SIN5U1 – Algorithmique Avancée
Devoir2 : Un correcteur orthographique

Le but de ce devoir est de réaliser un correcteur orthographique des mots isolés du français. C'est à dire que si on lui donne un mot, il dit si il appartient ou non au français (il est présent dans un dictionnaire du français). Si il est absent il propose des orthographes possibles. Pour cela on aura besoin des programmes suivants :

- Calcul de la distance de Levenshtein (ou distance d'édition)
- Construction et utilisation d'une table de hash-code.

Ce devoir sera fait en binôme et durera en principe pendant 5 séances de TP. Voici la progression que je vous propose (mais vous pouvez aller plus vite) :

1. TP 6 : implantation du calcul de la distance de Levenshtein et test du programme.
2. TP 7 : Implantation des tables de hash-code et de la recherche d'un mot
3. TP 8 : Lecture d'un dictionnaire et étude du remplissage de la table de hash-code.
4. TP 9 : Réalisation du correcteur orthographique.
5. TP 10 : Test et performance de votre correcteur orthographique (en temps et en nombre de mots corrigés)

Si vous avez ses difficultés pour comprendre utilisez le forum du cours.

Remarque préliminaire

Tous les fichiers de texte que je vais vous donner sont codés en UTF8. Si vous faites le devoir en Java, pas de problèmes, mais si vous travailler en C, il faut convertir ces fichiers avec le format ISO8859-15 (il faut le -15 pour avoir 'œ' et '€'). Gedit (ou la commande iconv d'unix) fait ce travail.

Calcul de la distance de **Levenshtein**

On appelle distance de Levenshtein entre deux mots M et P le coût minimal pour aller de M à P en effectuant les opérations élémentaires suivantes:

- substitution d'un caractère de M en un caractère de P.
- ajout dans M d'un caractère de P.
- suppression d'un caractère de M.

On associe ainsi à chacune de ces opérations un coût. Le coût est toujours égal à 1, sauf dans le cas d'une substitution de caractères identiques. En principe vous devriez avoir vu cet algorithme dans le TD 6. C'est le même esprit que le calcul de la plus longue sous séquence commune. Vous pouvez regarder également l'article dans Wikipedia. Ce programme prend en entrée deux chaînes et rend la distance. Plus cette distance est grande, plus les mots sont différents.

Vous passerez les exemples suivants et vous mesurerez le temps par exemple pour le calcul de la distance de Levenshtein entre baillonette (attention ce n'est pas la bonne orthographe) et les 424581 mots du dictionnaire fourni. Ce temps vous parait-il raisonnable par exemple si on veut corriger un texte de 1000 mots.

```
Levenshtein(acceuil, accueil) = 2
Levenshtein(arborigène, aborigène) = 1
Levenshtein(baillonette, baïonnette) = 4
Levenshtein(banquaire, bancaire) = 2
Levenshtein(déconnection, déconnexion) = 2
Temps de Levenshtein(baillonette,mot du  dico) = 687.8 ms
```

Attention les temps mesurés correspondent à mon ordinateur et mon programme.

Les tables de hash-code

Implanter les procédures pour la création et l'utilisation d'une table de hash-code. Ceci a été vu dans le cours 8. Vous pouvez également consulter les pages de Wikipedia. Voici deux conseils pour cette réalisation :

1. Vous choisirez une valeur de alpha aux environs de 0.5. Comme il y a 424581 mots, cela veut dire que la taille de votre table sera de N = 1000000 d'entrées.
2. Vous choisirez comme fonction de hashcode celle de java :
```
int hash = 0;
for (int i = 0; i < length(); i++) hash = (hash * 31 + charAt(i)) % N;
```
3. Vous représenterez les mots ayant le me hashcode sous forme d'une liste chaînée.

Vous serez amenés à créer plusieurs tables de hashcode.

Vous mesurerez le temps de lecture et construction de la table de hashcode pour le dictionnaire du français fourni. Vous regarderez également comment sont remplies les différentes entrées de la table de hashcode et vous comparerez ces chiffres avec la loi de poisson. Voici mes résultats :

```
Temps de lecture et construction du dictionnaire = 222 ms
Dictionnaire : alpha = 0.424581, nbMots = 424581, maxBoite = 7

k  nb Entrée   Poisson
0  654166      654043.77
1  277420      277694.56
2  59171       58951.92
3  8249        8343.29
4  906         885.6
5  81          75.2
6  6           5.32
7  1           0.32
```

Encore une fois, il est très improbable que vous trouviez exactement les mêmes résultats que moi.

Un dictionnaire du français

On trouve beaucoup de dictionnaires des mots du français, nous allons utiliser le suivant : `dico2012.txt` qui va se trouver dans le dossier `materiel4.zip` dont l'adresse est donnée à la fin de ce document. La liste des mots du français que je vous propose est *un dictionnaire de 424581 mots (formes fléchies comprises) - ce lexique est extrait du site : http://www.dicollecte.org/*

Le correcteur orthographique proposé

Voici maintenant la méthode que je vous propose. Cette méthode se décompose en deux étapes :

Etape 0 : On construit la table de hashcode correspondant au dictionnaire fournie

Etape 1 : On lit une liste de mots corrects (le dictionnaire). Pour chaque mot

1. on rajoute un caractère « $ » au début et à la fin du mot. « accueil » devient « $accueil$ »
2. Puis on construit la liste des tri-grammes apparaissant dans ce mot. {«$ac», «acc», «ccu», «cue», «uei», «eil», «il$»} .
3. On construit alors au fur et à mesure le dictionnaire des tri-grammes. Chaque tri-gramme pointant vers la liste des mots contenant ce tri-gramme. Par exemple «acc» pointe vers la liste {« accueil », « accent », « accélération », « raccourci », ….}

Etape 2 : La correction orthographique. Pour chaque mot que l'on cherche à corriger, c'est à dire qui n'apparait pas dans le dictionnaire, on procède ainsi :

1. Si le mot appartient au dictionnaire : c'est fini – il n'y a pas de faute d'orthographe.
2. on rajoute un caractère « $ » au début et à la fin du mot. « acceuil » devient « $acceuil$ »
3. Puis on construit la liste des tri-grammes apparaissant dans ce mot. {«$ac», «acc», «cce», «ceu», «eui», «uil», «il$»} .
4. En consultant notre liste de tri-grammes, on recherche les mots (corrects) du dictionnaire contenant le plus possible de ces tri-grammes. Par exemple « accueil » contient 3 tri-grammes en commun avec « acceuil », mais par exemple « fauteuil » contient aussi 3 tri-grammes commun avec « acceuil ».
5. On va trier cette liste de candidats à la correction en fonction du *coefficient de Jaccard* (http://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard). On ne va garder que les candidats ayant un coefficient > 0,2.
6. A partir de cette liste on va faire une autre classement. Parmi ces candidats potentiels, on va les trier en utilisant la distance de Levenshtein. En fait on va utiliser une variante « d » de cette distance :
   d(mot1,mot2) = 1.0 – dLevenshtein(mot1,mot2)/max(|mot1|,|mot2|)
   Et on va proposer comme liste de correction, les 5 (ou 10) premiers de cette liste par exemple.

Cette description est volontairement incomplète, mais en réfléchissant un peu vous devriez pouvoir comprendre. Et vous pouvez toujours me poser des questions.

Pour faire ce devoir je me suis servi d'un cours associé à un livre existant sur Internet : http://nlp.stanford.edu/IR-book/information-retrieval-book.html (vous pouvez regarder le chapitre 3 de ce livre [http://nlp.stanford.edu/IR-book/pdf/03dict.pdf] qui est en ligne. Vous pouvez aussi regarder le reste pour votre culture générale). Vous pouvez également regarder les transparents du cours (« slides ») toujours concernant le chapitre 3. Je trouve que c'est plus facile à comprendre.

Ce que vous devez faire

1. Vous devez programmer la méthode précédente.
2. Vous allez mesurer le temps nécessaire à la construction des deux dictionnaires (Celui du français et celui des trigrammes). Puis à la correction d'un mot.
3. Comme test de votre programme vous devez corriger les fautes qui se trouvent dans le fichier joint : « fautes2012.txt » (qui se trouve dans le dossier materiel4.zip). Cela serait bien d'expliquer pourquoi vous n'arrivez pas à corriger certains mots. Pour chaque mot, il y a 3 possibilités

   - Ou bien le premier mot proposé est la bonne orthographes
   - Ou bien parmi les 5 ou 15 mots proposés, il y a le bon
   - Ou bien vous n'êtes pas arrivé à trouver une correction du mots

   Vous devez calculer les pourcentages de ces 3 cas pour les mots proposés. Si le premier cas représente 100 % alors votre programme est parfait. Vous mesurerez également le temps total pour corriger ce fichier de fautes.
4. Ce programme est loin d'être parfait, faites moi des suggestions pour l'améliorer

Voici une partie de mes résultats :

```
424581 mots lus, ...L'indexation est terminée, t = 1578ms
temps pour acceuil : 12.16ms
temps pour baillonettes : 33.45ms
temps pour boudiste : 7.22ms

Bilan : 187 bonnes corrections, 242 dans la liste des suggestions sur 269 mots
   (nb candidats : 10)
temps pour corriger les 269 mots  : 3659ms

Pas de correction pour :
omnibulé - obnubilé : omnibus démantibulé omnium omniums dénébulé lobulé tabulé
   tubulé fabulé subulé déambulé omnivore affabulé omnicolore omnivores
Bénélux - Benelux : Bénédicte lux flux Bénin velux
…..
```

Ce que vous devez rendre

Vous allez travailler en binôme (Si possible le même binôme pour toute la durée du cours). Vous devez montrez a votre responsable de TP que votre programme marche . Et envoyer sur AMeTICE un dossier (dont le nom est une combinaison de vos noms et de devoir2) qui contient :

- Un fichier .pdf qui sera votre rapport : 10 pages pour décrire le sujet et la méthode, vos résultats (réponses aux questions) et éventuellement des commentaires.
- Votre programme (ce programme doit être fait de telle manière que si on le compile et on le lance on trouve les mêmes résultats que dans votre rapport – vous pouvez faire un fichier make qui fait tout cela). Cela n'est pas la peine de joindre les données !!

Le matériel se trouve ici :
http://michel.vancaneghem.perso.luminy.univmed.fr/cours/algo/devoir2/materiel4.zip

Ce document se trouve ici :
http://michel.vancaneghem.perso.luminy.univmed.fr/cours/algo/devoir2/devoir2.pdf

Rédigé en Octobre 2012 par Michel Van Caneghem