TP2 – IFT2015 – H17 – Major – Correcteur multilingue

Pour ce TP, vous allez programmer un correcteur, mieux connu sous la dénomination anglaise « spell checker ». Ce correcteur aura la particularité qu'il prendra en entrée une phrase et un dictionnaire de référence, rendant possible la correction de divers langages réels ou fictifs.

Principe du correcteur

Comme vous le savez déjà, un correcteur analyse un texte pour détecter les mots qui ne sont pas dans le dictionnaire de la langue sélectionnée. Pour ces mots, le correcteur fait des suggestions à l'utilisateur s'il en trouve d'autres qui ont une orthographe similaire. Par exemple, un correcteur pourrait suggérer « correcteur » si l'usager a entré « corrceteur ». Les types de suggestions que votre correcteur devra faire seront définis plus loin dans l'énoncé.

Entrée

L'entrée de votre correcteur sera divisée en deux parties : un dictionnaire dans un fichier **dict.txt** et une phrase dans un fichier **input.txt**. Votre programme doit comprendre un fichier `tp2_matricule1_matricule2.py` qui sera **exécuté dans un dossier** comprenant ces fichiers. Afin de rendre le correcteur capable de traiter les caractères spéciaux, les deux fichiers suivront l'encodage **utf-8** (qui est l'encodage par défaut en Python 3). Aucun argument en ligne de commande ne sera nécessaire.

Dictionnaire

Le fichier **dict.txt** contiendra un mot par ligne, dans un ordre quelconque. Un mot peut contenir un trait d'union (`-`), mais pas d'apostrophe (`'`) ou autres signes de ponctuation. C'est à partir de ce dictionnaire que vous allez construire votre dictionnaire interne qui sera implanté sous forme de table de hachage.

Phrase

La phrase que vous devrez corriger sera donnée dans le fichier **input.txt**. Elle sera sur la première ligne et pourra contenir les caractères de ponctuation suivants :

`.` `,` `:` `;` `!` `?` `'` `"`

Vous devrez ignorer ces caractères et ne considérer que les mots présents. Notez qu'une apostrophe sépare deux mots différents. Pour chacun des mots, votre programme vérifiera si le mot est contenu dans le dictionnaire. Si oui, vous produirez le mot tel quel en sortie. Sinon, vous produirez le mot entouré de crochets « [ ] », suivi d'une liste de suggestions entre parenthèses (potentiellement vide). Voici un exemple de phrase :

```
Je me demende si le correcteur va corrigé cette phrase.
```

Et voici ce que le correcteur devrait produire en sortie avec un dictionnaire de français :

```
je me [demende](demande) si le correcteur va corrigé cette phrase
```

Notez que le correcteur n'a pas corrigé l'erreur de la terminaison en « é », car il ne vérifie que l'orthographe. Notez aussi que la ponctuation et la majuscule initiale ont été retirées par rapport à la phrase d'origine. Ce comportement est permis pour vous simplifier la tâche, mais n'est pas requis (voir Bonus1).

Sortie

Vous ne devez produire que la phrase en sortie, avec erreurs identifiées et suggestions s'il y a lieu. Si vous ne trouvez aucune suggestion pour un mot absent du dictionnaire, ajoutez des parenthèses vides après le crochet fermant. Si vous avez plusieurs suggestions, séparez-les d'une virgule. Exemple :

```
il y a un [verr](vert, vers, verre, ver) dans ma [sadjkltyuiwe]()
```

Table de hachage

Vous devrez faire vous-même l'implantation de votre table de hachage. Cela veut donc dire que vous ne pourrez **pas utiliser dict()** qui est fourni par Python, ni **aucun objet/fonction de type abstrait dictionnaire** provenant de Python ou d'un module externe. Vous devrez aussi implanter votre propre fonction de hachage et ne devrez **pas utiliser la fonction hash de Python**. Vous implanterez donc une classe HashTable à l'intérieur de laquelle vous définirez les méthodes suivantes :

- get(*self,* key) : retourne la valeur associée à l'index correspondant à key
- set(*self,* key, value) : insère la valeur value à l'index correspondant à key
- del(*self,* key) : supprime la valeur à l'index correspondant à key
- autres méthodes internes pour votre fonction de hachage (voir plus bas)

Le constructeur de votre classe devra prendre la taille du tableau initial en entrée. Dans le cas de ce TP, vous n'aurez pas à implanter de méthode de réajustement de la taille car vous allez connaître le nombre d'entrées du dictionnaire en avance. Prenez une taille de tableau égale à environ 2 fois ce nombre d'entrées afin de minimiser à la fois la mémoire utilisée et le nombre de collisions.

Stratégie de hachage

Vous pouvez vous baser sur le document **hashtables.pdf** vu en cours pour la sélection d'un stratégie de hachage et de gestion des collisions. La plus simple est la table de hachage avec listes chaînées, mais vous pouvez utiliser les autres aussi. Pour le code de hachage, vous pouvez essayer différentes stratégies. La plus simple consiste à simplement considérer la chaîne de caractères à traiter comme un entier, mais elle engendrera beaucoup de collisions. L'accumulation polynomiale (**p. 8** du document) est la plus efficace pour les

chaînes de caractères. Assurez-vous d'avoir la version actuelle du document car quelques coquilles viennent d'être corrigées.

Suggestions

Votre correcteur devra fournir ses suggestions à partir de 5 types de modifications apportées au mot courant. Les voici :

- Intervertir chaque paire de caractères adjacents du mot
- Insérer chaque lettre de l'alphabet entre chaque paire de caractères adjacents du mot, de même qu'au début et à la fin du mot
- Supprimer chaque caractère du mot
- Remplacer chaque caractère du mot par chaque lettre de l'alphabet
- Séparer le mot en paire de mots de toutes les *n - 1* façons possibles (pour un mot de longueur *n)*. La paire sera suggérée seulement si les **deux mots** se trouvent dans le dictionnaire.

Afin de pouvoir faire ces modifications, vous devrez **mémoriser l'alphabet** du langage corrigé lors de la lecture du dictionnaire afin de pouvoir insérer chaque caractère possible. Autre remarque : il est possible que deux **suggestions identiques** (ou plus) soient générées. Dans ce cas, vous devrez en donner **une seule en sortie**.

Données de pratique

Vous trouverez sur Studium un fichier **dict.txt** et un fichier **input.txt** contenant une phrase couvrant chaque type de suggestion.

Structure du code

Avertissement : vous devez développer VOTRE PROPRE CODE et tout plagiat détecté entraînera automatiquement un échec.

Votre code doit être clair et bien documenté. Vous pouvez aller consulter PEP 8 qui est un guide de style pour Python à la page suivante : https://www.python.org/dev/peps/pep-0008/

Version de Python

Il est fortement recommandé d'utiliser Python 3.6.0. Il ne serait **pas surprenant** que des problèmes de compatibilité surviennent si vous utilisez Python 2 car l'encodage utf-8 n'est pas utilisé par défaut en Python 2.

Équipes

Vous pouvez faire votre travail seul ou en équipes de **deux** personnes **maximum**. Vous pouvez discuter avec les autres équipes évidemment, mais tout code dupliqué entre deux équipes sera considéré comme un plagiat des deux côtés.

Remise

Vous avez jusqu'au **29 avril à 23h55** pour remettre votre travail sur Studium. Remettez votre travail soit sous forme d'un seul fichier `tp2_matricule1_matricule2.py` ou sous la forme d'une archive `tp2_matricule1_matricule2.tar.gz` ou `tp2_matricule1_matricule2.zip` (**aucune** autre forme de compression ne sera tolérée) si vous avez plus d'un fichier. Dans ce cas, l'archive sera décompressée dans le dossier contenant les fichiers d'entrée et devra contenir entre autres le fichier `tp2_matricule1_matricule2.py` qui sera exécuté.

Correction négative

- 50% des points peuvent être perdus si votre code ne s'exécute pas ou ne produit pas de sortie
- 50% des points peuvent être perdus si vous n'avez **pas implanté** votre **propre table de hachage** ou votre **propre fonction de hachage**
- Une pénalité de **20% par jour de retard** sera appliquée dès la première seconde de chaque période de 24h, en commençant à 23h56 le 22 mars

Correction positive

- 30% : clarté du code
- 10% : respect du format de sortie
- 60% : tests de correction de phrases avec différents dictionnaires

Points boni

Bonus1

5% de bonus vous seront accordés si vous maintenez la ponctuation et les majuscules présentes dans la phrase d'origine. La ponctuation qui suit un mot mal orthographié devra être placée après la parenthèse fermante. Mentionnez que vous avez fait le Bonus1 ainsi : `tp2_b1_matricule1_matricule2.py`

5% supplémentaires seront accordés en fonction de la vitesse d'exécution (par rapport à la moyenne) de votre code :
Code rapide : +1,5%
Code très rapide : +3%
Code extrêmement rapide : +5%

Questions

Envoyez vos questions sur le travail à omailhot92@gmail.com.
Bon travail !!!