**Sistemes Operatius II - Pràctica 2**

Joaquim Comes Ubach

Samuel Roldán Robles

17 - 10 – 2019

# **Funcionament del programa**

Aquesta practica se centra en desenvolupar una aplicació C que permeti extreure i indexar informació dels vols dels Estats Units, per tal d’aconseguir aquest objectiu és necessari seguir els següents passos:

* **Lectura de la informació dels aeroports d’origen →** Per tal de poder llegir els codis IATA (formats per tres caràcters) s’utilitzarà la funció *abre\_aeroport* la qual ens permet emplenar el arbre balancejat amb tots els aeroports emmagatzemats en el fitxer *aeroports.csv*, les funcions que s'utilitzen pretereixen al fitxers *red-black-tree.h* i *.c* que han estat editats per tal de poder guardar cadenes de caràcters en comptes d'enters.
* **Lectura del fitxer de vols →** Per ta de poder llegir les columnes pertinents s'utilitza la funció *lista\_vuelos* la qual ens permet llegir les columnes 15, 17 i 18 les quals contenen les dades relacionades amb el retard del vol, l'aeroport origen i destí respectivament, els vols es guardaran en la llista enllaçada del node pertinent juntament amb el retard, el destí y el nombre de vols computats *(les funcions relacionades amb l'arbre son extretes de red-black.tree)*.
* **Computació d’estadístiques →** Un cop emplenat l'arbre s'han de computar les següents dades:
  + Es demanar computar, per a l’aeroport d’origen especificat com a tercer argument a l’hora d’executar l’aplicació, el retard mig per cadascun dels seus destins, per tal de fer això utilitzem les funcions *find\_node* i *find\_list*, del codi *red-black-tree* i *linked-list* respectivament. A continuació es llista la lògica seguida per computar la petició:
    - Agafem el node passat per paràmetre *origen*.
    - Agafem el primer vol associat al node.
    - Agafem la informació relacionada al vol i calculem el promig del retard per a cada vol.
    - Seleccionem el següent vol.
    - Es repeteix els passos 3 i 4 fins que no quedin vols pendents per mirar.
  + Es demanar computar quin és l’aeroport que té més destinacions. Per tal de poder això recorrem el arbre amb la funció *vuelos\_maximos* (*funció recursiva que ens permet explora tots els nodes*). A continuació es llista la lògica seguida per computar la petició:
    - Cridem la funció *max\_flights* amb el primer node del arbre com ha paràmetre.
    - Agafem el numero de vols que te el node.
    - Explorem els fills del node (c*ridem la funció vuelos\_maximos amb els fills com a paràmetre*).
    - Comprovem si el numero de vols del node actual, en cas de major al trobat fins ara actualitzem la variable global *contador\_destinos* i el nombre del aeroport en la variable global *aeroport.*
    - Imprimim la variables global *contador\_destinos* i *aeroport.*
* ***Alliberament de la memòria* →** Al finalitzar la execució o un mètode s’alliberarà tot el espai possible.

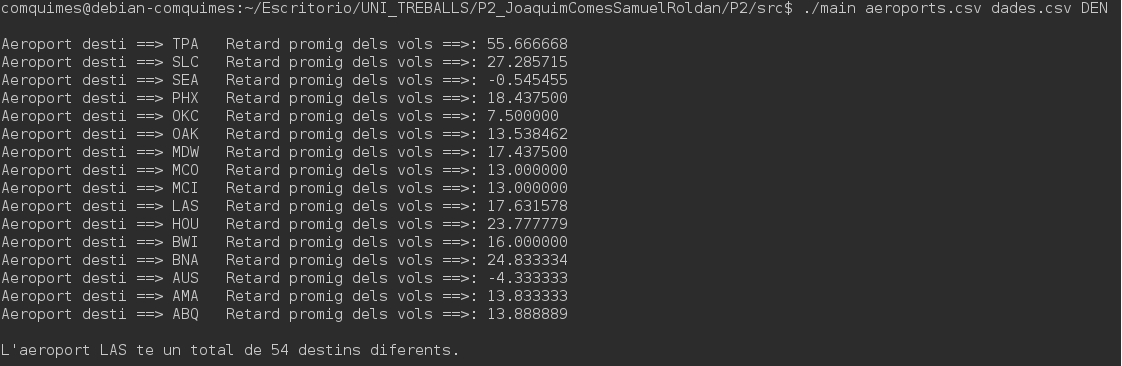
# **Qüestions**

Per tal de realitzar aquesta practica fem us del codi *red-black-tree*, aquest codi ens permet ordenar els aeroports mitjançant un arbre balancejat, ha estat necessari modificar el codi per tal de que pugui treballar amb cadenes de caràcters *(s'ha modificat el condicional de la funció que compare\_key1\_less\_than\_key2 [key1 < key2 => strcmp(key1, key2) < 0])*, també s'ha modificat la informació que conte el node, un struct data amb una key, que és el nom de l’aeroport, una linked-list per guardar els vols que surten de cada aeroport.

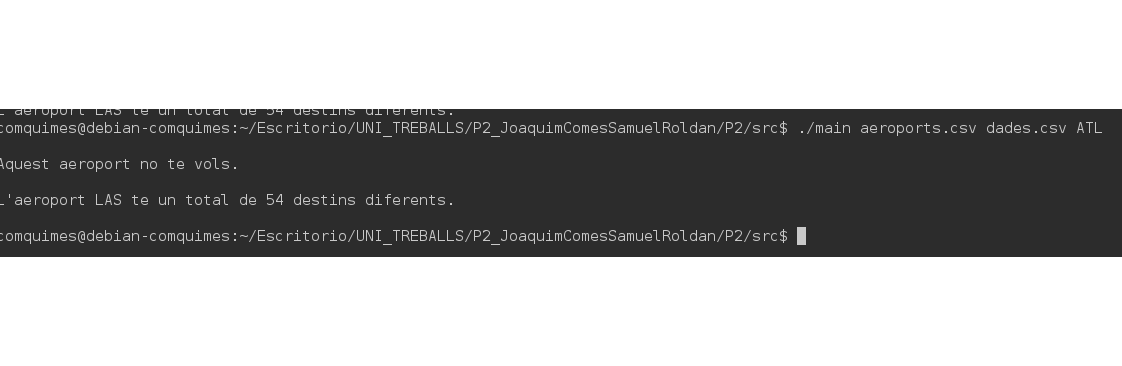
Per poder crear la llista enllaçada s'ha usat el codi *linked-list* on també s'ha modificat el mètode *compare\_key1\_equal\_key2* de la mateixa forma que en el codi *red-black-tree.*

Veiem ara un parell d’experiments amb diferents aeroports:

* Aeroport DEN:



* Aeroport ATL:



**Valgrind**

Utilitzem Valgrind per comprovar que la memòria s’allibera correctament. Veiem que no dona errors:

