Sistemes Operatius II

Informe de la Pràctica 2

Realitzat per Joaquim Yuste i Marcos Plaza

Índex

Introducció	3
Resposta a les diferents preguntes de l'enunciat	3
Primera pregunta. Funcions fgets vs fscanf	3
Segona pregunta. Adaptació de la clau de l'arbre	3
Tercera pregunta. Problema del codi d'extracció de paraules	4
Quarta pregunta. Especificacions	5
Cinquena pregunta. Top 10 paraules	5

Introducció

En aquest informe trobareu les respostes a les preguntes plantejades en els diferents apartats de l'enunciat de la pràctica.

Resposta a les diferents preguntes de l'enunciat

Primera pregunta. Funcions fgets vs fscanf

Inicialment se'ns pregunta sobre quina de les dues funcions (fgets o fscanf) és més segura per a llegir text pla.

La resposta a aquesta qüestió hem de respondre que la funció fgets, és més segura que la funció fscanf. Això es deu al fet que a la funció fgets, fem explícit la mida del buffer i d'aquesta manera evitem el BufferOverflow.

És a dir, amb la funció fscanf podem arribar a què el buffer intern se saturi (p.e: perquè hem llegit masses dades d'algun fitxer) i resulti en 'overflow' com hem mencionat anteriorment, amb la funció fgets no pot passar, ja que a més aquesta funció ja determina, per defecte, una mida màxima per a recollir les dades línia a línia.

Segona pregunta. Adaptació de la clau de l'arbre

La 'key' de cada node a l'arbre balancejat, segons la nostra implementació serà un punter de tipus char. Per a fer que la indexació sigui correcta (en ordre alfabètic), tal com se suggereix a l'enunciat de la pràctica, farem servir la funció strcasecmp (ignora majúscules/minúscules) la qual actua sobre 2 strings (string1 i string2), i les compara. Aquesta funció pot retornar diferents valors segons la següent taula:

Valor	Significat
Menor a 0	La string1 va abans que la string2 alfabéticament
Igual a 0	La string1 és igual a la string2
Major a 0	La string1 va després que la string2 alfabéticament

Aleshores a partir d'aquí hem modificat les funcions: compare_key1_less_than_key2 i compare_key1_equal_to_key2. Després aquestes funcions s'utilitzen internament per realitzar les rotacions a l'hora d'inserir un node a l'arbre (més concretament s'utilitzen a la funció insert_node).

D'altra banda, es demana que cada node emmagatzemi la mida mínima per a cada paraula, és a dir si una paraula com "house" té 5 lletres, que la mida de la key sigui de 6 bytes (1 per a cada char més el 'end of word'). Això ho hem aconseguit fent un malloc mitjançant la mida de la paraula (ús de la funció strlen) +1. D'aquesta manera reservem l'espai de memòria dinàmica just i necessari.

```
n_data->key = malloc(strlen(word)+1); //we allocate the exact memory we need to store the word
```

Tercera pregunta. Problema del codi d'extracció de paraules

En el codi proporcionat al directori extraccio-paraules, podem veure que existeixen alguns inconvenients segons les normes esmentades a l'enunciat que podem veure a continuació:

> Cal tenir en compte les següents regles per extreure les paraules:
>
> 1. Les paraules poden estar separades per espais, tabuladors o altres signes de puntuació. Es considerarà que aquests símbols no formen part de la paraula. És a dir, en trobar la cadena "why?" s'extraurà la paraula vàlida "why" ja que "?" és un signe puntuació. Les paraules unides per guions, com per exemple "taper-light", són paraules vàlides separades: "taper" i "light". Les paraules que continguin apòstrofs, com per exemple "wit's", són una única paraula vàlida.
>
> 2. L'aplicació no té perquè ser capaç de tractar paraules amb accents. Així, paraules com "Wäts" s'ignoraran. Es recomana no intentar tractar aquests casos ja que les lletres amb aquests tipus de símbols s'emmagatzemen de forma molt particular en un fitxer de text pla i són complicats de processar. Vegeu secció 3 per a més informació al respecte.
>
> 3. Paraules que continguin números o altres símbols s'ignoraran. Així, per exemple, una paraula com "hello123" o "##continue" s'ignoraran.

Podem observar que el primer punt es compleix parcialment, ja que el codi per defecte no agafa les paraules que contenen apòstrofs. El punt 2 i 3 sí que es compleix en la majoria dels casos, com que utilitza la funció isalpha per eliminar caràcters que no són lletres sense cap símbol excepcional com accents, dièresi o altres. El problema ve que quan tenim un caràcter invàlid al final o al principi d'una paraula. El programa el tracta com un signe de puntuació, ja que fa servir la funció ispunct la qual recull els següents signes: ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~, i aleshores agafem la paraula, quan en realitat l'hauríem de rebutjar.

Val a dir que en el nostre codi hem trobat solució per a tots aquests inconvenients. Hem fet que es puguin incloure apòstrofs dins una paraula. També hem considerat uns quants signes ortogràfics vàlids dins un text, els quals, si van enganxats a una paraula, aquesta no és rebutjada. Són els següents: . , : ; ? ! (i) (funció valid_ortographical_signs). Ho hem solucionat modificant els bucles while dins la funció process_line, la qual podeu veure al codi proporcionat a la carpeta src.

Quarta pregunta. Especificacions

Addicionalment hem afegit una nova variable a l'arbre, aquesta és de tipus int anomenada size, anirà guardant el nombre d'elements que conté l'arbre. A més a més, hem implementat una funció que ens permet imprimir l'arbre sencer en ordre.

```
/**
 *
 *  Function used to print a tree. Do not call directly
 *
 */
void inorder_print(node *x){
    if (x != NIL){
        inorder_print(x->left);
        printf("Key: %s | Times: %d\n", x->data->key, x->data->num_times);
        inorder_print(x->right);
    }
}

/**
 *
 *  Print the keys of the tree in order
 *
 */
void print_tree(rb_tree *tree){
    if(tree->root!=NIL)
        inorder_print(tree->root);
}
```

Cinquena pregunta. Top 10 paraules

Les 10 primeres paraules són les següents:

```
Key: the | Times: 17700
Key: AND | Times: 15221
Key: OF | Times: 9345
Key: to | Times: 6856
Key: A | Times: 5392
Key: that | Times: 4492
Key: I | Times: 4292
Key: HE | Times: 4003
Key: IN | Times: 3977
Key: IT | Times: 3180
```

El nombre mínim nombre de paraules assolit per les 10 paraules més utilitzades és de 3180 i la paraula més feta servir arriba a les 17700 aparicions (llista_2.cfg).
Per aconseguir aquesta informació hem fet ús de la funció abans esmentada que ens permet imprimir l'arbre. No obstant ens hem ajudat de les comandes de bash per ordenar i imprimir només les 10 paraules més trobades.

La comanda en qüestió és la següent:

```
./practica2 llista_2.cfg | sort -k5 -nr | head
```

Bàsicament redireccionem la sortida a la comanda sort, que ordenarà de més gran a més petit (-nr) les dades que li arribin en funció de la cinquena columna (-k5), que correspon al nombre d'aparicions.
Un cop s'ha ordenat, es torna a reencaminar la sortida a la comanda heap, que per defecte agafa les 10 primeres files, però això es podria modificant fent servir "-n N" on N és el nombre de files a imprimir.