Programación de actividades unitarias

Programación de actividades unitarias

Entrada: Un conjunto A = {$a_1, a_2, \ldots, a_N$} de actividades unitarias , es decir, que demoran una (1) unidad de tiempo. Cada actividad a_i tiene un identificador, un tiempo límite de finalización e_i ($1 \leq e_i \leq N$) y una penalización no negativa w_i en que se incurre al no terminar la tarea a_i en el tiempo e_i

Salida: Secuencia de tareas que minimiza la penalización total considerando que solo se puede hacer una tarea al tiempo

Programación de actividades unitarias

Ejemplo:

a_i	A	B	C	D
e_i	3	2	3	1
w_i	25	10	20	15

Soluciones: {D, A, C, B} ó {D, C, A, B}, ambas con una penalización total de 10

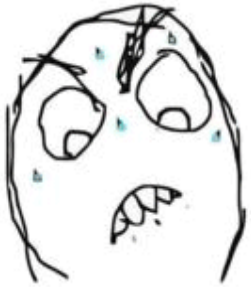
En general para *n* tareas, ¿Cuál es la cantidad de posibles soluciones para este problema?

$N!$

Programación de actividades unitarias

¿Se podrá resolver mediante una aproximación greedy? ¿Cuál sería el criterio a utilizar?

¿Ideas?



Idea 1: Ordenar las tareas descendentemente por la penalización y ejecutarlas en ese orden

En el ejemplo anterior daría: {A, C, D, B} con una penalización total de 25

Idea 2: Ordenar las tareas descendentemente por la penalización, luego ubicarlas una por una según ese orden pero en la posición donde dejen “el mayor margen posible” para las restantes. En caso de empate elegir primero la de menor tiempo límite de finalización

```
read A //N valores con atributos id, end, w
order(A) //Descendentemente por w, ascendentemente por end
for i = 0 to N-1:
     S_i = NULL
q = N-1
for i = 0 to N-1:
     p = A_i.end
     while S_p ≠ NULL:
          p = p-1
     if p ≥ 0:
          S_p = A_i.id
     else:
          while S_q ≠ NULL:
               q = q-1
          S_q = A_i.id
          q = q-1
print S
```

Ejemplo:

id	A	B	C	D	E	F	G
e	4	2	4	3	1	4	6
w	70	60	50	40	30	20	10

S	D	B	C	A	G	F	E

Iteración 1: Tomamos A, con tiempo límite 4 y lo ubicamos en la posición 4

Iteración 2: Tomamos B, con tiempo límite 2 y lo ubicamos en la posición 2

Iteración 3: Tomamos C, con tiempo límite 4 y lo ubicamos en la posición 3

Iteración 4: Tomamos D, con tiempo límite 3 y lo ubicamos en la posición 1

Iteración 5: Tomamos E, con tiempo límite 1 y lo ubicamos en la posición 7

Iteración 6: Tomamos F, con tiempo límite 4 y lo ubicamos en la posición 6

Iteración 7: Tomamos G, con tiempo límite 6 y lo ubicamos en la posición 5

¿Cuál es la eficiencia de este algoritmo? $O(N^2)$