



PROBLEMA DE LA MOCHILA 0/1 (CON PROGRAMACIÓN DINÁMICA)

CONTINUACIÓN

Programación 3
Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

W = 2, 3, 4, 5

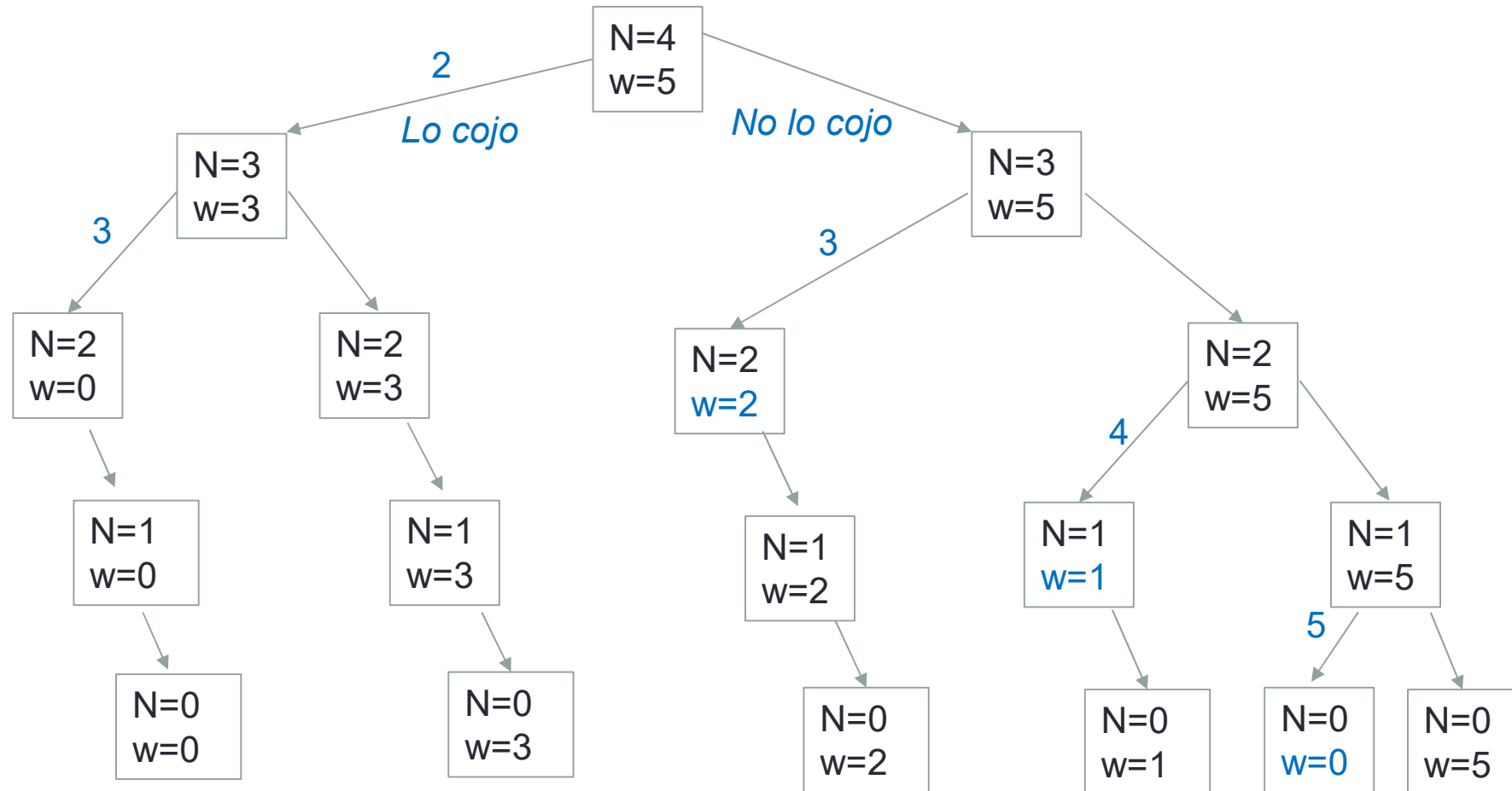
B = 3, 4, 5, 6

N = 4, 3, 2, 1

Peso

Beneficio

Debemos calcular la mejor combinación que tenga un peso máximo de 5 kilos

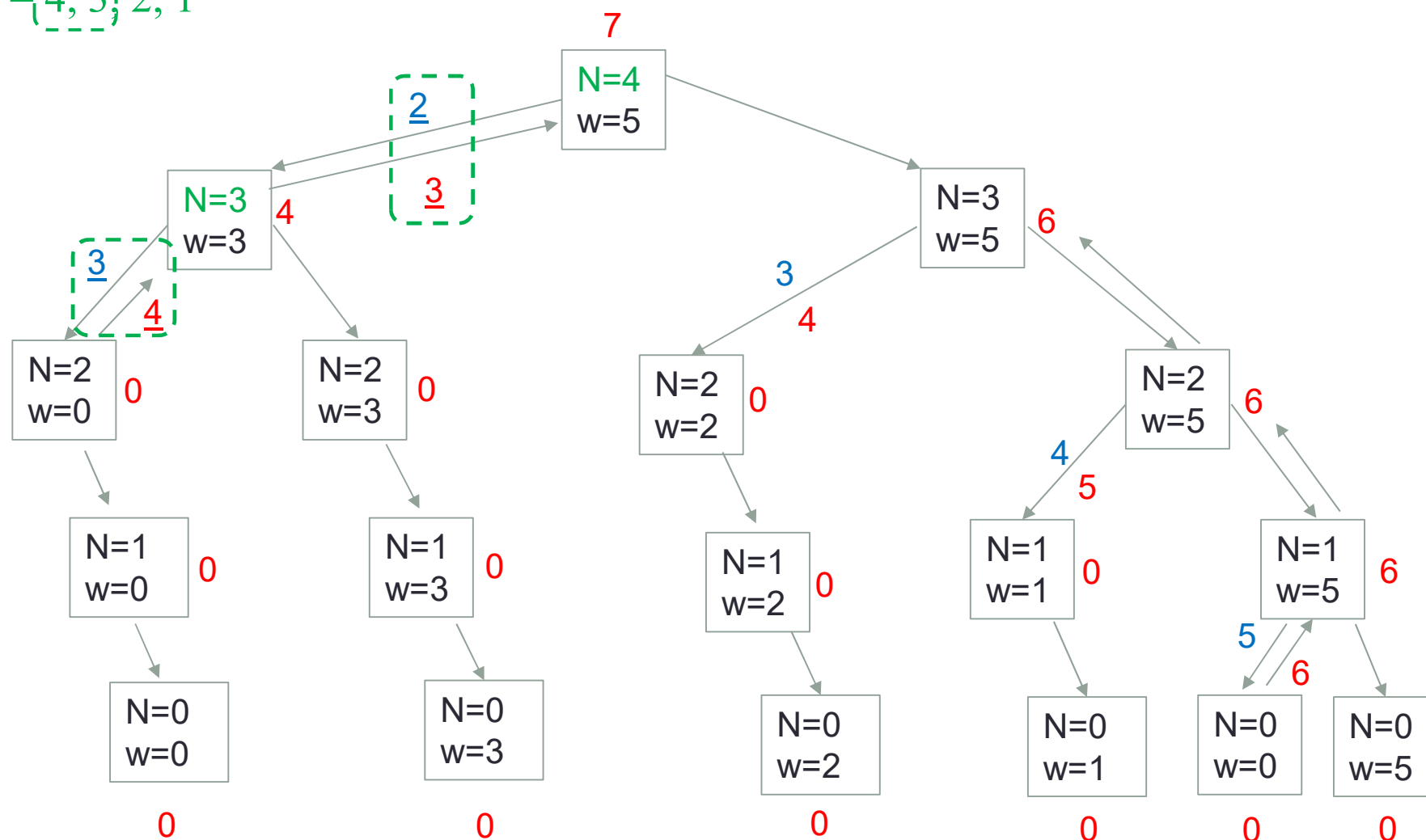


¡ Árbol completo de llamadas !

$W = \{ \underline{2}, \underline{3}, 4, 5 \}$
 $B = \{ 3, \underline{4}, 5, 6 \}$
 $N = \{ 4, 3, \underline{2}, 1 \}$

Peso
 Beneficio

Debemos calcular la mejor combinación que tenga un peso máximo de 5 kilos



Árbol de todas las posibles llamadas recursivas con su **beneficio**

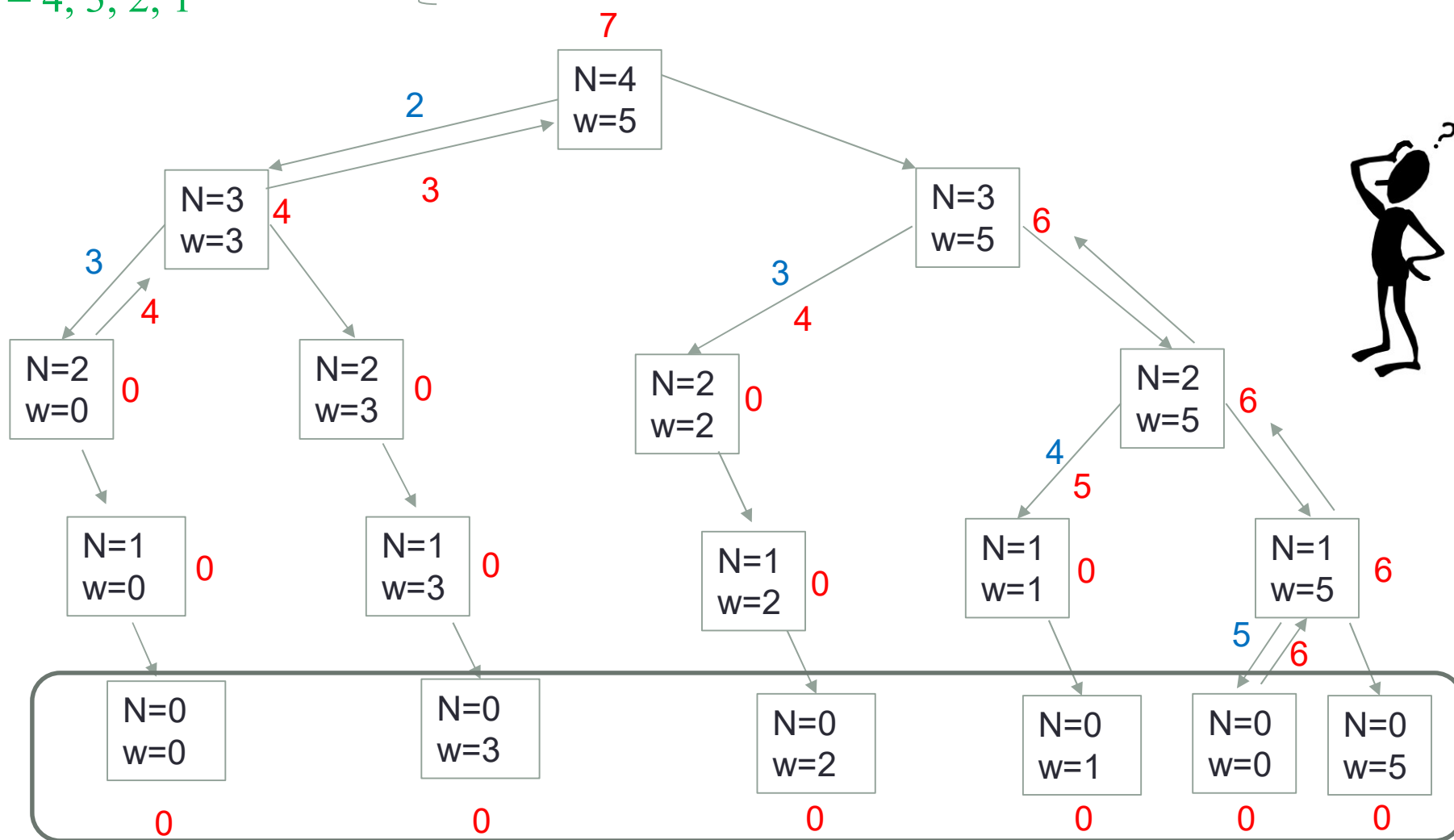
$W = 2, 3, 4, 5$

$B = 3, 4, 5, 6$

$N = 4, 3, 2, 1$

$t(n,w) \begin{cases} 0 \end{cases}$

: $n \leq 0$ ← *Caso Base*



$W = 2, 3, 4, 5$

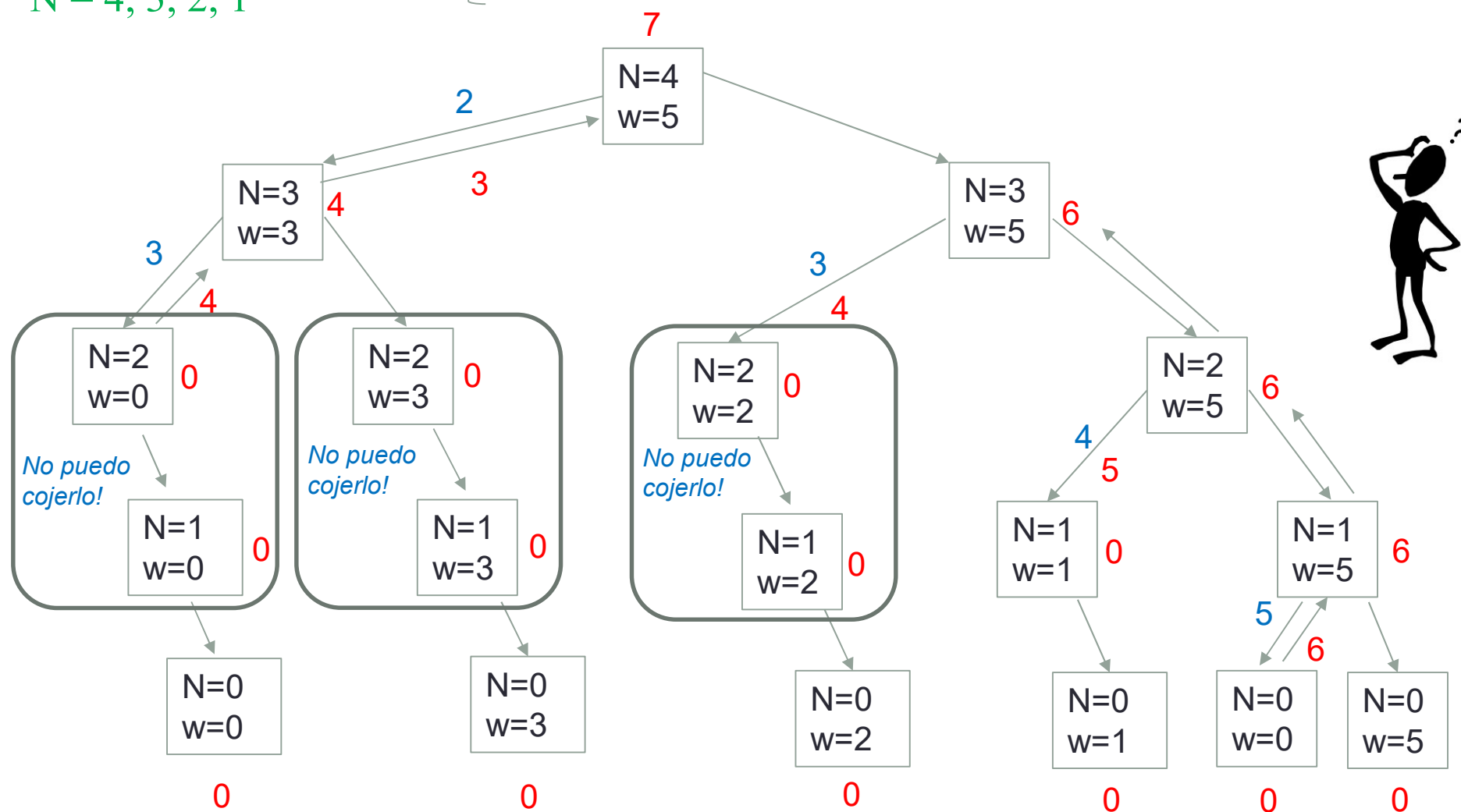
$B = 3, 4, 5, 6$

$N = 4, 3, 2, 1$

$$t(n, w) = \begin{cases} t(n-1, w) & : W_n > w \\ 0 & : n \leq 0 \end{cases}$$

$: W_n > w$

$: n \leq 0$



$W = 2, 3, 4, 5$

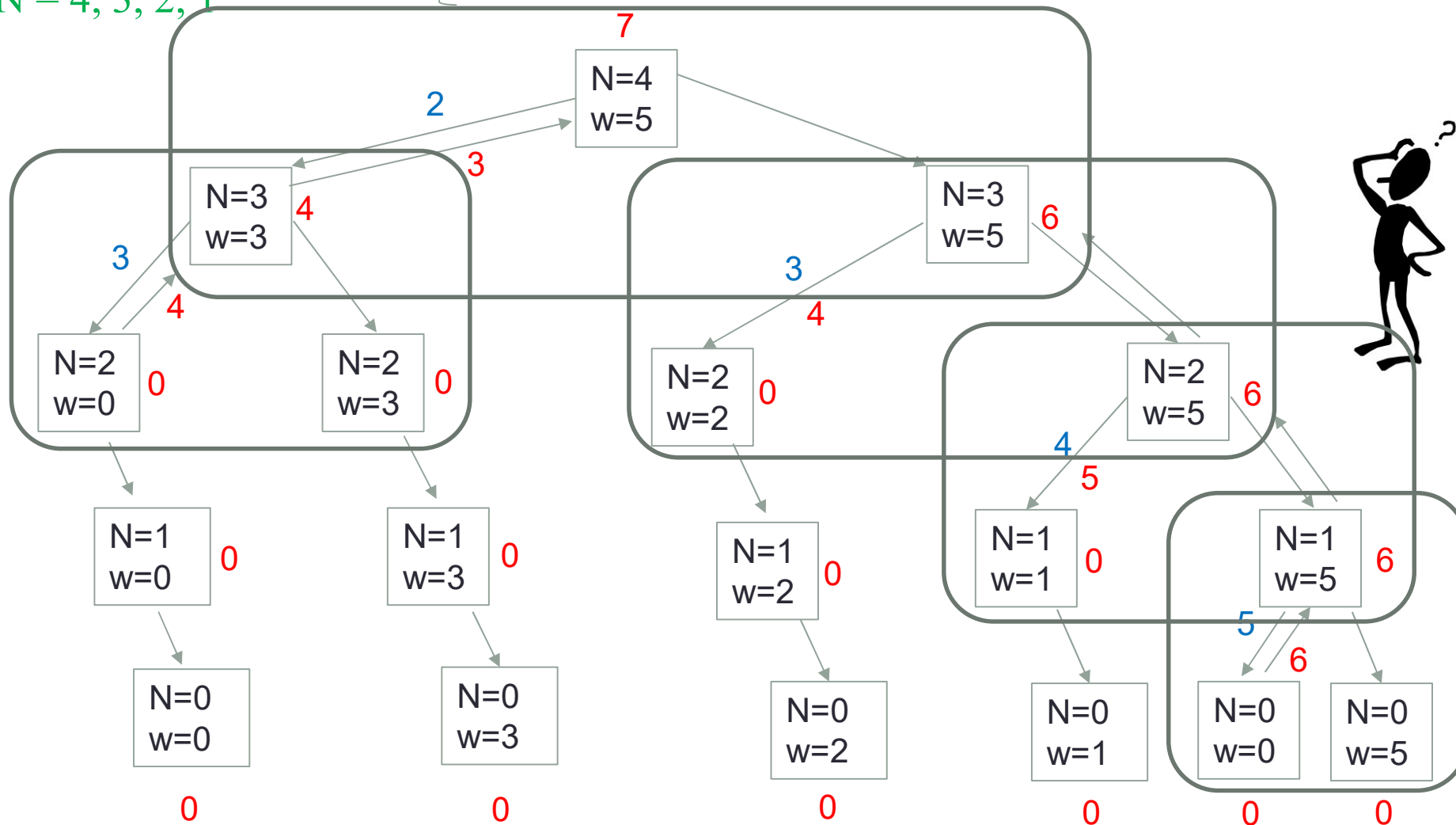
$B = 3, 4, 5, 6$

$N = 4, 3, 2, 1$

$$t(n, w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w - W_n) + B_n) & : n <= 0 \\ 0 & : n <= 0 \end{cases}$$

$: W_n > w$

$: n <= 0$



Lectura de la Recurrencia

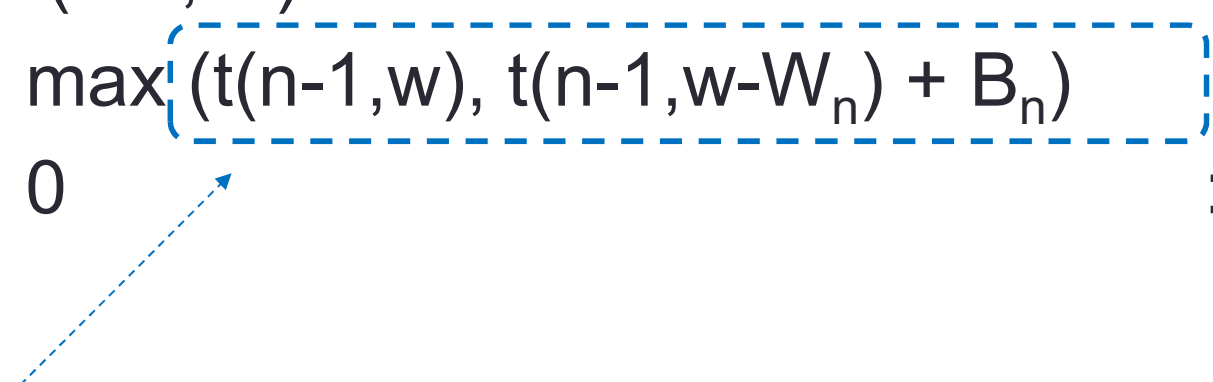
$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max (t(n-1,w), t(n-1,w-W_n) + B_n) & \\ 0 & : n \leq 0 \end{cases}$$

Lectura de la Recurrencia

$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max (t(n-1,w), t(n-1,w-W_n) + B_n) & \\ 0 & : n \leq 0 \end{cases}$$

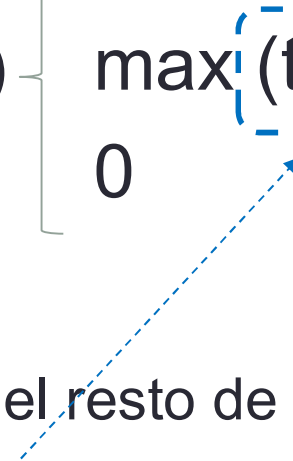
Si el peso del elemento n excede el peso máximo de la mochila, no podemos añadirlo y pasamos al siguiente

Lectura de la Recurrencia

$$t(n, w) \begin{cases} t(n-1, w) & : W_n > w \\ \max \{ (t(n-1, w), t(n-1, w - W_n) + B_n) \} & : n \leq 0 \\ 0 \end{cases}$$


En el resto de los casos, tenemos dos posibilidades:

Lectura de la Recurrencia

$$t(n, w) \begin{cases} t(n-1, w) & : W_n > w \\ \max \{ t(n-1, w), t(n-1, w - W_n) + B_n \} & \\ 0 & : n \leq 0 \end{cases}$$


En el resto de los casos, tenemos dos posibilidades:

1. No añadir el elemento n , con lo que el problema se reduce a calcular la mochila óptima sin tener en cuenta n

Lectura de la Recurrencia

$$t(n, w) \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w - W_n) + B_n) & : n \leq 0 \\ 0 \end{cases}$$

En el resto de los casos, tenemos dos posibilidades:

1. No añadir el elemento n , con lo que el problema se reduce a calcular la mochila óptima sin tener en cuenta n
2. Añadir n , con lo que ahora tenemos que calcular la mochila óptima restando su peso ($w - W_n$) pero teniendo en cuenta su beneficio (B_n)

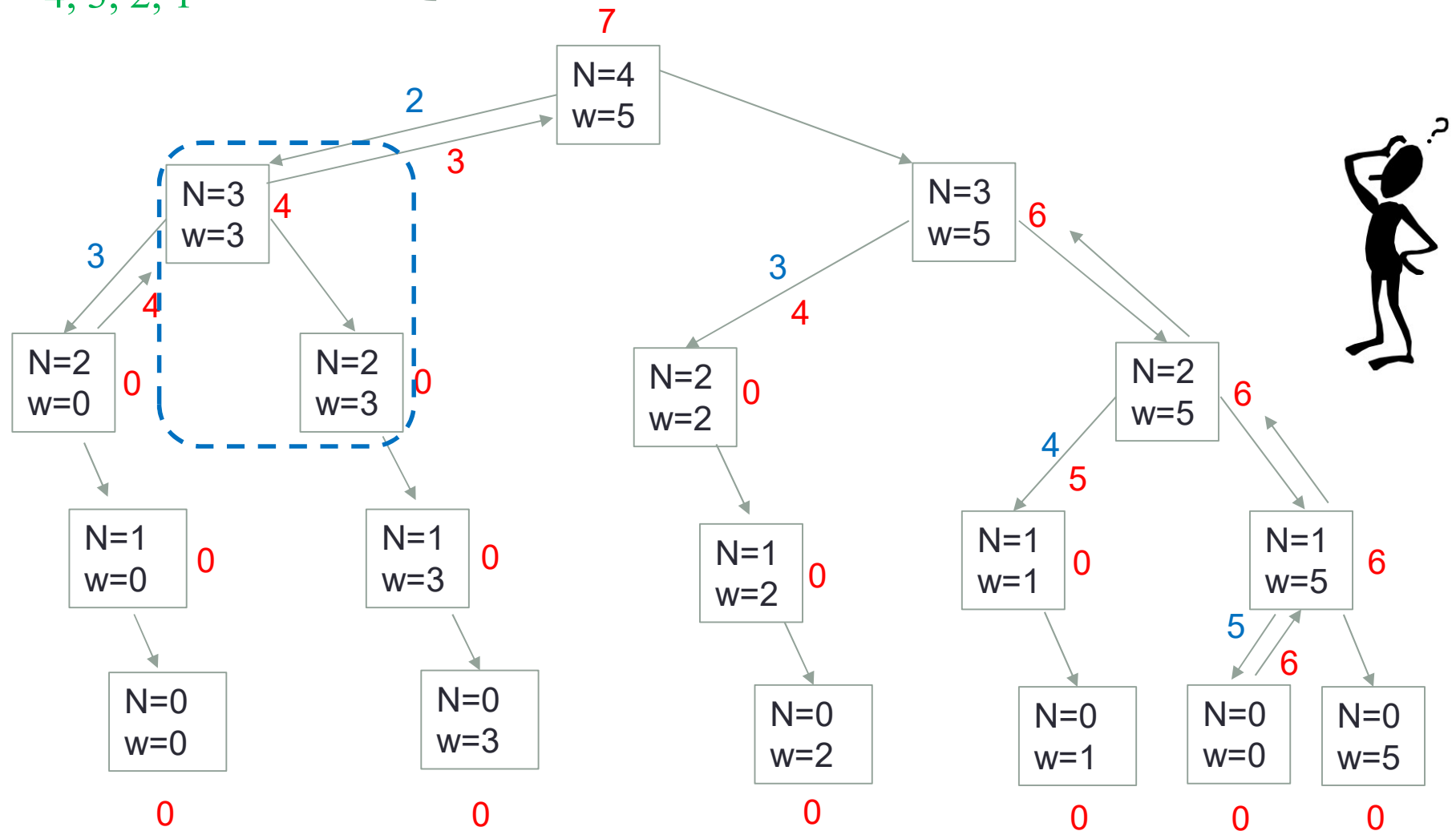
$B = 3, 4, 5, 6$

N = 4, 3, 2, 1

$$t(n,w) = \begin{cases} t(n-1, w) \\ \max \{ t(n-1, w), t(n-1, w-W_n) + B_n \} \\ 0 \end{cases}$$

$$: W_n > w$$

```
: n <= 0
```

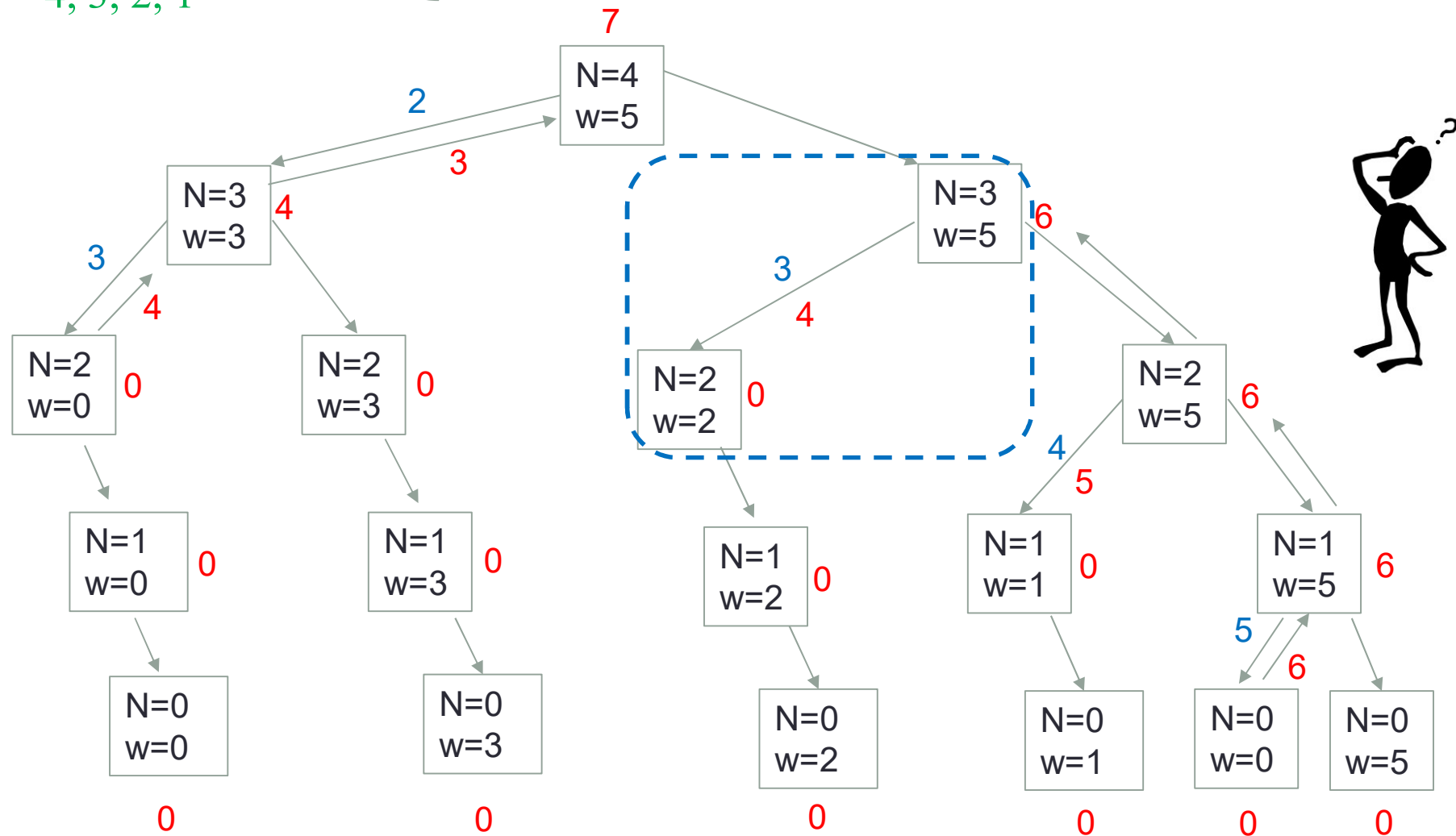


$W = 2, 3, 4, 5$

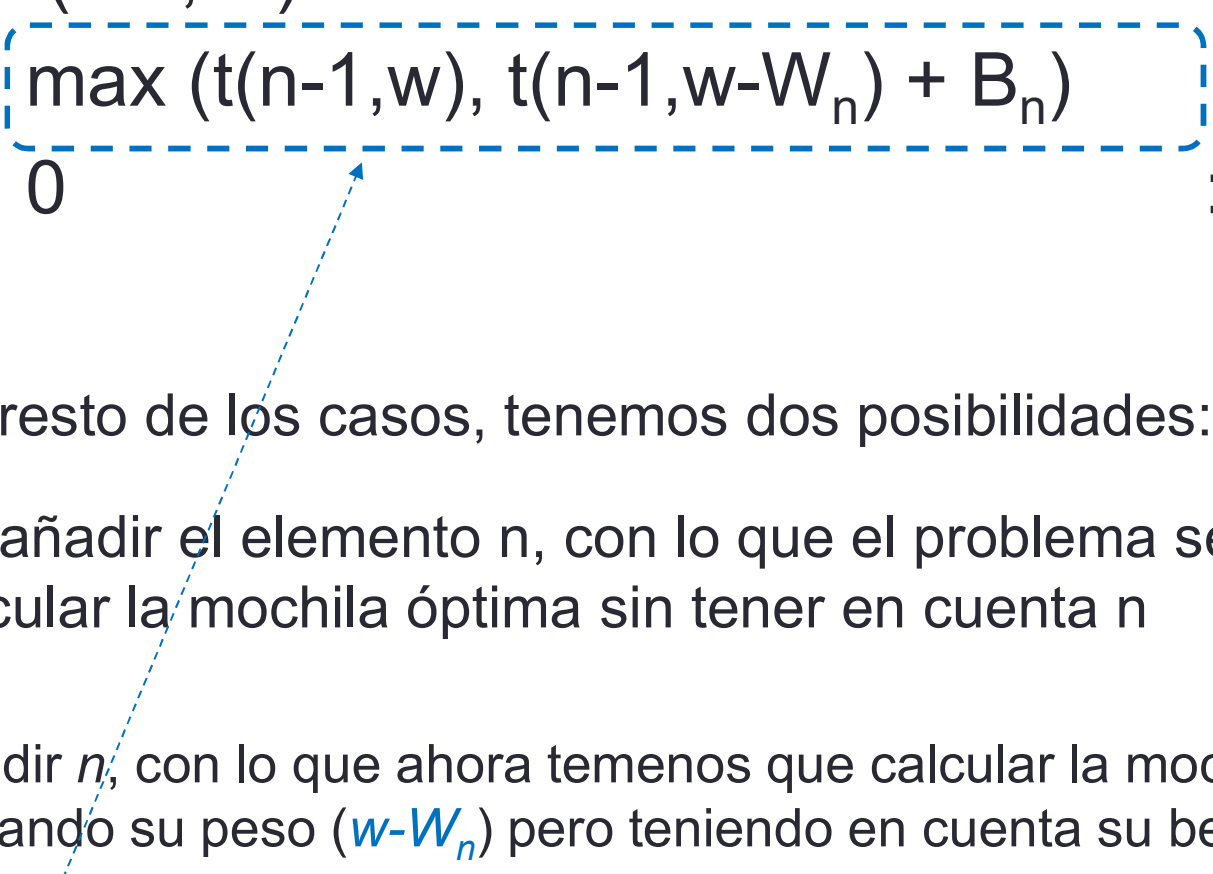
$B = 3, 4, 5, 6$

$N = 4, 3, 2, 1$

$$t(n, w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w - W_n) + B(n)) & : n \leq 0 \\ 0 & \end{cases}$$



Lectura de la Recurrencia

$$t(n, w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w - W_n) + B_n) & : n \leq 0 \\ 0 & \end{cases}$$


En el resto de los casos, tenemos dos posibilidades:

1. No añadir el elemento n , con lo que el problema se reduce a calcular la mochila óptima sin tener en cuenta n
2. Añadir n , con lo que ahora tenemos que calcular la mochila óptima restando su peso ($w - W_n$) pero teniendo en cuenta su beneficio (B_n)

Y nos quedamos con la opción que maximice nuestro beneficio!

$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n \leq 0 \\ 0 & \end{cases}$$

Programación Dinámica: Tabulation

Debemos crear una tabla para evitar repetir cálculos
... y como vamos a implementar tabulation calculamos
la recurrencia a partir de los casos base (**bottom-up**)

¿Cuál es el número mínimo de filas de la tabla ?

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)



$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n \leq 0 \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

Debemos crear una tabla para evitar repetir cálculos
... y como vamos a implementar tabulation calculamos
la recurrencia a partir de los casos base (**bottom-up**)

¿Cuál es el número mínimo de filas de la tabla ?
El número de ítems (o sea, 4)

¿Cuál es el número mínimo de columnas de la tabla ?

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)



$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max (t(n-1,w), t(n-1,w-W_n) + B_n) & : n > 0 \\ 0 & : n \leq 0 \end{cases}$$

Programación Dinámica: Tabulation

Debemos crear una tabla para evitar repetir cálculos
... y como vamos a implementar tabulation calculamos
la recurrencia a partir de los casos base (**bottom-up**)

¿Cuál es el número mínimo de filas de la tabla ?
El número de ítems (o sea, 4)

¿Cuál es el número mínimo de columnas de la tabla ?

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

*Debemos calcular la mejor combinación que tenga un peso
máximo de 5 kilos*



$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max (t(n-1,w), t(n-1,w-W_n) + B_n) & : n > 0 \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

Número mínimo de filas = 4

Número mínimo de columnas = 5

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n \leq 0 \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

Número mínimo de filas = 4

Número mínimo de columnas = 5

... pero si nos fijamos bien ...

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

Número mínimo de filas = 4

Número mínimo de columnas = 5

... pero si nos fijamos bien ...

1) El caso base es cuando $n \leq 0$

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n > 0 \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

Número mínimo de filas = 5

Número mínimo de columnas = 5

... pero si nos fijamos bien ...

- 1) El caso base es cuando $n \leq 0$ (*añadimos una fila*)
- 2) Si lleno la mochila $w = 0$

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n <= 0 \\ 0 & : n <= 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

Número mínimo de filas = 5

Número mínimo de columnas = 6

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

n\W	0	1	2	3	4	5
0						
1						
2						
3						
4						

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n > 0 \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

$n \backslash W$	0	1	2	3	4	5
0						
1						
2						
3						
4						

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n > 0 \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

for $w = 0$ to W
 $t[0, w] = 0$

Items:
 1: (2,3)
 2: (3,4)
 3: (4,5)
 4: (5,6)

$n \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n \leq 0 \\ 0 & \end{cases}$$

*Programación
Dinámica: Tabulation*

for $w = 0$ to W
 $t[0, w] = 0$
 for $i = 1$ to n
 $t[i, 0] = 0$

Items:
 1: (2,3)
 2: (3,4)
 3: (4,5)
 4: (5,6)

$n \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n > 0 \\ 0 & : n \leq 0 \end{cases}$$

*Programación
Dinámica: Tabulation*

...

for i = 1 to n
 for w = 0 to W

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

n\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n \leq 0 \\ 0 & \end{cases}$$

 $: W_n > w$
 $: n \leq 0$

*Programación
Dinámica: Tabulation*

...

for i = 1 to n

 for w = 0 to W

 if $w_i \leq w$

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

else

n\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & : n \leq 0 \\ 0 & : n \leq 0 \end{cases}$$

Programación Dinámica: Tabulation

...

for i = 1 to n
 for w = 0 to W
 if $w_i \leq w$

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

else $t[i,w] = t[i-1,w]$

n\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

$$t(n,w) = \begin{cases} t(n-1, w) & : W_n > w \\ \max(t(n-1, w), t(n-1, w-W_n) + B_n) & \\ 0 & : n \leq 0 \end{cases}$$

Programación Dinámica: Tabulation

...

for i = 1 to n

 for w = 0 to W

 if $w_i \leq w$

 if $b_i + t[i-1, w-w_i] > t[i-1, w]$

$t[i, w] = b_i + t[i-1, w-w_i]$

 else

$t[i, w] = t[i-1, w]$

 else $t[i, w] = t[i-1, w]$

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

n\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Ejemplo (1/17)

$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

for $w = 0$ to W
 $V[0,w] = 0$

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

Ejemplo (2/17)

$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

for $i = 1$ to n
 $V[i,0] = 0$

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

Ejemplo (3/17)

		w ↓				
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	↓ 0				
2	0					
3	0					
4	0					

Items:

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

 $i=1$ $b_i=3$ $w_i=\underline{2}$ $w=1$ $w-w_i=-1$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (4/17)

			w ↓			
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3			
2	0					
3	0					
4	0					

Items:

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

 $i=1$ $b_i=3$ $w_i=\underline{2}$ $w=2$ $w-w_i=0$ if $w_i \leq w$ // item i can be part of the solutionif $b_i + V[i-1, w-w_i] > V[i-1, w]$ $V[i, w] = b_i + V[i-1, w-w_i]$

else

 $V[i, w] = V[i-1, w]$ else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (5/17)

				w ↓		
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
$i \rightarrow$ 1	0	0	3	3		
2	0					
3	0					
4	0					

Items:

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

$i=1$

$b_i=3$

$w_i=\underline{2}$

$w=3$

$w-w_i=1$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (6/17)

		w ↓					
i \ W		0	1	2	3	4	5
i →	0	0	0	0	0	0	0
	1	0	0	3	3	3	
	2	0					
	3	0					
	4	0					

Items:

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

 $i=1$ $b_i=3$ $w_i=\underline{2}$ $w=4$ $w-w_i=2$ if $w_i \leq w$ // item i can be part of the solutionif $b_i + V[i-1, w-w_i] > V[i-1, w]$ $V[i, w] = b_i + V[i-1, w-w_i]$

else

 $V[i, w] = V[i-1, w]$ else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (7/17)

						w ↓
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0					
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

 $i=1$ $b_i=3$ $w_i=\underline{2}$ $w=5$ $w-w_i=3$ if $w_i \leq w$ // item i can be part of the solutionif $b_i + V[i-1, w-w_i] > V[i-1, w]$ $V[i, w] = b_i + V[i-1, w-w_i]$

else

 $V[i, w] = V[i-1, w]$ else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (8/17)

		w ↓				
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	↓ 0				
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=2$

$b_i=4$

$w_i=\underline{3}$

$w=1$

$w-w_i=-2$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (9/17)

			w ↓			
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	↓ 3			
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=2$

$b_i=4$

$w_i=\underline{3}$

$w=2$

$w-w_i=-1$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (10/17)

				w ↓		
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4		
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=2$

$b_i=4$

$w_i=\underline{3}$

$w=3$

$w-w_i=0$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (11/17)

					w ↓	
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	
3	0					
4	0					

Items:

1: (2,3)

2: (3,**4**)

3: (4,5)

4: (5,6)

$i=2$

$b_i=4$

$w_i=3$

$w=4$

$w-w_i=1$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (12/17)

						w ↓
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=2$

$b_i=4$

$w_i=\underline{3}$

$w=5$

$w-w_i=2$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (13/17)

		w ↓					
i \ W	0	1	2	3	4	5	
0	0	0	0	0	0	0	
1	0	0	3	3	3	3	
2	0	0	3	4	4	7	
3	0	↓ 0	↓ 3	↓ 4			
4	0						

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i=3

$b_i=5$

$w_i=\underline{4}$

w = 1..3

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (14/17)

					w ↓	
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
$i \rightarrow$ 3	0	0	3	4	5	
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,**5**)

4: (5,6)

$i=3$

$b_i=5$

$w_i=4$

$w=4$

$w - w_i = 0$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (15/17)

		w ↓					
i \ W		0	1	2	3	4	5
i →	0	0	0	0	0	0	0
	1	0	0	3	3	3	3
	2	0	0	3	4	4	7
	3	0	0	3	4	5	↓ 7
	4	0					

Items:

1: (2,3)

2: (3,4)

3: (4, 5)

4: (5,6)

i=3

$b_i = 5$

$w_i = 4$

$w = 5$

$w - w_i = 1$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (16/17)

		w ↓					
i \ W		0	1	2	3	4	5
0		0	0	0	0	0	0
1		0	0	3	3	3	3
2		0	0	3	4	4	7
3		0	0	3	4	5	7
i → 4		0	↓ 0	↓ 3	↓ 4	↓ 5	

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i=4

$b_i=6$

$w_i=\underline{5}$

$w=1..4$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

Ejemplo (17/17)

		w ↓					
i \ W		0	1	2	3	4	5
0		0	0	0	0	0	0
1		0	0	3	3	3	3
2		0	0	3	4	4	7
3		0	0	3	4	5	7
i → 4		0	0	3	4	5	↓ 7

Items:

- 1: (2,3)
- 2: (3,4)
- 3: (4,5)
- 4: (5,6)

i=4

$b_i=6$

$w_i=5$

$w=5$

$w - w_i = 0$

if $w_i \leq w$ // item i can be part of the solution

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else $V[i, w] = V[i-1, w]$ // $w_i > w$

¿ Qué elementos contiene la mochila ?

- La información está en la tabla
... sólo tenemos que recorrerla hacia atrás!

$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	<u>7</u>

↑
Beneficio
Máximo

→ $i=n$, $k=W$

if $V[i,k] \neq V[i-1,k]$ **then**

 // El i^{th} elemento está en la mochila

$i = i-1$, $k = k-w_i$

else

 // El i^{th} elemento no está en la mochila

$i = i-1$

Encontrando los elementos

						k ↓
i \ W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
i → 4	0	0	3	4	5	7

i=4

k=5

b_i=6w_i=5 $V[i,k] = 7$ $V[i-1,k] = 7$

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i=n, k=W

while i,k > 0

if $V[i,k] \neq V[i-1,k]$ then*el elemento i^{th} está en la mochila* $i = i-1, k = k-w_i$

else

 $i = i-1$

Encontrando los elementos

						k ↓
i \ W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
i → 4	0	0	3	4	5	7

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=4$

$k=5$

$b_i=6$

$w_i=5$

$V[i,k] = 7$

$V[i-1,k] = 7$

$i=n, k=W$

while $i,k > 0$

if $V[i,k] \neq V[i-1,k]$ then

el elemento i^{th} está en la mochila

$i = i-1, k = k-w_i$

else

$i = i-1$

Encontrando los elementos

						k ↓
i \ W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
i → 3	0	0	3	4	5	7
4	0	0	3	4	5	7

i=3

k=5

b_i=5w_i=4

V[i,k] = 7

V[i-1,k] = 7

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i=n, k=W

while i,k > 0

if V[i,k] ≠ V[i-1,k] then

*el elemento ith está en la mochila**i = i-1, k = k-w_i*

else

i = i-1

Encontrando los elementos

						k ↓
$i \backslash W$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
$i \rightarrow$ 2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=2$

$k=5$

$b_i=4$

$w_i=3$

$V[i,k] = 7$

$V[i-1,k] = 3$

$k - w_i = 2$

$i=n, k=W$

while $i,k > 0$

if $V[i,k] \neq V[i-1,k]$ then

el elemento i^{th} está en la mochila

$i = i-1, k = k-w_i$

else

$i = i-1$

Encontrando los elementos

			k ↓			
i \ W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Items:

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

$i=1$

$k=2$

$b_i=3$

$w_i=2$

$V[i,k] = 3$

$V[i-1,k] = 0$

$k - w_i = 0$

$i=n, k=W$

while $i, k > 0$

if $V[i,k] \neq V[i-1,k]$ then

el elemento i^{th} está en la mochila

$i = i-1, k = k-w_i$

else

$i = i-1$

Encontrando los elementos

		k ↓						
		$i \backslash W$	0	1	2	3	4	5
$i \rightarrow$	0	0	0	0	0	0	0	0
	1	0	0	3	3	3	3	3
	2	0	0	3	4	4	7	7
	3	0	0	3	4	5	7	7
	4	0	0	3	4	5	7	7

$i=0$
 $k=0$

Items:

- 1: (2,3)
- 2: (3,4)
- 3: (4,5)
- 4: (5,6)

La mochila
óptima contiene
{1, 2}

$i=n, k=W$

while $i, k > 0$

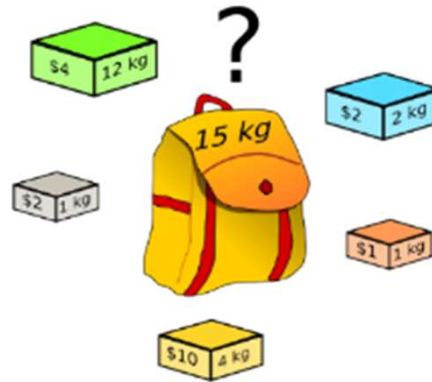
if $V[i, k] \neq V[i-1, k]$ then

el elemento i^{th} está en la mochila k

$i = i-1, k = k-w_i$

else

$i = i-1$



Programación Dinámica: Tabulation

Fase 1 del algoritmo: Rellenar la tabla

for $w = 0$ to W

$V[0, w] = 0$

for $i = 1$ to n

$V[i, 0] = 0$

for $i = 1$ to n

for $w = 0$ to W

if $w_i \leq w$

if $b_i + V[i-1, w-w_i] > V[i-1, w]$

$V[i, w] = b_i + V[i-1, w-w_i]$

else

$V[i, w] = V[i-1, w]$

else

$V[i, w] = V[i-1, w]$

Fase 2 del algoritmo: Utilizando el contenido
de la tabla identificar los items elegidos

$i = n, k = W$

while

if $V[i, k] \neq V[i-1, k]$ then

// El i^{th} elemento está en la mochila

$i = i-1, k = k-w_i$

else

// El i^{th} elemento no está en la mochila

$i = i-1$