $A_3^{5\times50}$:

- $((A_1A_2)A_3)$ supone $10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 7500$ multiplicaciones escalares.
- $(A_1(A_2A_3))$ supone $100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50 = 75000$ multiplicaciones escalares.

Resulta por lo tanto fundamental encontrar la asociación optima para realizar la multiplicación.

- ¿Cuántos posibles órdenes hay? Respuesta: Muchos.
  Sea P(n)el número de órdenes posibles en una cadena de n matrices.

  - Si tenemos solo 1 matriz, es obvio que solo hay una posibilidad.
  - En otro caso, tendremos la suma de los posibles órdenes para k por los posibles órdenes para n-k.

$$P(n) = \begin{cases} 1 & \text{si } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{si } n \geq 2 \end{cases}$$

Por ejemplo, sea $A_1^{2\times3}$, $A_2^{3\times5}$ y $A_3^{5\times4}$:

Con este ejemplo pequeño:

- (A_1A_2) -> $2 \cdot 3 \cdot 5 = 30$ multiplicaciones.
- (A_2A_3) -> $3 \cdot 5 \cdot 4 = 60$ multiplicaciones.

Para A de 1 a 3, entonces solo hay dos opciones:

- $((A_1A_2)A_3)$ -> $2 \cdot 3 \cdot 5 + 2 \cdot 5 \cdot 4 = 30 + 40 = 70$ multiplicaciones.
- $(A_1(A_2A_3))$ -> $3 \cdot 5 \cdot 4 + 2 \cdot 3 \cdot 4 = 60 + 24 = 84$ multiplicaciones.

i	1	2	3
1	0	30	X
2	0	0	60
3	0	0	0

Tenemos que:

- $((A_1A_2)A_3)$ -> $2 \cdot 3 \cdot 5 + 2 \cdot 5 \cdot 4 = C[1,2] + 40$
- $(A_1(A_2A_3))$ -> $3 \cdot 5 \cdot 4 + 2 \cdot 3 \cdot 4 = C[2,3] + 24$

¿Entonces que son el 40 y el 24?

- $40 = 2 \cdot 5 \cdot 4$ = filas1 · filas 3 · columnas 3
- $24 = 2 \cdot 3 \cdot 4$ = filas1 · filas 2 · columnas 3

- $((A_1A_2)A_3)$ -> C[1,2] + filas1 · filas 3 · columnas 3
- $(A_1(A_2A_3))$ -> C[2,3] + filas1 · filas 2 · columnas 3

- X = C[1,3] = min { (C[1,2] + C[3,3] + f1·f3·c3), (C[2,3] + C[3,3] + f1·f2·c3) }
- X = min { (30 + 0 + 40), (24 + 0 + 60) } = min { 70, 84 } = 70

Por ejemplo, sea $A_1^{2\times3}$, $A_2^{3\times5}$, $A_3^{5\times4}$ y $A_4^{4\times6}$:

Con este ejemplo pequeño:

- (A_1A_2) -> $2 \cdot 3 \cdot 5 = 30$ multiplicaciones.
- (A_2A_3) -> $3 \cdot 5 \cdot 4 = 60$ multiplicaciones.
- (A_3A_4) -> $5 \cdot 4 \cdot 6 = 120$ multiplicaciones.

i	1	2	3	4
1	0	30	70	X
2	0	0	60	132
3	0	0	0	120
4	0	0	0	0

Para A de 2 a 4, entonces solo hay dos opciones:

- $((A_2A_3)A_4)$ -> $3 \cdot 5 \cdot 4 + 3 \cdot 4 \cdot 6 = 60 + 72 = 132$
- $(A_2(A_3A_4))$ -> $5 \cdot 4 \cdot 6 + 3 \cdot 5 \cdot 6 = 120 + 90 = 210$

Para A de 1 a 4, entonces hay 5 opciones:

- $((A_1A_2)A_3)A_4)$ -> $2 \cdot 3 \cdot 5 + 2 \cdot 5 \cdot 4 + 2 \cdot 4 \cdot 6 = 118$
- $(A_1(A_2A_3))A_4)$ -> $3 \cdot 5 \cdot 4 + 2 \cdot 3 \cdot 4 + 2 \cdot 4 \cdot 6 = 132$
- $(A_1((A_2A_3)A_4)$ -> $3 \cdot 5 \cdot 4 + 3 \cdot 4 \cdot 6 + 2 \cdot 3 \cdot 6 = 168$
- $(A_1(A_2(A_3A_4))$ -> $5 \cdot 4 \cdot 6 + 3 \cdot 4 \cdot 6 + 2 \cdot 4 \cdot 6 = 240$
- $((A_1A_2)(A_3A_4))$ -> $2 \cdot 3 \cdot 5 + 5 \cdot 4 \cdot 6 + 2 \cdot 5 \cdot 6 = 210$

Tenemos que:

a) $((A_1A_2)A_3)A_4)$ -> $C[1,3] + C[4,4] + f_1 \cdot f_4 \cdot c_4 = 70+0+48 = 118$
b) $(A_1(A_2A_3))A_4)$ -> propiedad de subestructura óptima a mejor b
c) $(A_1((A_2A_3)A_4)$ -> $C[1,1] + C[2,4] + f_1 \cdot f_2 \cdot c_4 = 0+132+36 = 168$
d) $(A_1(A_2(A_3A_4))$ -> propiedad de subestructura óptima c mejor d
e) $((A_1A_2)(A_3A_4))$ -> $C[1,2] + C[3,4] + f_1 \cdot f_3 \cdot c_4 = 30+120+60 = 210$

- $X = C[1,4] = \min \{ a, c, e\}$
- $X = \min_{1 \le k \le 4} \{ C[1, k] + C[k+1, 4] + f_1 \cdot f_{k+1} \cdot c_4\}$

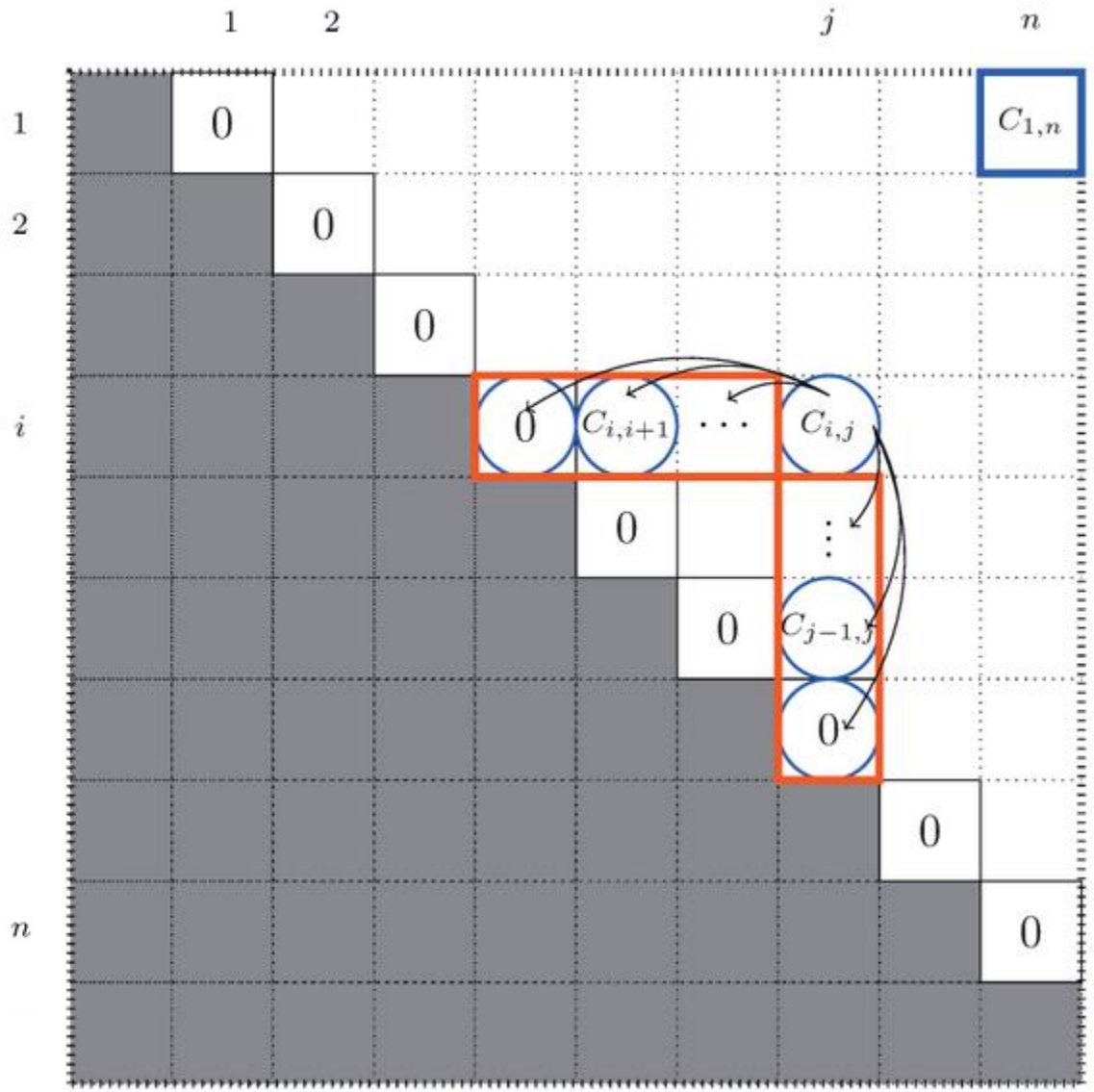
- Supongamos que tenemos la forma óptima de multiplicar las matrices $\langle A_1, \ldots, A_n \rangle$. Al nivel más alto, la solución se verá como la multiplicación de dos matrices que resultan de calcular los productos $A_1 \cdots A_k$ y $A_{k+1} \cdots A_n$,ambos en forma óptima, para algún k $(1 \leq k \leq n)$. Lo anterior implica que el problema tiene subestructura óptima.

- Supongamos que llamamos $C[i, j]$ al número óptimo de multiplicaciones escalares a realizar al $\langle A_i, A_{i+1} \ldots, A_j \rangle$. $C[i, j]$ se puede escribir recursivamente por:

$$C[i,j] = \begin{cases} 0 & \text{si } i = j \\ \min_{i \leq k \leq j} \{C[i,k] + C[k+1,j] + f_i f_{k+1} c_j\} & \text{si } i < j \end{cases}$$

  Donde f_i es el número de filas de la matriz i, f_{k+1} el nº de filas de k+1, y c_j el nº de columnas de j.

- Necesitamos emplear una estructura bidimensional con filas y columnas indexadas entre 1 y n (únicamente emplearemos las posiciones (i, j) con $i \leq j$, esto es, la diagonal principal y las posiciones sobre la misma). El caso base se halla distribuido a lo largo de la diagonal principal ($C_{i,i} = 0$), y cada posición $C_{i,j}$depende de todas las situadas a la izquierda en la misma fila ($C_{i,k}$ con $k < j$), y debajo en la misma columna ($C_{k+1,j}$ con $k \geq i$), por lo que un posible orden para rellenar la tabla es desde abajo hacia arriba y de izquierda a derecha.



Input: array f (entero) de rango $[1 \ldots n+1]$
Output: coste óptimo c
Data: tablas C, D (enteras) de rango $[1 \ldots n, 1 \ldots n]$, enteros i, j, k, q

```
for i ← 1 to n do
    C_{i,i} ← 0;
end
for i ← n − 1 down to 1 do
    for j ← i + 1 to n do
        C_{i,j} ← ∞;
        for k ← i to j − 1 do
            q ← C_{i,k} + C_{k+1,j} + f_i · f_{k+1} · f_{j+1};
            if q < C_{i,j} then
                C_{i,j} ← q;
                D_{i,j} ← k;
            end
        end
    end
end
c ← C_{1,n};
```

2. Se desea construir una antena de telefonía de M metros de altura. Para ello se dispone de un suministro ilimitado de bloques de armazón de n tipos distintos, cada uno de altura a_i y peso p_i. El objetivo es determinar cuántos bloques hay que emplear de cada tipo, de manera que la antena mida exactamente M metros y tenga el peso mínimo.

Por ejemplo, si tuviéramos 4 tipos de bloques caracterizados como (altura, peso):
[b1, b2, b3, b4] = [(2,1), (3,2), (4,3), (8,3)]

- Si la antena tuviera que tener M=6 -> cogeríamos 3 bloques (2,1) que daría peso 3.
- Si la antena tuviera que tener M=7 -> cogeríamos 2 bloques (2,1) y 1 bloque (3,2) que daría peso 4.
- Si la antena tuviera que tener M=10 -> cogeríamos 1 bloques (2,1) y 1 bloque (8,3) que daría peso 4.

i, j	0	1	2	3	4	5	6	7	8	9	10
1	0										
2	0										
3	0										
4	0										

Tenemos una tabla con filas que representan los n bloques que tenemos, y columnas que representan la altura de 0 a M.

Por tanto, si la altura es 0, tenemos que no es necesario poner ningún bloque, por lo que su peso mínimo va a ser 0.

Cada celda será representada por A[i, j].

2. Se desea construir una antena de telefonía de M metros de altura. Para ello se dispone de un suministro ilimitado de bloques de armazón de n tipos distintos, cada uno de altura a_i y peso p_i. El objetivo es determinar cuántos bloques hay que emplear de cada tipo, de manera que la antena mida exactamente M metros y tenga el peso mínimo.

Por ejemplo, si tuviéramos 4 tipos de bloques caracterizados como (altura, peso):
[b1, b2, b3, b4] = [(2,1), (3,2), (4,3), (8,3)]

- Si la antena tuviera que tener M=6 -> cogeríamos 3 bloques (2,1) que daría peso 3.
- Si la antena tuviera que tener M=7 -> cogeríamos 2 bloques (2,1) y 1 bloque (3,2) que daría peso 4.
- Si la antena tuviera que tener M=10 -> cogeríamos 1 bloques (2,1) y 1 bloque (8,3) que daría peso 4.

i, j	0	1	2	3	4	5	6	7	8	9	10
1	0										
2	0										
3	0										
4	0	∞	∞	∞	∞	∞	∞	∞	3	∞	∞

Empezamos a rellenar de abajo a arriba y de izquierda a derecha.

Si vemos que no podemos utilizar ese bloque para poner la antena, ponemos infinito, que significa que no se puede resolver ese problema. Ejemplo: solo con bloques de tamaño a_i = 8 no podemos hacer una antena de tamaño 3, 4 o 10.

2. Se desea construir una antena de telefonía de M metros de altura. Para ello se dispone de un suministro ilimitado de bloques de armazón de n tipos distintos, cada uno de altura a_i y peso p_i. El objetivo es determinar cuántos bloques hay que emplear de cada tipo, de manera que la antena mida exactamente M metros y tenga el peso mínimo.

Por ejemplo, si tuviéramos 4 tipos de bloques caracterizados como (altura, peso):
[b1, b2, b3, b4] = [(2,1), (3,2), (4,3), (8,3)]

- Si la antena tuviera que tener M=6 -> cogeríamos 3 bloques (2,1) que daría peso 3.
- Si la antena tuviera que tener M=7 -> cogeríamos 2 bloques (2,1) y 1 bloque (3,2) que daría peso 4.
- Si la antena tuviera que tener M=10 -> cogeríamos 1 bloques (2,1) y 1 bloque (8,3) que daría peso 4.

i, j	0	1	2	3	4	5	6	7	8	9	10
1	0	∞									
2	0	∞	∞								
3	0	∞	∞	∞	3	∞	∞	∞	3	∞	∞
4	0	∞	∞	∞	∞	∞	∞	∞	3	∞	∞

Ahora, tenemos bloques b3 y b4.

Para M=4, ahora si podemos resolver el problema poniendo 1 bloque de peso 3.

Para M=8, tenemos que o ponemos 2 bloques b3 o uno b4. El peso mínimo es:

min(A[i, j-ai] + pi, A[i+1,j]) = min(3+3, 3) = 3

2. Se desea construir una antena de telefonía de M metros de altura. Para ello se dispone de un suministro ilimitado de bloques de armazón de n tipos distintos, cada uno de altura a_i y peso p_i. El objetivo es determinar cuántos bloques hay que emplear de cada tipo, de manera que la antena mida exactamente M metros y tenga el peso mínimo.

Por ejemplo, si tuviéramos 4 tipos de bloques caracterizados como (altura, peso):
[b1, b2, b3, b4] = [(2,1), (3,2), (4,3), (8,3)]

- Si la antena tuviera que tener M=6 -> cogeríamos 3 bloques (2,1) que daría peso 3.
- Si la antena tuviera que tener M=7 -> cogeríamos 2 bloques (2,1) y 1 bloque (3,2) que daría peso 4.
- Si la antena tuviera que tener M=10 -> cogeríamos 1 bloques (2,1) y 1 bloque (8,3) que daría peso 4.

i, j	0	1	2	3	4	5	6	7	8	9	10
1	0	∞									
2	0	∞	∞	2	3	∞	4	5	3	6	∞
3	0	∞	∞	∞	3	∞	∞	∞	3	∞	∞
4	0	∞	∞	∞	∞	∞	∞	∞	3	∞	∞

Ahora, tenemos bloques b2, b3 y b4.

Para M=3, ponemos 1 bloque b2 de peso 2.
Para M=4, ponemos 1 bloque b3 de peso 3.
Para M=6, ponemos 2 bloques b2, peso 4.
Para M=7, ponemos 1 b2 y 1 b3, peso 5.
Para M=8, ponemos 1 bloques b4, peso 3.

$$\min(A[i, j-a_i] + p_i, A[i+1,j])$$

2. Se desea construir una antena de telefonía de M metros de altura. Para ello se dispone de un suministro ilimitado de bloques de armazón de n tipos distintos, cada uno de altura a_i y peso p_i. El objetivo es determinar cuántos bloques hay que emplear de cada tipo, de manera que la antena mida exactamente M metros y tenga el peso mínimo.

Por ejemplo, si tuviéramos 4 tipos de bloques caracterizados como (altura, peso):
[b1, b2, b3, b4] = [(2,1), (3,2), (4,3), (8,3)]

- Si la antena tuviera que tener M=6 -> cogeríamos 3 bloques (2,1) que daría peso 3.
- Si la antena tuviera que tener M=7 -> cogeríamos 2 bloques (2,1) y 1 bloque (3,2) que daría peso 4.
- Si la antena tuviera que tener M=10 -> cogeríamos 1 bloques (2,1) y 1 bloque (8,3) que daría peso 4.

i, j	0	1	2	3	4	5	6	7	8	9	10
1	0	∞	1	2	2	3	3	4	3	5	4
2	0	∞	∞	2	3	∞	4	5	3	6	∞
3	0	∞	∞	∞	3	∞	∞	∞	3	∞	∞
4	0	∞	∞	∞	∞	∞	∞	∞	3	∞	∞

Por último, tenemos bloques b1, b2, b3 y b4.

Para M=2, ponemos 1 bloque b1 de peso 1.
Para M=3, ponemos 1 bloque b2 de peso 2.
Para M=4, ponemos 2 bloques b1, peso 2.
Para M=7, ponemos 2 b1 y 1 b2, peso 4.
Para M=10, ponemos 1 b1 y 1 b4, peso 4.

$$\min(A[i, j-a_i] + p_i, A[i+1,j])$$

- Con lo anterior tenemos una tabla de resultados que nos índica el peso mínimo, con lo que es fácil recomponer una salida. La salida será un vector de tamaño m, donde para cada posición tenemos el nº de veces que se coge cada bloque [B_1, B_2, …, B_n]. Ahora hay que ver si se cumple la propiedad de subestructura óptima. Se tiene que la solución tiene que cumplir lo siguiente:

$$\sum_{i=1}^{n} B_i \cdot a_i = M \qquad \sum_{i=1}^{n} B_i \cdot p_i = P^*$$

Si tuviéramos un problema más pequeño, sin el bloque n, tendremos que:

$$\sum_{i=1}^{n-1} B_i \cdot a_i = M - B_n \cdot a_n \qquad \sum_{i=1}^{n-1} B_i \cdot p_i = P^* - B_n \cdot p_n$$

Ahora supongamos que esa no es una solución óptima, significa que hay otra solución B' tal que:

$$\sum_{i=1}^{n-1} B'_i \cdot a_i = M - B_n \cdot a_n \qquad \sum_{i=1}^{n-1} B'_i \cdot p_i = P' < P^* - B_n \cdot p_n$$

De ser así, si volviéramos al problema original e incluir de nuevo el bloque n, tendríamos:

$$\sum_{i=1}^{n} B'_i \cdot a_i + B_n \cdot a_n = M \qquad \sum_{i=1}^{n} B'_i \cdot p_i = P' + B_n \cdot p_n < P^*$$

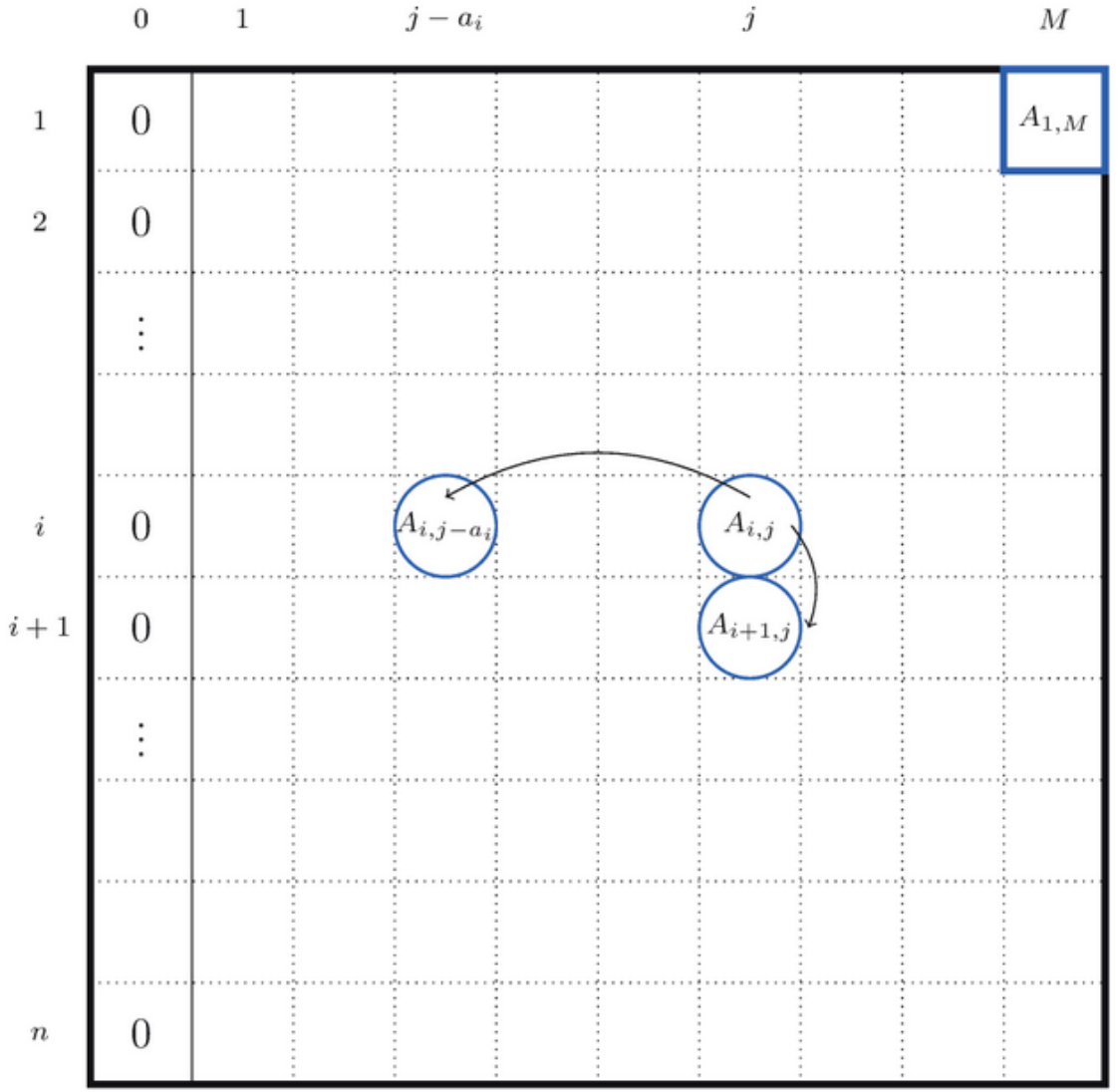
Dado que habíamos dicho que P^* era mínimo, que haya una solución $P' + B_n \cdot p_n < P^*$, es una contradicción.

- Con todo lo anterior podríamos ya formular la ecuación de Bellman:

$$A_{i,j}\begin{cases} 0 & \text{si } j = 0 \\ \infty & \text{si } j > 0, i = n, a_i > j \\ A_{i+1,j} & \text{si } j > 0, i < n, a_i > j \\ A_{i,j-a_i} + p_i & \text{si } j \geq a_i, i = n \\ \min(A_{i,j-a_i} + p_i, A_{i+1,j}) & \text{si } j \geq a_i, i < n \end{cases}$$

El cálculo de la recurrencia debe apoyarse en una estructura de datos bidimensional con filas indexadas entre 1 y n (los tipos de bloque) y columnas indexadas entre 0 y M (altura de la antena). De acuerdo con las mismas, debemos rellenar la tabla empezando por la fila n e ir ascendiendo hasta la fila 1, completando cada fila de izquierda a derecha.

Observamos que el tamaño de la tabla es $\Theta(nM)$ y que el cómputo de cada posición se realiza en tiempo constante (el mínimo entre dos valores). Por lo tanto el proceso completo tiene complejidad temporal $\Theta(nM)$.



- Cómputo del peso óptimo de la antena:

Input: arrays p (real) y a (entero) [1...n], entero M>0

Output: peso óptimo w (real)

Data: enteros i, j, tabla A (real) [1...n,0...M]

for $i = 1$ to n **do**
    $A_{i,0} = 0;$
for $i = n$ down to 1 **do**
    for $j = 1$ to M **do**
        if $a_i >$ j **then**
            if $i = n$ **then**
                $A_{i,j} = \infty;$
            else
                $A_{i,j} = A_{i+1,j};$
        else
            if $i = n$ **then**
                $A_{i,j} = A_{i,j-a_i} + p_i;$
            else
                $A_{i,j} = \min(A_{i,j-a_i} + p_i, A_{i+1,j});$
$w = A_{1,M}$

Para reconstruir la solución óptima con la tabla, empezando en la posición (1, M), basta comparar la posición actual con la de la fila inferior: si coinciden no se usan más bloques del tipo actual y se baja a la siguiente fila; en caso contrario se usa un bloque de dicho tamaño, se queda en la misma fila y desplaza tantas columnas a la izquierda como altura tenga el bloque correspondiente. Se finaliza cuando la columna j = 0.

Input: tabla A (real) [1...n,0... M], array a (entero) [1...n]

Output: array s (entero) [1...n] **Data**: enteros i, j

for $i = 1$ to n **do**
    $s_i = 0;$

$i = 1; \quad j = M;$
if $A_{1,M} \neq \infty$ **then**
    while $j > 0$ **do**
        if $(i = n) \vee (A_{i,j} \neq A_{i+1,j})$ **then**
            $s_i = s_i + 1; \quad j = j - a_i;$
        else
            $i = i + 1;$

3. Un estudiante aventajado intenta organizar sus horas de estudio para maximizar la nota acumulada que obtendría en las n asignaturas del cuatrimestre. Para ello, analiza los temarios y obtiene una estimación c_{ij} de la nota numérica que obtendría en la asignatura i si le dedicara j de las H horas de estudio de las que dispone. Su objetivo ahora es determinar cuántas horas debe dedicar a cada asignatura.

Por ejemplo, el alumno tiene 4 asignaturas y 7 horas para dedicar, y el alumno obtiene la siguiente estimación:

est	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	6	8	10	10	10	10
3	0	4	4	5	6	7	9	10
4	0	2	4	7	7	8	8	9

n,h	0	1	2	3	4	5	6	7
1								
2								
3								
4								

Dada la estimación, tenemos que crear otra tabla similar con filas que representan las n asignaturas, y columnas que representan las h horas disponibles.

Cada celda será representada por A[i, j].

3. Un estudiante aventajado intenta organizar sus horas de estudio para maximizar la nota acumulada que obtendría en las n asignaturas del cuatrimestre. Para ello, analiza los temarios y obtiene una estimación c_{ij} de la nota numérica que obtendría en la asignatura i si le dedicara j de las H horas de estudio de las que dispone. Su objetivo ahora es determinar cuántas horas debe dedicar a cada asignatura.

Por ejemplo, el alumno tiene 4 asignaturas y 7 horas para dedicar, y el alumno obtiene la siguiente estimación:

est	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	6	8	10	10	10	10
3	0	4	4	5	6	7	9	10
4	0	2	4	7	7	8	8	9

n,h	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2								
3								
4								

Hay varias formas de rellenar la tabla, una de las más simples es empezar de izquierda a derecha y de arriba a abajo. ¿Por qué? Porque si tenemos solo la asignatura 1, las calificaciones máximas serán las que se pueden obtener con esta asignatura.

3. Un estudiante aventajado intenta organizar sus horas de estudio para maximizar la nota acumulada que obtendría en las n asignaturas del cuatrimestre. Para ello, analiza los temarios y obtiene una estimación c_{ij} de la nota numérica que obtendría en la asignatura i si le dedicara j de las H horas de estudio de las que dispone. Su objetivo ahora es determinar cuántas horas debe dedicar a cada asignatura.

Por ejemplo, el alumno tiene 4 asignaturas y 7 horas para dedicar, y el alumno obtiene la siguiente estimación:

est	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	6	8	10	10	10	10
3	0	4	4	5	6	7	9	10
4	0	2	4	7	7	8	8	9

n,h	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	8	9	11	13	14	16
3								
4								

¿Ahora como procedemos?

- Si podemos dedicar 0 horas, y tenemos las asignaturas 1 y 2, tendremos el máximo de dedicar $A_{1,0}$ y $c_{2,0}$
- Si h=2, y tenemos las asignaturas 1 y 2, tendremos el máximo de dedicar 2 horas a a_2 y 0 horas a a_1, dedicar 2 horas a a_1 y 0 horas a a_2, o dedicar 1 hora1 a a_1 y 1 hora a a_2

3. Un estudiante aventajado intenta organizar sus horas de estudio para maximizar la nota acumulada que obtendría en las n asignaturas del cuatrimestre. Para ello, analiza los temarios y obtiene una estimación c_{ij} de la nota numérica que obtendría en la asignatura i si le dedicara j de las H horas de estudio de las que dispone. Su objetivo ahora es determinar cuántas horas debe dedicar a cada asignatura.

Por ejemplo, el alumno tiene 4 asignaturas y 7 horas para dedicar, y el alumno obtiene la siguiente estimación:

est	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	6	8	10	10	10	10
3	0	4	4	5	6	7	9	10
4	0	2	4	7	7	8	8	9

n,h	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	8	9	11	13	14	16
3	1	5	9	12	13	15	17	18
4								

- Si h=3, y tenemos las asignaturas 1, 2 y 3, tendremos el máximo de dedicar 3 horas a a_1 y 0 al resto, dedicar 3 horas a a_2 y 0 al resto, 3 horas a a_3 y 0 al resto, 2 horas a a_1 y 1 hora a a_2, 2 horas a a_1 y 1 hora a a_3, …
- Todos los resultados implican a la fila anterior $A_{2,3-k}$ más la estimación $c_{3,k}$

3. Un estudiante aventajado intenta organizar sus horas de estudio para maximizar la nota acumulada que obtendría en las n asignaturas del cuatrimestre. Para ello, analiza los temarios y obtiene una estimación c_{ij} de la nota numérica que obtendría en la asignatura i si le dedicara j de las H horas de estudio de las que dispone. Su objetivo ahora es determinar cuántas horas debe dedicar a cada asignatura.

Por ejemplo, el alumno tiene 4 asignaturas y 7 horas para dedicar, y el alumno obtiene la siguiente estimación:

est	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	6	8	10	10	10	10
3	0	4	4	5	6	7	9	10
4	0	2	4	7	7	8	8	9

n,h	0	1	2	3	4	5	6	7
1	0	3	4	6	8	9	10	10
2	1	5	8	9	11	13	14	16
3	1	5	9	12	13	15	17	18
4	1	5	9	12	14	16	19	20

- Nuestro razonamiento se ratifica con el caso de 4 asignaturas. Así que cada posición $A_{i,j}$ es:

$$A_{i,j} = \max_{0 \le k \le j} \{A_{i-1,j-k} + c_{i,k}\}$$

- Con lo anterior tenemos una tabla de resultados que nos índica la calificación máxima, con lo que es fácil recomponer una salida. La salida será un vector de tamaño n, donde para cada posición tenemos el nº de horas que se dedica a cada asignatura [A_1, A_2, …, A_n]. Ahora hay que ver si se cumple la propiedad de subestructura óptima. Se tiene que la solución tiene que cumplir lo siguiente:

$$\sum_{i=1}^{n} A_i = H \qquad\qquad \sum_{i=1}^{n} c_{i,\,A_i} = C^*$$

Si tuviéramos un problema más pequeño, sin la asignatura n, tendremos que:

$$\sum_{i=1}^{n-1} A_i = H - A_n \qquad\qquad \sum_{i=1}^{n-1} c_{i,\,A_i} = C^* - c_{n,\,A_n}$$

Ahora supongamos que esa no es una solución óptima, significa que hay otra solución A' tal que:

$$\sum_{i=1}^{n-1} A'_i = H - A_n \qquad\qquad \sum_{i=1}^{n-1} c_{i,\,A'_i} = C' > C^* - c_{n,\,A_n}$$

De ser así, si volviéramos al problema original e incluir de nuevo la asignatura n, tendríamos:

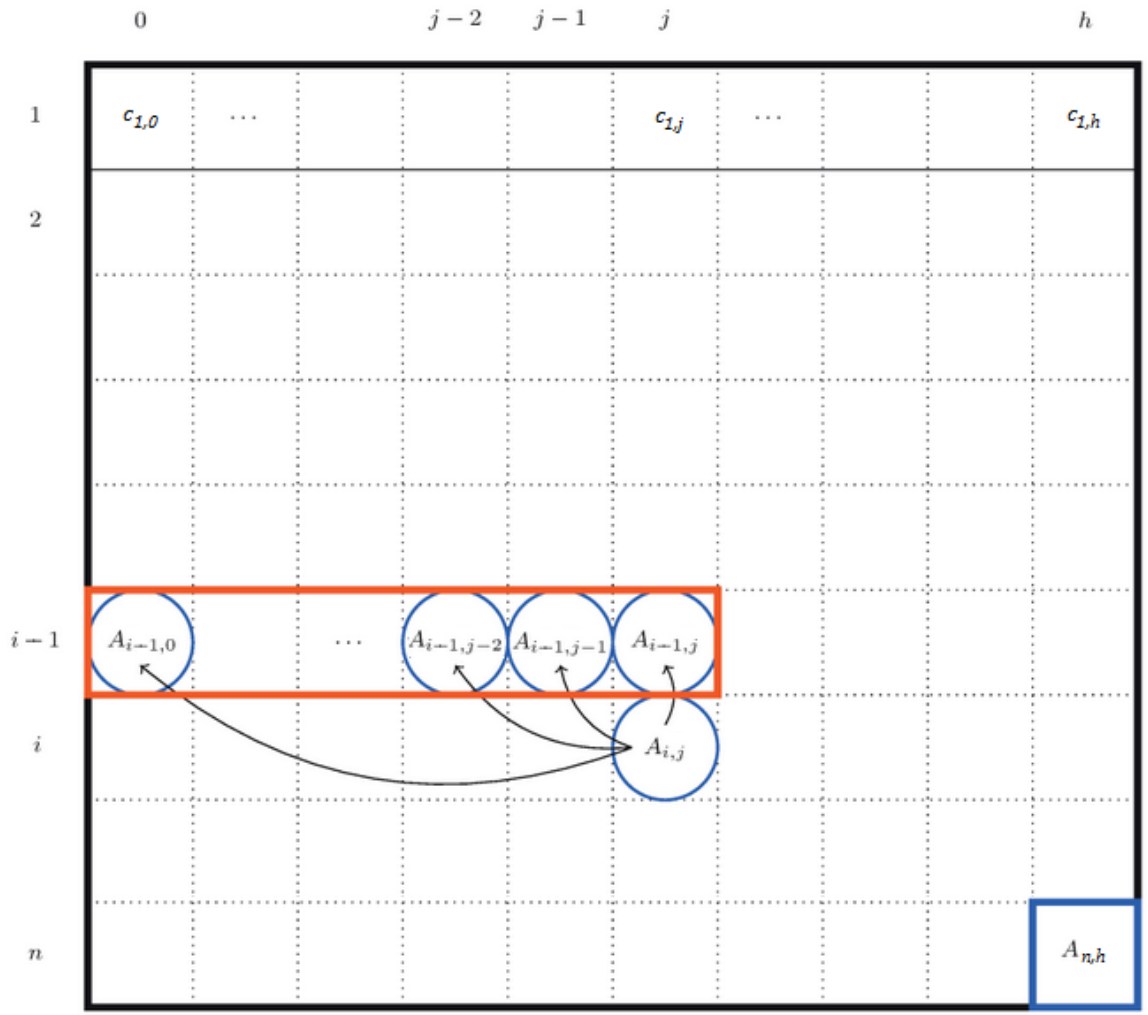
$$\sum_{i=1}^{n} A'_i + A_n = H \qquad\qquad \sum_{i=1}^{n} c_{i,\,A'_i} = C' + c_{n,\,A_n} > C^*$$

Dado que habíamos dicho que C^* era máximo, no puede pasar que haya una solución $C' + c_{n,\,A_n} > C^*$, es una contradicción, por lo que el problema exhibe la propiedad de subestructura óptima.

- Con todo lo anterior podríamos ya formular la ecuación de Bellman:

$$A_{i,j}\begin{cases} c_{i,j} & \text{si } i = 1 \\ \max_{0\le k\le j}\{A_{i-1,j-k} + c_{i,k}\} & \text{si } i > 1 \end{cases}$$

El cálculo de la recurrencia debe apoyarse en una estructura de datos bidimensional con filas indexadas entre 1 y n (las diferentes asignaturas posibles) y columnas indexadas entre 0 y H (máximo de horas posible a dedicar). De acuerdo con las mismas, debemos rellenar la tabla empezando por la fila 1 e ir descendiendo hasta la fila n, completando cada fila de izquierda a derecha. De esta forma, el resultado de la calificación máxima está en la posición $A_{n,H}$. A la vez se rellena otra tabla *D* en la que se almacena el número de horas que se utiliza para cada asignatura.



Observamos que el tamaño de la tabla es $\Theta(nH)$ y cada posición requiere computar el máximo entre $\Theta(H)$ valores, por lo que la complejidad temporal es $\Theta(nH^2)$.

- Cómputo de la calificación máxima y de la tabla auxiliar de horas dedicadas:

Input: arrays c (entero) [1...n]
Output: tabla D (real) [1...n,0...H]
Data: enteros i, j, k, tabla A (real) [1...n,0...H]

for $j = 0$ to H **do**
    $A_{0,j} = c_{0,j}$;
    $D_{0,j} = j$;
for $i = 1$ to n **do**
    for $j = 0$ to H **do**
        $A_{i,j} = c_{i,j} + A_{i-1,j}$;
        $D_{i,j} = 0$;
        for $k = 0$ to j **do**
            $aux = c_{i,k} + A_{i-1,j-k}$
            if $aux > A_{i,j}$ **then**
                $A_{i,j} = aux$;
                $D_{i,j} = k$;

Para reconstruir la solución óptima vamos directamente a la tabla *D*, empezando en la posición (n, H), y vamos metiendo en el vector resultado desde n hasta que i < 1.

D	0	1	2	3	4	5	6	7
1	0	1	2	3	4	5	6	7
2	0	1	1	1	1	1	1	3
3	0	0	1	1	1	1	1	1
4	0	0	0	0	1	2	3	3

Input: tabla D (real) [1...n,0... M]
Output: array s (entero) [1...n] **Data**: enteros i, j

$i = n; \quad j = H;$
while $i > 0$ **do**
    $s_i = D_{i,j}$;
    $j = j - D_{i,j}$;
    $i = i - 1$;

5. A lo largo de un río en el que sólo se puede navegar en una dirección se encuentran n embarcaderos. Conocemos el coste T_{ij} para ir directamente del embarcadero i al embarcadero j (y que puede ser más barato o más caro que un recorrido entre i y j con paradas intermedias). Se desea encontrar el coste mínimo para viajar entre cualesquiera embarcaderos i y j.

Ti,j	1	2	3	4	5
1	0	4	3	5	8
2	-	0	1	2	5
3	-	-	0	2	4
4	-	-	-	0	1
5	-	-	-	-	0

- Como con todo, lo primero es ponernos un ejemplo.
  Imaginemos que tenemos n=5 embarcaderos, y tenemos un coste T_{ij} que representa ir del embarcadero i al embarcadero j.

  Pero, ¿cómo viene dada esta información?
- Lo que se nos debe ocurrir es que esta información la tenemos almacenada en una tabla cuyas dimensiones son n x n.

  Bien, pero, ¿y los datos?
- Pues tendremos que si quiero ir del embarcadero i al i, ya estaré en él, por lo que no tiene coste y por tanto $T_{ii} = 0$.
- Si quiero ir del embarcadero i al j, pero i > j, no puedo, porque solo se puede navegar en una dirección.
- Por último, en cualquier otro caso, si tiene un coste, y como es un ejemplo pues nos los inventamos.

5. A lo largo de un río en el que sólo se puede navegar en una dirección se encuentran n embarcaderos. Conocemos el coste T_{ij} para ir directamente del embarcadero i al embarcadero j (y que puede ser más barato o más caro que un recorrido entre i y j con paradas intermedias). Se desea encontrar el coste mínimo para viajar entre cualesquiera embarcaderos i y j.

Ti,j	1	2	3	4	5
1	0	4	3	5	8
2	-	0	1	2	5
3	-	-	0	2	4
4	-	-	-	0	1
5	-	-	-	-	0

C	1	2	3	4	5
1					
2					
3					
4					
5					

Dado el coste de cada viaje, lo más factible es crear otra tabla similar con filas y columnas que también representan los embarcaderos, pero ahora en cada celda [i,j] estará el valor del coste mínimo para llegar a ese embarcadero j desde el correspondiente embarcadero i, pudiendo haber pasado por los i-1 embarcaderos anteriores. Cada celda será representada por C[i, j].

5. A lo largo de un río en el que sólo se puede navegar en una dirección se encuentran n embarcaderos. Conocemos el coste T_{ij} para ir directamente del embarcadero i al embarcadero j (y que puede ser más barato o más caro que un recorrido entre i y j con paradas intermedias). Se desea encontrar el coste mínimo para viajar entre cualesquiera embarcaderos i y j.

Ti,j	1	2	3	4	5
1	0	4	3	5	8
2	-	0	1	2	5
3	-	-	0	2	4
4	-	-	-	0	1
5	-	-	-	-	0

C	1	2	3	4	5
1	0				
2	-	0			
3	-	-	0		
4	-	-	-	0	
5	-	-	-	-	0

Al igual que en el coste, los casos base son simples.

Si queremos ir del embarcadero i al embarcadero j, pero i=j, ya estamos en el y el camino mínimo será 0.

Y por supuesto, al ser un río con una dirección, no podemos ir entre los embarcaderos i y j, si i > j.

5. A lo largo de un río en el que sólo se puede navegar en una dirección se encuentran n embarcaderos. Conocemos el coste T_{ij} para ir directamente del embarcadero i al embarcadero j (y que puede ser más barato o más caro que un recorrido entre i y j con paradas intermedias). Se desea encontrar el coste mínimo para viajar entre cualesquiera embarcaderos i y j.

Ti,j	1	2	3	4	5
1	0	4	3	5	8
2	-	0	1	2	5
3	-	-	0	2	4
4	-	-	-	0	1
5	-	-	-	-	0

C	1	2	3	4	5
1	0				
2	-	0			
3	-	-	0		
4	-	-	-	0	1
5	-	-	-	-	0

¿Ahora como procedemos?

- Al revés que en la mayoría de problemas, aquí vamos a empezar a rellenar la tabla de abajo a arriba.
- Así pues, empezamos a rellenar con i=4, el coste de ir desde este embarcadero al último es el mismo que el coste de la tabla T. De forma que C[4,5] = T[4,5].

5. A lo largo de un río en el que sólo se puede navegar en una dirección se encuentran n embarcaderos. Conocemos el coste T_{ij} para ir directamente del embarcadero i al embarcadero j (y que puede ser más barato o más caro que un recorrido entre i y j con paradas intermedias). Se desea encontrar el coste mínimo para viajar entre cualesquiera embarcaderos i y j.

Ti,j	1	2	3	4	5
1	0	4	3	5	8
2	-	0	1	2	5
3	-	-	0	2	4
4	-	-	-	0	1
5	-	-	-	-	0

C	1	2	3	4	5
1	0				
2	-	0			
3	-	-	0	2	3
4	-	-	-	0	1
5	-	-	-	-	0

- Seguimos con i = 3, de forma que para ir al embarcadero 4 será el coste directo de T[3,4]. Si bien, si lo queremos hacer recursivo, es en realidad C[3,4] = T[3,4] + C[4,4].
- Para ir del embarcadero 3 al 5, ya tenemos dos opciones, o pasamos por el embarcadero 4 o no, de forma que tendremos que escoger entre T[3,5] + C[5,5] (4 + 0 = 4) o T[3,4] + C[4,5] (2 + 1 = 3).

5. A lo largo de un río en el que sólo se puede navegar en una dirección se encuentran n embarcaderos. Conocemos el coste T_{ij} para ir directamente del embarcadero i al embarcadero j (y que puede ser más barato o más caro que un recorrido entre i y j con paradas intermedias). Se desea encontrar el coste mínimo para viajar entre cualesquiera embarcaderos i y j.

Ti,j	1	2	3	4	5
1	0	4	3	5	8
2	-	0	1	2	5
3	-	-	0	2	4
4	-	-	-	0	1
5	-	-	-	-	0

C	1	2	3	4	5
1	0				
2	-	0	1	2	3
3	-	-	0	2	3
4	-	-	-	0	1
5	-	-	-	-	0

- Seguimos con i = 2, e ir al embarcadero 3 será C[2,3] = T[2,3] + C[3,3], e ir del embarcadero 2 al 4, tenemos dos opciones, T[2,4] + C[4,4] (2 + 0 = 2) o T[2,3] + C[3,4] (1 + 2 = 3).
- Para ir del embarcadero 2 al 5, tenemos tres opciones, pasar directamente, pasar por el 4 o pasar por el 3, de forma que tendremos que escoger entre T[2,5] + C[5,5], T[2,4] + C[4,5] o T[2,3] + C[3,5].

5. A lo largo de un río en el que sólo se puede navegar en una dirección se encuentran n embarcaderos. Conocemos el coste T_{ij} para ir directamente del embarcadero i al embarcadero j (y que puede ser más barato o más caro que un recorrido entre i y j con paradas intermedias). Se desea encontrar el coste mínimo para viajar entre cualesquiera embarcaderos i y j.

Ti,j	1	2	3	4	5
1	0	4	3	5	8
2	-	0	1	2	5
3	-	-	0	2	4
4	-	-	-	0	1
5	-	-	-	-	0

C	1	2	3	4	5
1	0	4	3	5	6
2	-	0	1	2	3
3	-	-	0	2	3
4	-	-	-	0	1
5	-	-	-	-	0

- Por último tenemos el embarcadero i=1, y como ya nos habremos dado cuenta, cada una de las celdas de la tabla va a venir dada por la siguiente fórmula:

$$C_{i,j} = \min_{i<k\leq j}\{T_{i,k} + C_{k,j}\}$$

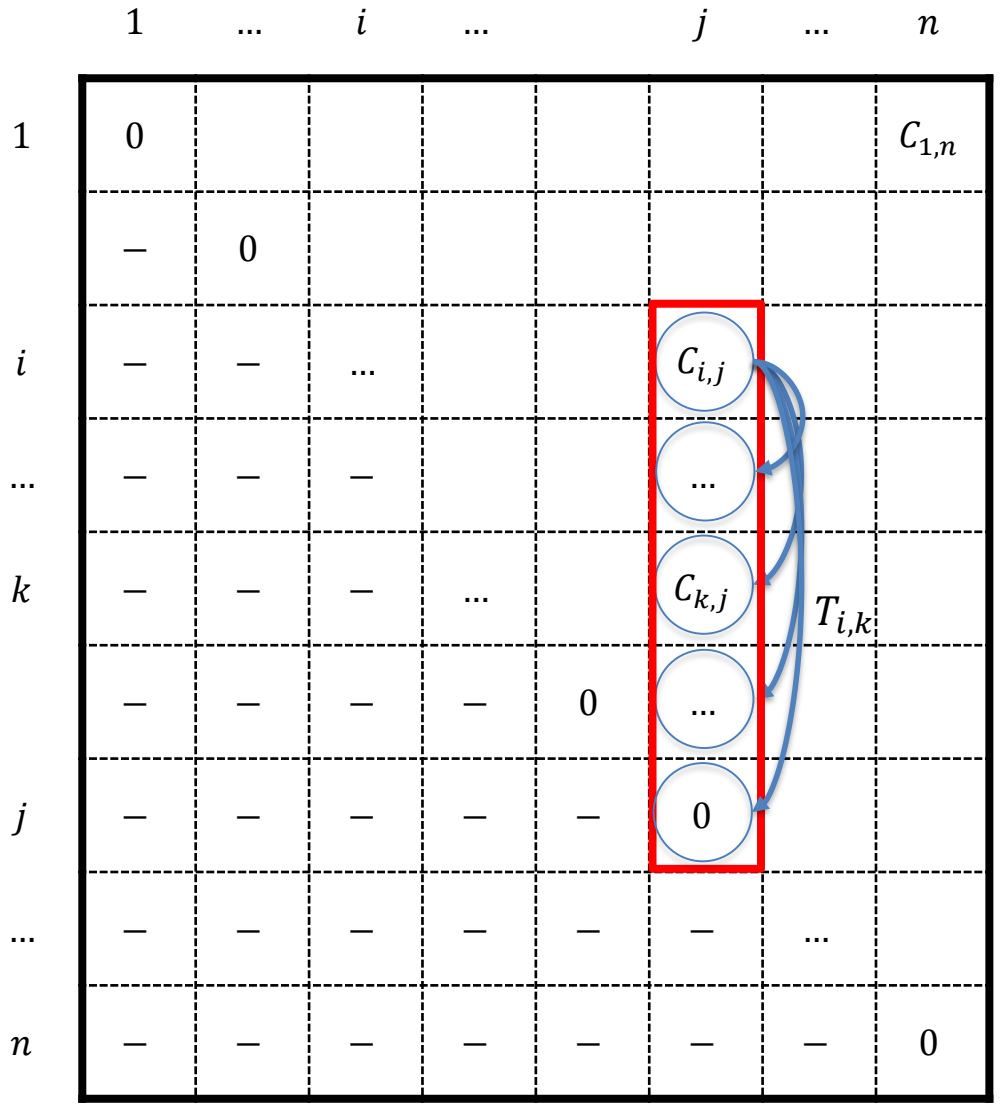
- En esta última ecuación se contempla el viaje directo, que corresponde al caso en el que k coincide con j. Esta ecuación verifica también que la solución buscada C[i,j] satisface el principio del óptimo, pues el coste C[k,j], que forma parte de la solución, ha de ser, a su vez, óptimo.

- Con todo lo anterior podríamos ya formular la ecuación de Bellman:

$$C_{i,j} = \begin{cases} 0 & \text{si } i = j \\ \min_{i<k\leq j}\{T_{i,k} + C_{k,j}\} & \text{si } i < j \end{cases}$$

El cálculo de la recurrencia debe apoyarse en una estructura de datos bidimensional con filas y columnas indexadas entre 1 y n. De acuerdo con las mismas, debemos rellenar la tabla empezando por la fila n e ir ascendiendo hasta la fila 1, completando cada fila de izquierda a derecha. De esta forma, el resultado está en la posición $C_{1,n}$.

Observamos que el tamaño de la tabla es $\Theta(n^2)$, pero la complejidad temporal tiene en cuenta además la búsqueda de $i < k \leq j$, por lo que es $\Theta(n^3)$.



6. Como oficial de comunicaciones de un crucero de combate, su labor es la de gestionar las comunicaciones de la manera más eficiente. Supóngase que se quiere transmitir un mensaje $S = s_1 \ldots s_m$ dado como una cadena de m símbolos. A tal fin, dispone de r códigos distintos. Sea b_{ij} el numero de bits necesarios para codificar el i-ésimo símbolo en el j-ésimo código. Inicialmente el trasmisor del puente esta fijado al código #1 pero puede ser cambiado durante la transmisión cuando se desee. Para ello se necesita enviar un código de control compuesto de C_{ij} bits si se desea cambiar del código i actual al código j. Su objetivo es determinar como enviar el mensaje empleando el número mínimo de bits.

La solución al problema será un vector $E = \langle e_1, \ldots, e_m \rangle$, donde $e_i \in \{1, \ldots, r\}$ indica el código empleado para transmitir el i-ésimo símbolo del mensaje. Las decisiones a las que nos enfrentamos son por lo tanto múltiples: en cada momento hay que elegir qué código de entre los disponibles debemos emplear.

La longitud en bits asociada a esta solución será:

$$B(E) = \sum_{i=1}^{m} \left(C_{e_{i-1},e_i} + b_{\sigma(s_i),e_i} \right)$$

donde $\sigma : \Sigma \rightarrow \mathbb{N}$ es el índice de un símbolo en el alfabeto. Dicha longitud corresponde a la codificación de cada símbolo en el código correspondiente ($b_{\sigma(s_i),e_i}$) más los cambios de código efectuados (C_{e_{i-1},e_i}).

A efectos de este último término suponemos que $e_0 = 1$ es el código inicial y que $C_{i,i} = 0$ para todo i, esto es, cuando no se cambia de código no hay que transmitir ningún bit adicional.

Supongamos que conocemos la solución óptima $E^* = \langle e_1^*, \dots, e_m^* \rangle$. Consideremos ahora e_i^*, la i-ésima decisión de esta solución óptima. La longitud en bits de la solución se podrá expresar como:

$$B(\langle e_1^*, \dots, e_m^* \rangle) = \underbrace{\sum_{k=1}^{i} \left(C_{e_{k-1}^*, e_k^*} + b_{\sigma(s_k), e_k^*} \right)}_{i \text{ primeros símbolos}} + \underbrace{\sum_{k=i+1}^{m} \left(C_{e_{k-1}^*, e_k^*} + b_{\sigma(s_k), e_k^*} \right)}_{m - i \text{ últimos símbolos}}$$

Cada uno de los términos se pueden caracterizar mediante funciones auxiliares $B_{i \to k} : \mathbb{N} \times \mathbb{N}^{k-i+1} \to \mathbb{N}$ definidas:

$$B_{i \to k}(e, \langle e_1^*, \dots, e_m^* \rangle) = C_{e, e_i} + b_{\sigma(s_i), e_i} + \sum_{l=i+1}^{k} \left(C_{e_{l-1}, e_l} + b_{\sigma(s_l), e_l} \right)$$

Así, tenemos que: $B(E^*) = B_{1 \to m}(e_0, \langle e_1^*, \dots, e_m^* \rangle) = B_{1 \to i}(e_0, \langle e_1^*, \dots, e_i^* \rangle) + B_{(i+1) \to m}(e_i, \langle e_{i+1}^*, \dots, e_m^* \rangle)$

Es fácil demostrar que $E_{(i+1) \to m}^* = \langle e_{i+1}^*, \dots, e_m^* \rangle$ es la solución óptima para transmitir los símbolos $s_{i+1} \dots s_m$ partiendo del código inicial e_i^*, es decir, es la secuencia de códigos que minimiza $B_{(i+1) \to m}\left(e_i^*, E_{(i+1) \to m}^*\right)$. Si existiera otra solución $E'_{(i+1) \to m} = \langle e'_{i+1}, \dots, e'_m \rangle$ mejor en dichas condiciones, esto es,

$$B_{(i+1) \to m}(e_i^*, E') < B_{(i+1) \to m}\left(e_i^*, E_{(i+1) \to m}^*\right)$$

entonces podríamos construir $E'' = \langle e_1^*, \dots, e_i^*, e'_{i+1}, e'_m \rangle$ que implicaría una longitud total

$$B(E'') = B_{1 \to i}(e_0, \langle e_1^*, \dots, e_i^* \rangle) + B_{(i+1) \to m}(e_i^*, \langle e'_{i+1}, \dots, e'_m \rangle) < B(E^*)$$

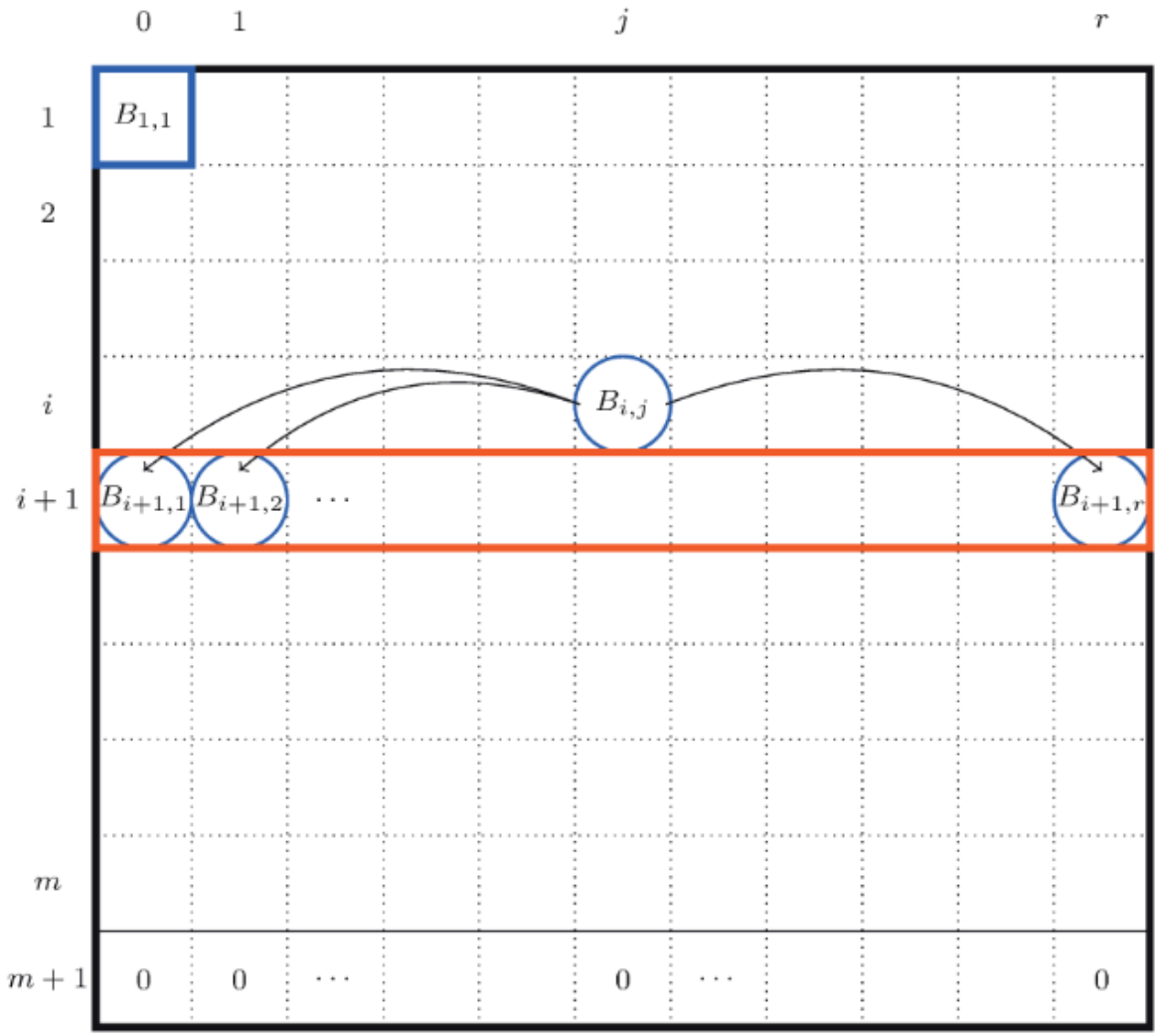
lo que es imposible ya que habíamos supuesto que E^* era la solución óptima. Por lo tanto no existe E' mejor, y se exhibe la propiedad de subestructura óptima, siendo únicamente necesario considerar el código activo en cada momento.

$B_{i,j}$ = número mínimo de bits necesario para codificar los símbolos $s_i \ldots s_m$, partiendo del código inicial j.

Estos subproblemas tienen una solución trivial cuando el mensaje tiene una longitud nula ($i > m$). En dicho caso la longitud del mensaje es 0. En el caso general ($i \leq m$) habrá que tomar una decisión sobre cuál de entre todos los códigos disponibles se elige, teniendo en cuenta que usar el código k-ésimo tendrá un coste $C_{j,k} + b_{\sigma(s_i),k}$ correspondiente al cambio de código (si lo hay) y la codificación del símbolo i-ésimo, y el resto del mensaje requerirá $B_{i+1,k}$ bits. La ecuación de Bellman correspondiente es por lo tanto:

$$B_{i,j} = \begin{cases} 0 & \text{si } i > m \\ \min_{1 \leq k \leq r} \{C_{j,k} + b_{\sigma(s_i),k} + B_{i+1,k}\} & \text{si } i \leq m \end{cases}$$

El cálculo de la recurrencia debe apoyarse en una estructura de datos bidimensional con filas entre 1 y m+1 (longitud del mensaje más una fila para actuar de centinela ante la terminación del mismo) y columnas entre 1 y r (los códigos disponibles). La tabla debe rellenarse desde abajo hacia arriba, siendo indiferente el sentido de izq y der.



Dado que la tabla tiene un tamaño $O(mr)$ y cada posición requiere un cómputo de complejidad $\Theta(r)$, la complejidad temporal completa del proceso es $\Theta(mr^2)$. Para la reconstrucción de la solución es conveniente emplear una tabla adicional en la que ir almacenando la decisión óptima para cada subproblema, la cuál se empleará para recorrer hacia atrás el camino óptimo sobre la tabla B, empezando en $B_{1,1}$.

Input: array s (carácter) [1...m], tabla b (entero) [1...n,1...r], tabla C' (entero) [1...r,1...r]

Output: longitud óptima L (entero)

Data: tabla B (entero) [1...m+1,1...r], tabla D (entero) (1...m,l...r]

for $j = 1$ to r **do**
    $B_{m+1,j} = 0;$
for $i = m$ down to 1 **do**
    for $j = 1$ to r **do**
        $B_{i,j} = \infty;$
        for $k = 1$ to r **do**
            $q = C_{j,k} + b_{\sigma(s_i),k} + B_{i+1,k}$
            if $B_{i,j} > q$ **then**
                $B_{i,j} = q;$
                $D_{i,j} = k;$
$L = B_{1,1};$

Reconstrucción de la secuencia óptima

Input: tabla D (real) [1...m,1... r]

Output: array E (entero) [1...m] con la secuencia de códigos

Data: enteros i, j

$j = 1;$
for $i = 1$ to m **do**
    $E_i = D_{i,j};$
    $j = E_i;$

[Enero 2020] Dado un vector de números naturales $[a_1,\cdots,a_n]$, donde cada elemento a_i representa el número máximo de pasos que se pueden avanzar desde ese elemento, diseñar e implementar un algoritmo para devolver el número mínimo de saltos para llegar al final del vector (comenzando desde el primer elemento). Si un elemento es 0, entonces no puede moverse a través de ese elemento.

Por ejemplo, si el vector es [1,3,5,9,4,3,8,1,2,6,9] la salida podría ser: 3 (saltar de la posición #1 a la posición #2, de ahí a la posición #4 y de ahí a la última posición).

Este ejercicio es similar al problema del embarcadero.

De forma que tenemos una tabla n x n, en la cual la diagonal está rellena de valores $T[i,i]=0$, ya que ir desde la posición i a sí misma no conlleva gasto ninguno.

Sin embargo, si hay una diferencia, y es que para ir desde la posición i a la j tenemos que ver si el valor del vector $V[i]$ es mayor que la posición j, es decir,

Si $i=3$ tenemos $V[3]=5$, y $5+3=8$ por lo que si queremos ir desde $i=3$ hasta $j=6$ podemos usar este salto como único ya que con él podremos ir como máximo a $j=8$.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	0										
2	-	0									
3	-	-	0								
4	-	-	-	0							
5	-	-	-	-	0						
6	-	-	-	-	-	0					
7	-	-	-	-	-	-	0				
8	-	-	-	-	-	-	-	0			
9	-	-	-	-	-	-	-	-	0		
10	-	-	-	-	-	-	-	-	-	0	
11	-	-	-	-	-	-	-	-	-	-	0

[Enero 2020] Dado un vector de números naturales $[a_1,\cdots,a_n]$, donde cada elemento a_i representa el número máximo de pasos que se pueden avanzar desde ese elemento, diseñar e implementar un algoritmo para devolver el número mínimo de saltos para llegar al final del vector (comenzando desde el primer elemento). Si un elemento es 0, entonces no puede moverse a través de ese elemento.

Por ejemplo, si el vector es [1,3,5,9,4,3,8,1,2,6,9] la salida podría ser: 3 (saltar de la posición #1 a la posición #2, de ahí a la posición #4 y de ahí a la última posición).

Al igual que en el caso del embarcadero, tiene sentido ir rellenando la tabla desde abajo, así pues tendremos:

$T[10,11] = 1$ porque $V[10] = 6$ (máximo hasta $j = 16$)

$T[9,10] = 1$ porque $V[9] = 2$ (máximo hasta $j = 11$)
$T[9,11] = 1$ porque $V[9] = 2$ (máximo hasta $j = 11$)

$T[8,9] = 1$ porque $V[8] = 1$ (máximo hasta $j = 9$)
$T[8,10] = 2$ porque será ir a $j = 9$ y de aquí a $j = 10$
$T[8,11] = 2$ porque será ir a $j = 9$ y de aquí a $j = 11$

Ya se ve una dinámica en los resultados.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	0										
2	-	0									
3	-	-	0								
4	-	-	-	0							
5	-	-	-	-	0						
6	-	-	-	-	-	0					
7	-	-	-	-	-	-	0				
8	-	-	-	-	-	-	-	0	1	2	2
9	-	-	-	-	-	-	-	-	0	1	1
10	-	-	-	-	-	-	-	-	-	0	1
11	-	-	-	-	-	-	-	-	-	-	0

[Enero 2020] Dado un vector de números naturales $[a_1,\cdots,a_n]$, donde cada elemento a_i representa el número máximo de pasos que se pueden avanzar desde ese elemento, diseñar e implementar un algoritmo para devolver el número mínimo de saltos para llegar al final del vector (comenzando desde el primer elemento). Si un elemento es 0, entonces no puede moverse a través de ese elemento.

Por ejemplo, si el vector es [1,3,5,9,4,3,8,1,2,6,9] la salida podría ser: 3 (saltar de la posición #1 a la posición #2, de ahí a la posición #4 y de ahí a la última posición).

Seguimos:

$T[7,8] = 1$ porque $V[7] = 8$ (máximo hasta $j = 15$)
$T[7,9] = 1$ porque $V[7] = 8$ (máximo hasta $j = 15$)
$T[7,10] = 1$ porque $V[7] = 8$ (máximo hasta $j = 15$)
$T[7,11] = 1$ porque $V[7] = 8$ (máximo hasta $j = 15$)

$T[6,7] = 1$ porque $V[6] = 3$ (máximo hasta $j = 9$)
$T[6,8] = 1$ porque $V[6] = 3$ (máximo hasta $j = 9$)
$T[6,9] = 1$ porque $V[6] = 3$ (máximo hasta $j = 9$)
$T[6,10] = 2$ porque será ir a $j = 9$ y de aquí a $j = 10$
$T[6,11] = 2$ porque será ir a $j = 9$ y de aquí a $j = 11$

Seguimos con la misma dinámica en los resultados.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	0										
2	-	0									
3	-	-	0								
4	-	-	-	0							
5	-	-	-	-	0						
6	-	-	-	-	-	0	1	1	1	2	2
7	-	-	-	-	-	-	0	1	1	1	1
8	-	-	-	-	-	-	-	0	1	2	2
9	-	-	-	-	-	-	-	-	0	1	1
10	-	-	-	-	-	-	-	-	-	0	1
11	-	-	-	-	-	-	-	-	-	-	0

[Enero 2020] Dado un vector de números naturales $[a_1,\cdots,a_n]$, donde cada elemento a_i representa el número máximo de pasos que se pueden avanzar desde ese elemento, diseñar e implementar un algoritmo para devolver el número mínimo de saltos para llegar al final del vector (comenzando desde el primer elemento). Si un elemento es 0, entonces no puede moverse a través de ese elemento.

Por ejemplo, si el vector es [1,3,5,9,4,3,8,1,2,6,9] la salida podría ser: 3 (saltar de la posición #1 a la posición #2, de ahí a la posición #4 y de ahí a la última posición).

Seguimos:

$T[5,10] = 2$ porque será ir a $j = 9$ y de aquí a $j = 10$,
si bien también puede ser ir a $j = 7$ y de aquí a $j = 10$
$T[5,11] = 2$ porque será ir a $j = 9$ y de aquí a $j = 10$,
si bien también puede ser ir a $j = 7$ y de aquí a $j = 10$

$T[4,11] = 1$ porque $V[4] = 9$ (máximo hasta $j = 13$)

$T[1,3] = 2$ porque será ir a $j = 2$ y de aquí a $j = 3$
$T[1,4] = 2$ porque será ir a $j = 2$ y de aquí a $j = 4$

$T[i,j] = \min_{i<k\leq i+V[i]}\{1 + T[k,j]\}$ si i $+ V[i] < j$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	0	1	2	2	2	3	3	3	3	3	3
2	-	0	1	1	1	2	2	2	2	2	2
3	-	-	0	1	1	1	1	1	2	2	2
4	-	-	-	0	1	1	1	1	1	1	1
5	-	-	-	-	0	1	1	1	1	2	2
6	-	-	-	-	-	0	1	1	1	2	2
7	-	-	-	-	-	-	0	1	1	1	1
8	-	-	-	-	-	-	-	0	1	2	2
9	-	-	-	-	-	-	-	-	0	1	1
10	-	-	-	-	-	-	-	-	-	0	1
11	-	-	-	-	-	-	-	-	-	-	0

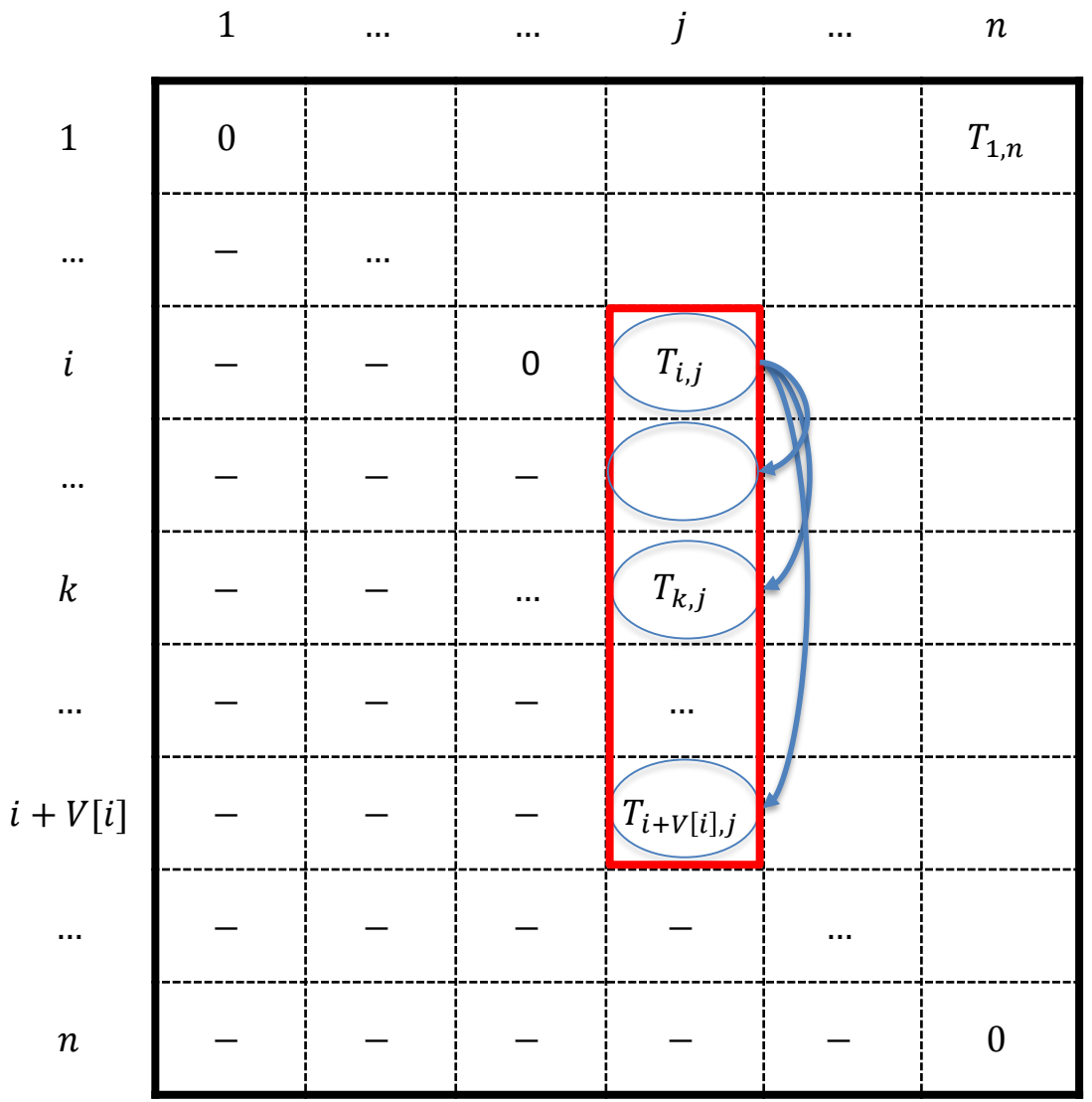
- Esta ecuación verifica que la solución buscada T[i,j] satisface el principio del óptimo, pues el coste T[k,j] siendo k como máximo i+V[i], que forma parte de la solución, ha de ser, a su vez, óptimo.

- Con todo lo anterior podríamos ya formular la ecuación de Bellman:

$$T[i,j] = \begin{cases} 0 & \text{si } i = j \\ 1 & \text{si } i + V[i] \geq j \\ \min_{i<k\leq i+V[i]}\{1 + T[k,j]\} & \text{si } i + V[i] < j \end{cases}$$

El cálculo de la recurrencia debe apoyarse en una estructura de datos bidimensional con filas y columnas indexadas entre 1 y n. De acuerdo con las mismas, debemos rellenar la tabla empezando por la fila n e ir ascendiendo hasta la fila 1, completando cada fila de izquierda a derecha. De esta forma, el resultado está en la posición $T_{1,n}$.

Observamos que el tamaño de la tabla es $\Theta(n^2)$, pero la complejidad temporal tiene en cuenta además la búsqueda de $i < k \leq i + V[i]$, por lo que es $\Theta(n^3)$.



[Ejercicio Voluntario] Vd. es el director de logística de una empresa de componentes informáticos, y su trabajo es determinar al final de cada semana cómo se deben hacer llegar los componentes manufacturados de la fábrica al distribuidor. Para ello tiene dos opciones con su compaña de transportes: pagar en función del peso transportado, cargando K euros/kg, o pagar una tarifa plana de T euros semanales con independencia del peso. En este segundo caso, el contrato de tarifa plana debe hacerse durante s semanas consecutivas. Sabiendo que la producción de la fábrica es de p_i kg durante la semana i, $1 \leq i \leq n$, el objetivo es encontrar la forma menos costosa de distribuir los componentes.

Se pide:

a) Demostrar que el problema exhibe la propiedad de subestructura óptima.

b) Plantear la ecuación de Bellman para el coste del transporte.

c) Confeccionar un algoritmo de programación dinámica *bottom-up* para resolver el problema.

d) Determinar la complejidad del algoritmo anterior en tiempo y en espacio.

e) Aplicar a mano este algoritmo para resolver la instancia determinada por $K = 1$, $T = 5$, $s = 3$, y

$p_i \in \langle 4, 10, 3, 5, 1, 6, 4, 4 \rangle$. Se desea conocer el coste total óptimo, así como la solución que lo produce.