SISTEMES OPERATIUS II:

PRÀCTICA 2

Curs 2019-2020

Juan Cano i Sergi Romero

1. A l'enunciat de la pràctica es comenta que la funció fgets és més segura que fscanf per llegir text pla. Podeu comentar per què? Quines avantatges aporta la funció fgets a nivell de seguretat?
En aquesta pràctica fem servir la funció fgets ja que és més segura que fscanf per llegir text pla, això és perquè fgets() va llegint per línies i ho guarda en el buffer passat per paràmetre on també s'assigna la seva mida màxima. Es guarda el salt de línia ('\n') i un terminating null byte al buffer ('\0'). Acaba de llegir quan troba un salt de línia o EOF. En canvi, si fem ser fscanf() podem obtenir una entrada no desitjada (com per exemple una paraula molt llarga que excedís la mida del *buffer*) i deixaria el file pointer en una posició desconeguda en cas d'error.

Per tant, fgets aporta més seguretat perquè sempre coneixerem que ha llegit fins al salt de línia i no tindrem errors en cas que l'input no tingui el format adequat. Per tant, utilitzar fscanf pot ser perillós si el que volem es llegir línia per línia.

2. Com s'ha adaptat el codi de la manipulació de l'arbre perquè la clau d'indexació sigui una cadena. Indiqueu de forma clara quines funcions s'han adaptat. Recordeu que, tal com s'indica a la secció 2.1, cada node ha d'emmagatzemar la paraula amb la mida mínima necessària (i.e. no es pot suposar que la paraula té una mida màxima de, per exemple, 100 caràcters).
Per poder adaptar el codi de la manipulació de l'arbre perquè la clau d'indexació sigui una cadena hem hagut de fer les següents modificacions:

Hem editat les funcions de comparació de claus de la llibreria *red-black-tree* (*compare_key1_less_than_key2* i *compare_key1_equal_to_key2*) perquè es faci seguint l'ordre lexicogràfic. En aquestes funcions hem canviat el tipus d'input per *strings* i la relació d'ordre dels enters (>, <, =) per la funció *strcasecmp(a, b)* que compara alfabèticament, sense diferenciar entre minúscules i majúscules, els strings *a* i *b*. També hem hagut d'editar la funció *delete_tree_recursive()* per alliberar l'espai de memòria dinàmica que ocupem amb la clau en forma d'*string* (abans era entera i no ocupava espai de manera dinàmica).

Com que cada node ha d'emmagatzemar la paraula amb la mida mínima necessària i no amb una mida màxima predeterminada hem fet que es reservés a memòria exactament l'espai que ocupa la paraula a guardar. Primerament guardem en una variable temporal la paraula a entrar a l'arbre. Aleshores reservem a memòria exactament l'espai que ocuparà la paraula (longitud de la paraula de la variable temporal per la mida d'un caràcter) i finalment copiem (amb strncpy) la paraula a guardar a la zona reservada de memòria. En aquesta copia eliminem el caràcter *"\n"* que marca el final de la línia.

3. Quin és el problema que té el codi d'extracció de paraules de la secció 2.2? Comenteu breument com ho heu arreglat.

El problema que té el codi d'extracció de paraules de la secció 2.2 és que els caràcters no alfabètics no els tractava de la mateixa manera que es demanava en l'enunciat de la practica, en particular les paraules que contenen un apòstrof les separa en dues de diferents i nosaltres volem que compti com una sola paraula. Per solucionar-ho, fem servir el codi ASCII de l'apòstrof per tal de que no es consideri com a principi d'una nova paraula.

Així doncs, hem afegit aquesta condició dins dels bucles *while* que determinen quan comencen i quan acaben les paraules (bucles extrets del codi d'extracció de paraules) i en el *if* que comprova que cada caràcter sigui una lletra.

A més, hem fet una funció que retorna un booleà indicant si un caràcter és un símbol per tal de poder ignorar les paraules com ##continue (com s'indica a l'enunciat).

4. En cas que incloeu alguna cosa excepcional (diferent) de la demanada a l'enunciat, podeu incloure també l'explicació de la funcionalitat implementada.

A la classe *red-black-tree,* hem hagut d'implementar una funció que fa un recorregut *Inorder* de l'arbre i va imprimint totes les paraules que apareixen almenys un cop en els fitxers cercats i també s'imprimeix el nombre de vegades que han estat comptades.

5. Quines són les 10 paraules que apareixen més cops als fitxers de texts proporcionats? Comenteu quantes vegades apareixen cadascuna d'aquestes 10 primeres paraules i indiqueu clarament com heu obtingut el resultat. Sou capaços d'obtenir el resultat ajudant-vos de les instruccions de la línia de comandes? És a dir, podeu obtenir el resultat sense haverde programar-ho tot en C?

Per poder obtenir la sortida desitjada hem utilitzat les instruccions del SO al terminal juntament amb la funció en C que acabem de comentar (inOrder()) que mostra per la consola una llista de dues columnes, a la primera columna trobem les paraules que apareixen un cop o més i a la segona columna la quantitat de vegades. Per tant, hem fet servir la següent instrucció:

./main <nom fitxer> | sort -nrk2 | head

On fem ús de les canonades i de la comanda sort per tal d'ordenar les paraules segons la segona columna (quantitat d'aparicions) i per últim utilitzem una altra canonada amb la comanda head per mostrar només les 10 primeres paraules.

A continuació adjuntem les captures de pantalla amb els resultats obtinguts depenent del fitxer:

```
jcanopra7.alumnes@ub052098:~/Baixades/Practica2/src/base_dades$ ./main llista_2.cfg | sort -nrk2 | head
the: 17812
AND: 15470
OF: 9352
to: 6899
A: 5419
I: 4574
that: 4556
HE: 4036
IN: 3999
IT: 3277
```

Imatge 1: llista_2.cfg

```
jcanopra7.alumnes@ub052098:~/Baixades/Practica2/src/base_dades$ ./main llista_10.cfg | sort -nrk2 | head
the: 63143
AND: 34928
OF: 34303
to: 21789
A: 19300
IN: 15882
that: 10725
HE: 9337
I: 8954
IT: 8900
```

Imatge 2: llista_10.cfg

```
jcanopra7.alumnes@ub052098:~/Baixades/Practica2/src/base_dades$ ./main llista.cfg | sort -nrk2 | head
the: 4629457
OF: 2598540
AND: 2546943
to: 1978343
A: 1367185
IN: 1246383
that: 860600
I: 801542
HE: 786367
was: 705081
```

Imatge 3: llista.cfg