

TAREAS CON PLAZO Y BENEFICIO

Problema de planificación en el que todas las tareas requieren el mismo tiempo para ejecutarse, pero tienen un plazo límite asociado. De manera que, solamente si la tarea se realiza antes de que venza el plazo le aporta un beneficio. Objetivo es maximizar el beneficio total obtenido.



UNIVERSIDAD
COMPLUTENSE
MADRID

ALBERTO VERDEJO

Tareas con plazo y beneficio

Da igual que sean minutos, horas...

- ▶ Tenemos n tareas, cada una requiriendo una unidad de tiempo en ejecutarse, y con un plazo p_i y un beneficio b_i .
Si terminamos la tarea antes de su plazo conseguiremos ese beneficio. No hay obligación de hacer todas las tareas. Si hay una tarea que no la podemos hacer antes de su plazo es mejor NO hacerla.
- ▶ 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*.

Hay que hacerlas unas detrás de otras. NO se pueden hacer en paralelo.

Ejemplo

[1,2] NO SERÍA POSIBLE, PORQUE LA TAREA 2 SOLO PUEDE HACERSE EL PRIMER DÍA

tarea	plazos	beneficios
i	p_i	b_i
1	2	30
2	1	35
3	2	25
4	1	40

Primero la tarea 1 y después la tarea 3.

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]	$40 + 25 = 65$

ÓPTIMA.

Que la tarea 1 tenga plazo 2, significa que la tarea se puede hacer el día 1 o el día 2.

En este ejercicio, la tarea con mayor beneficio, que es la tarea 4, está incluida en la planificación óptima pero no la tarea con segundo mayor beneficio (2). Creo que habría que coger la de mayor de cada plazo. Ordenar tareas por orden decreciente de beneficio, añadiéndolas a la planificación si es posible.

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 *admisible* si

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

Puede que el día asignado a la tarea NO SEA el índice de esa permutación. Mientras que se respete el plazo basta.

- ▶ 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.

En cada etapa tomamos la tarea con mayor beneficio para ese plazo.

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

$$S = \emptyset$$

Al principio tenemos un conjunto vacío de tareas seleccionadas

Las tareas están ordenadas de manera decreciente.

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

Consideramos la primera tarea

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

$$S = \emptyset$$

$$S = \{ 1 \}$$

$$[1]$$

$$40$$

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

$$[2, 1]$$

$$75$$

Consideramos la tarea 2 el conjunto 1,2 es FACTIBLE porque la permutación [2,1] es admisible, pero no [1,2].

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

$$S = \emptyset$$

$$S = \{ 1 \}$$

$$[1]$$

$$40$$

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

$$[2, 1]$$

$$75$$

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

Infactible porque las tareas 2 y 3 tienen el mismo plazo

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

este conjunto es factible porque esta permutación es admisible

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

La tarea 6 es factible. A pesar de que 4 tiene plazo 3 la podemos hacer antes, y la 6 hacerla el día 3. Por tanto es factible, y la permutación admisible es [2,4,6,1] o [2,6,4,1]

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

$$S = \emptyset$$

$$S = \{ 1 \}$$

[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

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

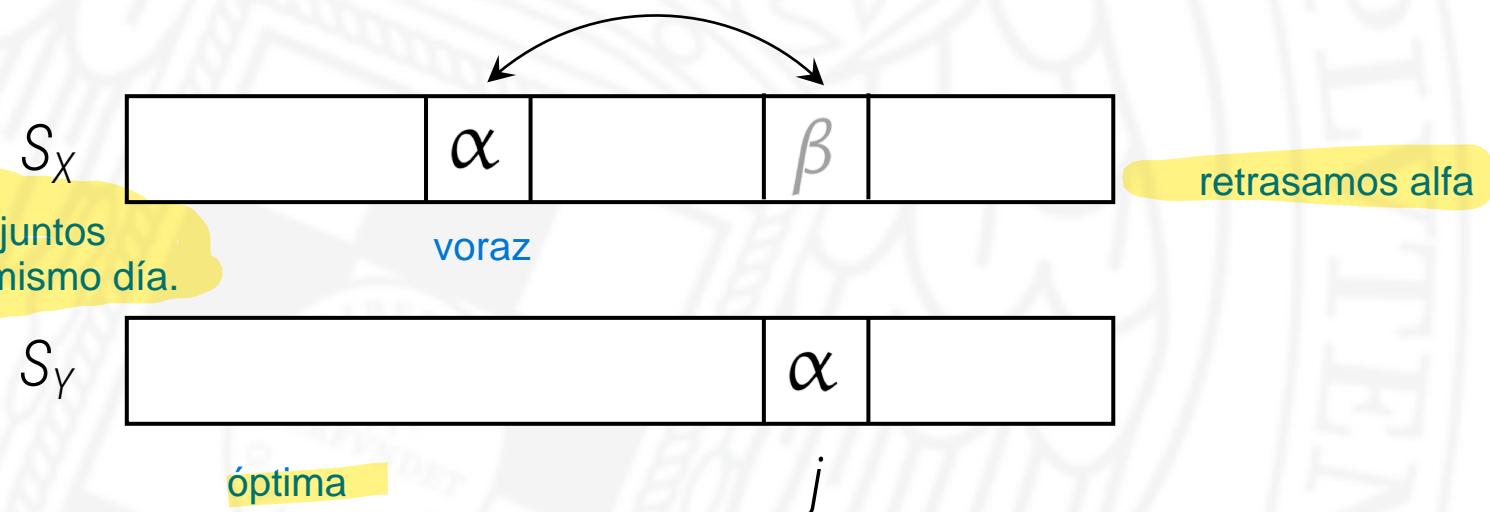
7 rechazada

beneficio

Demostración de optimalidad

Cómo saber si un subconjunto de tareas es factible o NO.

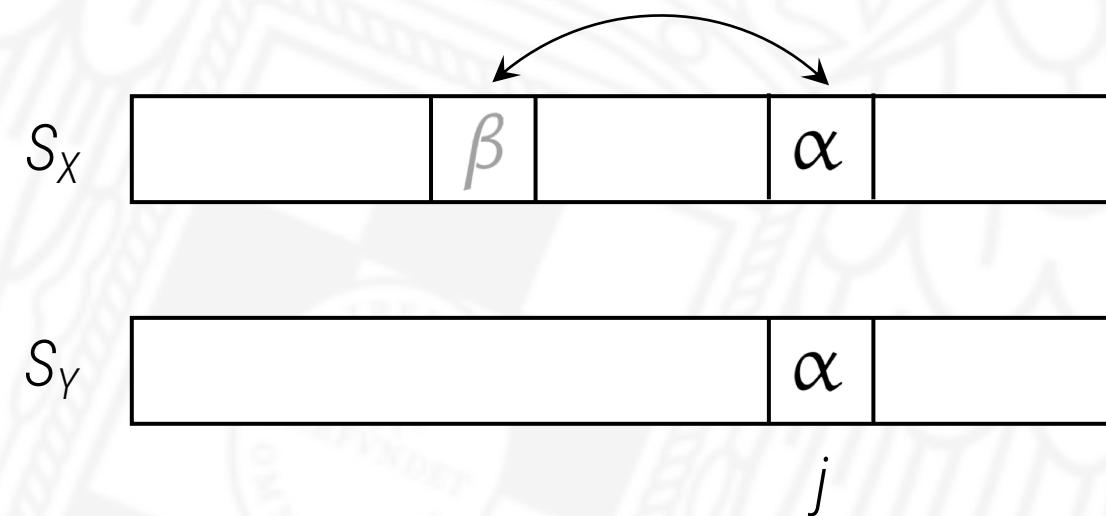
- ▶ Llamamos X a la solución voraz, e Y a una solución óptima cualquiera.
- ▶ Sean S_X y S_Y secuencias admisibles de las tareas de X e Y .
Cardinal no tiene por qué ser el mismo.
- ▶ Primero se transforman las secuencias de forma que las tareas comunes se realicen en el mismo momento.



Cambiamos el orden de las tareas de los conjuntos para que las tareas de ambos se realicen el mismo día.

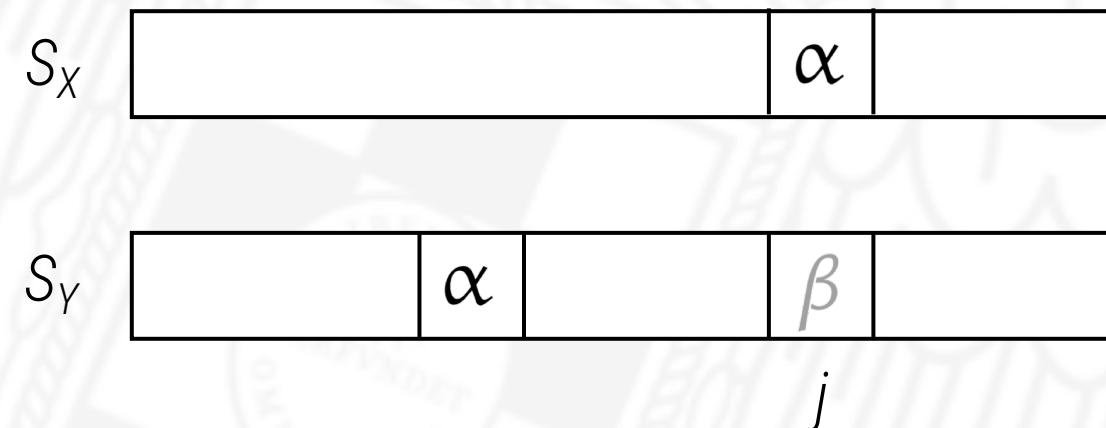
Demostración de optimalidad

- ▶ 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

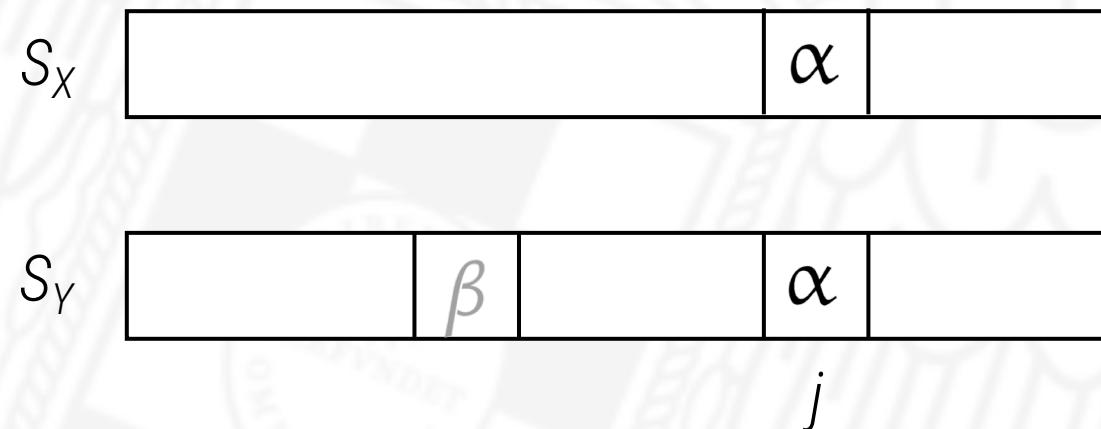
- ▶ Llamamos X a la solución voraz, e Y a una solución óptima cualquiera.
- ▶ Sean S_X y S_Y secuencias admisibles de las tareas de X e Y .
- ▶ Primero se transforman las secuencias de forma que las tareas comunes se realicen en el mismo momento.



También la podemos adelantar en S_Y

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 .

S'_X		α	
	=	\neq	
S'_Y			
		j	

Las vamos comprobando de izquierda a derecha. j es la primera posición donde difieren

- ▶ Este tipo de diferencia no puede darse.

Demostración de optimalidad

- ▶ Comparamos ahora las secuencias transformadas S'_X y S'_Y .

S'_X		
	=	\neq
S'_Y	α	

j

- ▶ Este tipo de diferencia tampoco puede darse.

Porque cuando el algoritmo voraz consideró la tarea alpha, la rechazó. En el momento en el que intentó ingresar alpha en el conjunto S_X no constituyó el conjunto factible.

Demostración de optimalidad

- Comparamos ahora las secuencias transformadas S'_X y S'_Y .

S'_X	α	
=	\neq	
S'_Y	β	

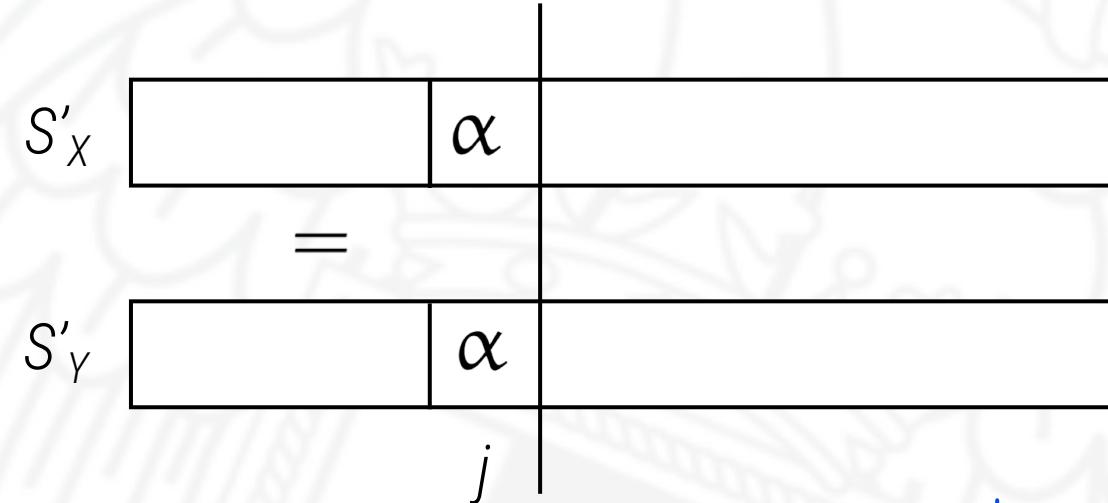
j

Esto si se puede dar, que el día j ambas secuencias planifiquen distintas tareas.

- $b_\alpha > b_\beta$ Si lo fuera, si sustituyeremos en la solución Y beta por alfa, obtendríamos una solución factible mejor. Eso es imposible por ser Y la solución óptima.
- $b_\alpha < b_\beta$ No es posible. La estrategia voraz considera las tareas en orden creciente de beneficios, la habría seleccionado si fuera óptima.
- $b_\alpha = b_\beta$ Esta es la única opción posible.

Demostración de optimalidad

- ▶ Comparamos ahora las secuencias transformadas S'_X y S'_Y .



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

Única opción posible.

Test de factibilidad 1

Hacer permutaciones hasta encontrar una admisible nos llevaría un tiempo factorial

- ▶ 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 $p_1 \leq p_2 \leq \dots \leq p_k$. Enumeradas de menor a mayor plazo.
- ▶ 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

Se planifica después de su plazo.

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

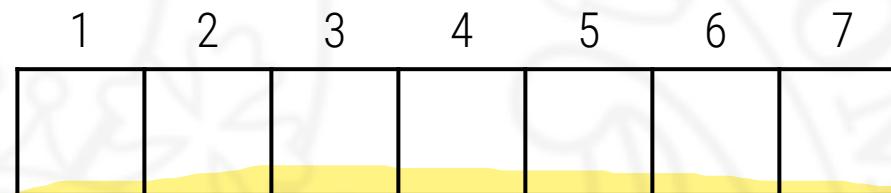
- ▶ T no es factible.

El test de factibilidad consistirá en comprobar si la secuencia de tareas ordenada por plazo es admisible.

Test de factibilidad 1, ejemplo

Ordenadas de mayor a menor beneficio

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



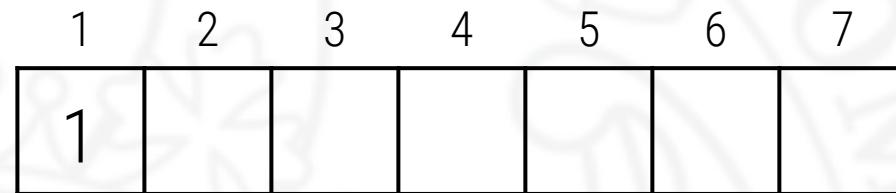
Como solo tenemos 7 tareas, solamente necesitamos un vector de 7 posiciones, 1 tarea por día.

Por otro lado, como el mayor plazo es 4, solamente deberán ocuparse como máximo 4

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

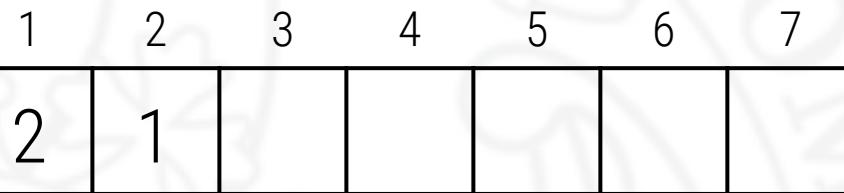
Se asigna provisionalmente al día 1.



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

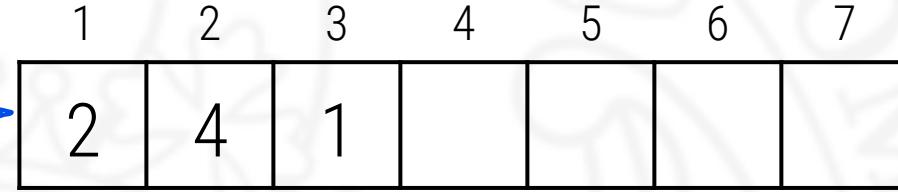
Tarea 2 también puede ser seleccionada, desplazamos a la tarea 1 para mantener a la planificación ordenada por plazos.



La tarea 3 se rechaza.

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

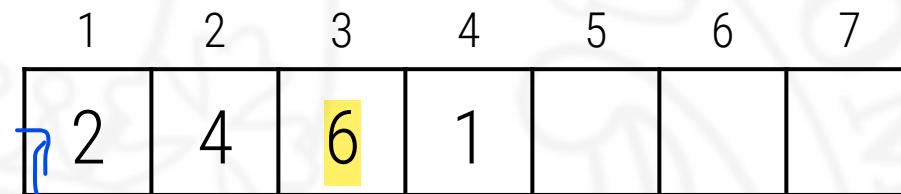


Podemos desplazar en la planificación a la tarea 1 porque tiene plazo 4.

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

En el caso peor esto tendría coste N^2 .



Rechazada

Podemos volver a mover la tarea 1 con plazo 4. Ahora ya si que no la podríamos mover.

Rechazada.

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

Veremos una alternativa con una implementación más eficiente de la estrategia voraz.

Test de factibilidad 2

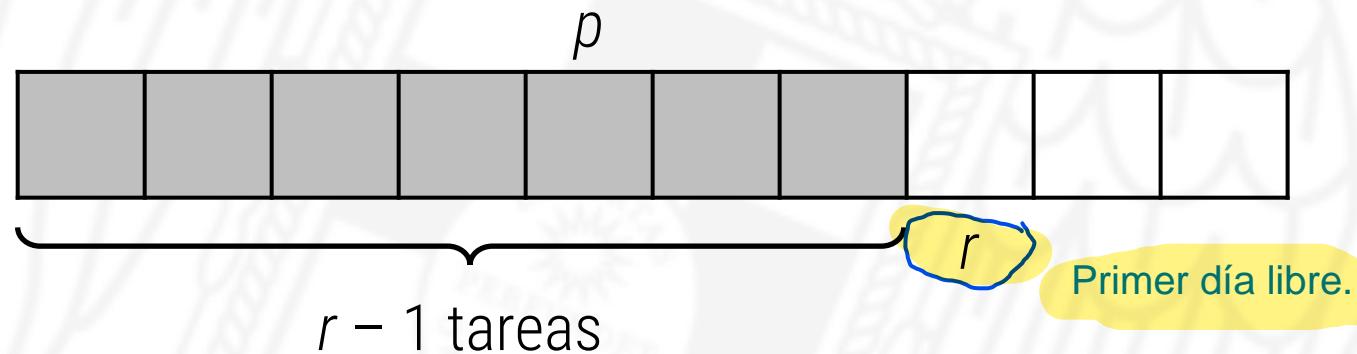
- ▶ Un conjunto de tareas T es factible si y solo si la secuencia que planifica las tareas *lo más tarde posible* es admisible.

Para cada tarea se elige el momento libre más tardío y que no se pase de su plazo

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

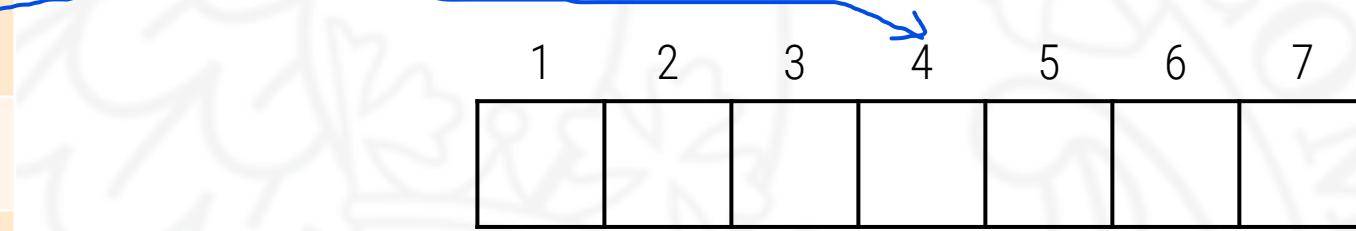
mayor momento disponible no superior a su plazo y que no haya sido asignado a una tarea anteriormente.

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



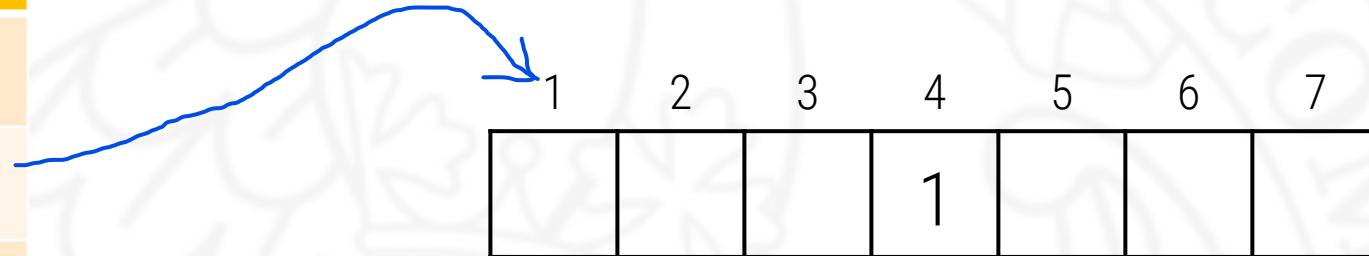
Test de factibilidad 2, ejemplo

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



Test de factibilidad 2, ejemplo

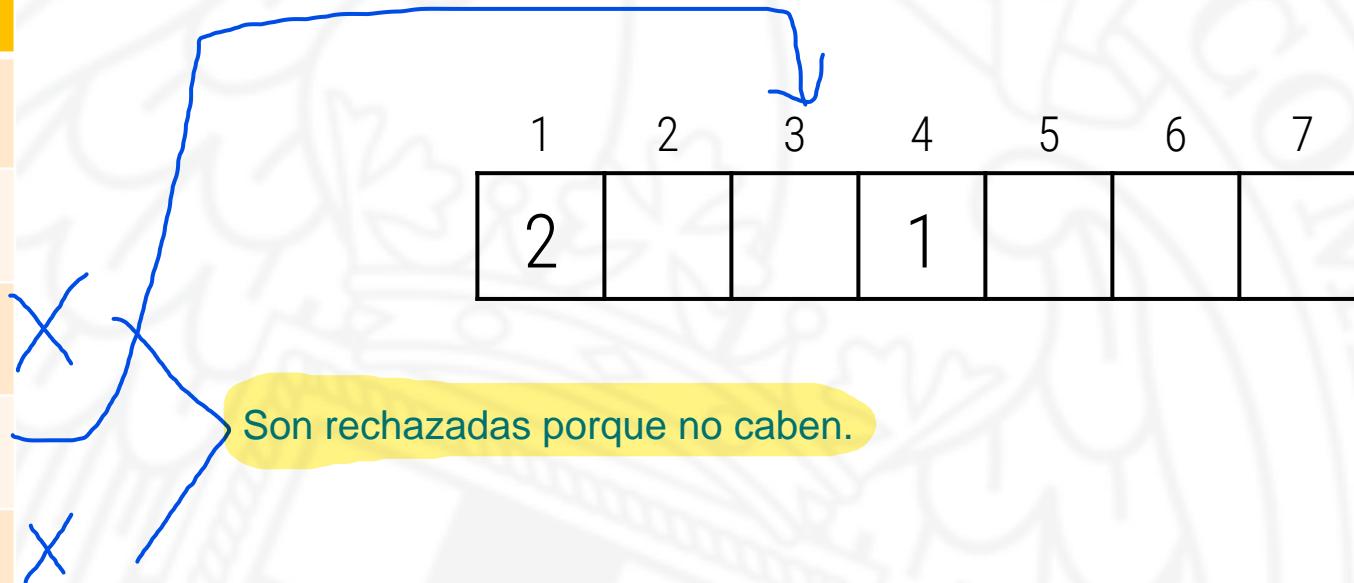
i	p_i	b_i
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10



La ponemos lo más tarde posible.

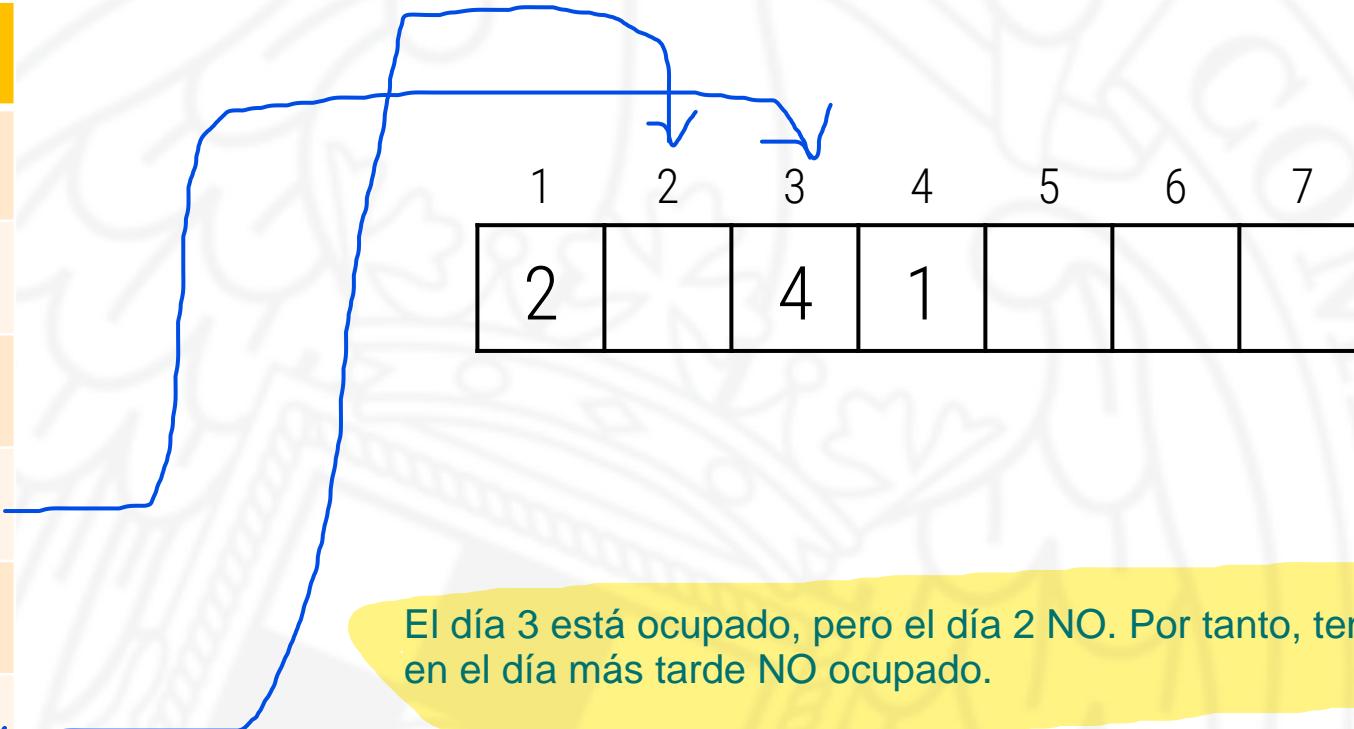
Test de factibilidad 2, ejemplo

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



Test de factibilidad 2, ejemplo

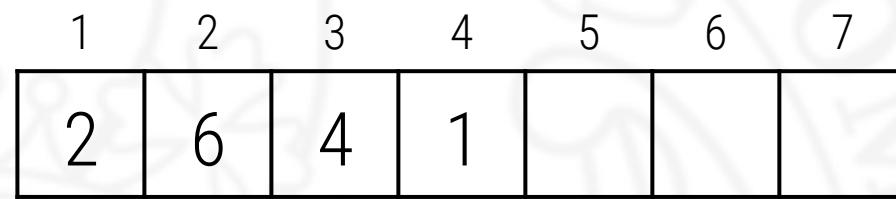
i	p_i	b_i
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10



El día 3 está ocupado, pero el día 2 NO. Por tanto, tenemos que colocarla en el día más tarde NO ocupado.

Test de factibilidad 2, ejemplo

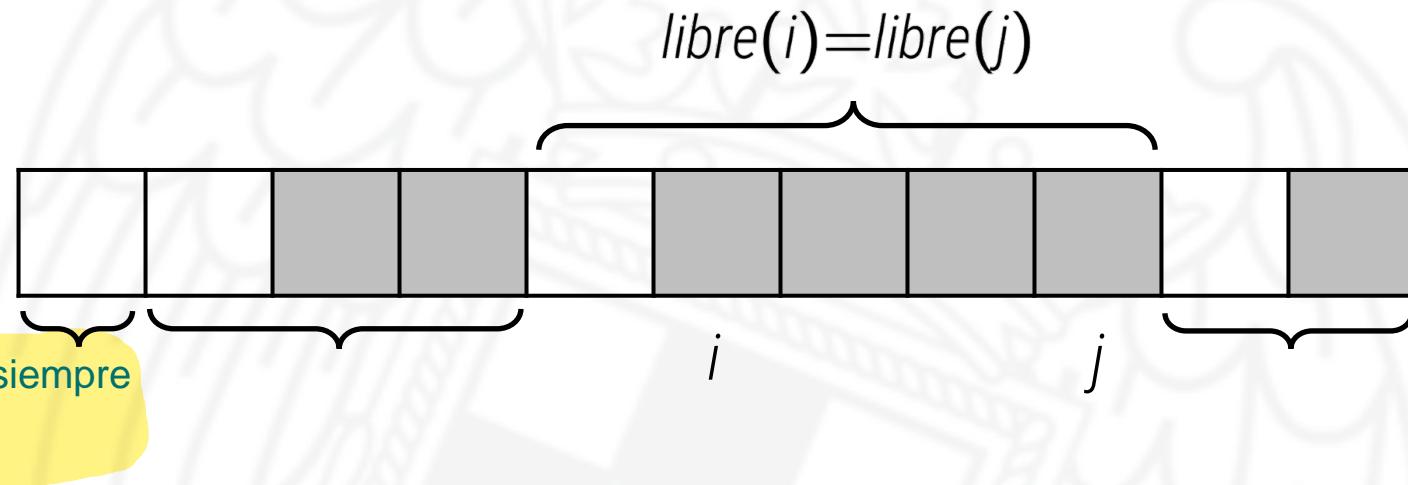
i	p_i	b_i
1	4	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10



Cómo sabemos el plazo mayor disponible para meter una tarea.

Test de factibilidad 2

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

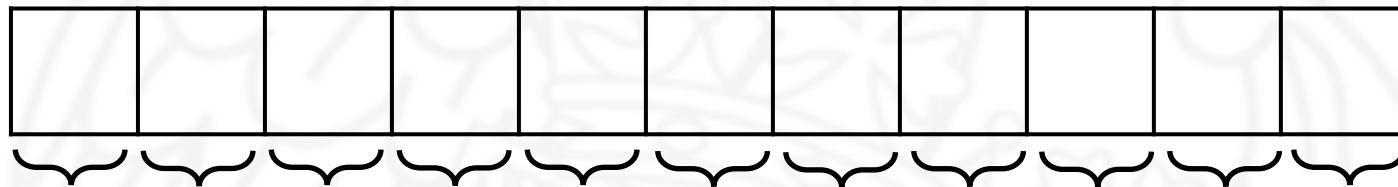


- ▶ Cada tarea t_i debería realizarse en el día $libre(p_i)$, que representa el último día libre que respeta su plazo.

Si es 0 es que están todos completados porque la tarea 0 siempre es libre.

Test de factibilidad 2

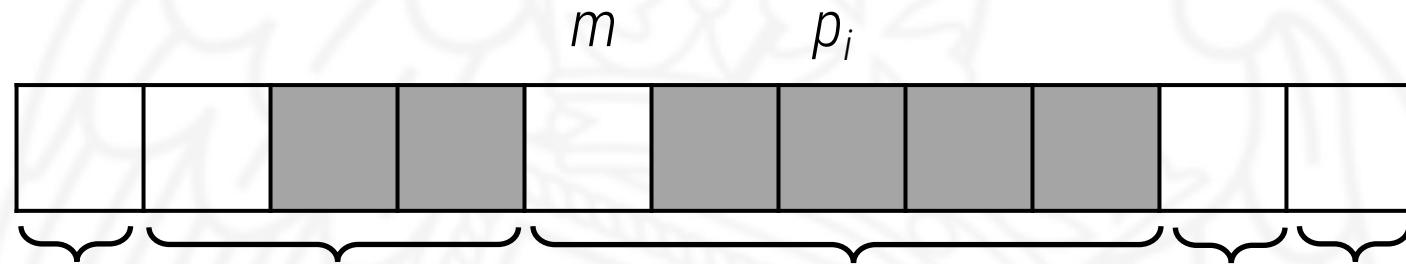
- ▶ Inicialmente $\forall i : 0 \leq i \leq n : libre(i) = i$



Cuando no hay ninguno puesto, el mayor libre es i .

Test de factibilidad 2

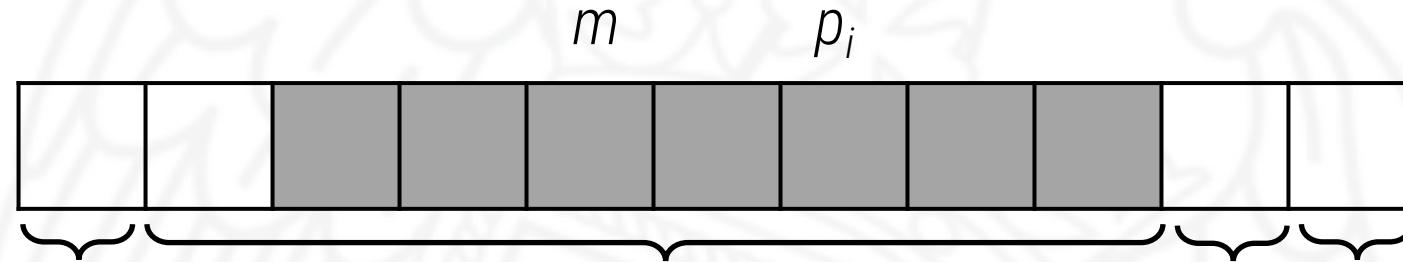
- ▶ Si se planifica la tarea t_i el día $\text{libre}(p_i) = m$, hay que unir su clase de equivalencia con el del día anterior.



Test de factibilidad 2

Vamos a utilizar CONJUNTOS DISJUNTOS PARA REPRESENTAR LAS CLASES DE EQUIVALENCIA.

- ▶ Si se planifica la tarea t_i el día $\text{libre}(p_i) = m$, 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;  
}
```

Las ordenamos de mayor a menor por beneficio

Implementación

Tareas seleccionadas.

Beneficio óptimo

// 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

O(N)

vector<int> libre(N + 1, 0); Para saber para cada día, cual es el día NO POSTERIOR más cercano libre.

for (int i = 0; i <= N; ++i)

libre[i] = i; Al principio, libre de i es = a i en todas sus posiciones.

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

Guardamos las tareas seleccionadas, cada una lo más tarde posible (puede haber huecos)

ConjuntosDisjuntos particion(N + 1);

Representan las clases de equivalencia de días de forma eficiente

int beneficio = 0;

Implementación

$O(N \log^* N)$

Cuasi lineal

```
// recorrer las tareas de mayor a menor beneficio
```

```
for (int i = 0; i < N; ++i) { Recorremos las tareas.
```

```
    int c1 = particion.buscar(min(N, tareas[i].plazo));
```

```
    int m = libre[c1]; Día libre de esa clase de equivalencia.
```

```
    if (m != 0) { // podemos colocar la tarea i
```

```
        plan[m] = tareas[i].id; tarea planificada para el día m.
```

```
        beneficio += tareas[i].beneficio;
```

```
        int c2 = particion.buscar(m-1);
```

```
        particion.unir(c1, c2);
```

```
        libre[c1] = libre[c2];
```

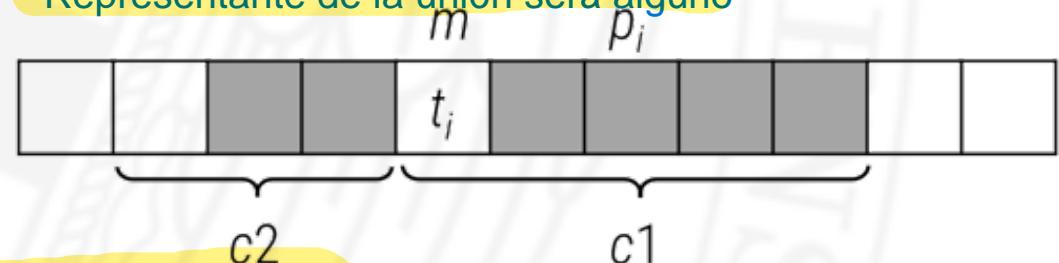
```
}
```

```
}
```

Buscamos la clase de equivalencia que corresponde a su plazo.

Unimos clases de equivalencia con el del día anterior.

Representante de la unión será alguno



Tarea se descartaría porque no cabe y pasaríamos a la siguiente.

Implementación

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

$O(N)$

TOTAL = $O(N \log N)$

POR REALIZAR EL SORT CON ALGORITHM (LA ORDENACIÓN)