Documentation
Indexeur de texte

Yoann Thomann
Mohammed Bouabdellah
ythomann@univ-mlv.fr
mbouabde@univ-mlv.fr

12/04/2010

Table des matières

1 Description générale **2**

2 Manuel Utilisateur **2**

2.1 Conseils d'utilisation 3

2.2 Recommandation 3

3 Manuel Developpeur **3**

3.1 Parseur de mot 3

3.2 Parseur de mot 4

3.3 Contrôle qualité 4

1 Description générale

Ce programme permet, à partir d'un texte, de représenter un dictionnaire en utilisant une table de hachage. Il indexe notamment des mots, en les associant à toutes les positions des phrases dans lequels ils peuvent apparaître dans le texte.

Il est donc possible en executant ce programme, pour un mot, de tester :

- l'appartenance d'un mot au texte
- l'affichage des positions d'un mot dans le texte
- l'affichage des phrases contenant un mot
- l'affichage de la liste triée des mots et de leur position
- l'affichage des mots commençant par un préfixe donné
- la sauvegarde de l'index

2 Manuel Utilisateur

Il existe deux manières d'éxecuter ce programme :

```
./Index [fichier]
```

et

```
./Index [option] [mot] fichier
```

Ces informations sont accessible a partir de l'executable via la commande :

```
yoann@hp_laptop# ./Index -h

SYNOPSIS:
Index [option] file
        or
Index [file]

Examples:
Index -a word file      | Check if word is in file.
Index -p word file      | Print word positions in file.
Index -P word file      | Print sentences containing word in file.
Index -l text           | Print sorted list of text's words.
Index -d word file      | Print words begining with word in the text.
```

```
Index -D out  file      | Save sorted list of file's words in out
    .DICO
Index -h out  file      | Print this help
```

La première commande donne accès à un menu permettant de manipuler les diverses fonctions de l'index.
La deuxième permet d'effectuer des opérations ponctuelles selon les arguments de la commande, cette deuxieme est particulièrement adaptée lors de l'utilisation du programme dans un script.
L'utilisateur pour ce faire peut rediriger la sortie standard, puis la traiter...

2.1 Conseils d'utilisation

Le fichier passé en argument doit être un fichier texte. Il est possible lors du traitement du texte de voir l'evolution du traitement en utilisant l'option verbose **-v**.
Example :

```
./Index -pv oreste Andromaque.txt
```

2.2 Recommandation

La principale recomandation d'utilisation dans le cas du test d'appartenance d'un mot dans le texte en mode interactif, un hash est systématiquement éffectué dans ce mode, ainsi si l'utilisateur souhaite uniquement rechercher la présence d'un mot dans le texte, il serait plus efficace d'utiliser la commande :

```
./Index -a word Colomba.txt
```

En effet, cette commande n'engendre elle pas la création d'un hash, elle recherche directement dans le fichier en le parcourant alors qu'en mode interractif, un hash du fichier est crée et le mot est recherché dans celui çi.

3 Manuel Developpeur

3.1 Parseur de mot

Pour récuperer les mots du textes, on aurait pu utiliser une fonction du type *fgets/fscanf*, mais ces fonctions 'voient' un mot est comme une suite

de caractère séparés par des espaces, retour chariot, tabulation ; ce qui ne correspond pas à notre définition d'un mot.
En effet, un mot est pour nous une suite de caractères espacés par des espaces, retour chariot, tabulation, et divers caractères de ponctuations. Il nous a donc fallu recuperer les mots caractères par caractère.
Dans la récupération de mots, on peut distinguer 3 cas :

- fin de mot
- fin de phrase
- fin de texte

Face à ces 3 cas, on agit de la sorte : si une fin de mot se présente, on stocke le mot dans la table de hashage ou met a jour sa position (si déjà présent), si une fin de phrase se présente, on met à jour la valeur de l'offset (position dans le texte en octet) ; et enfin dans le cas d'une fin de texte on s'arrête là.

3.2 Parseur de mot

3.3 Contrôle qualité

Une fois le programme fonctionnel, des tests ont été effectués, afin de vérifier et corriger des éventuels *bugs*.
Nous avons dans un premier temps utilisé le programme valgrind qui est un outil de debugging et de profiling.

```
valgrind ./Index -vp camion Colomba.txt
...
==4255== HEAP SUMMARY:
==4255==     in use at exit: 0 bytes in 0 blocks
==4255==   total heap usage: 10,043,999 allocs, 10,043,999 frees, 168,883,240 byte
...
==4255== ERROR SUMMARY: 0 errors from 0 contexts
```

On constate que a la fin de l'exécussion, la totalité de la mémoire utilisée est bien libérée, et que l'on a aucune erreur.

Dans un deuxième temps nous avons testé les *entrées* en utilisant le fichier **/dev/random**.

```
yoann@hp_laptop# ./Index -p tt /dev/random
^C
ctrl-c caught, exiting...
```

Apres un certain temps, on entre au clavier *ctrl-c* afin d'arréter le processus. On constate ici que l'on a pas rencontré d'erreurs ni d'erreur mémoire *segmentation fault*.