Sistemes Operatius 2

Memòria pràctica 3

1. *Funcionament de l'aplicació*

   En aquesta pràctica se'ns demana una petita ampliació de l'anterior. Concretament, definim una mica més d'interfície amb un menú per consola, guardem i carreguem l'arbre que guarda paraules i la seva informació i per últim definim un mètode per consultar aquesta informació. Així doncs, el menú està format per 5 opcions:

   1.1. **Carregar l'arbre:** en comptes de carregar un sol fitxer *pdf* que passem per paràmetre, ara ho fem amb varis. Aquests, estan definits en un fitxer *.txt*, el qual llegim per anar analitzant un per un cada fitxer *pdf*. D'aquesta manera, ara guardem en l'arbre varies paraules de varis fitxers *pdf.*

   1.2. **Guardar l'arbre:** en aquesta pràctica se'ns demana poder guardar l'arbre a memòria, és a dir, guardar la informació de les paraules de l'arbre (no la informació pare-fill, etc). També guardem el número de nodes d'aquest i el nombre màgic del fitxer. Per poder-ho fer, hem implementat dos funcions. La primera, guarda la informació que ha de ser guardada un sol cop, és a dir, el nombre màgic i el número de nodes. La segona és recursiva, i va guardant al fitxer la longitud, la paraula i el número de vegades que es repeteix una paraula per cada paraula de l'arbre. Per escriure a un fitxer fem servir *fwrite*.

   1.3. **Carregar l'arbre:** al igual que guardem l'arbre a un fitxer, hem de poder carregar-lo des d'aquest. Per fer-ho, els nombres màgics han de coincidir. Per llegir d'un fitxer fem servir *fread*. Per tant, la primera cosa que llegim es aquest nombre màgic. En segon lloc, i si es compleix la condició, llegim el número de nodes. D'aquesta manera, amb un for de 0 al número de nodes, anem llegint la longitud d'una paraula, la paraula en sí i les vegades que aquesta es repeteix. S'ha de tenir en compte el *n* de cada paraula. També s'ha de controlar que no es llegeixin nombres negatius o zero. Finalment, inserim el node a l'arbre.

   1.4. **Consulta d'informació:** aquesta funció permet buscar quantes vegades es repeteix una paraula qualsevol en l'arbre. Així mateix, permet comprovar el correcte funcionament de l'aplicació. Per fer-ho, fem ús de la donada funció *findNode* per trobar el node de la paraula donada, de manera que podem accedir així al seu número de vegades que es repeteix.

   1.5. **Sortir:** ordre que surt de l'aplicació. Abans, però, s'ha d'alliberar la memòria de l'arbre amb *deleteTree* i alliberar també cada espai de memòria per cada paraula, amb *free(tree)*.

2. *Proves realitzades:*

Per comprovar el correcte funcionament de l'aplicació només hem fet servir l'opció 4 del menú, juntament amb la informació donada al informe de quantes vegades es repeteix certa paraula i el fitxer on es troba un arbre guardat concret.

De totes maneres, al anar implementar les funcions hem fet ús de:

2.1. ***Printf*:** sovint per comprovar que llegíem una dada de manera correcte hem fet servir *printf* per imprimir-la. Això requereix abans declarar correctament aquesta paraula, és a dir, reservar espai per ella. Un exemple seria imprimir el nombre màgic llegit a la funció de carregar arbre.

2.2. **El fitxer *fitxer_pdf***: per comprovar que llegim correctament varis fitxers *pdf's*, hem fet ús del fitxer i *pdf's* donats.

3. *Problemes trobats:*

3.1. **Guardar Arbre:** al guardar l'arbre no ens funcionava la funció recursiva. No podia guardar la longitud d'una paraula, però si podia llegir i imprimir aquesta paraula i aquesta longitud. Finalment vam comprovar que el problema estava en com accedíem a la informació, és a dir, el que fèiem era obtenir l'*RBData* mitjançant el node, i d'allà la longitud, la paraula, etc. Aquesta implementació (no sabem per què) no funciona, així que vam accedir a la informació sense declarar un *RBData:*

```
void guardarRecursiu(Node *a,FILE *fp) {

    int len;
    len = strlen(a->data->key);

    //Guardant longitud paraula
    fwrite(&len, sizeof(int), 1, fp);
    //Guardant paraula
    fwrite(a->data->key, sizeof(char),len,fp);
    //Guardant cops que surt
    int numcops = a->data->num_vegades;
    fwrite(&numcops, sizeof(int), 1,fp);

    if(a->right != NIL) {
        guardarRecursiu(a->right,fp);
    }
    if (a->left != NIL) {
        guardarRecursiu(a->left,fp);
    }

}
```

3.2. **Carregar arbre:** al carregar l'arbre vam estar una bona estona per entendre que s'havia de reservar espai per cada paraula segon la seva longitud **més 1**, ja que hi ha el \n. Després, és clar, se l'hi ha de treure.

3.3. **Inicialització i destrucció de l'arbre:** l'arbre es modifica, destrueix i es crea nombroses vegades en l'aplicació. En l'opció de carregar arbre, per exemple, es destrueix si ja n'hi ha un i es crea de nou amb la informació del fitxer. La nostra primera idea, per exemple, era que la funció *carregaArbre* retornes un arbre, per després assignar-lo al menú. Finalment, vam optar per fer una variable global.
De totes maneres, el codi quedava més net amb una funció dedicada a crear i destruir l'arbre.

3.4. **Valgrind**: ens ha ajudat per definir o retocar algun *free* que no existia i havia d'estar present. De totes maneres, hi ha un *free* que no aconseguim fer, el de la línia 145, ja que si és "descomentat" i es selecciona l'opció 1 del menú dóna error (el Valgrind). La resta d'opcions i implementacions les analitza correctament.