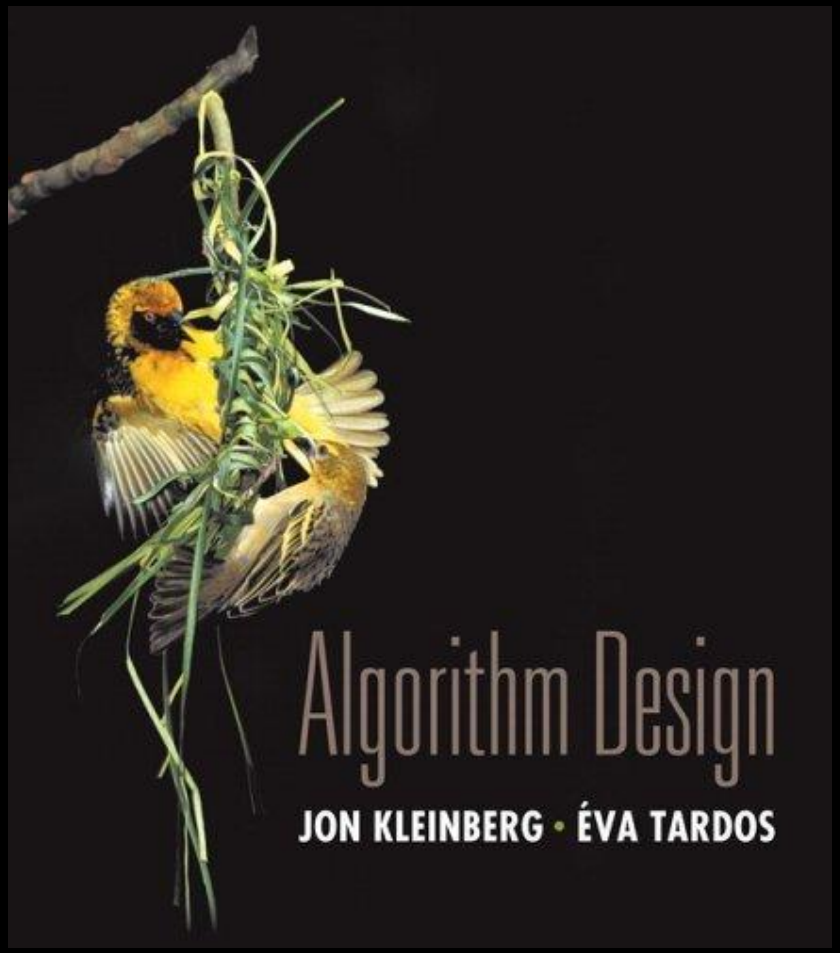


Capítulo 4

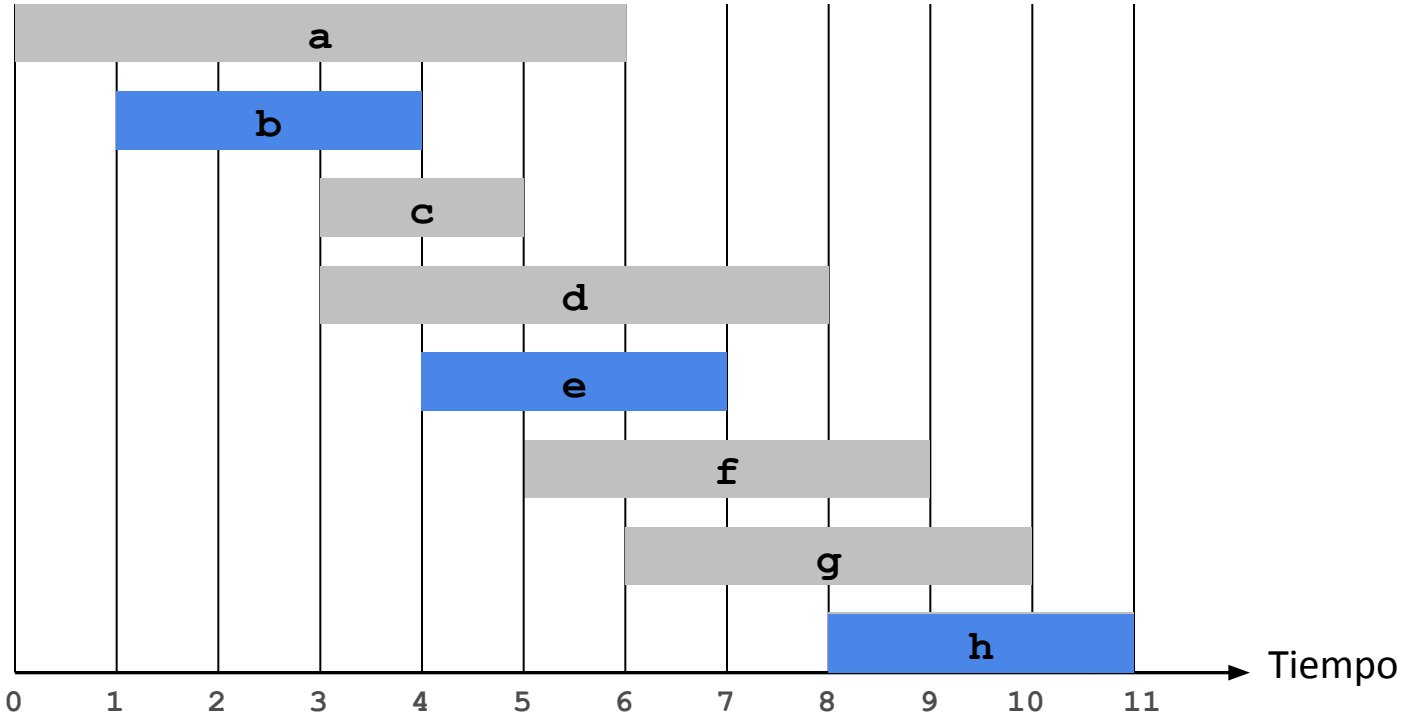
Algoritmos voraces (Greedy)

Un algoritmo es voraz si construye una solución en pequeños pasos, en cada paso tomando una decisión local (sin contexto) para optimizar algún criterio subyacente.

4.1 Planificación de intervalos

Planificación de intervalos.

- Trabajo j empieza en s_j y termina en f_j.
- Dos trabajos son compatibles si no solapan.
- Objetivo: encontrar un subconjunto de intervalos compatibles entre sí de tamaño máximo.



Planificación de intervalos: Algoritmos greedy

Greedy template. Considerar los trabajos en algún orden natural.

Tomar cada trabajo siempre que sea compatible con los ya tomados.

- Inicializar una solución vacía **A**.
- Mientras haya trabajos compatibles con los de **A**:
  - Seleccionar el trabajo compatible **j**, que sea el mejor según algún criterio.
  - Añadirlo a **A**.
- Devolver **A** como la solución.

Planificación de intervalos: Algoritmos greedy

Algunos ejemplos de criterios para seleccionar el mejor trabajo.

- [Tiempo de inicio más temprano] Considerar los trabajos en orden ascendente de $\mathbf{s_j}$.
- [Tiempo de finalización más temprano] Considerar los trabajos en orden ascendente de $\mathbf{f_j}$.
- [Intervalo más corto] Considerar los trabajos en orden ascendente de $\mathbf{f_j - s_j}$.
- [Mínimos conflictos] Para cada trabajo **j**, contar el número de trabajos en conflicto $\mathbf{c_j}$. Planificar en orden ascendente de $\mathbf{c_j}$.

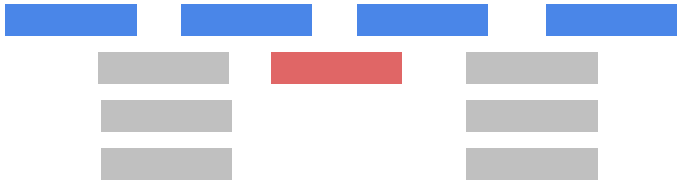
Planificación de intervalos: Algoritmos greedy

Algunos contraejemplos.

Solución óptima
Primer intervalo elegido

contraejemplo para el inicio más temprano

contraejemplo para el intervalo más corto



contraejemplo para mínimos conflictos

Planificación de intervalos: Algoritmos greedy

Algoritmo greedy. Considere los trabajos en orden creciente de tiempo de finalización. Tomar cada trabajo siempre que sea compatible los ya tomados.

```
1.  Ordenar trabajos tal que f_1 ≤ f_2 ≤ ... ≤ f_n.

    Conjunto de trabajos seleccionados
2.  A ← φ
3.  for j = 1 to n {
4.     if (trabajo j compatible con A)
5.        A ← A ∪ {j}
6.  }
7.  return A
```

Implementación. O(n log n).

- Recordar el trabajo j* que fue añadido último a A.
- j es compatible con A si cumple que $s_j \geq f_{j^*}$.

Planificación de intervalos: Análisis

Teorema. El algoritmo greedy retorna una solución óptima.

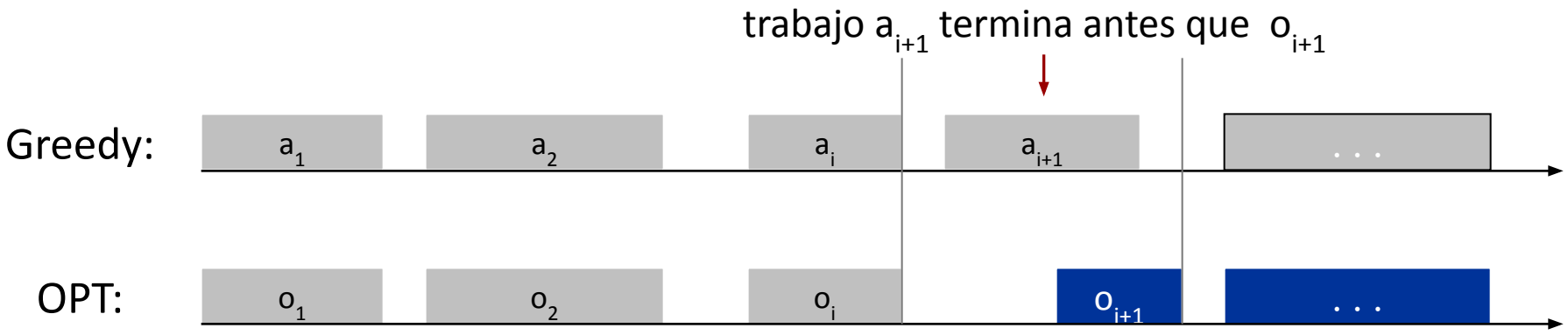
Dem.

- Termina. ■
- Retorna un conjunto de trabajos compatibles. ■
- La solución es óptima (es de tamaño máximo). (?)

Estrategia: **"The greedy algorithm stays ahead"**, se mantiene por delante.

- Consideramos una solución óptima **O**.
- Encontramos una forma de **medir** nuestra solución **A**, y mostramos que **siempre** "está por delante" que la solución **O**.
- Utilizamos lo que mostramos para probar que $|\mathbf{A}| = |\mathbf{O}|$.

Planificación de intervalos: Análisis



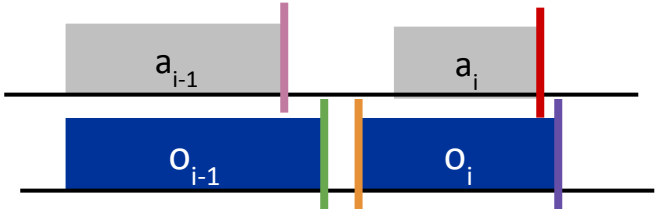
Prop. Sea **O** un conjunto de requerimientos compatibles.

Sean $o_1, o_2, \ldots o_m$ los elementos de **O** en orden de inicio.

Sean $a_1, a_2, \ldots a_k$ los elementos de **A** en orden de inicio.

Queremos probar que $f(a_i) \leq f(o_i) \forall i, 1 \leq i \leq min\{m, k\}$

Planificación de intervalos: Análisis



Dem. Por inducción

- **Paso base:** Para i = 1, se cumple por el paso 4 toma un elemento con mínimo valor de **f(i)**.
- **Paso inductivo:** Para i > 1, asumimos que se cumple para i-1 **(HI)**.
- Como $\mathbf{o_i}$ es compatible y posterior a $\mathbf{o_{i-1}}$, $f(o_{i-1}) \leq s(o_i)$.
- Por otro lado, por **(HI)**, $f(a_{i-1}) \leq f(o_{i-1})$.
- Entonces $f(a_{i-1}) \leq s(o_i)$.
- Por lo tanto $\mathbf{o_i}$ es compatible con $A=\{a_1, \ldots, a_{i-1}\}$
- En consecuencia, como en el paso 4 el algoritmo elige $\mathbf{a_i}$ como el trabajo compatible de menor tiempo de finalización, debe cumplirse que $f(a_i) \leq f(o_i)$. ■

Planificación de intervalos: Análisis

Teorema. El algoritmo greedy propuesto es óptimo.

Dem. Por absurdo

- Sea **O={**o_1, o_2, ... o_m**}** una solución óptima y **A={**a_1, a_2, ... a_k**}** la solución de nuestro algoritmo.

- Supongamos por absurdo que **m > k**.
- Como $\mathbf{o_{k+1}}$ es compatible con $\mathbf{o_k}$, $\mathbf{s(o_{k+1}) \geq f(o_k) \geq f(a_k)}$.
- Entonces $\mathbf{o_{k+1}}$ es compatible con **A**.
- Sin embargo, para obtener **A** el algoritmo terminó, lo que implica que, por la condición del paso 2, no hay ningún otro trabajo compatible con **A**.
- Llegamos a un absurdo, y concluímos que **m** = **k**. ■

4.2 Planificación para minimizar la tardanza

Planificación para minimizar la tardanza

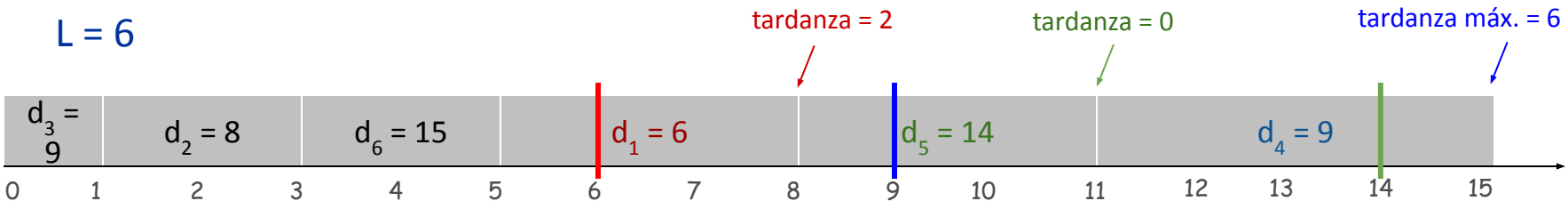
El problema de minimizar la tardanza.

- Un solo recurso procesa un trabajo a la vez.
- El trabajo j requiere t_j unidades de tiempo de procesamiento y tiene como plazo el tiempo d_j.
- Si j comienza en el tiempo s_j, termina en el tiempo $f_j = s_j + t_j$.
- Tardanza: $l_j = \max \{ 0, f_j - d_j \}$.
- Objetivo: planificar todos los trabajos para minimizar la tardanza máxima $L = \max l_j$.

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

Ej: 3, 2, 6, 1, 5, 4

L = 6



Minimizando la tardanza: Algoritmos greedy

Greedy template. Considerar los trabajos en algún orden.

- [Menor tiempo de procesamiento primero] Considerar los trabajos en orden ascendente de tiempo de procesamiento t_j.

- [Plazo más temprano primero] Considerar los trabajos en orden ascendente de plazo d_j.

- [Menor holgura] Considerar los trabajos en orden asc. de holgura $d_j - t_j$.

Minimizando la tardanza: Algoritmos greedy

Greedy template. Considerar los trabajos en algún orden.

- [Menor tiempo de procesamiento primero] Considerar los trabajos en orden ascendente de tiempo de procesamiento t_j.

	1	2
t_j	1	10
d_j	100	10

contraejemplo

- [Menor holgura] Considerar los trabajos en orden asc. de holgura $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

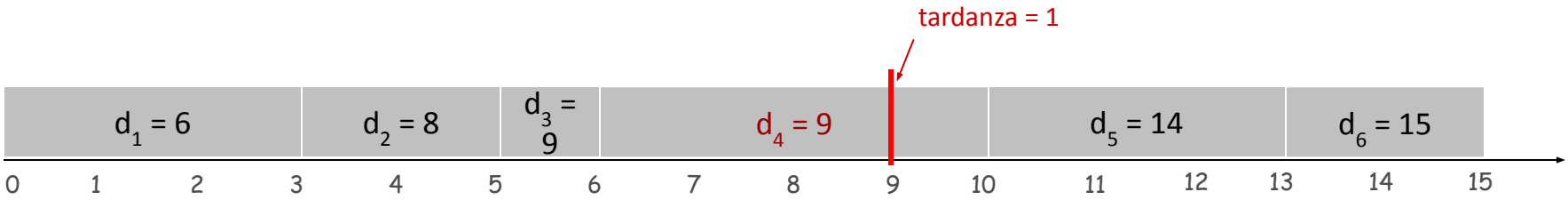
contraejemplo

Minimizando la tardanza: Algoritmos greedy

Algoritmo greedy. Plazo más temprano primero.

```
1.  Ordenar n trabajos por plazo tal que d_1 ≤ d_2 ≤
    … ≤ d_n

2.  t ← 0
3.  for j = 1 to n
4.     Asignar a j el intervalo [t, t + t_j]
5.     s_j ← t, f_j ← t + t_j
6.     t ← t + t_j
7.  retornar intervalos [s_j, f_j]
```



Minimizando la tardanza : Análisis

Teorema. El algoritmo greedy retorna una solución óptima.

Dem.

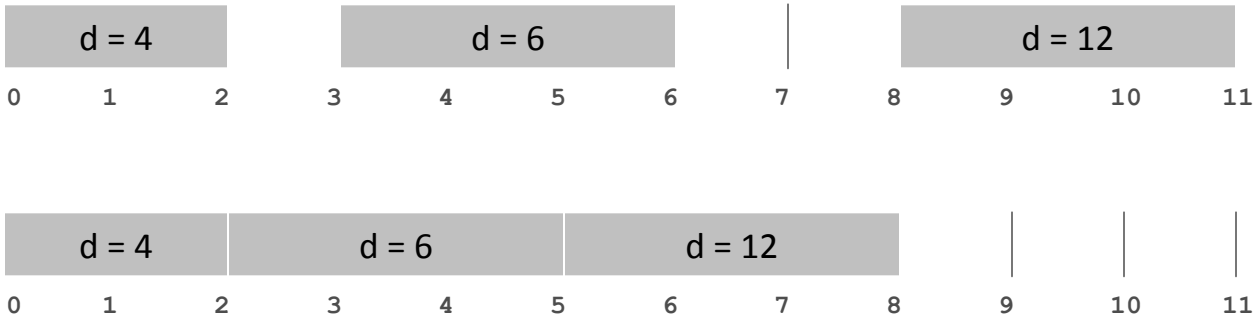
- Termina. ■
- La solución es óptima (tiene mínima tardanza máxima). (?)

Estrategia: **"Exchange argument"**, por argumento de intercambio.

- Consideramos una solución óptima **O**.
- Definimos algún sentido en el que **O** y **A** difieren.
- Intercambiamos partes de **O** de manera que se asemeje más a nuestra solución **A**, y mostramos que ese intercambio *no empeora* la calidad de la solución.
- Intercambiamos partes hasta que **O = A**, mostrando que **A** también es óptima.

Minimizando la tardanza: Sin tiempo ocioso

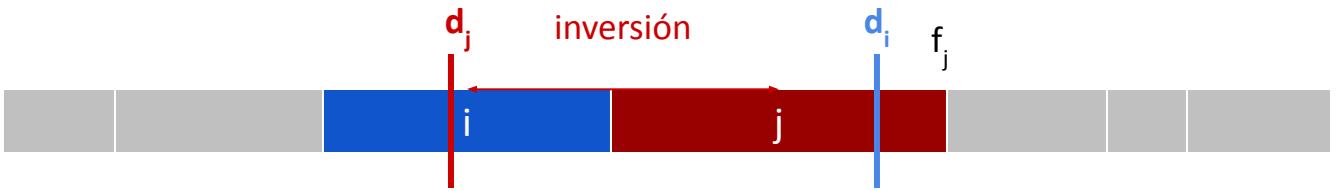
Prop. Existe una planificación óptima sin tiempo ocioso.



Observación. La planificación greedy no tiene tiempo ocioso.

Minimizando la tardanza: Inversiones

Def. Dada una planificación S, una inversión es un par de trabajos i y j tales que: $i < j$ pero $d_j < d_i$.



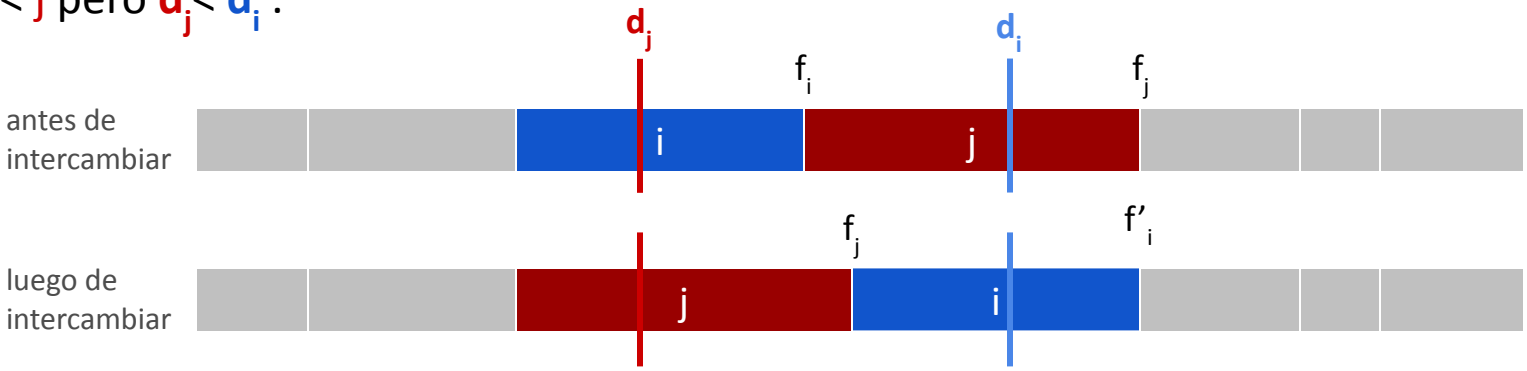
[como antes, asumimos a los trabajos enumerados de forma que $d_1 \leq d_2 \leq \ldots \leq d_n$]

Observación. La planificación greedy **no** tiene inversiones.

Prop. Si una planificación (sin tiempo ocioso) tiene una inversión, tiene al menos una con un par de trabajos invertidos planificados de forma **consecutiva**.

Minimizando la tardanza: Inversiones

Def. Dada una planificación S, una inversión es un par de trabajos i y j tales que: $i < j$ pero $d_j < d_i$.



Prop. Intercambiar dos trabajos consecutivos invertidos *reduce* el número de inversiones en uno y *no aumenta* la tardanza máxima.

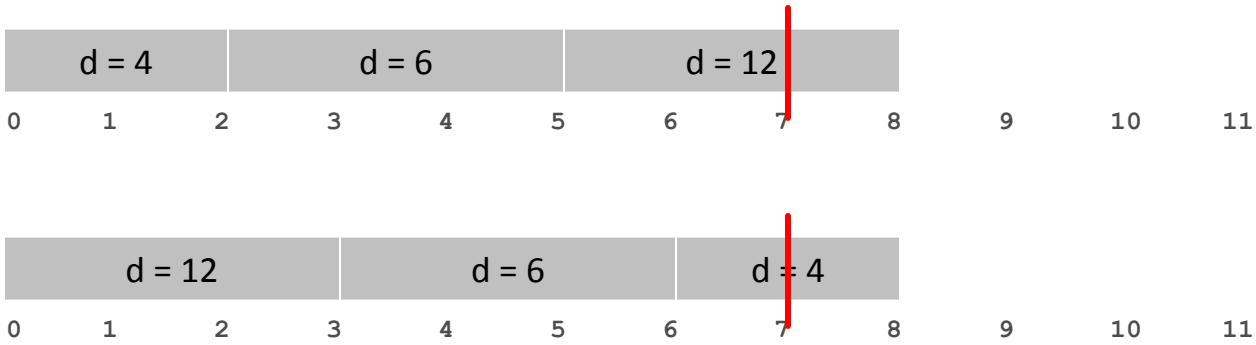
Dem. Sea L la tardanza antes del intercambio, y sea L' luego.

- $l'_k = l_k$ para todo $k \neq i, j$, *porque* no cambia ningún otro tiempo de finalización
- $l'_j \leq l_j$, *porque* $f'_j \leq f_j$
- $l'_i \leq ?$

$$l'_i = f'_i - d_i = f_j - d_i < f_j - d_j = l_j$$

Minimizando la tardanza: Sin tiempo ocioso ni inversiones

Observación. Todas las planificaciones sin tiempo ocioso y sin inversiones tienen la misma tardanza máxima. Si tienen el mismo plazo, el orden en que estén no cambia la tardanza máxima.



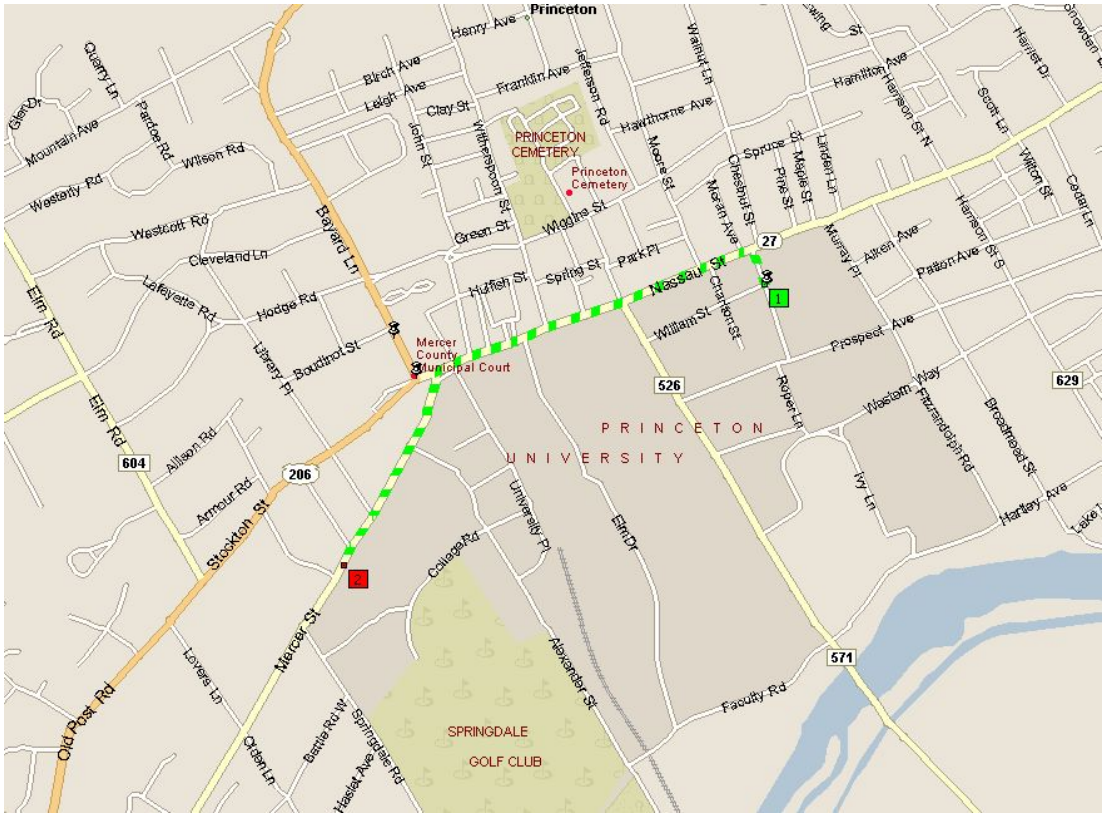
Minimizando la tardanza: Análisis del algoritmo greedy

Teorema. El algoritmo greedy retorna una solución óptima **S**.

Dem. Sea **O** una planificación óptima sin tiempo ocioso.

- Podemos deshacer todas las inversiones en **O** en una cantidad finita de pasos (hay una cantidad máxima de inversiones y cada vez que deshacemos una no se forma ninguna otra), obteniendo una nueva planificación óptima **O'** que tiene mínima tardanza máxima.
- Como **O'** no tiene tiempo ocioso ni inversiones, tiene la misma tardanza máxima que **S**, probando que **S** es óptima. ■

4.4 Camino más corto en un grafo



camino más corto desde el departamento de ciencias de la computación de Princeton a la casa de Einstein

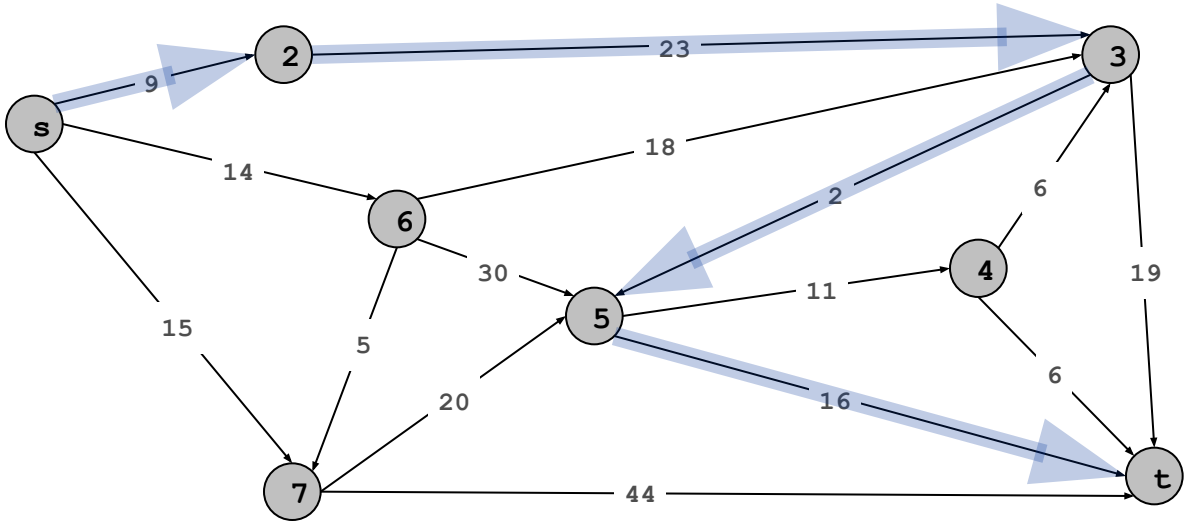
Problema del camino más corto

Red del camino más corto.

- Grafo dirigido **G = (V, E)**.
- Fuente **s**, destino **t**.
- Largo $\mathbf{l_e}$ = largo de la arista **e**.

Problema del camino más corto: encontrar el camino dirigido de menor costo de **s** a **t**.

costo del camino = suma de los costos de las aristas del camino



Costo del camino s-2-3-5-t
= 9 + 23 + 2 + 16
= 50.