Problema 2.5: Interval Partitioning Problem

A.22 Grup 2

Enunciat

El problema de la partició interval (Interval Partitioning Problem) ós similar al problema de la selecció d'activitats vist a classe però, en lloc de tenir un únic recurs, tenim molts recursos (és a dir, diverses còpies del mateix recurs). Doneu un algorisme que permeti programar totes les activitats fent servir el menor número possible de recursos.

Solució

Pseudocodi

Siguin les activitats conegudes per temps inicial i temps final.
Sigui t = [] el vector amb les activitats
Aplicar algorisme d'ordenació ascendent en t per temps inicial
Sigui p una priority queue d'activitats ordenades per temps final
Afegir t[0] a p
n = t.size()
for (i = 1; i < n; ++i) {
    if (t[i].temps inici > p.front().temps final) invalidar el front actual
    afegir t[i] a la cua
}

Idea general

La idea d'aquest algorisme seria fer ús d'una priority queue per comparar el temps d'inici de la nova activitat a fer amb el temps final de l'activitat registrada al front de la priority queue ja que serà el recurs que tindrà l'activitat que finalitzarà abans.
Si el temps d'inici de la nova activitat és abans que el primer recurs que es queda lliure s'haurà de crear un nou recurs.

Correcció

Sigui n el nombre de recursos que hem obtingut a l'aplicar l'algoritme.
Aleshores sabem que hi ha una activitat al recurs n que no podia anar al recurs n-1 ja que l'activitat començava abans de finalitzar l'última activitat de n-1.
Gràcies a que les activitats estan ordenades per temps inicial sabem que, pel que hem comentat abans, l'última activitat de cada recurs *i* finalitza després que l'inici de les activitats de tots els recursos *j* on j > i.

Cost de l'algorisme

- Ordenar t -> O(nlogn)
- for -> O(n)
- cos del for -> O(logn)

Si sumem això:

- O(nlogn) + O(n)*O(logn) = O(nlogn)