TP 6 – Concordance

0 Introduction

Une concordance est une liste de toutes les occurrences de tous les mots d'un texte (voir Section 5 pour un exemple). Dans ce tp, on utilise un dictionnaire pour créer une concordance d'un fichier texte quelconque. Dans un premier temps, on implante le dictionnaire par une liste chaînée et dans un deuxième temps par une table de hachage.

Il n'y a pas de "squelette de code" pour ce tp. Vous êtes libre de structurer vos fichiers comme vous voulez. Seulement les opérations les plus importantes sont mentionnées dans les questions. Il y en a d'autres (par exemple, liberation de mémoire) qui sont également importantes.

1 Dictionnaire (liste chaînée)

▸ **Exercice 1 :** Dans cet exercice, on implante un dictionnaire par une liste de couples "mot – nombre d'occurrences". On utilise la structure suivante.

```
typedef struct _link {
    char *word;
    int occurrence;     /* nombre d'occurrences */
    struct _link *next;
} link;
```

(a) Écrire une fonction `link *insert_first_list(link *lst, char word[])` qui ajoute un mot au début de la liste `lst` avec un compte d'occurrences égal à 1. La fonction renvoie un pointeur sur la première cellule de la nouvelle liste.

(b) Écrire un programme qui lit un fichier texte et insère les mots [1] du texte dans une liste.

(c) Écrire une fonction `link *find_list(link *lst, char word[])` qui cherche le mot `word` dans la liste `lst`. Si le mot est présent, la fonction renvoie un pointeur sur la cellule qui le contient. Sinon, la fonction renvoie `NULL`.

(d) Modifier votre programme pour qu'il cherche chaque mot dans la liste avant de l'insérer. Si le mot est déjà présent dans la liste, on incrémente le champs `occurrence` par 1. Sinon, on ajoute le mot au début de la liste grace à la fonction `insert_first_list`.

(e) Tester votre programme sur les fichiers `old.txt` (petit), `2city11.txt` (grand) ou sur un autre fichier texte de taille importante, par exemple du site http://www.gutenberg.org/.

1. Un "mot" est une séquence de caractères. Les mots sont séparés par des espaces ou des retours chariot. On va retrouver des mots de type `"Why?"`, `"Think,"` ou bien `"(or"`. Pour l'instant, on les laisse comme ça.

2 Occurrences (liste de listes)

▸ **Exercice 2 :** Dans cet exercice, on remplace le simple compte d'occurrences de chaque mot par une liste chaînée qui stocke toutes les positions où un mot apparait dans le texte. La liste devient donc une liste de listes. On utilise (et modifie) les structures suivantes.

```
typedef struct _olink {
        int pos;
        struct _olink *next;
} olink;

typedef struct _link {
    char *word;
    olink *occurrence;  /* liste des occurrences */
    struct _link *next;
} link;
```

(a) Écrire une fonction `void add_occurrence(link *lnk, int pos)` qui ajoute une occurrence (dans une nouvelle cellule `olink`) à la liste `lnk->occurrence`.

(b) Modifier vos fonctions et programme principal pour qu'ils fonctionnent avec cette nouvelle représentation des occurrences.

3 Table de hachage

▸ **Exercice 3 :** Dans cet exercice, on réimplante le dictionnaire par une table de hachage. On utilise la structure suivante.

```
typedef struct _table {
    link **bucket;
    int  M;       /* nombre de seaux */
    int  size;    /* nombre de mots dans la table */
} table;
```

(a) Écrire une fonction `table *create_table(int M)` qui crée une nouvelle table de hachage à `M` seaux. Tous les seaux sont initialement vides.

(b) Écrire une fonction `void add_occ_table(table *tab, char word[], int pos)` qui insère une occurrence du mot `word` sur la position `pos` dans la table de hachage.

1. La fonction calcule d'abord le seau correspondant au mot `word`.

2. Si le mot est déjà présent dans le seau, alors la fonction insère une nouvelle occurrence dans la liste d'occurrences grace à la fonction `add_occurrence` de l'exercice précédent.
3. Sinon, la fonction insère le nouveau mot au début du seau.

(c) Modifier le programme principal pour qu'il fonctionne avec cette nouvelle représentation du dictionnaire.
   — Tester le programme sur le fichier `2city11.txt` avec 1, 50 et 2000 seaux. Comparer le temps d'exécution.
   — Tester le programme sur le fichier http://norvig.com/big.txt.

(d) Écrire une fonction `int size_table(table *tab)` qui renvoie le nombre de mots présents dans la table. Vérifier que le compte est bon.

▸ **Exercice 4 :** Modifier le programme principal pour qu'il prenne deux arguments sur la ligne de commande : le nom d'un fichier d'entrée et le nom d'un fichier de sortie sur lequel la concordance sera écrite.

4 Pour aller plus loin [FACULTATIF]

▸ **Exercice 5 :** Les questions suivantes sont des suggestions pour explorer davantage la table de hachage et la concordance. Certaines sont plus laborieuses que d'autres. Vous pouvez les faire (ou pas) dans l'ordre que vous voulez.

(a) Permettre le nombre de seaux à augmenter si la table devient trop pleine.

(b) Créer une visualisation des tailles des seaux (longueur des listes) pour vérifier que la fonction de hachage distribue les mots de façon uniforme.

(c) Faire en sorte que l'affichage de la concordance liste les mots en ordre lexicographique et les occurrences en ordre croissant.

(d) Nettoyer les mots du texte d'entrée. Par exemple, vous pouvez supprimer tout caractère sauf les alphanumériques et les convertir en minuscules.

(e) Écrire un programme qui lit une concordance et recrée le texte. (Il est possible que le résultat ne soit pas exactement le texte d'origine, par exemple si vous avez nettoyé les mots.)

5 Exemple

Pour le texte suivant :

```
Old MACDONALD had a farm
E-I-E-I-O
And on his farm he had a cow
E-I-E-I-O
With a moo moo here
And a moo moo there
Here a moo, there a moo
Everywhere a moo moo
Old MacDonald had a farm
E-I-E-I-O
```

on a la concordance (les positions commencent par 0) :

```
a : 3 12 16 21 26 29 32 38
And : 6 20
cow : 13
E-I-E-I-O : 5 14 40
Everywhere : 31
farm : 4 9 39
had : 2 11 37
he : 10
here : 19
Here : 25
his : 8
MACDONALD : 1
MacDonald : 36
moo : 17 18 22 23 30 33 34
moo, : 27
Old : 0 35
on : 7
there : 24 28
With : 15
```