Algoritmos codiciosos

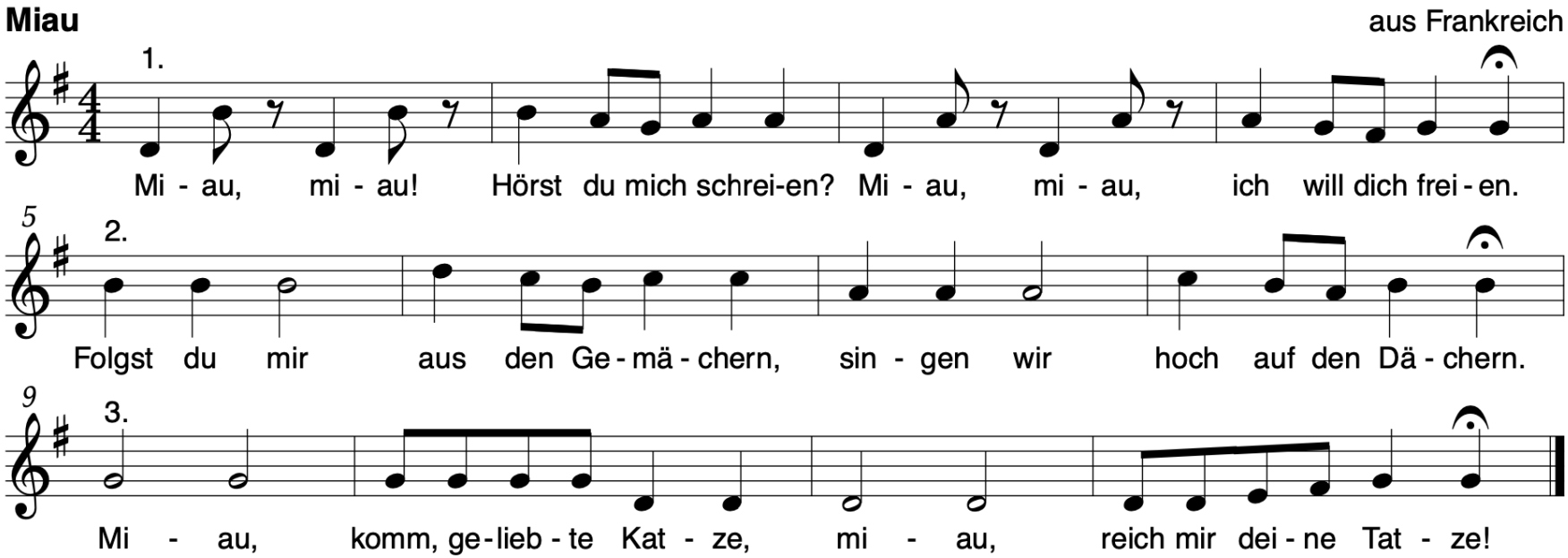
Clase 15

IIC 2133 - Sección 1

Prof. Sebastián Bugedo

Sumario

¿Cómo están?



Miau, miau, hörst du mich schreien?
Miau, miau, ich will dich freien.

Folgst du mir aus den Gemächern,
singen wir hoch auf den Dächern.

Miau, komm, geliebte Katze,
miau, reich mir deine Tatze!

Tercer Acto: Los jinetes de la salvación

Estrategias de diseño de algoritmos

Playlist 3

Playlist: DatiWawos Tercer Acto

Además sigan en instagram:
@orquesta_tamen

Problemas en computación

Hasta este punto hemos estudiado diversos tipos de problemas

- Ordenar una secuencia
- Buscar en un diccionario
- Determinar si un CSP tiene solución
- Determinar las soluciones de un CSP, en caso que tenga
- Determinar la mejor solución de un CSP, en caso que tenga
- ...

¿Podríamos agrupar estos problemas según rasgos comunes?

Problemas en computación

Diremos que un **problema computacional** es un problema que puede resolverse con un **algoritmo**

¿Qué tipos de problemas computacionales existen?

Distinguiremos 5 tipos

Tipos de problemas computacionales

1. Problemas de decisión
   - Pregunta binaria (Sí o No)
   - E.g. "*Determine si el tablero de Sudoku T tiene solución*"
2. Problemas de búsqueda
   - Secuencia de estados hasta alcanzar un estado objetivo
   - E.g. "*Determine un camino para salir del laberinto L desde Θ*"
3. Problemas de conteo
   - Número de soluciones diferentes
   - E.g. "*Determine el número de configuraciones válidas de 8 reinas*"
4. Problemas de optimización
   - Mejor solución de acuerdo a alguna métrica
   - E.g. "*Determine el camino más corto de A a B en el mapa G*"
5. Problemas de función
   - Solución explícita
   - E.g. "*Determine una solución del Sudoku T*"

Backtracking y optimización

Backtracking es una técnica de diseño muy flexible

- Podemos atacar muchos tipos de problemas con ella
- PERO, no es la mejor solución para algunos

Especialmente, Backtracking no suele ser la mejor solución a los problemas de **optimización**

¿Tenemos una mejor estrategia para optimización?

Problema de la mochila

Ejemplo

Considere el **problema de la mochila con objetos fraccionables**.

Tenemos n objetos y una mochila

- Los objetos tienen pesos $\{w_1, \ldots, w_n\}$
- Los objetos tienen ganancias por unidad de peso $\{p_1, \ldots, p_n\}$
- La mochila tiene una capacidad m, en peso
- Incluir una fracción x_k del objeto k proporciona ganancia $p_k x_k$

Problema de la mochila

Ejemplo

Interesa llenar la mochila cumpliendo tres condiciones

- Queremos maximizar la ganancia total

$$\sum_{k=1}^{n} p_k x_k$$

  Esta es la **función objetivo.**

- No podemos exceder la capacidad de la mochila

$$\sum_{k=1}^{n} w_k x_k \leq m$$

- Las fracciones deben cumplir

$$0 \leq x_k \leq 1, \quad 1 \leq k \leq n$$

Problema de la mochila

Ejemplo

Interesa llenar la mochila cumpliendo tres condiciones

- Maximizar función objetivo $\sum_{k=1}^{n} p_k x_k$
- No podemos exceder la capacidad $\sum_{k=1}^{n} w_k x_k \leq m$
- Las fracciones deben cumplir $0 \leq x_k \leq 1$ para $1 \leq k \leq n$

Una **solución factible** es $\{x_1, \ldots, x_n\}$ que cumple las dos últimas condiciones. Una **solución óptima** es una solución factible que maximiza la función objetivo.

¿Cómo escoger los valores x_k adecuados
para encontrar una solución óptima? ¿Alguna idea?

Objetivos de la clase

- ☐ Comprender el paradigma de algoritmos codiciosos
- ☐ Demostrar que una estrategia no es correcta como estrategia codiciosa
- ☐ Aplicar la estrategia codiciosa para obtener óptimos en problemas particulares

Sumario

Algoritmos codiciosos

Los **algoritmos codiciosos** plantean una estrategia algorítmica basada en el paradigma de subconjuntos

1. Tenemos conjunto $S = \{s_1, \ldots, s_n\}$ con n inputs
2. Queremos un subconjunto $S' \subseteq S$ que satisfaga restricciones
3. Queremos solución factible que maximice o minimice una **función objetivo**

Un subconjunto S' que cumple las restricciones se llama **factible**. Una solución que maximiza/minimiza se llama **óptima**

Esta es una concepción teórica
Veremos su aplicación a problemas concretos

Algoritmos codiciosos

Los **algoritmos codiciosos** trabajan en etapas

- Consideran un input a la vez
- Una vez que se decide sobre un input, la decisión **es final**
- Ninguna decisión posterior cambia la actual

En el caso del problema de la mochila

- Se trata de seleccionar un subconjunto de objetos
- Y determinar la fracción x_k

Algoritmos codiciosos

Para lograr este objetivo, debemos considerar los input en cierto orden

- Se usa un **procedimiento de selección**
- Si la inclusión del próximo input en la solución óptima parcial produce solución infactible, no lo consideramos
- En otro caso, el input se agrega a la solución

¿Qué diferencia hay con backtracking?

Algoritmos codiciosos

El procedimiento de selección se basa en una **medida de optimización** o **estrategia codiciosa**

- Seleccionamos un input de forma **localmente óptima**
- Esperamos que esa selección nos lleve a una solución globalmente óptima

¿Qué estrategias codiciosas podemos usar?

Algoritmos codiciosos

Ejemplo

Para el problema de la mochila, podemos considerar las siguientes estrategias codiciosas *para la etapa actual del algoritmo*

- Incluir el objeto con mayor ganancia
- Incluir el con menor peso
- Incluir el que tenga mayor cuociente ganancia/peso

Normalmente, la mayoría de las estrategias no producen soluciones óptimas

Ejemplo

Considere la siguiente instancia del problema de la mochila con $m = 20$ y $n = 3$

- pesos $w_1 = 18,\ w_2 = 15,\ w_3 = 10$
- ganancias $p_1 = 25,\ p_2 = 24,\ p_3 = 15$

Algunas soluciones **factibles**

estrategia	x_1	x_2	x_3	$\sum_{k=1}^{n} w_k x_k$	$\sum_{k=1}^{n} p_k x_k$
mayor ganancia	1	2/15	0	20	28.2
menor peso	0	2/3	1	20	31
mayor ganancia/peso	0	1	1/2	20	31.5

En este problema, el óptimo se encuentra privilegiando ganancia/peso

Algoritmos codiciosos

Dado un problema

- Varias estrategias codiciosas pueden ser plausibles
- La mayoría produce soluciones **subóptimas**

Para garantizar que una estrategia produce soluciones óptimas es necesario **demostrarlo**

En este curso, no nos preocuparemos de ese último aspecto

Algoritmos codiciosos

input : Arreglo de inputs $A[0 \ldots n-1]$, cantidad de inputs n

`Greedy`(A, n)**:**

```
1    S ← ∅
2    for i = 1 ... n :
3        x ← Select(A)
4        if Feasible(S, x) :
5            S ← Union(S, x)
6    return S
```

Tal como en backtracking, esta es una idea abstracta...
Su implementación dependerá de cada problema

Sumario

Selección de tareas

Consideremos el problema de escoger tareas

- La tarea i tiene un plazo d_i (día del mes)
- Además tiene una ganancia p_i que se obtiene si la tarea se hace a tiempo (antes del plazo)

Una tarea toma **un día en completarse** y solo se puede realizar **una tarea al día** y

Objetivo: maximizar la ganancia total

Selección de tareas

Una solución factible será un subconjunto T se tareas que se pueden realizar en algún orden

- El **valor** de T será

$$p(T) = \sum_{k \in T} p_k$$

Una solución factible T es óptima si su valor $p(T)$ es máximo

Selección de tareas

Ejemplo

Suponiendo que hoy es el día 0, sean las tareas $\{1, 2, 3, 4\}$ tales que

- Sus plazos son $[d_1, d_2, d_3, d_4] = [2, 1, 2, 1]$
- Sus ganancias son $[p_1, p_2, p_3, p_4] = [100, 10, 15, 27]$

Consideremos la estrategia: escoger la tarea que entrega más ganancia cada día

Es decir, estamos usando la función objetivo como estrategia codiciosa

Selección de tareas

Ejemplo

Suponiendo que hoy es el día 0, sean las tareas $\{1, 2, 3, 4\}$ tales que

- Sus plazos son $[d_1, d_2, d_3, d_4] = [2, 1, 2, 1]$
- Sus ganancias son $[p_1, p_2, p_3, p_4] = [100, 10, 15, 27]$

Tenemos entonces

Solución factible T	Orden de procesam.	Valor $p(T)$
$\{1, 2\}$	2,1	110
$\{1, 3\}$	1,3 o 3,1	115
$\{1, 4\}$	4,1	127
$\{2, 3\}$	2,3	25
$\{3, 4\}$	4,3	42
$\{1\}$	1	100
$\{2\}$	2	10
$\{3\}$	3	15
$\{4\}$	4	27

Programación de charlas

Consideremos ahora el problema de asignar charlas en una misma sala

- Tenemos n charlas por asignar
- La charla i tiene hora de inicio s_i y de término f_i
- Es decir, se define el intervalo de tiempo $[s_i, f_i)$

Solo se puede realizar **una charla a la vez**

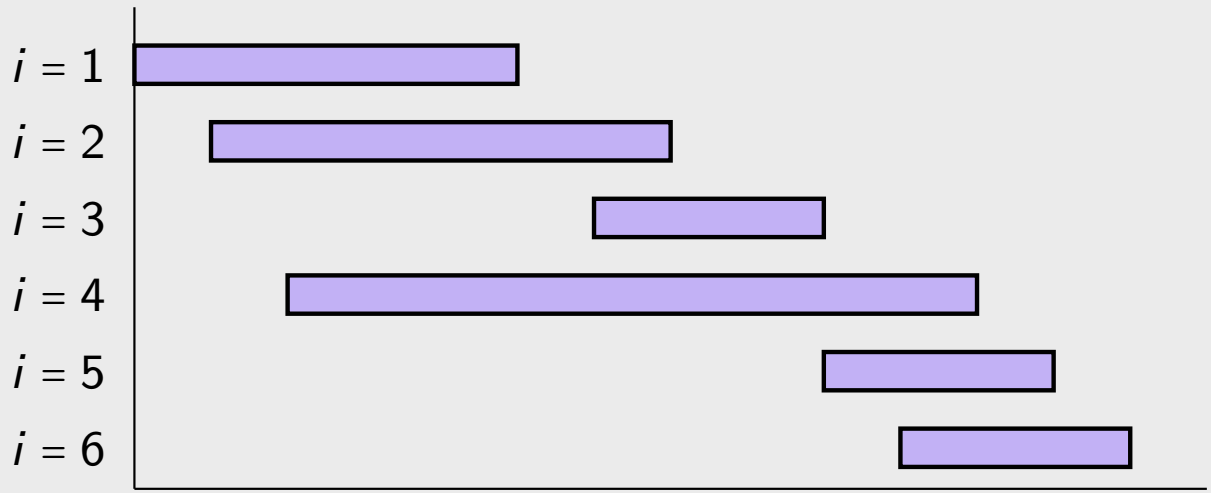
Objetivo: maximizar el número de charlas ofrecidas en la sala

Programación de charlas

Ejemplo

Sean las siguientes charlas con sus intervalos

- $i = 1$, $[0, 5)$
- $i = 2$, $[1, 7)$
- $i = 3$, $[6, 9)$
- $i = 4$, $[2, 11)$
- $i = 5$, $[9, 12)$
- $i = 6$, $[10, 13)$



Programación de charlas

Ejemplo

Posibles estrategias codiciosas: elegir primero la charla...

- que empieza más temprano
- más corta
- tiene menos incompatibilidades con otras charlas

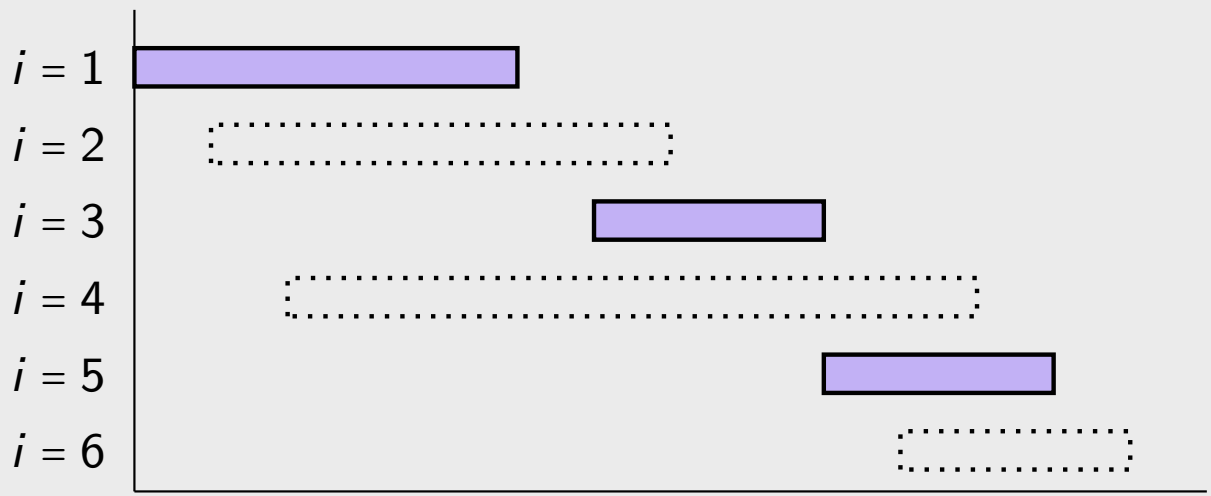
En general, ninguna de estas estrategias produce una solución óptima

Programación de charlas

Ejemplo

Escojamos según la charla **que termina más temprano**

- $i = 1$, $[0, 5)$
- $i = 2$, $[1, 7)$
- $i = 3$, $[6, 9)$
- $i = 4$, $[2, 11)$
- $i = 5$, $[9, 12)$
- $i = 6$, $[10, 13)$



Veamos otro ejemplo

Intervalos

Ejemplo I3 2022-2

Dado un intervalo cerrado $[s, f]$ considere un conjunto de intervalos cerrados no necesariamente disjuntos $S = \{[s_i, f_i] \mid i = 1, \ldots, n\}$ tales que $s \leq s_i < f_i \leq f$ para todo i.

El conjunto $\Omega \subseteq S$ es un **cubrimiento** de $[s, f]$ si la unión de sus elementos contiene a $[s, f]$ y diremos que es un cubrimiento óptimo si es el cubrimiento más pequeño (en cantidad de intervalos) para el S dado.

(a) Muestre un intervalo $[s, f]$ y un conjunto S que sirvan como contraejemplo para la siguiente estrategia codiciosa que no es óptima: *el siguiente intervalo de S que se escoge es $[s_i, f_i]$ tal que se solapa con el último intervalo escogido y s_i es el menor posible.*

Intervalos

Ejemplo

Sea el intervalo $[s, f] = [0, 3]$ y el conjunto de intervalos

$$S = \{[0, 2], [1, 2], [2, 3]\}$$

- La estrategia propuesta escoge los siguientes intervalos en orden
  - $[0, 2]$, pues tiene el menor s_i posible
  - $[1, 2]$, pues de los dos que se solapan con el escogido antes, es el que empieza primero
  - $[2, 3]$, pues es el que queda y llega al extremo del intervalo

  Es decir, $|\Omega| = 3$ con esta estrategia. Notamos que la unión de estos intervalos cubre $[0, 3]$ por completo, por lo que es un cubrimiento válido.
- Por inspección notamos que un cubrimiento más pequeño es $\Omega^* = \{[0, 2], [2, 3]\}$ que tiene $|\Omega^*| = 2$, por lo que Ω no es óptimo.

Intervalos

Ejemplo

(b) Proponga el pseudocódigo de un algoritmo codicioso que efectivamente encuentre un cubrimiento óptimo para $[s, f]$ y S cualesquiera. *Hint:* escoja el intervalo que se solapa con el último escogido y que tiene mayor f_i.

Intervalos

Ejemplo

input : límite inferior s, límite superior f, conjunto de intervalos S

`Greedy`(s, f, S)**:**

1 $S \leftarrow S$ ordenado por f_i
2 $\Omega \leftarrow \varnothing$
3 $t \leftarrow s$
4 **while** $t < f$ **:**
5 $f_i \leftarrow \max\{f_j \mid s_j \leq t \leq f_j\}$
6 $\Omega \leftarrow \Omega \cup \{i\}$
7 $t \leftarrow f_i$
8 **return** S

Sumario

Programación de charlas 2.0

Consideremos el problema de asignar charlas en una misma sala

- Tenemos n charlas por asignar
- La charla i tiene hora de inicio s_i y de término f_i
- Es decir, se define el intervalo de tiempo $[s_i, f_i)$

Solo se puede realizar **una charla a la vez**

ADEMÁS: si la charla i es realizada, produce una ganancia v_i

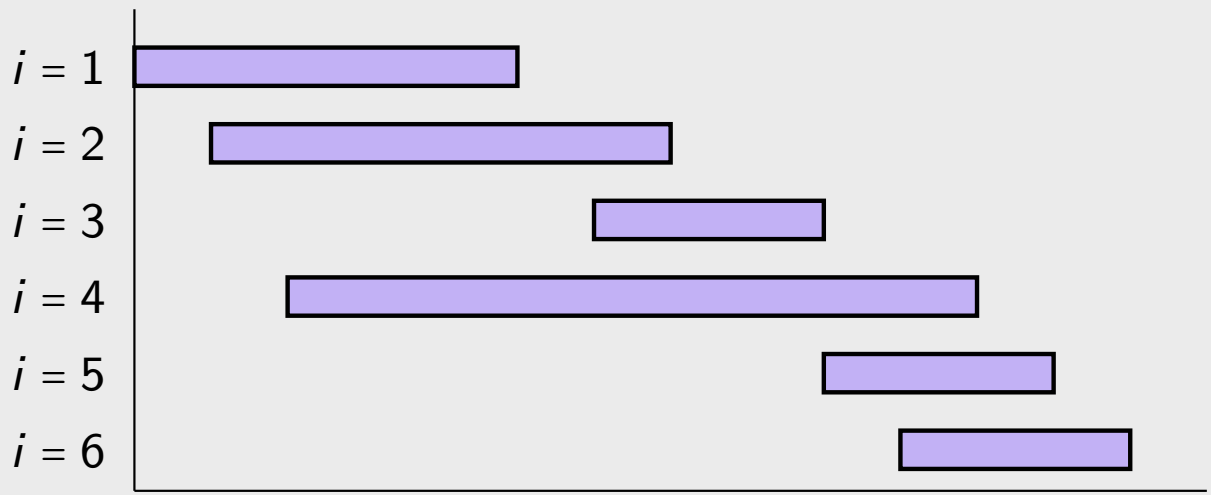
¿La estrategia codiciosa sigue siendo correcta?

Programación de charlas 2.0

Ejemplo

Sean las siguientes charlas con sus intervalos y ganancias

- $i = 1$, $[0, 5)$, $v_1 = 2$
- $i = 2$, $[1, 7)$, $v_2 = 4$
- $i = 3$, $[6, 9)$, $v_3 = 4$
- $i = 4$, $[2, 11)$, $v_4 = 7$
- $i = 5$, $[9, 12)$, $v_5 = 2$
- $i = 6$, $[10, 13)$, $v_6 = 1$

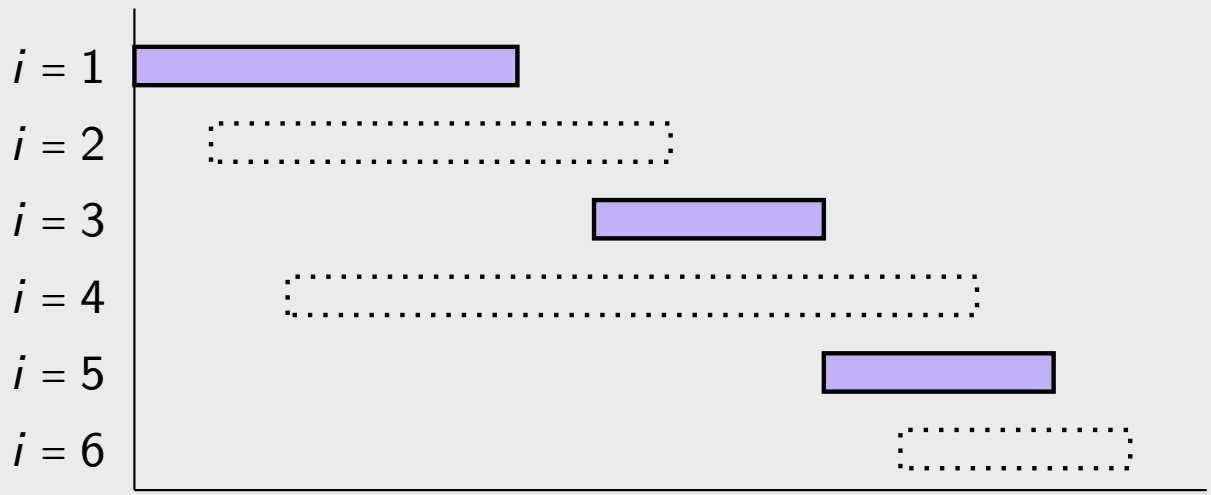


Programación de charlas 2.0

Ejemplo

El caso que sabemos resolver consideraba $v_i = c$ para cada charla

- $i = 1$, $[0, 5)$, $v_1 = c$
- $i = 2$, $[1, 7)$, $v_2 = c$
- $i = 3$, $[6, 9)$, $v_3 = c$
- $i = 4$, $[2, 11)$, $v_4 = c$
- $i = 5$, $[9, 12)$, $v_5 = c$
- $i = 6$, $[10, 13)$, $v_6 = c$



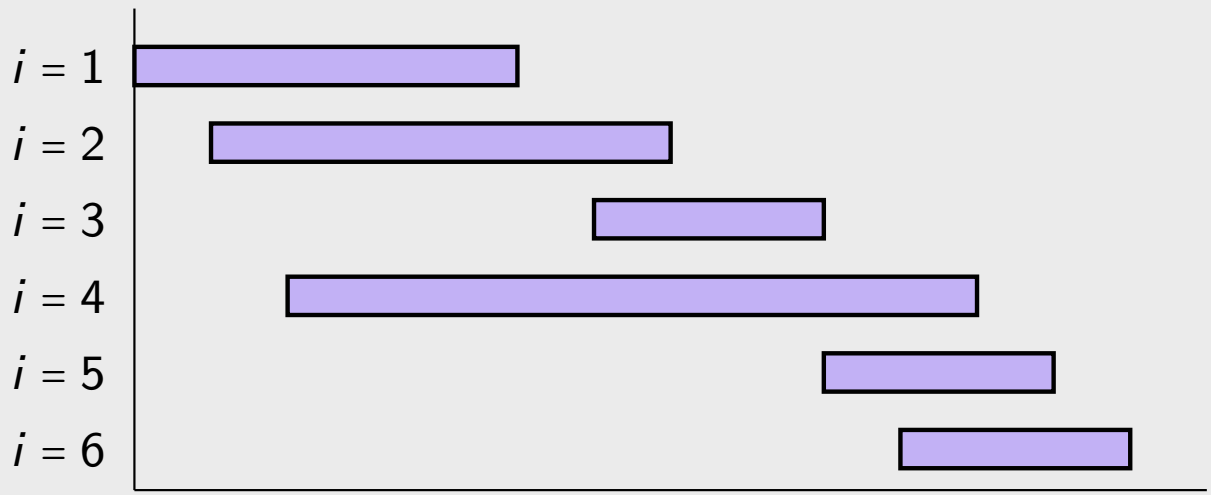
Cuando la ganancia es la misma, la estrategia codiciosa de **elegir la charla que termina antes** es óptima

Programación de charlas 2.0

Ejemplo

Sean las siguientes charlas con sus intervalos y ganancias

- $i = 1$, $[0, 5)$, $v_1 = 2$
- $i = 2$, $[1, 7)$, $v_2 = 4$
- $i = 3$, $[6, 9)$, $v_3 = 4$
- $i = 4$, $[2, 11)$, $v_4 = 7$
- $i = 5$, $[9, 12)$, $v_5 = 2$
- $i = 6$, $[10, 13)$, $v_6 = 1$

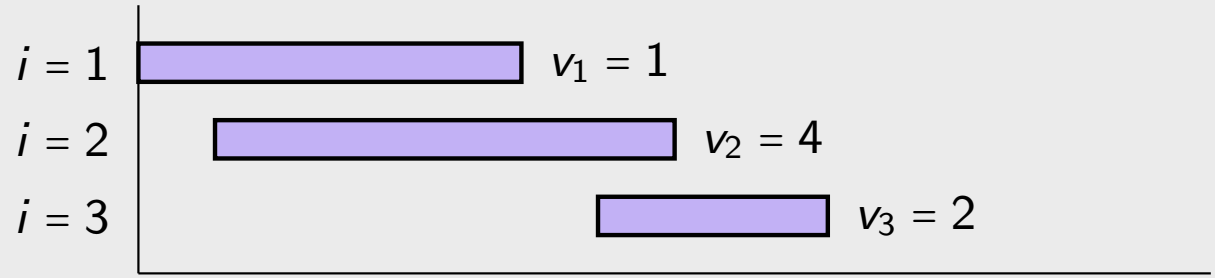


Con ganancias diferentes, el problema
no es equivalente a maximizar el número de charlas

Programación de charlas 2.0

Ejemplo

Podemos pensar en una instancia del problema de forma que la estrategia codiciosa mencionada **no funciona**



En este caso,

estrategia	charlas	ganancia
codiciosa	$\{1, 3\}$	3
óptima	$\{2\}$	4

Con ganancias diferentes, la estrategia codiciosa no sirve...
¿Qué hacemos?

Objetivos de la clase

- ☐ Comprender el paradigma de algoritmos codiciosos
- ☐ Demostrar que una estrategia no es correcta como estrategia codiciosa
- ☐ Aplicar la estrategia codiciosa para obtener óptimos en problemas particulares