.UBAfiuba
FACULTAD DE INGENIERÍA

Greedy: Planificación para minimizar latencia

Teoría de Algoritmos I (75.29 / 95.06)
Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Planificación para minimizar el latencia (retraso)

Sea

P un conjunto de n pedidos $\{p_1, p_2, \ldots, p_n\}$

Cada pedido i tiene

Una duración ti (no fraccionable)

Una fecha de entrega (deadline) di

Queremos

programar el inicio de cada unos de los pedidos a realizar sin superposición de forma de minimizar la máxima latencia

Latencia

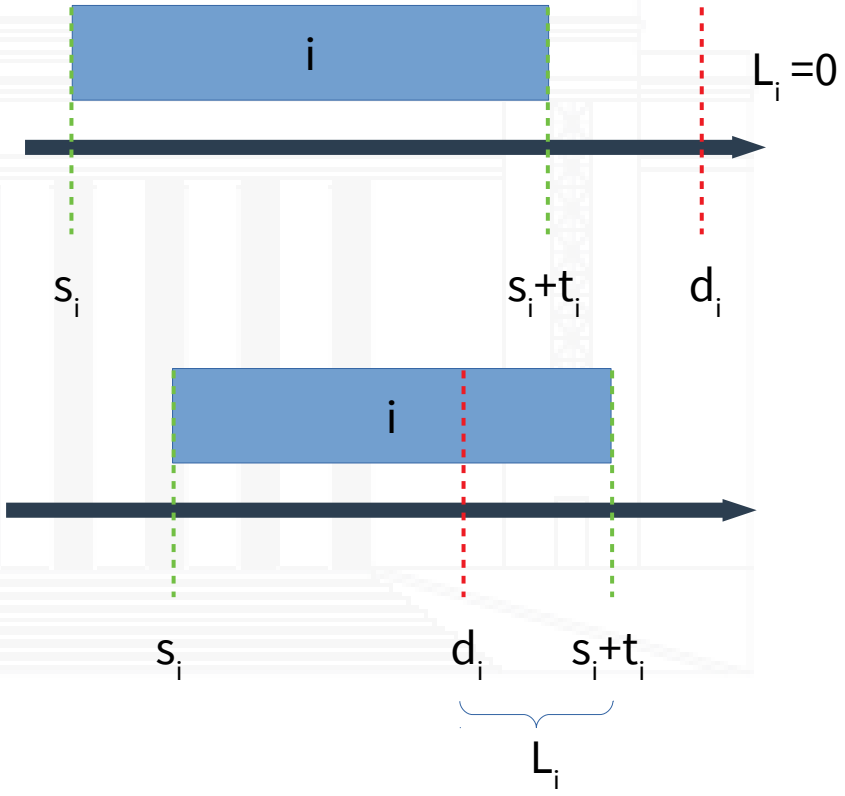
Si la tarea $i=(t_i,d_i)$

Se programa para comenzar en el tiempo s_i

Diremos que su latencia

$l_i = s_i + t_i - d_i$ si $(s_i + t_i - d_i) \geq 0$

$l_i = 0$ si $(s_i + t_i - d_i) < 0$



La latencia máxima

de una secuencia de pedidos programados

Corresponde a la maxima de las latencias

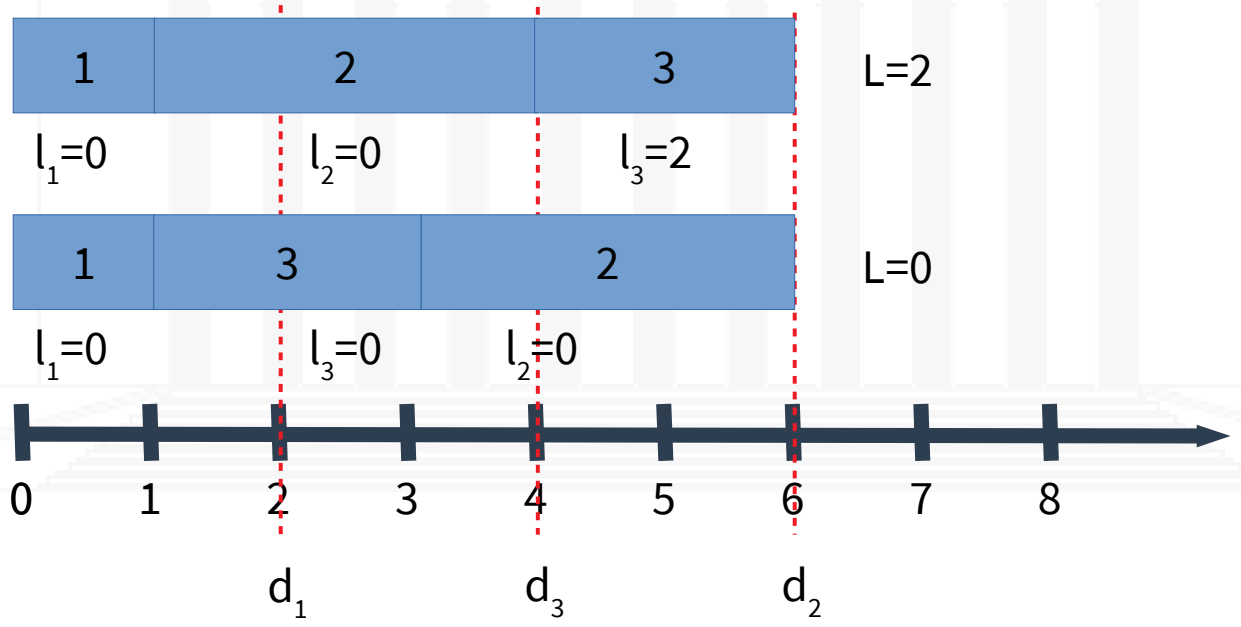
$L = Max_{1 a n}\ l_x$

Ejemplo

Contamos con los siguientes pedidos

1=(1,2) 2=(3,6) 3=(2,4)

¿Cómo los programamos?



Estrategia greedy

Podemos procesar los pedidos

De acuerdo a la duración de cada uno (del mas corto al más largo o viceversa)

De acuerdo a la diferencia entre el deadline y la duración ($d_i - t_i$)

De acuerdo al deadline (del más cercano al mas lejano)

Los primeros 2 no sirven

Podemos rápidamente encontrar contraejemplos que los desacredite

El tercero parece funcionar

Debemos demostrarlo!

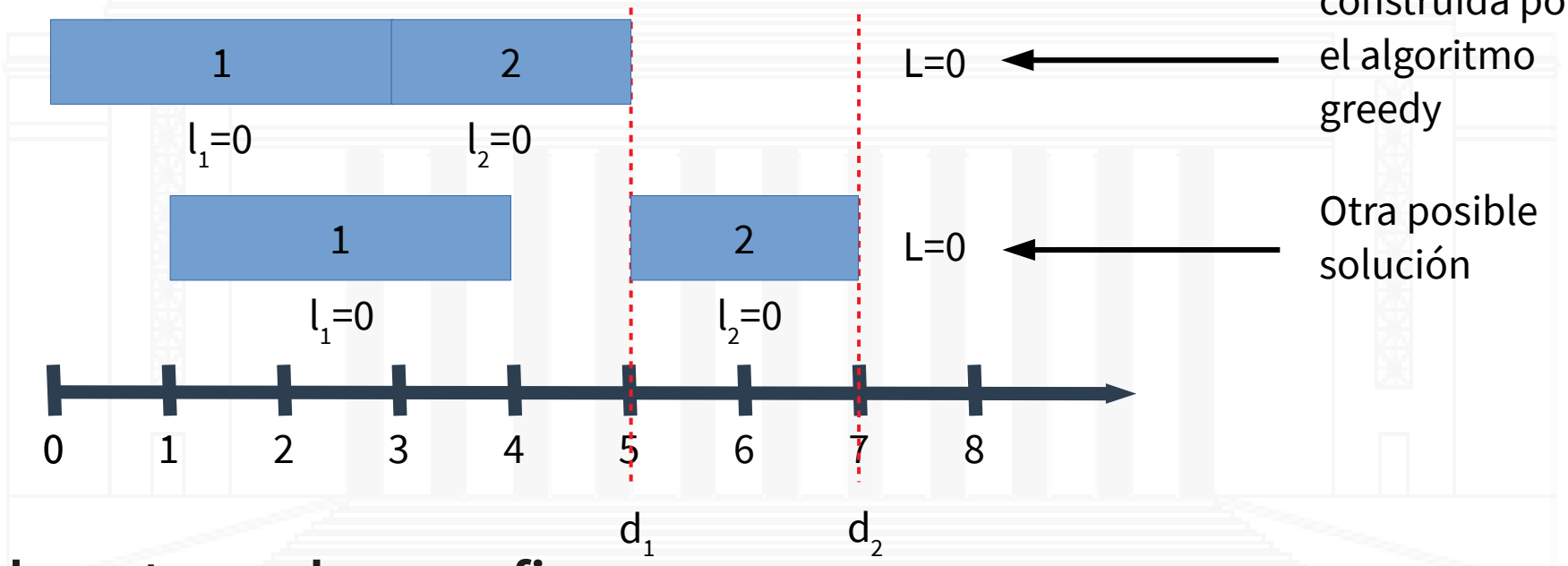
Pseudocódigo

```
Ordenar los pedidos en función de su deadline
(Asumiremos por simplicidad la notación d1 ≤ d2 ≤ … ≤ dn)

Inicialmente f=s    // s es la hora de inicio (podemos suponerlo como 0)
Desde el pedido i=1 al n
    Asignar el pedido i al intervalo s(i)=f a f(i)=f+ti
    f = f + ti

Retornar el set de intervalos programados [s(i), f(i)] para i=1 a n
```

Análisis de la solución: Tiempos muertos

La solución A que construye el algoritmo greedy

No deja tiempo "muertos" o de inactividad



Igualmente, podemos afirmar que

Existe una solución optima sin tiempos muertos

Optimalidad de la solución

Consideraremos una solución optima O

Esta puede ser cualquier solución optima posible

Intentaremos transformarla paso a paso

sin perder la optimalidad

Llegando finalmente a una programación

Idéntica a “A” la conseguida por el algoritmo greedy

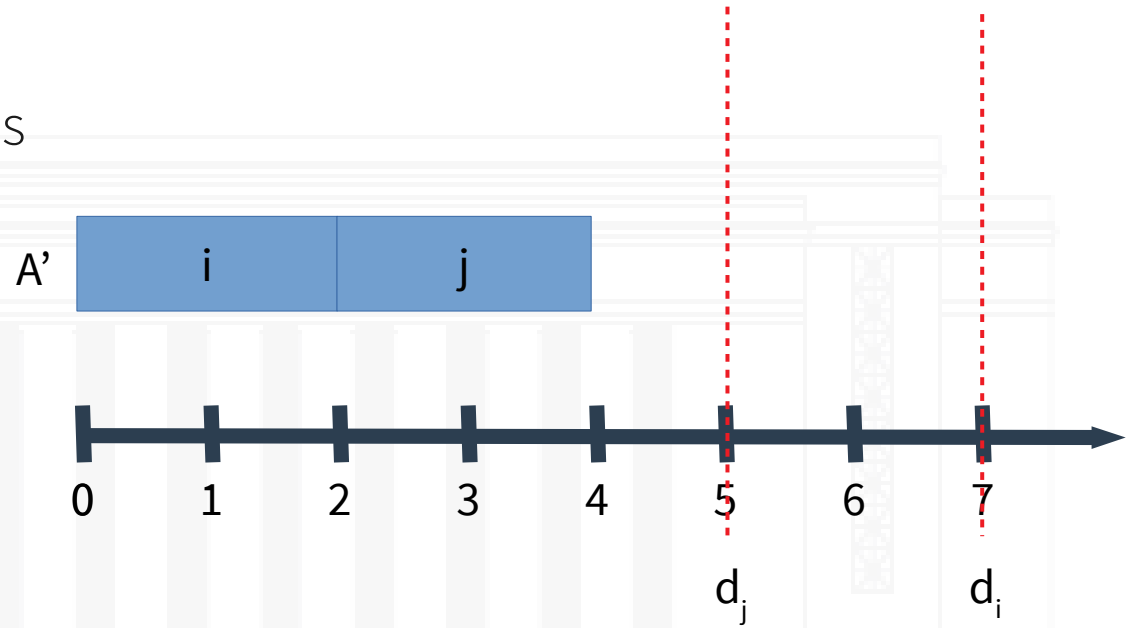
Llamaremos a este método de probar la optimalidad

Como un “argumento de intercambio”

Inversiones

Sea

A’ una programación de pedidos



Diremos

Que A’ tiene una <u>inversión</u>

Si

Un pedido $i=(t_i,d_i)$ esta programado antes de otro pedido $j=(t_j,d_j)$ y $d_j < d_i$

Programación sin inversión ni tiempos muertos

Sean

B,C programaciones sin inversiones ni tiempos muertos.

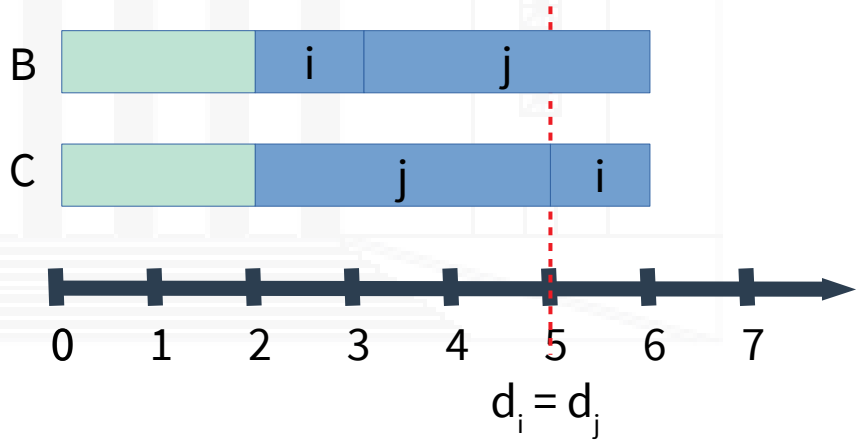
Entonces

B y C tienen el mismo tiempo de latencia total

Si B y C no tienen inversiones pero son diferentes

Entonces existen pedidos que comparten el mismo deadline

Invertir el orden de esos pedidos no altera la latencia máxima (*1)



Demostración: Paso 1

Sea

O una programación optima

Si O tiene tiempos muertos

Podemos eliminarlos desplazando los pedidos sin dejar de ser optimo

Si O tiene inversiones

Entonces existen al menos 2 pedidos contiguos i,i+1 tal que $d_{i+1} < d_i$

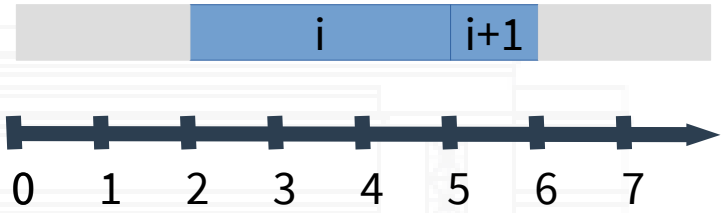
Podemos intercambiarlas para tener una inversión menos

Demostración: Inversión

Sea

O una programación optima con inversiones

i,i+1 una inversión de pedidos en O $(d_{i+1} < d_i)$



Realizamos una inversión entre i e i+1

Sabemos que la latencia máxima no puede disminuir (O no sería óptimo)

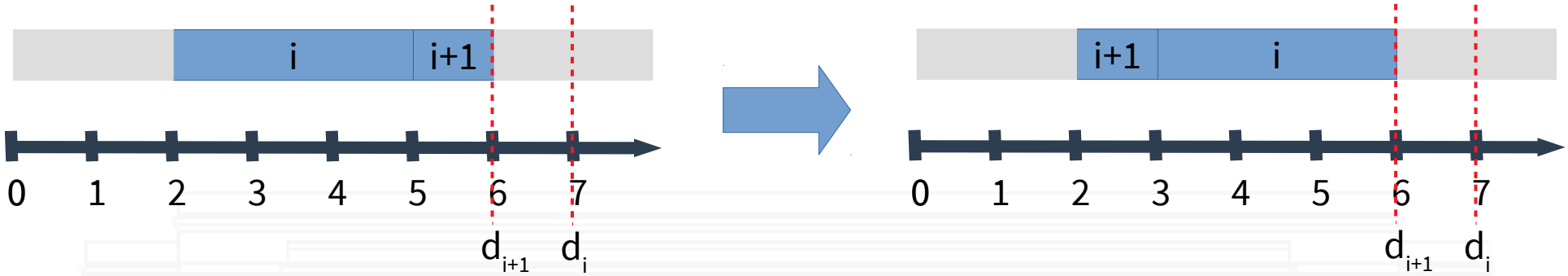
La latencia de i+1 puede mantenerse o disminuir

La latencia de i puede aumentar o mantenerse

La cantidad de inversiones disminuye en 1

No modifica la latencia de los pedidos no involucrados

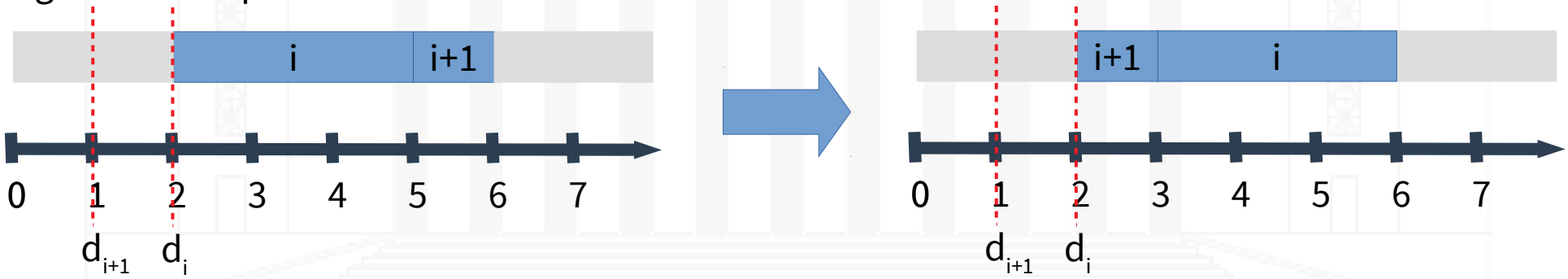
Casos de inversiones (cont.)



Ambos deadline después del termino de i+1
Intercambiar no empeora la latencia total.
Sigue siendo optimo

$L'_{i+1} = 2 + 1 - 6 < 0 \rightarrow l_{i+1}=0$

$L'_i = 3 + 3 - 7 < 0 \rightarrow l_i=0$



$L_{i+1} = 5 + 1 - 1 = 5$

$L_i = 2 + 3 - 2 = 3$

Mejoran ambos.
Podría empeorar?

$L'_{i+1} = 2 + 1 - 1 = 2$

$L'_i = 3 + 3 - 2 = 4$

Inversión - Generalización

Partimos de

O óptimo con inversiones

Llamamos

O' al optimo luego de intercambio

Podemos ver que

La tarea i+1 terminara antes. Su latencia se mantiene o disminuye.

La tarea i puede aumentar su latencia.

Puede aumentar el máximo de la programación?

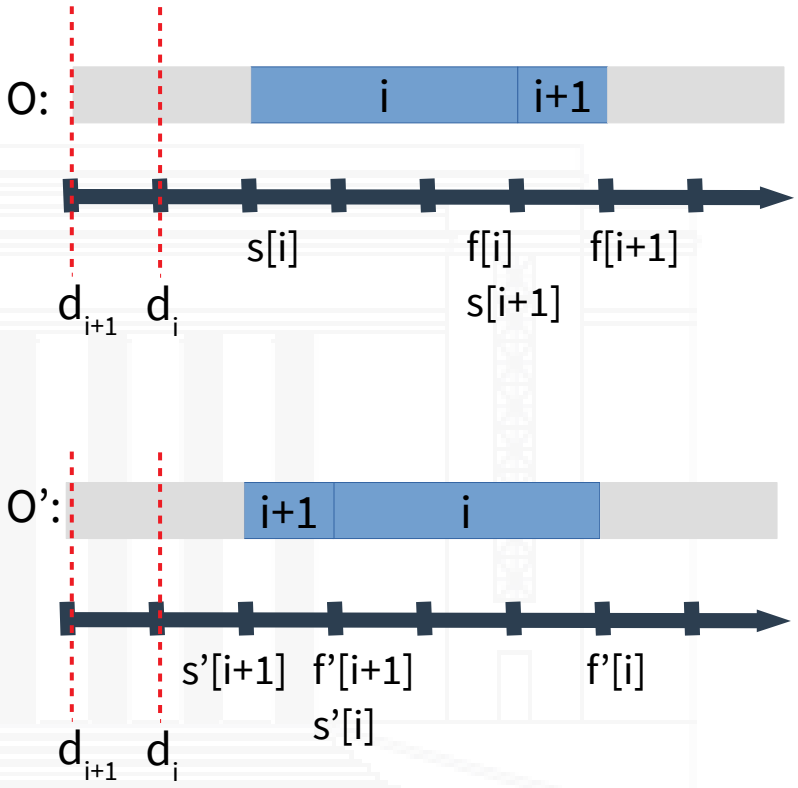
l'[i]=f'[i]-d[i]

Vemos que f[i+1] = f'[i]

Por lo tanto l'[i]=f[i+1]-d[i]

Como d[i]>d[i+1]

Entonces l'[i]=f[i+1]-d[i] < f[i+1]-d[i+1] = l[i+1] → **l'[i] < l[i+1]**



Una resolver una inversión mejora o mantiene la latencia máxima

Programación optima sin inversiones

Como

$L \geq l_{i+1} > l'_i$

Entonces

El intercambio no incrementa la latencia máxima de la programación

Mientras exista una inversión

Realizaremos otros intercambio.

Al finalizar

nos quedaremos con una programación optima, sin inversiones. (*2)

Colorario

Como

Existe un optimo sin tiempos muertos y sin inversiones (*2)

Y

Cualquier programación sin tiempos muertos e inversiones es equivalente o otra programación sin tiempos muertos e inversiones (*1)

Y

El algoritmo greedy retorna una programación sin tiempos muertos e inversiones

Entonces

La solución entregada por greedy es óptima

Complejidad

Complejidad temporal

Ordenar los pedidos: O(n log n)

Iterar sobre ellos O(n)

Complejidad total: O(n log n)

Complejidad espacial:

Almacenar los intervalos o(n)

```
Ordenar los pedidos en función de su deadline
(Asumiremos por simplicidad la notación d_1 ≤
d_2 ≤ … ≤ d_n)

Inicialmente f=s    // s es la hora de inicio
(podemos suponerlo como 0)
Desde el pedido i=1 al n
     Asignar el pedido i al intervalo s(i)=f a
f(i)=f+t_i
     f = f + t_i

Retornar el set de intervalos programados
[s(i), f(i)] para i=1 a n
```

.UBAfiuba

FACULTAD DE INGENIERÍA

Presentación realizada en Abril de 2021