Selección de procesos

Selección de procesos

Entrada: Un conjunto $P = \{p_1, p_2, \ldots, p_N\}$ de procesos que necesitan de un recurso para ser llevadas a cabo, el cual solo puede atender un proceso al tiempo. Cada proceso p_i tiene un identificador, un tiempo de inicio s_i y uno de finalización e_i donde $0 \leq s_i < e_i < \infty$

Salida: Un subconjunto procesos mutuamente compatibles que maximiza la cantidad de procesos llevadas a cabo. Los procesos p_i y p_j son compatibles si $s_i \geq e_j$ ó $s_j \geq e_i$

id	a	b	c	d	e	f	g	h
s	1	0	5	3	6	8	8	12
e	4	6	7	9	10	11	12	16

{a, c, f, h}

{a, c, g, h}

Dadas *n* actividades, ¿cuántos subconjuntos diferentes pueden haber?

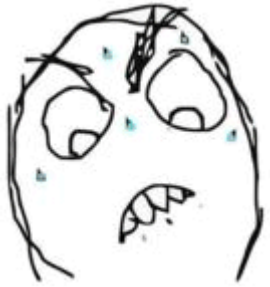
$$2^n - 1$$

Es decir, una solución por fuerza bruta sería generar un arreglo de binario de *n* elementos y evaluar las $2^n - 1$ posibilidades

¿Se puede hacer mejor?

Para este problema ¿cuál sería la decisión greedy?

¿Ideas?



¿Qué tal escoger la actividad que deje la mayor cantidad de recursos disponibles para las actividades siguientes, es decir, aquella con un menor tiempo de finalización?

Selección de procesos

Para utilizar esta estrategia debemos:

- En caso que ya no lo estén, ordenar las actividades de forma ascendente por tiempo de finalización.
- Escoger la primera actividad
- Escoger sistemáticamente la siguiente actividad compatible según la decisión greedy hasta llegar a la *n*-ésima.

Pero para este problema ¿la intuición empleada es correcta?, es decir, ¿la decisión greedy utilizada guía la estrategia definida hacia una solución óptima?

Resulta que si (ver teorema 16.1 y la prueba correspondiente en *Introduction to algoritmhs,* p. 418)

Selección de procesos

```
read P //N valores con atributos id, start, end
order(P) //Ascendentemente por tiempos de finalizacion
R.add(P_0.id)
k = 1
for i = 1 to N-1:
      if P_i.start ≥ P_k.end :
            R.add(P_i.id)
            k = i
print R
```

¿Cuál es la eficiencia de este algoritmo?

O(n*log(n)) debido al ordenamiento, sin duda mucho mejor que O($n2^n$) del algoritmo por fuerza bruta

Volviendo al ejemplo:

id	a	b	c	d	e	f	g	h
s	1	0	5	3	6	8	8	12
e	4	6	7	9	10	11	12	16