Práctica 5 - Algoritmos *greedy*

NOTAS PRELIMINARES
Los objetivos de esta práctica son:

- Introducir la técnica de algoritmos *greedy* (golosos, avaros o codiciosos en castellano).
- Identificar los pasos requeridos para alcanzar una solución por esta técnica.
- Aprender a justificar que un algoritmo *greedy* encuentra la solución óptima.

Ejercicio 1 *

Teniendo en cuenta el problema de selección de actividades (*activity selection* problem) visto en la clase teórica, resuelva lo siguiente.

a. Suponga que en cada paso del algoritmo, en lugar de seleccionar la actividad que primero finalizará, tomamos siempre la última en comenzar que sea compatible con las ya elegidas. Indicar porque este algoritmo es *greedy* y demostrar que encuentra un conjunto actividades de tamaño máximo.
b. Qué ocurre si seleccionamos en cada paso la de menor duración que sea compatible con las ya elegidas? Y si elegimos siempre a la que incia más tempranamente y sea compatible con las seleccionadas?

Ejercicio 2 *

Se desea asignar cursos, que tienen una hora de comienzo y fin, a cierta cantidad de aulas. Obviamente no se pueden dictar dos cursos simultáneamente en una misma aula y se desea minimizar la cantidad de aulas utilizadas. Dar un algoritmo $O(n \cdot \log n)$ para resolver el problema, donde n es la cantidad de cursos.

Ejercicio 3 *

Se quiere dar el vuelto a un cliente usando el mínimo número de monedas posibles. Hay monedas de 1, 5, 10 y 25 centavos (hay al menos una de cada tipo). Analizar el siguiente algoritmo goloso para resolver el problema:

```
Elegir en cada paso la moneda de mayor valor entre las disponibles.
```

a. Probar que si hay una solución, el algoritmo goloso siempre encuentra una solución para estos valores de las monedas.
b. Comparar la complejidad de este algoritmo con aquel propuesto en la práctica de programación dinámica.
c. Mostrar ejemplos que muestren que si también hay monedas de 12 centavos, o si no hay al menos una moneda de cada tipo, puede ocurrir que el algoritmo no encuentre una solución aunque la haya.

Ejercicio 4 *

Sean $D_1, D_2, \ldots, D_n$ conjuntos de datos que se quieren almacenar en una cinta. El conjunto D_i requiere m_i Gb de memoria. La cinta tiene capacidad para almacenar todos los datos. Se conoce la frecuencia π_i con que se usa el conjunto D_i. La densidad de la cinta y la velocidad del lector son constantes. Después de que un conjunto se carga en memoria desde la cinta, la misma se rebobina hasta el principio. Si los conjuntos se almacenan en orden $i_1, i_2, \ldots, i_n$ el tiempo promedio de carga de un programa es:

$$T = c \sum_j \left(\pi_{i_j} \sum_{k \leq j} m_{i_k} \right)$$

donde la constante c depende de la densidad de grabado y la velocidad del dispositivo. Queremos construir un algoritmo goloso para determinar el orden en que se almacenan los datos de forma de minimizar T.

Decidir si cada una de las siguientes ideas resolverían el problema (según corresponda demostrar o dar un contraejemplo).

Almacenar los conjuntos:

a. en orden no decreciente de los m_i
b. en orden no creciente de los π_i
c. en orden no creciente de π_i / m_i