

# Técnicas de Diseño de Algoritmos (Ex Algoritmos y Estructuras de Datos III)

Primer cuatrimestre 2024

14 de abril de 2024

# Heurísticas

- ▶ Una **heurística** es un procedimiento computacional que intenta obtener soluciones de buena calidad para un problema, intentando que su comportamiento sea lo más preciso posible.

# Heurísticas

- ▶ Una **heurística** es un procedimiento computacional que intenta obtener soluciones de buena calidad para un problema, intentando que su comportamiento sea lo más preciso posible.
- ▶ Por ejemplo, una heurística para un problema de optimización obtiene una solución con un valor que se espera sea cercano (idealmente igual) al valor óptimo.

# Heurísticas

- ▶ Una **heurística** es un procedimiento computacional que intenta obtener soluciones de buena calidad para un problema, intentando que su comportamiento sea lo más preciso posible.
- ▶ Por ejemplo, una heurística para un problema de optimización obtiene una solución con un valor que se espera sea cercano (idealmente igual) al valor óptimo.
- ▶ Decimos que  $A$  es un **algoritmo  $\epsilon$ -aproximado** ( $\epsilon \geq 0$ ) para un problema si

$$\left| \frac{x_A - x_{OPT}}{x_{OPT}} \right| \leq \epsilon.$$

# Heurísticas

- ▶ Una **heurística** es un procedimiento computacional que intenta obtener soluciones de buena calidad para un problema, intentando que su comportamiento sea lo más preciso posible.
- ▶ Por ejemplo, una heurística para un problema de optimización obtiene una solución con un valor que se espera sea cercano (idealmente igual) al valor óptimo.
- ▶ Decimos que  $A$  es un **algoritmo  $\epsilon$ -aproximado** ( $\epsilon \geq 0$ ) para un problema si

$$\left| \frac{x_A - x_{OPT}}{x_{OPT}} \right| \leq \epsilon.$$

- ▶ Hay otra forma de definir el **factor/ratio de aproximación** en la literatura. **APX** (approximable) y **PTAS** (polynomial-time approximation scheme) son 2 clases de Teoría de Complejidad Computacional relacionadas.  
(<https://en.wikipedia.org/wiki/APX>)

# Algoritmos golosos

**Idea:** Construir una solución seleccionando en cada paso la mejor alternativa, sin considerar (o haciéndolo débilmente) las implicancias de esta selección.

# Algoritmos golosos

**Idea:** Construir una solución seleccionando en cada paso la mejor alternativa, sin considerar (o haciéndolo débilmente) las implicancias de esta selección.

- ▶ Habitualmente, proporcionan **heurísticas** sencillas para **problemas de optimización**.
- ▶ En general permiten construir soluciones razonables (pero sub-óptimas) en tiempos eficientes.
- ▶ Sin embargo, en ocasiones nos pueden dar interesantes sorpresas!

# El problema de la mochila

## Datos de entrada:

- ▶ Capacidad  $C \in \mathbb{Z}_+$  de la mochila (peso máximo).
- ▶ Cantidad  $n \in \mathbb{N}$  de objetos.
- ▶ Peso  $p_i \in \mathbb{Z}_{>0}$  del objeto  $i$ , para  $i = 1, \dots, n$ .
- ▶ Beneficio  $b_i \in \mathbb{Z}_{>0}$  del objeto  $i$ , para  $i = 1, \dots, n$ .

**Problema:** Determinar qué objetos debemos incluir en la mochila sin excedernos del peso máximo  $C$ , de modo tal de **maximizar** el beneficio total entre los objetos seleccionados.



# El problema de la mochila

- ▶ **Algoritmo(s) goloso(s):** Mientras no se haya excedido el peso de la mochila, agregar a la mochila el objeto  $i$  que ...
  - ▶ ... tenga mayor beneficio  $b_i$ .
  - ▶ ... tenga menor peso  $p_i$ .
  - ▶ ... maximice  $b_i/p_i$ .

# El problema de la mochila

- ▶ **Algoritmo(s) goloso(s):** Mientras no se haya excedido el peso de la mochila, agregar a la mochila el objeto  $i$  que ...
  - ▶ ... tenga mayor beneficio  $b_i$ .
  - ▶ ... tenga menor peso  $p_i$ .
  - ▶ ... maximice  $b_i/p_i$ .
- ▶ ¿Qué podemos decir en cuanto a la **calidad** de las soluciones obtenidas por estos algoritmos?

# El problema de la mochila

- ▶ **Algoritmo(s) goloso(s):** Mientras no se haya excedido el peso de la mochila, agregar a la mochila el objeto  $i$  que ...
  - ▶ ... tenga mayor beneficio  $b_i$ .
  - ▶ ... tenga menor peso  $p_i$ .
  - ▶ ... maximice  $b_i/p_i$ .
- ▶ ¿Qué podemos decir en cuanto a la **calidad** de las soluciones obtenidas por estos algoritmos?
- ▶ ¿Qué podemos decir en cuanto a su **complejidad**?

# El problema de la mochila

- ▶ **Algoritmo(s) goloso(s):** Mientras no se haya excedido el peso de la mochila, agregar a la mochila el objeto  $i$  que ...
  - ▶ ... tenga mayor beneficio  $b_i$ .
  - ▶ ... tenga menor peso  $p_i$ .
  - ▶ ... maximice  $b_i/p_i$ .
- ▶ ¿Qué podemos decir en cuanto a la **calidad** de las soluciones obtenidas por estos algoritmos?
- ▶ ¿Qué podemos decir en cuanto a su **complejidad**?
- ▶ ¿Qué sucede si se puede poner una **fracción** de cada elemento en la mochila?

# El problema de la mochila

- Supongamos que los objetos están ordenados de mayor a menor cociente  $b_i/p_i$ .

# El problema de la mochila

- Supongamos que los objetos están ordenados de mayor a menor cociente  $b_i/p_i$ .

---

```
 $L \leftarrow C;$   
 $i \leftarrow 1;$   
while  $L > 0$  and  $i \leq n$  do  
     $x \leftarrow \min\{1, L/p_i\};$   
    Agregar una fracción de  $x$  del objeto  $i$  a la solución;  
     $L \leftarrow L - x p_i;$   
     $i \leftarrow i + 1;$   
end while
```

---

# El problema de la mochila

- Supongamos que los objetos están ordenados de mayor a menor cociente  $b_i/p_i$ .

---

```
 $L \leftarrow C;$   
 $i \leftarrow 1;$   
while  $L > 0$  and  $i \leq n$  do  
     $x \leftarrow \min\{1, L/p_i\};$   
    Agregar una fracción de  $x$  del objeto  $i$  a la solución;  
     $L \leftarrow L - x p_i;$   
     $i \leftarrow i + 1;$   
end while
```

---

- **Teorema.** El algoritmo goloso por cocientes encuentra una solución óptima del problema de la mochila fraccionario.

## El problema del cambio

- **Problema:** Supongamos que queremos dar el vuelto a un cliente usando el mínimo número de monedas posibles, utilizando monedas de 1, 5, 10 y 25 centavos. Por ejemplo, si el monto es \$0,69, deberemos entregar 8 monedas: 2 monedas de 25 centavos, una de 10 centavos, una de 5 centavos y cuatro de un centavo.



## El problema del cambio

- ▶ **Problema:** Supongamos que queremos dar el vuelto a un cliente usando el mínimo número de monedas posibles, utilizando monedas de 1, 5, 10 y 25 centavos. Por ejemplo, si el monto es \$0,69, deberemos entregar 8 monedas: 2 monedas de 25 centavos, una de 10 centavos, una de 5 centavos y cuatro de un centavo.
- ▶ **Algoritmo goloso:** Seleccionar la moneda de mayor valor que no exceda la cantidad restante por devolver, agregar esta moneda a la lista de la solución, y sustraer la cantidad correspondiente a la cantidad que resta por devolver (hasta que sea 0).

## El problema del cambio

- ▶ Sean  $a_1, \dots, a_k \in \mathbb{Z}_+$  las denominaciones de las monedas ( $a_i > a_{i+1}$  para  $i = 1, \dots, k - 1$ ), y sea  $t$  el valor del cambio.

## El problema del cambio

- Sean  $a_1, \dots, a_k \in \mathbb{Z}_+$  las denominaciones de las monedas ( $a_i > a_{i+1}$  para  $i = 1, \dots, k-1$ ), y sea  $t$  el valor del cambio.
- 

$s \leftarrow 0;$

$i \leftarrow 1;$

**while**  $s < t \wedge i \leq k$  **do**

$c \leftarrow \lfloor (t - s) / a_i \rfloor;$

    Agregar  $c$  monedas de tipo  $i$  a la solución;

$s \leftarrow s + c a_i;$

$i \leftarrow i + 1;$

**end while**

---

## El problema del cambio

- ▶ Este algoritmo siempre produce la mejor solución **para estos valores de monedas**, es decir, retorna la menor cantidad de monedas necesarias para obtener el valor *cambio*.

## El problema del cambio

- ▶ Este algoritmo siempre produce la mejor solución **para estos valores de monedas**, es decir, retorna la menor cantidad de monedas necesarias para obtener el valor *cambio*.
- ▶ Sin embargo, si también hay monedas de 12 centavos, puede ocurrir que el algoritmo no encuentre una solución óptima: si queremos devolver 21 centavos, el algoritmo retornará una solución con 6 monedas, una de 12 centavos, 1 de 5 centavos y cuatro de 1 centavos, mientras que la solución óptima es retornar dos monedas de 10 centavos y una de 1 centavo.

## El problema del cambio

- ▶ Este algoritmo siempre produce la mejor solución **para estos valores de monedas**, es decir, retorna la menor cantidad de monedas necesarias para obtener el valor *cambio*.
- ▶ Sin embargo, si también hay monedas de 12 centavos, puede ocurrir que el algoritmo no encuentre una solución óptima: si queremos devolver 21 centavos, el algoritmo retornará una solución con 6 monedas, una de 12 centavos, 1 de 5 centavos y cuatro de 1 centavos, mientras que la solución óptima es retornar dos monedas de 10 centavos y una de 1 centavo.
- ▶ El algoritmo es goloso porque en cada paso selecciona la moneda de mayor valor posible, sin preocuparse que esto puede llevar a una mala solución o inclusive que no conduzca a ninguna solución, y nunca modifica una decisión tomada.

## El problema del cambio

- ▶ Sean  $a_1, \dots, a_k \in \mathbb{Z}_+$  las denominaciones de las monedas ( $a_i > a_{i+1}$  para  $i = 1, \dots, k - 1$ ), y sea  $t$  el valor del cambio.

# El problema del cambio

- ▶ Sean  $a_1, \dots, a_k \in \mathbb{Z}_+$  las denominaciones de las monedas ( $a_i > a_{i+1}$  para  $i = 1, \dots, k - 1$ ), y sea  $t$  el valor del cambio.
- ▶ **Teorema.** Si existen  $m_2, \dots, m_k \in \mathbb{Z}_{\geq 2}$  tales que  $a_i = m_{i+1}a_{i+1}$  para  $i = 1, \dots, k - 1$ , entonces toda solución óptima usa  $\lfloor t/a_1 \rfloor$  monedas de tipo  $a_1$ .



## El problema del cambio

- ▶ Sean  $a_1, \dots, a_k \in \mathbb{Z}_+$  las denominaciones de las monedas ( $a_i > a_{i+1}$  para  $i = 1, \dots, k - 1$ ), y sea  $t$  el valor del cambio.
- ▶ **Teorema.** Si existen  $m_2, \dots, m_k \in \mathbb{Z}_{\geq 2}$  tales que  $a_i = m_{i+1}a_{i+1}$  para  $i = 1, \dots, k - 1$ , entonces toda solución óptima usa  $\lfloor t/a_1 \rfloor$  monedas de tipo  $a_1$ .
- ▶ **Corolario.** Si existen  $m_2, \dots, m_k \in \mathbb{Z}_{\geq 2}$  tales que  $a_i = m_{i+1}a_{i+1}$  para  $i = 1, \dots, k - 1$ , entonces el algoritmo goloso proporciona una solución óptima del problema del cambio.

## Tiempo de espera total en un sistema

- **Problema:** Un servidor tiene  $n$  clientes para atender, y los puede atender en cualquier orden. Para  $i = 1, \dots, n$ , el tiempo necesario para atender al cliente  $i$  es  $t_i \in \mathbb{R}_+$ . El objetivo es determinar en qué orden se deben atender los clientes para minimizar **la suma de los tiempos de espera** de los clientes.

# Tiempo de espera total en un sistema

- **Problema:** Un servidor tiene  $n$  clientes para atender, y los puede atender en cualquier orden. Para  $i = 1, \dots, n$ , el tiempo necesario para atender al cliente  $i$  es  $t_i \in \mathbb{R}_+$ . El objetivo es determinar en qué orden se deben atender los clientes para minimizar **la suma de los tiempos de espera** de los clientes.
- Si  $I = (i_1, i_2, \dots, i_n)$  es una permutación de los clientes que representa el orden de atención, entonces la suma de los tiempos de espera es

$$\begin{aligned} T &= t_{i_1} + (t_{i_1} + t_{i_2}) + (t_{i_1} + t_{i_2} + t_{i_3}) + \dots \\ &= \sum_{k=1}^n (n - k + 1) t_{i_k}. \end{aligned}$$

# Tiempo de espera total en un sistema

- ▶ **Algoritmo goloso:** En cada paso, atender al cliente pendiente que tenga menor tiempo de atención.

# Tiempo de espera total en un sistema

- ▶ **Algoritmo goloso:** En cada paso, atender al cliente pendiente que tenga menor tiempo de atención.
  1. Este algoritmo retorna una permutación  $I_{\text{GOL}} = (i_1, \dots, i_n)$  tal que  $t_{i_j} \leq t_{i_{j+1}}$  para  $j = 1, \dots, n - 1$ .

# Tiempo de espera total en un sistema

- ▶ **Algoritmo goloso:** En cada paso, atender al cliente pendiente que tenga menor tiempo de atención.
  1. Este algoritmo retorna una permutación  $I_{\text{GOL}} = (i_1, \dots, i_n)$  tal que  $t_{i_j} \leq t_{i_{j+1}}$  para  $j = 1, \dots, n - 1$ .
  2. ¿Cuál es la **complejidad** de este algoritmo?

# Tiempo de espera total en un sistema

- ▶ **Algoritmo goloso:** En cada paso, atender al cliente pendiente que tenga menor tiempo de atención.
  1. Este algoritmo retorna una permutación  $I_{\text{GOL}} = (i_1, \dots, i_n)$  tal que  $t_{i_j} \leq t_{i_{j+1}}$  para  $j = 1, \dots, n - 1$ .
  2. ¿Cuál es la **complejidad** de este algoritmo?
- ▶ **Teorema.** El algoritmo goloso por menor tiempo de atención proporciona una solución óptima del problema de minimizar el tiempo total de espera en un sistema.