Técnicas de diseño de algoritmos
Algoritmos voraces
Ejercicios (Bloque 1): con soluciones

Luis Javier Rodríguez Fuentes
Amparo Varona Fernández

Departamento de Electricidad y Electrónica
Facultad de Ciencia y Tecnología, UPV/EHU
luisjavier.rodriguez@ehu.es
amparo.varona@ehu.es

OpenCourseWare 2015
Campus Virtual UPV/EHU

Algoritmos voraces – Ejercicios (Bloque 1)

(B1.1) Escribir en lenguaje Python una función que devuelva el árbol de recubrimiento de coste mínimo según el algoritmo de Kruskal (explicado en los apuntes), utilizando una implementación en la que los costes de los arcos se almacenen en un diccionario indexado por tuplas (que representan los arcos del grafo).

(B1.2) Un cierto servidor (un procesador, un cajero automático, una impresora, etc.) ha de atender a n clientes/peticiones, y el tiempo estimado de servicio para cada uno de ellos es conocido: t_i, $i = 1, 2, \ldots, n$. Escribir en lenguaje Python una función que, utilizando un esquema voraz, devuelva el orden en que deben ser atendidos los n clientes para minimizar el tiempo estimado de espera. Demostrar que el criterio de selección voraz conduce a la solución óptima del problema.
Importante: En el **Apéndice A** de esta sección se da una explicación más detallada del problema, así como un ejemplo de resolución.

Algoritmos voraces – Ejercicios (Bloque 1)

(B1.3) Se han planificado n actividades, de las cuales se conoce la hora de inicio (intervalo cerrado) y la hora de terminación (intervalo abierto): actividad $i \rightarrow [t_i^{\text{ini}}, t_i^{\text{fin}})$, pero tan sólo se dispone de una sala. Se desea diseñar un calendario tal que se realice el máximo número de actividades mutuamente compatibles, es decir, sin que se solapen entre sí. Escribir en lenguaje Python una función que, aplicando un esquema voraz, obtenga la solución óptima al problema planteado. **Importante:** se deberá escoger como siguiente actividad aquella que, siendo compatible con las escogidas anteriormente, tenga una hora de terminación más temprana (es decir, que deje el máximo de tiempo disponible tras su terminación). Para ello, lo más conveniente será ordenar las tareas según su hora de terminación. Demostrar que este criterio conduce a la solución óptima.

(B1.4) Considerese el mismo problema del ejercicio anterior. Escribir en lenguaje Python una función que, aplicando un esquema voraz, obtenga una solución basada en un criterio distinto: escoger aquella actividad que, siendo compatible con las anteriormente escogidas, tenga una duración más corta. Demostrar mediante un contraejemplo que este algoritmo no conduce a una solución óptima del problema planteado.

Algoritmos voraces – Ejercicios (Bloque 1)

(B1.5) Se tienen n esquiadoras de alturas $H = [h_1, h_2, \ldots, h_n]$ y n pares de esquíes de longitudes $S = [s_1, s_2, \ldots, s_n]$. Escribir en lenguaje Python una función que, aplicando un esquema voraz, asigne los esquíes a las esquiadoras de modo que el promedio de la diferencia (en valor absoluto) entre la altura de la esquiadora y la longitud de los esquíes asignados sea mínima. Es decir, se trata de minimizar la siguiente función:

$$f(H, S, \Pi) = \frac{1}{n} \sum_{i=1}^{n} |h_i - s_{\pi_i}|$$

donde $\Pi = [\pi_1, \pi_2, \ldots, \pi_n]$ representa la asignación de esquíes a las esquiadoras $1, 2, \ldots, n$. **Importante:** se deberán asignar los esquíes más cortos a la esquiadora más baja, los siguientes esquíes más cortos a la segunda esquiadora más baja, etc. Para ello, deberán ordenarse las listas H y S. Demostrar que este criterio de asignación conduce a una solución óptima.

Apéndice A: Minimización del tiempo de espera

- Se trata de un problema de planificación de tareas.
- Un servidor (un procesador, un cajero automático, etc.) tiene que atender a n clientes que llegan todos a la vez al sistema.
- El tiempo de servicio para cada cliente es t_i, $i = 1, 2, \ldots, n$.
- Se desea minimizar el tiempo de espera global.
- Formulación:
  - Variables:

    n: número de clientes
    $t = (t_i)$, con $i \in [1, n]$: tiempo de servicio para el cliente i
    $X = (x_i)$ con $x_i \in [1, n]$: cliente atendido en la posición i
  - Restricciones: $X \in \text{Permutación}\{1, 2, \ldots, n\}$
  - Función objetivo (a minimizar): $T(X) = \sum_{i=1}^{n} \text{tiempo\_espera}(x_i)$
    donde $\text{tiempo\_espera}(x_i) = \sum_{j=1}^{i-1} t_{x_j}$

Ejemplo de minimización del tiempo de espera

$n = 3$
$t = (5, 10, 3)$

```
  X                      T(X)
-----------------------------------------
1 2 3:    5 + (5 + 10) + (5 + 10 + 3) = 38
1 3 2:    5 +  (5 + 3) + (5 + 3 + 10) = 31
2 1 3:   10 + (10 + 5) + (10 + 5 + 3) = 43
2 3 1:   10 + (10 + 3) + (10 + 3 + 5) = 41
3 1 2:    3 +  (3 + 5) + (3 + 5 + 10) = 29  <-- óptimo !!!
3 2 1:    3 + (3 + 10) + (3 + 10 + 5) = 34
```