

# TAREAS CON PLAZO Y BENEFICIO



U N I V E R S I D A D  
COMPLUTENSE  
M A D R I D

**ALBERTO VERDEJO**

# Tareas con plazo y beneficio

- ▶ Tenemos  $n$  tareas, cada una requiriendo una unidad de tiempo en ejecutarse, y con un plazo  $p_i$  y un beneficio  $b_i$ .
- ▶ Si la tarea se realiza no después de su plazo, se obtiene el beneficio.
- ▶ No todas las tareas tienen por qué realizarse. Solamente las que puedan hacerse antes de que venza el plazo.
- ▶ El objetivo es planificar las tareas de forma que se *maximice el beneficio total obtenido*.

# Ejemplo

$i$	$p_i$	$b_i$
1	2	30
2	1	35
3	2	25
4	1	40

Planificación	Beneficio total
[1, 3]	$30 + 25 = 55$
[2, 1]	$35 + 30 = 65$
[2, 3]	$35 + 25 = 60$
[3, 1]	$25 + 30 = 55$
[4, 1]	$40 + 30 = 70$
[4, 3]	$40 + 25 = 65$

# Estrategia voraz

- ▶ Un conjunto de tareas es *factible* si existe alguna secuencia de ejecución admisible, que permita realizar todas las tareas dentro de sus plazos.
- ▶ Una permutación  $(i_1, i_2, \dots, i_k)$  es *admisibile* si

$$\forall j : 1 \leq j \leq k : p_{i_j} \geq j$$

- ▶ La estrategia voraz considera las tareas *de mayor a menor beneficio*, y cada tarea se selecciona si al añadirla al conjunto de tareas seleccionadas, este sigue siendo factible.

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$

[1]

40



# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{ 1 \}$                        $[ 1 ]$                       40

$S = \{ 1, 2 \}$                        $[ 2, 1 ]$                       75



# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{ 1 \}$                        $[ 1 ]$                       40

$S = \{ 1, 2 \}$                        $[ 2, 1 ]$                       75

$S = \{ 1, 2, 3 \}$

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$  [1] 40

$S = \{1, 2\}$  [2, 1] 75

$S = \{1, 2, 3\}$  3 rechazada

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$  [1] 40

$S = \{1, 2\}$  [2, 1] 75

$S = \{1, 2, 3\}$  3 rechazada

$S = \{1, 2, 4\}$  [2, 1, 4] 100

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$  [1] 40

$S = \{1, 2\}$  [2, 1] 75

$S = \{1, 2, 3\}$  3 rechazada

$S = \{1, 2, 4\}$  [2, 1, 4] 100

$S = \{1, 2, 4, 5\}$

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$  [1] 40

$S = \{1, 2\}$  [2, 1] 75

$S = \{1, 2, 3\}$  3 rechazada

$S = \{1, 2, 4\}$  [2, 1, 4] 100

$S = \{1, 2, 4, 5\}$  5 rechazada

# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$  [1] 40

$S = \{1, 2\}$  [2, 1] 75

$S = \{1, 2, 3\}$  3 rechazada

$S = \{1, 2, 4\}$  [2, 1, 4] 100

$S = \{1, 2, 4, 5\}$  5 rechazada

$S = \{1, 2, 4, 6\}$  [2, 4, 6, 1] 115



# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$  [1] 40

$S = \{1, 2\}$  [2, 1] 75

$S = \{1, 2, 3\}$  3 rechazada

$S = \{1, 2, 4\}$  [2, 1, 4] 100

$S = \{1, 2, 4, 5\}$  5 rechazada

$S = \{1, 2, 4, 6\}$  [2, 4, 6, 1] 115

$S = \{1, 2, 4, 6, 7\}$



# Ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

$S = \emptyset$

$S = \{1\}$  [1] 40

$S = \{1, 2\}$  [2, 1] 75

$S = \{1, 2, 3\}$  3 rechazada

$S = \{1, 2, 4\}$  [2, 1, 4] 100

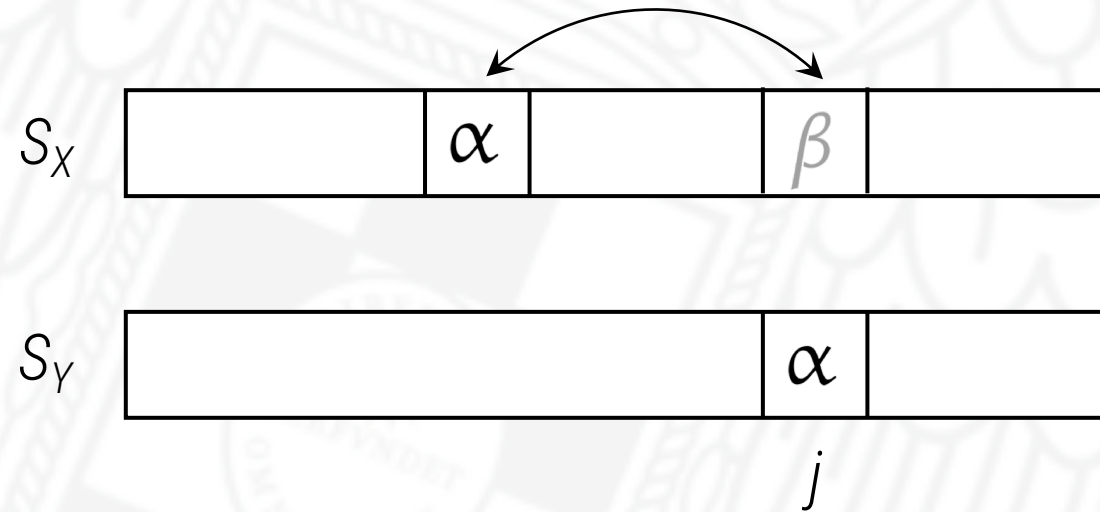
$S = \{1, 2, 4, 5\}$  5 rechazada

$S = \{1, 2, 4, 6\}$  [2, 4, 6, 1] 115

$S = \{1, 2, 4, 6, 7\}$  7 rechazada

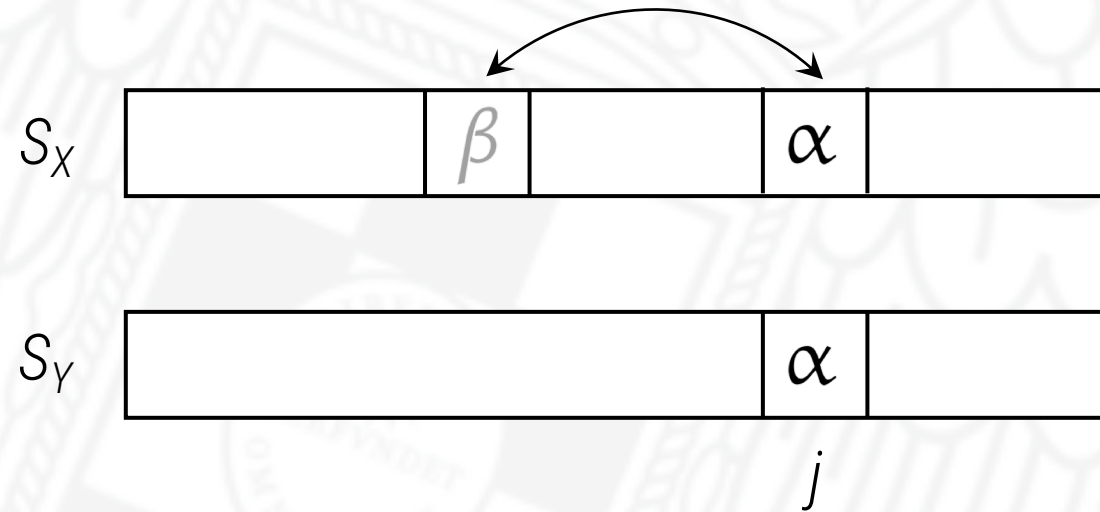
# Demostración de optimalidad

- ▶ Llamamos  $X$  a la solución voraz, e  $Y$  a una solución óptima cualquiera.
- ▶ Sean  $S_X$  y  $S_Y$  secuencias admisibles de las tareas de  $X$  e  $Y$ .
- ▶ Primero se transforman las secuencias de forma que las tareas comunes se realicen en el mismo momento.



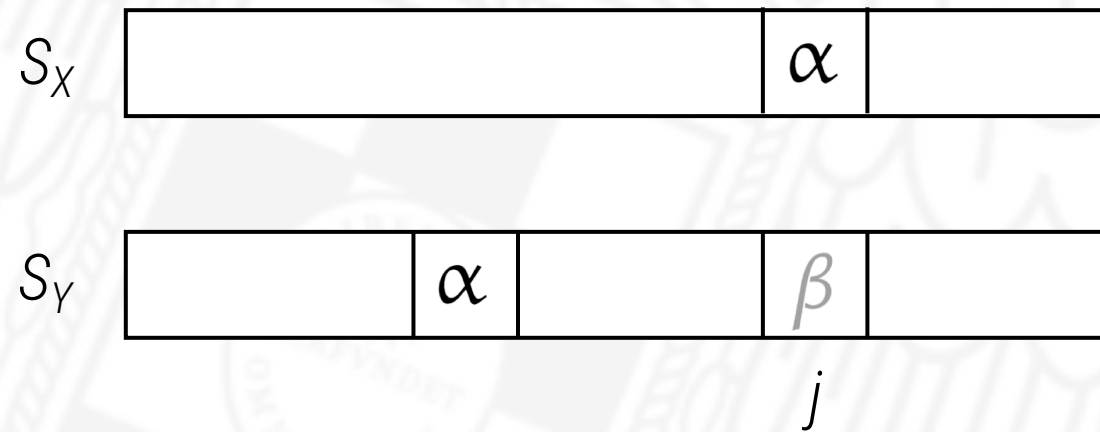
# Demostración de optimalidad

- ▶ Llamamos  $X$  a la solución voraz, e  $Y$  a una solución óptima cualquiera.
- ▶ Sean  $S_X$  y  $S_Y$  secuencias admisibles de las tareas de  $X$  e  $Y$ .
- ▶ Primero se transforman las secuencias de forma que las tareas comunes se realicen en el mismo momento.



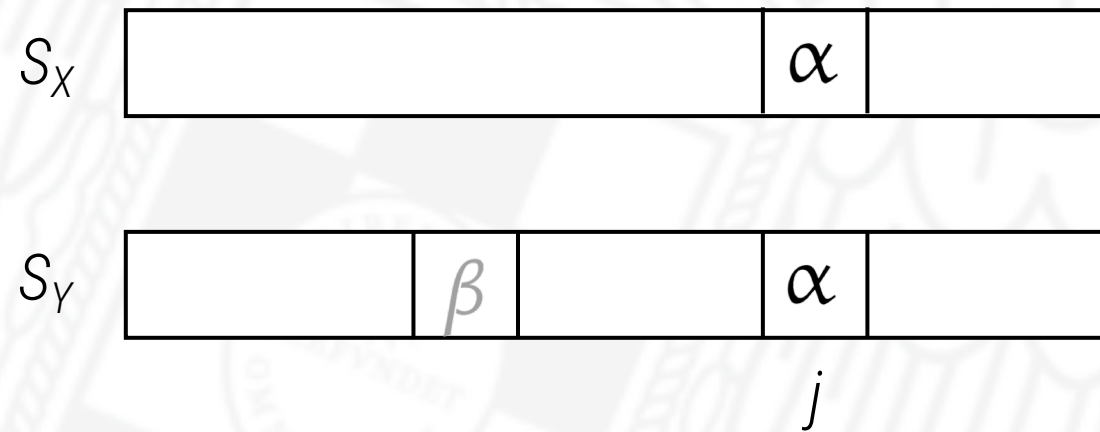
# Demostración de optimalidad

- ▶ Llamamos  $X$  a la solución voraz, e  $Y$  a una solución óptima cualquiera.
- ▶ Sean  $S_X$  y  $S_Y$  secuencias admisibles de las tareas de  $X$  e  $Y$ .
- ▶ Primero se transforman las secuencias de forma que las tareas comunes se realicen en el mismo momento.



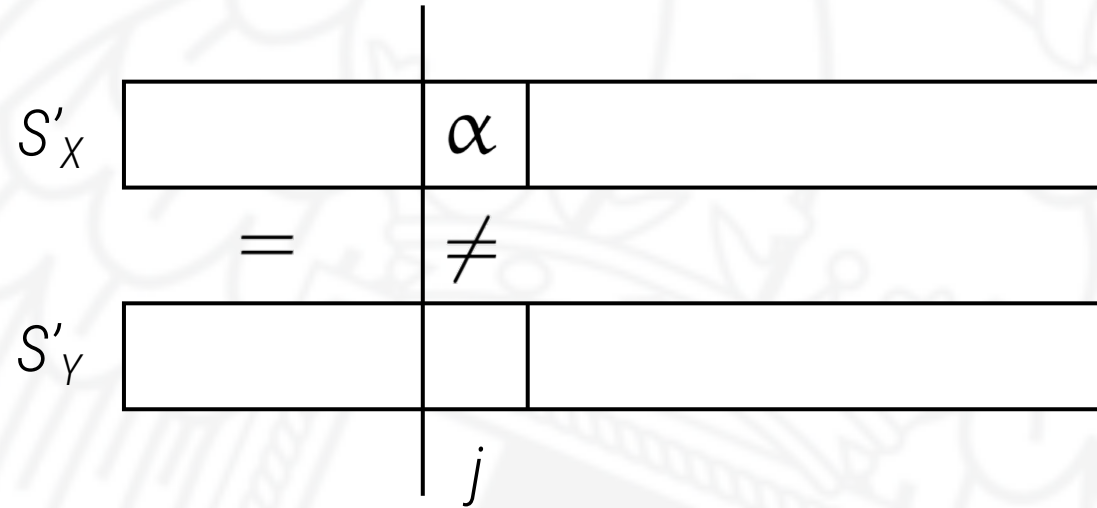
# Demostración de optimalidad

- ▶ Llamamos  $X$  a la solución voraz, e  $Y$  a una solución óptima cualquiera.
- ▶ Sean  $S_X$  y  $S_Y$  secuencias admisibles de las tareas de  $X$  e  $Y$ .
- ▶ Primero se transforman las secuencias de forma que las tareas comunes se realicen en el mismo momento.



# Demostración de optimalidad

- ▶ Comparamos ahora las secuencias transformadas  $S'_x$  y  $S'_y$ .

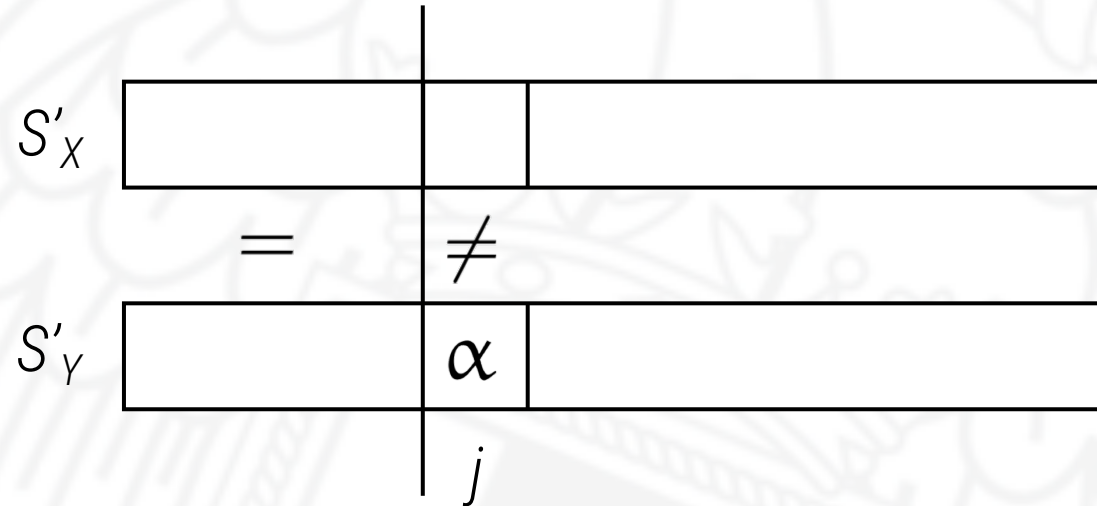


- ▶ Este tipo de diferencia no puede darse.



# Demostración de optimalidad

- ▶ Comparamos ahora las secuencias transformadas  $S'_x$  y  $S'_y$ .

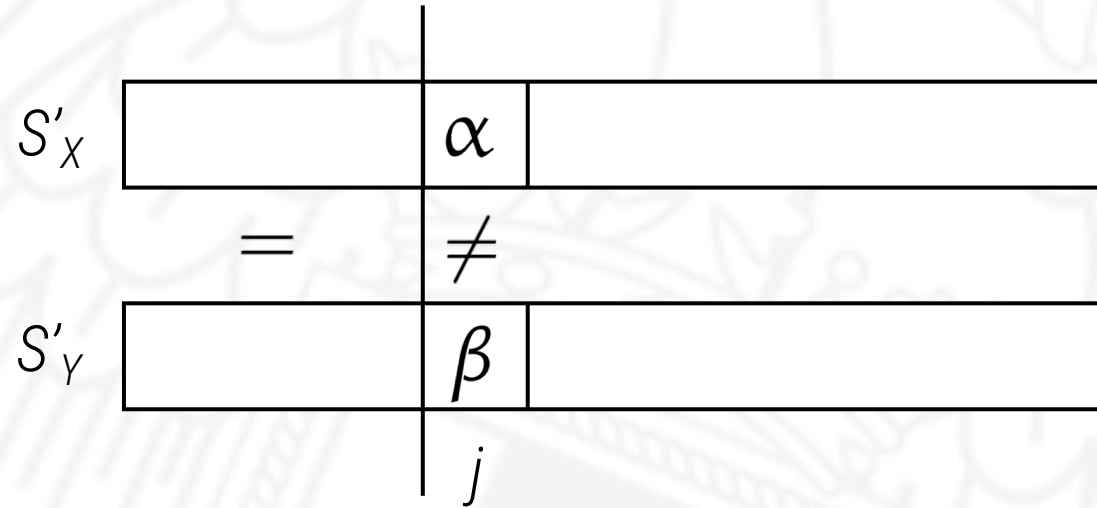


- ▶ Este tipo de diferencia tampoco puede darse.



# Demostración de optimalidad

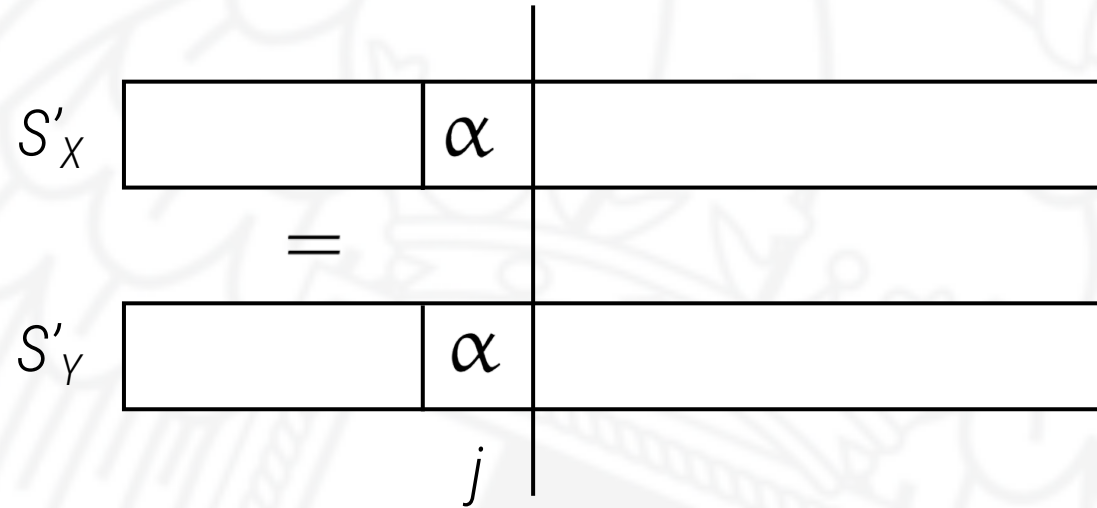
- ▶ Comparamos ahora las secuencias transformadas  $S'_x$  y  $S'_y$ .



- ▶  $b_\alpha > b_\beta$
- ▶  $b_\alpha < b_\beta$
- ▶  $b_\alpha = b_\beta$

# Demostración de optimalidad

- ▶ Comparamos ahora las secuencias transformadas  $S'_x$  y  $S'_y$ .



- ▶  $b_\alpha > b_\beta$
- ▶  $b_\alpha < b_\beta$
- ▶  $b_\alpha = b_\beta$

# Test de factibilidad 1

- ▶ Un conjunto de tareas  $T$  es factible si y solo si la secuencia que ordena las tareas por orden creciente de plazos es admisible.
- ▶ Sea  $T = \{t_1, \dots, t_k\}$  con  $p_1 \leq p_2 \leq \dots \leq p_k$ .
- ▶ Si la secuencia  $t_1, t_2, \dots, t_k$  no es admisible, existe alguna tarea  $t_r$  tal que  $p_r < r$ . Pero entonces se cumple

$$p_1 \leq p_2 \leq \dots \leq p_{r-1} \leq p_r \leq r-1.$$

- ▶  $T$  no es factible.

# Test de factibilidad 1, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7

# Test de factibilidad 1, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
1						

# Test de factibilidad 1, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
2	1					



# Test de factibilidad 1, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
2	4	1				



# Test de factibilidad 1, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
2	4	6	1			

# Test de factibilidad 1, ejemplo que requiere el máximo trabajo

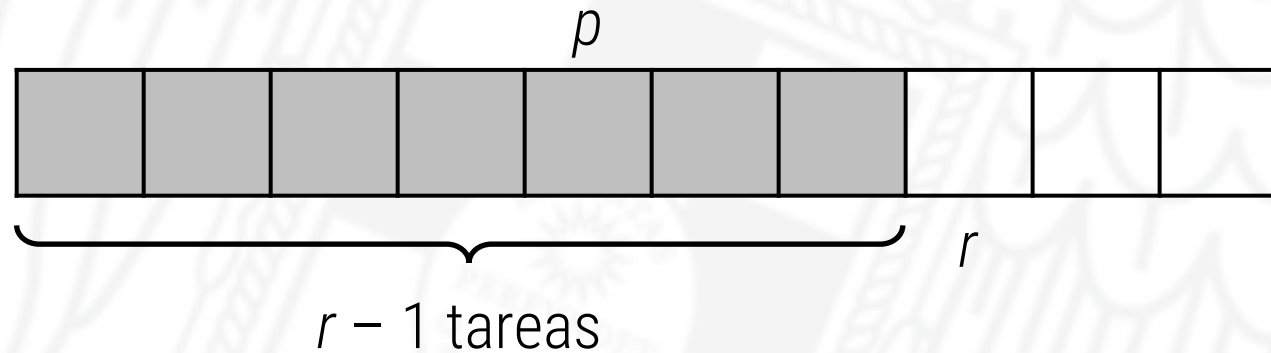
$i$	$p_i$	$b_i$
1	9	40
2	8	35
3	6	30
4	5	25
5	4	20
6	2	15
7	1	10

## Test de factibilidad 2

- Un conjunto de tareas  $T$  es factible si y solo si la secuencia que planifica las tareas *lo más tarde posible* es admisible.

$$d(i) = \max \{ d \mid 1 \leq d \leq \min(n, p_i) \wedge (\forall j : 1 \leq j < i : d \neq d(j)) \}$$

- Tarea en  $T$  con plazo  $p$ , que al ir a planificarla no “cabe”:



# Test de factibilidad 2, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7

# Test de factibilidad 2, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
			1			

# Test de factibilidad 2, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
2			1			



# Test de factibilidad 2, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
2		4	1			



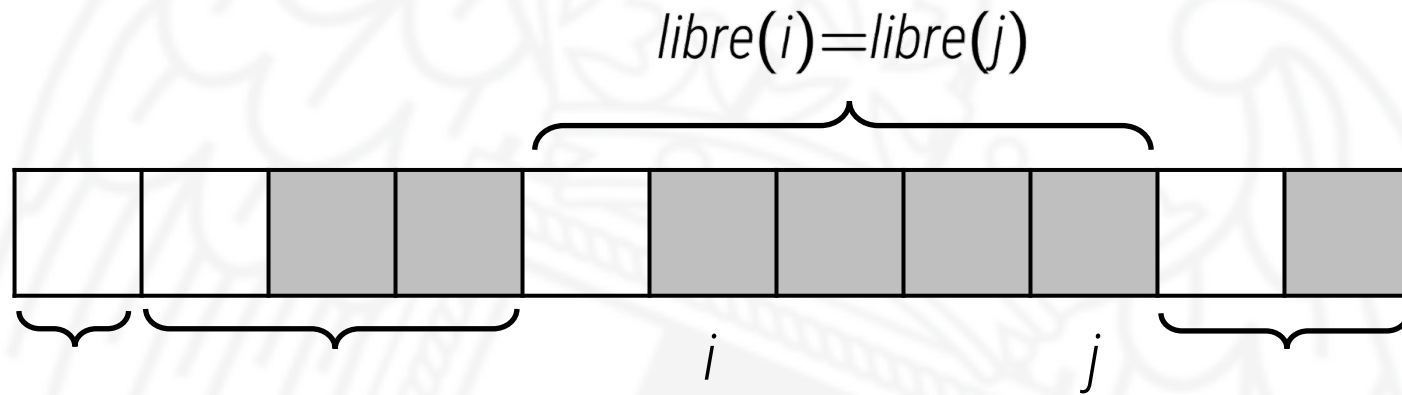
# Test de factibilidad 2, ejemplo

$i$	$p_i$	$b_i$
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1	2	3	4	5	6	7
2	6	4	1			

## Test de factibilidad 2

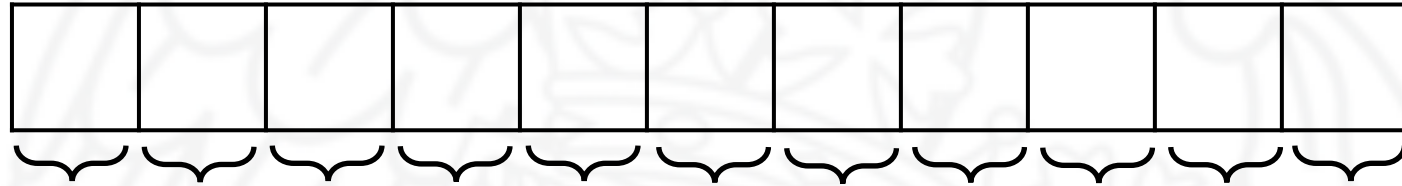
$$\text{libre}(i) = \text{máx} \{ d \leq i \mid d \text{ libre} \}$$



- Cada tarea  $t_i$  debería realizarse en el día  $\text{libre}(p_i)$ , que representa el último día libre que respeta su plazo.

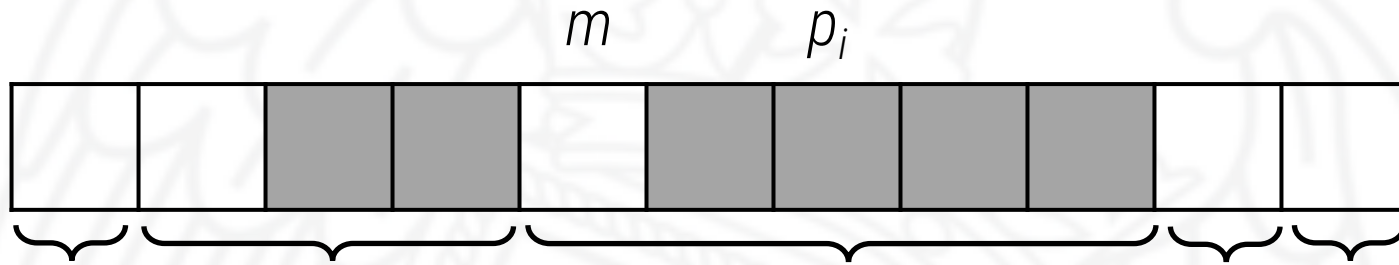
## Test de factibilidad 2

- Inicialmente  $\forall i : 0 \leq i \leq n : libre(i) = i$



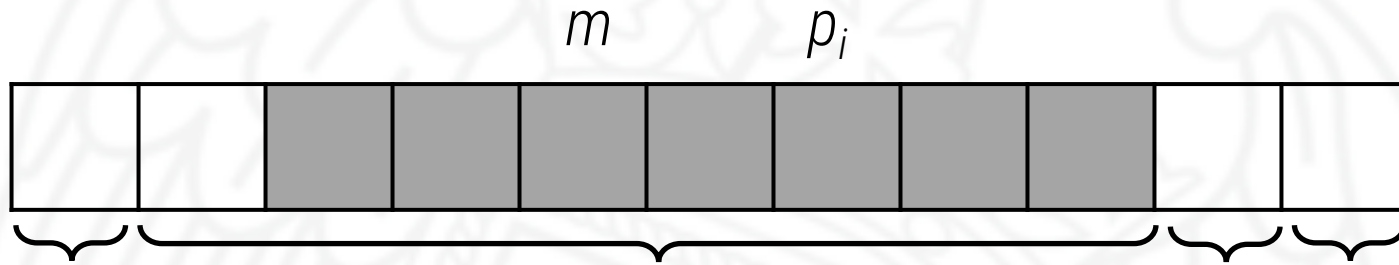
## Test de factibilidad 2

- Si se planifica la tarea  $t_i$  el día  $libre(p_i) = m$ , hay que unir su clase de equivalencia con el del día anterior.



## Test de factibilidad 2

- Si se planifica la tarea  $t_i$  el día  $libre(p_i) = m$ , hay que unir su clase de equivalencia con el del día anterior.



# Implementación

```
struct Tarea {  
    int plazo;  
    int beneficio;  
    int id;  
};  
  
bool operator>(Tarea a, Tarea b) {  
    return a.beneficio > b.beneficio;  
}
```



# Implementación

```
// las tareas están ordenadas de mayor a menor beneficio
int resolver(vector<Tarea> const& tareas, vector<int> & sol) {
    int N = tareas.size(); // número de tareas

    vector<int> libre(N + 1, 0);
    for (int i = 0; i <= N; ++i)
        libre[i] = i;

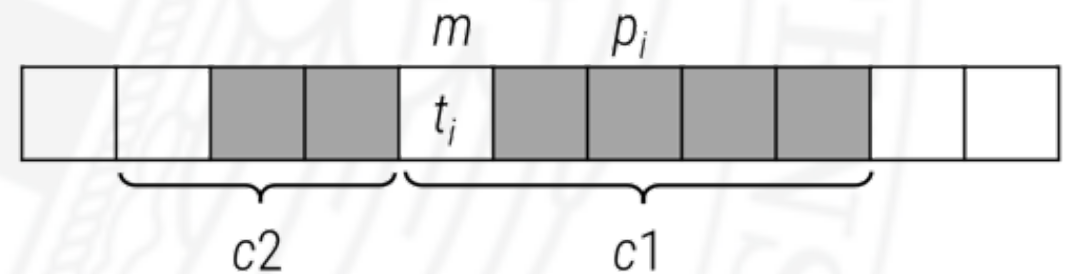
    vector<int> plan(N + 1); // 0 es que no está usado

    ConjuntosDisjuntos particion(N + 1);

    int beneficio = 0;
```

# Implementación

```
// recorrer las tareas de mayor a menor beneficio
for (int i = 0; i < N; ++i) {
    int c1 = particion.buscar(min(N, tareas[i].plazo));
    int m = libre[c1];
    if (m != 0) { // podemos colocar la tarea i
        plan[m] = tareas[i].id;
        beneficio += tareas[i].beneficio;
        int c2 = particion.buscar(m-1);
        particion.unir(c1, c2);
        libre[c1] = libre[c2];
    }
}
```



# Implementación

```
// compactamos la solución
for (int i = 1; i <= N; ++i) {
    if (plan[i] > 0)
        sol.push_back(plan[i]);
}
return beneficio;
}
```