TP de Langages de script n° 4 bis : Correcteur d'orthographe

I) Description du projet

Le but du projet est d'implémenter un correcteur d'orthographe. Ce correcteur doit être capable de trouver des mots mal-écrits dans un texte fourni et de donner des suggestions de correction interactivement. Le programme doit aussi permettre de sauvegarder le texte corrigé après avoir fait les corrections.

Le projet est réparti en plusieurs parties de différentes difficultés :

1. *Mode non-interactif:* Écrire un programme `correcteur`, qui affiche une liste des mots mal-écrits avec les numéros de ligne où l'erreur se trouve lorsqu'il est lancé par la commande :

   ```
   correcteur -f monfichier.txt
   ```

   Le programme doit vérifier l'emploi correct des majuscules et en particulier doit s'assurer que les mots au début d'une phrase ou d'un paragraphe commencent par une majuscule.

2. *Mode non-interactif avec suggestions:* Lancé avec la ligne de commande

   ```
   correcteur -f monfichier.txt -s
   ```

   le programme doit afficher une liste de suggestion de correction pour chaque erreur, si possible.

3. *Mode interactif:* Avec la ligne de commande

   ```
   correcteur -f monfichier.txt -i
   ```

   le programme affiche une par une les erreurs **et** la ligne de texte correspondante, et offre une liste de suggestions de correction pour chaque erreur. L'utilisateur peut choisir une des suggestions ou fournir une correction à la main si la bonne ne se trouve pas parmi les suggestions.

   À la fin, l'utilisateur peut choisir d'enregistrer le texte corrigé, soit dans le même fichier, soit dans un fichier spécifié par l'utilisateur (à vous de décider).

4. Extensions:
   - Offrir une option pour sauvegarder dans un fichier les mots qui n'apparaissent pas dans la liste des mots corrects mais que l'utilisateur considère comme correct. Ce fichier sera chargé au prochain lancement du programme. Prévoir le cas d'un mot consideré comme correct apparaissant plusieurs fois dans le texte.
   - Optimisation des suggestions: Présenter des suggestions de correction de sorte que les plus probables soient au début de la liste (voir section c)).

II) Indications

Dans la suite du sujet, quelques indications sont présentées afin de vous aider à construire votre programme.

a) Trouver les erreurs et les suggestions

Pour déterminer si un mot est écrit correctement, on vérifie s'il est inclus dans un ensemble de mots connus. Pour cet ensemble, on peut prendre les mots du dictionnaire `aspell` qui est normalement inclus dans des systèmes UNIX (en particulier au SCRIPT). Pour écrire tous les mots français connus d'aspell dans un fichier `mots`, on utilise la commande

```
aspell -d fr dump master > mots
```

Si le dictionnaire aspell français n'est pas installé sur votre machine, vous trouverez le fichier `mots` également sur DidEL. Ce fichier est assez grand et pour que le test, qui évalue si une chaîne de caractères est un mot connu, soit performant, il est important de stocker les mots dans une structure de données adaptée de votre programme python.

Si vous trouvez un mot dans le texte qui n'est pas connu, il faut trouver des mots connus semblables comme suggestions de correction. Il existe plusieurs heuristiques pour trouver les suggestions. En voici quelques unes :

- Créer des variations de votre chaîne inconnue `s` en faisant de petites modifications comme
  - changer des accents
  - échanger deux lettres consécutives
  - enlever une lettre
  - ajouter une lettre
  - ...
- Parmi les modifications engendrées, prendre les connues comme suggestions

b) Analyse syntaxique

Afin de pouvoir analyser l'orthographe des mots d'un texte, on doit découper le texte en mots individuels. Pour chaque mot, on a aussi besoin de savoir s'il se trouve au début d'une phrase ou d'un paragraphe et du numéro de ligne. Pour pouvoir reconstruire le texte après avoir fait les corrections, il faut en plus stocker les caractères qui se trouvent entre les mots (par exemple espaces, retours à la ligne et ponctuation).

A cette fin, on peut écrire une fonction

```
def decomposition(texte):
```

qui prend une chaîne de caractères (le texte à analyser) en argument et renvoie une liste de « tokens », où un token est un tuple `(chaine, ligne, mot, debut)` dans lequel

- `chaine` est une sous-chaine de `texte`,
- `ligne` est le numéro de ligne correspondant,
- `mot` est une valeur booléenne qui est `True` si `chaine` est un mot à analyser, et
- `debut` est une valeur booléenne qui détermine si `chaine` est au début d'une phrase ou d'un paragraphe.

Le dernier champ est inutile pour les chaînes qui ne sont pas des « mots », on peut lui donner n'importe quelle valeur dans ce cas.

Attention: il doit être possible de reconstruire le `texte` correctement à partir de la liste des tokens.

c) Optimisation des suggestions

Il est désirable de présenter les suggestions afin que les plus probables soient au début de la liste. Pour cela, il faut deviner ce que l'auteur du texte voulait écrire quand il a fait une erreur. Dans l'estimation des corrections les plus probables, il faut prendre en compte deux facteurs:

- Est-ce qu'il est probable de faire une erreur donnée ? En général, il est plus probable d'oublier une lettre que deux, et d'oublier un accent que d'écrire une autre lettre par exemple.
- Est-ce qu'une suggestion est un mot fréquent dans la langue française ? Une correction est plus probable si elle consiste en un mot fréquent.

Il paraît difficile de prendre en compte la fréquence d'un mot dans la langue français, mais en fait il s'agit d'une tache faisable. Une stratégie est de télécharger plusieurs grands textes, par exemple des livres électroniques du site `http://www.gutenberg.org/wiki/FR_Principal` et de créer un dictionnaire python qui compte la fréquence de tous les mots dans ces textes. Après cela, on peut ordonner les suggestions en tenant compte des valeurs associées dans ce dictionnaire.