Algoritmos voraces

Algoritmos voráces

Aunque no es una definición formal, podríamos decir que un algoritmo puede ser considerado como ‘voraz’ si involucra un proceso iterativo que va tomando decisiones ‘miopes’ previendo que de esa manera se llega al final a la decisión deseada.

En este contexto miope significa que una decisión se toma con la información que se tiene en ese momento y que, una vez tomada, es irrevocable.

Generalmente, no es complejo definir un algoritmo ‘greedy’ no siempre conlleva a soluciones óptimas, sin embargo para muchos problemas si lo hacen y con una eficiencia mejor que otras aproximaciones.

Problema de planificación de tareas

Dado un recurso compartido (maquina, procesador, trabajador) y un conjunto de tareas por realizar, ¿en qué orden se deben llevar a cabo considerando que el recurso solo puede ejecutar una al tiempo y no se pueden realizar parcialmente?, más formalmente:

Entrada: Un conjunto $T = \{T_1, T_2, \ldots, T_N\}$ de tareas donde cada tarea *i* una tiene un identificador, un peso o prioridad w_i y un tiempo de finalización e_i

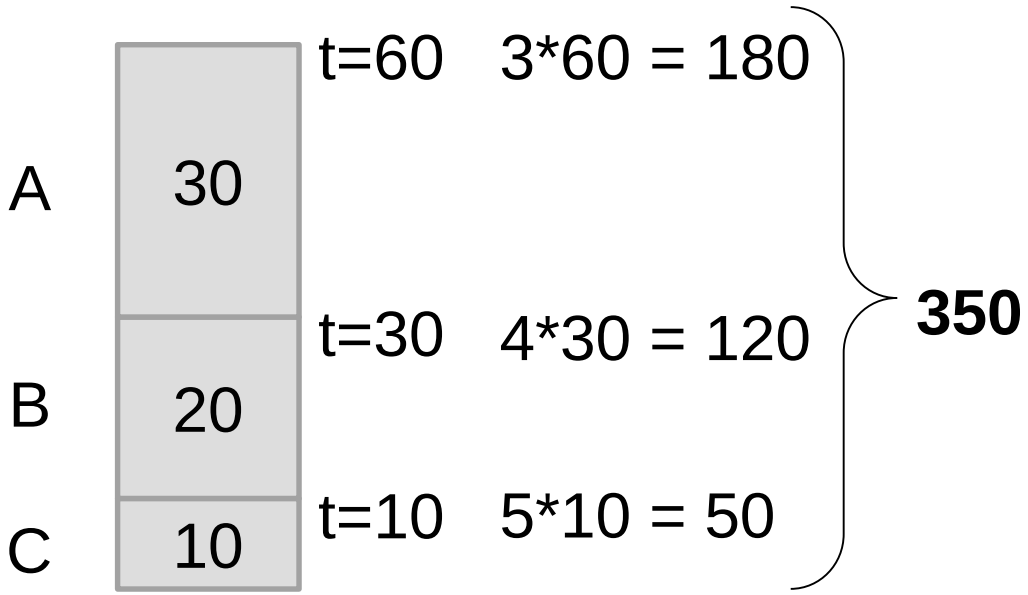
Salida: Secuencia de tareas que minimiza el producto de las prioridades por los tiempos de finalización, es decir:

$min \sum_{i=1}^{N} w_i * f_i$ donde $f_i = \sum_{j=1}^{i} e_j$

Problema de planificación de tareas

Ejemplo:

T_i	A	B	C
w_i	3	4	5
e_i	30	20	10



Para *N* tareas, ¿Cuál es la cantidad de posibles soluciones para este problema? *N*!

Es decir, una solución por fuerza bruta sería generar las n! permutaciones y evaluarlas todas

Problema de planificación de tareas

Preguntémonos primero: Si todas las tareas tuvieran la misma longitud, ¿Cuáles deberían programarse primero, las de mayor peso o las de menor?

Las de mayor peso

Y si todas las tareas tuvieran el mismo peso o prioridad, ¿Cuáles deberían programarse primero, las de mayor tiempo de finalización o las de menor?

Las de menor tiempo

A partir de la generalización de estos casos especiales, un posible criterio greedy es escoger las tareas con mayor peso y menor tiempo o, lo que es lo mismo, aquellas con mayor relación peso/tiempo

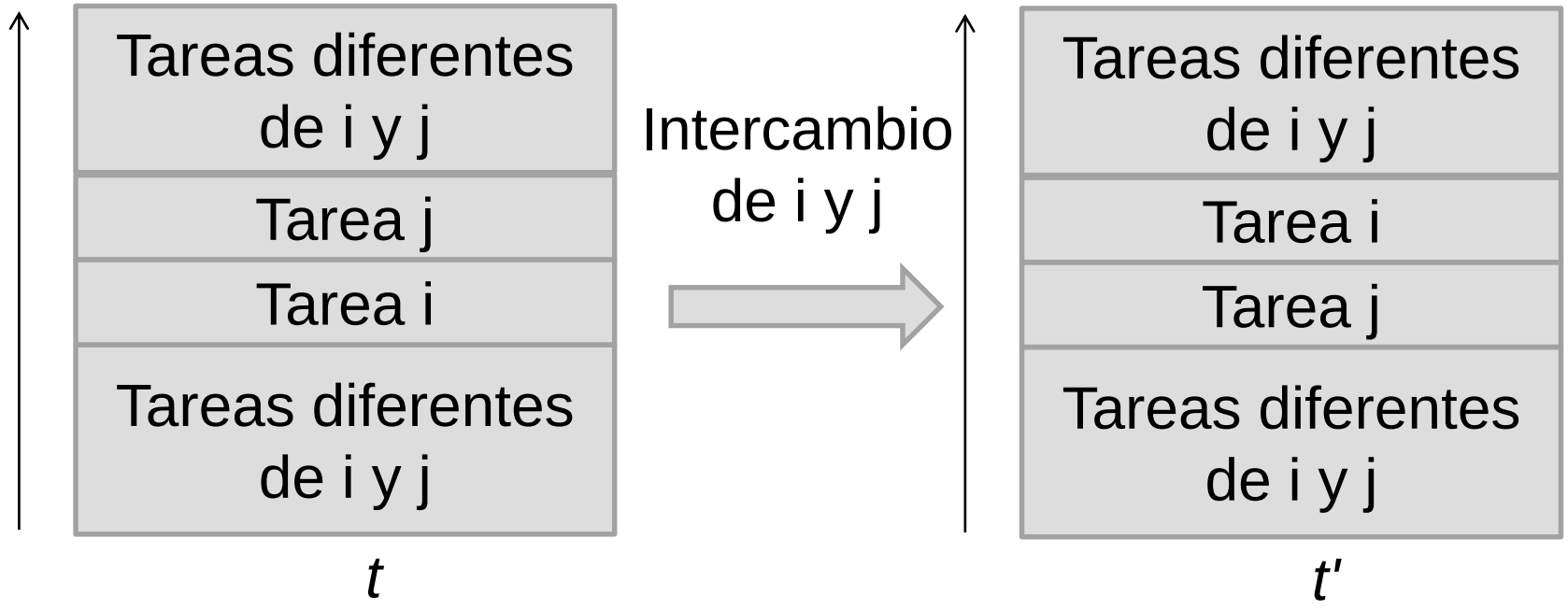
Problema de planificación de tareas

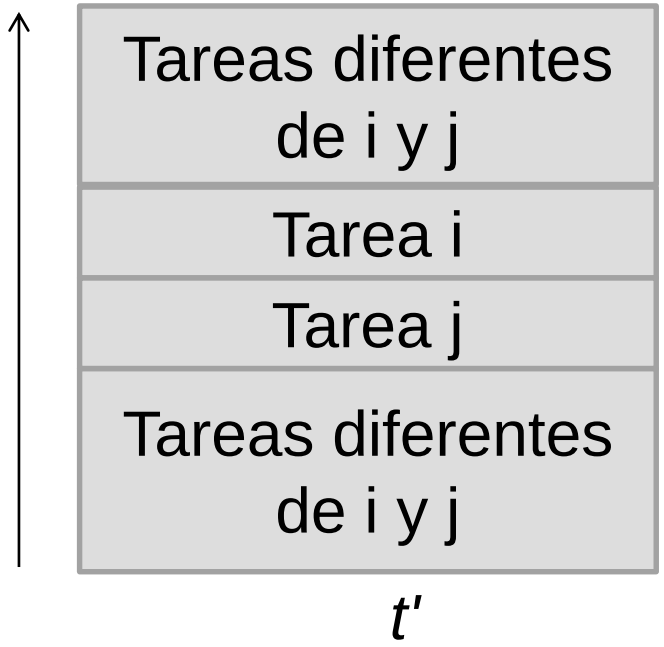
```
read T //N valores con atributos id, w, end
for i = 0 to N-1:
     S_i.id = T_i.id
     S_i.r = T_i.w/T_i.end
order(S) //Descendentemente por r
print S
```

¿Cuál es la eficiencia de este algoritmo? $O(N*log(N))$

Para demostrar que el algoritmo es correcto (que encuentra la solución óptima) vamos a utilizar un método conocido como “intercambio de argumentos”.

Digamos que *t’* es una solución diferente a *t* donde por simplicidad solo se intercambian dos tareas consecutivas *i*, *j*





¿Cuál es el efecto de este intercambio en los tiempos de finalización de las tareas diferentes de *i* y *j*? Ninguno

¿Cuál es el efecto en el tiempo de finalización de *i*?

Aumenta t_j, es decir, el <u>costo</u> del intercambio es $w_i * e_j$

¿Cuál es el efecto en el tiempo de finalización de *j*?

Disminuye t_i, es decir, el <u>beneficio</u> del intercambio es $w_j * e_i$

Sin embargo nosotros sabemos de *t* que:

$$\frac{w_i}{e_i} > \frac{w_j}{e_j} \rightarrow w_i * e_j > w_j * e_i$$

Es decir, <u>costo</u> > <u>beneficio</u>, por tanto t' no puede ser óptimo