Correcteur orthographique

Résumé

Ce projet consiste à réaliser un correcteur orthographique de mots isolés qui s'appuie sur la distance de Damerau-Levenshtein pour proposer des corrections plausibles. Deux programmes seront écrits : le premier interactif et le second en ligne de commande, automatique. Dans le cadre de ce projet, un rapport écrit sera rédigé, une présentation orale et une démonstration seront réalisées. Ce document décrit les différents aspects de ce projet : cahier des charges, organisation, livrables, échéances et critères de notation.

1 Cahier des charges

L'objectif de ce projet est de réaliser un correcteur orthographique de mots isolés qui s'appuie sur un dictionnaire. Ce correcteur signalera les mots n'appartenant pas au dictionnaire et suggérera des corrections en s'appuyant sur le calcul de distance entre mots de Damerau–Levenshtein. L'utilisateur pourra alors choisir l'une des propositions pour remplacer le mot erroné, l'ajouter dans un dictionnaire personnel, le conserver ou le corriger lui-même. Un correcteur automatique sera aussi réalisé.

1.1 Le dictionnaire

Le dictionnaire est un fichier texte (en UTF-8) qui ne contient qu'un seul mot par ligne.

1.2 Distance de Damerau–Levenshtein

La distance de Levenshtein entre deux mots M1 et M2 est le coût minimal pour passer de M1 à M2 en effectuant les opérations élémentaires suivantes :

— la substitution d'un caractère ('algorythme' et 'algorithme' : substitution de 'y' par 'i'),

— l'ajout d'un caractère ('lontemps' et 'longtemps' : ajout de 'g') et

— la suppression d'un caractère ('malgrés' et 'malgré' : suppression du 's' final).

À ces trois opérations élémentaires, Damerau ajoute la transposition de deux caractères adjacents (par exemple 'infractus' et 'infarctus' en permutant les lettres 'a' et 'r'). Dans la distance de Levenshtein, la transposition est comptée comme deux opérations élémentaires (suppression d'un caractère et ajout d'un caractère : 'infractus' $\longrightarrow$ 'infactus' $\longrightarrow$ 'infarctus').

On associe à chacune de ces opérations un coût de 1. On peut montrer alors que la distance de Damerau-Levenshtein entre deux mots a et b est obtenue par l'algorithme ci-après où l'on note $|x|$ la longueur du mot x. Cet algorithme calcule de manière itérative la distance entre a et b en stockant dans une matrice d de taille $(|a|+1) \times (|b|+1)$ les distances entre tous les préfixes

de a et tous les préfixes de b. À la fin de l'algorithme, le coefficient $d[i, j]$ est le nombre minimal d'opérations permettant de transformer le préfixe (la sous-chaîne de caractères) $a_{0...i-1}$ en $b_{0...j-1}$.

```
Comment « Calculer la distance entre deux mots a et b » ?
    Initialiser la matrice d
    Pour i De 1 À |a| Faire
        Pour j De 1 À |b| Faire
            Si a[i-1] = b[j-1] Alors
                cout <- 0
            Sinon
                cout <- 1
            FinSi
            d[i, j] <- minimum(d[i-1, j] + 1,       // suppression
                               d[i, j-1] + 1,       // ajout
                               d[i-1, j-1] + cout)  // substitution
            Si i > 1 Et j > 1 Et a[i] = b[j-1] Et a[i-1] = b[j] Alors
                d[i, j] <- minimum(d[i, j],
                                   d[i-2, j-2] + 1)  // transposition
            FinSi
        FinPour
    FinPour
    Résultat <- d[|a|, |b|]
```

La matrice de distances d est initialisée de la manière suivante :

$$d = \left(\begin{array}{c|ccc} 0 & 1 & \cdots & |b| \\ \hline 1 & & & \\ \vdots & & 0 & \\ |a| & & & \end{array}\right) \tag{1}$$

1.3 Correction d'un mot

L'idée pour corriger un mot est de s'appuyer sur la distance précédente pour proposer les mots du dictionnaire qui sont proches du mot erroné. Une proposition de correction est d'autant plus plausible que sa distance au mot à corriger est petite. On peut envisager plusieurs stratégies.

Une première stratégie pour corriger un mot consiste à chercher dans le dictionnaires les mots les plus proches de celui à corriger. On utilise alors la distance précédente et on ne garde que les mots dont la distance est inférieure à un seuil donné (2 par exemple).

Une deuxième stratégie consiste à produire, à partir du mot erroné, tous les mots qui en sont à une certaines distance. Par exemple, tous les mots qui sont à une distance de 1 sont obtenus en utilisant l'une des opérations élémentaires (suppression, substitution, transposition et ajout [1]). On ne garde alors que les mots ainsi obtenus présents dans le dictionnaire.

1. Pour cette dernière opération, il faut connaître l'alphabet utilisé par le dictionnaire. Une solution pour l'obtenir est de considérer tous les caractères des mots du dictionnaire.

1.4 Correcteur interactif

Le programme `icorrecteur.py` permettra de corriger un texte en s'appuyant sur un dictionnaire. Le programme demandera à l'utilisateur le nom du fichier qui contient le dictionnaire à utiliser, puis le fichier contenant le texte à corriger et un seuil. Le programme proposera alors à l'utilisateur, pour tous les mots du texte qui ne sont pas dans le dictionnaire, les mots du dictionnaires dont la distance au mot à corriger est inférieure au seuil. Les propositions seront classées de la plus plausible (distance faible) à la moins plausible (distance grande). L'utilisateur pourra alors choisir de corriger le mot (il saisira le numéro de la proposition retenue), d'ajouter le mot dans un dictionnaire personnel (il tape « + »), de garder le mot erroné sans l'ajouter dans le dictionnaire (il ne saisit rien) ou de corriger lui-même le mot (il tape le nouveau mot qui sera lui aussi vérifié). Le dictionnaire personnel (s'il existe) sera utilisé en plus du dictionnaire principal. Il s'appelle `perso.dic`. Toutes les corrections faites seront enregistrées dans un fichier `corrections.txt`. Les corrections apparaîtront dans l'ordre, en précisant le numéro de ligne du mot, le mot erroné et la correction éventuellement apportée. Le texte corrigé sera écrit dans le fichier `corrige.txt`.

Voici une interface possible pour ce programme.

```
Dictionnaire principal : frgut.dic
Nom du fichier à corriger : exemple1.txt
Seuil : 3
Le mot 'ortografique' (ligne 1) n'est pas dans les dictionnaires
1. orographique (3)
2. orthographique (3)
'+' ajouter 'ortografique' dans le dictionnaire personnel
Correction (rien pour le garder) : 2
Le mot 'conjuguaison' (ligne 3) n'est pas dans les dictionnaires
1. conjugaison (1)
2. conjugaisons (2)
3. conjuguais (2)
4. conjuguaient (3)
5. conjuguasses (3)
6. conjuguasse (3)
7. conjuguassent (3)
8. conjuguions (3)
9. conjuguait (3)
10. conjuguassions (3)
11. conjuguai (3)
12. conjuguas (3)
'+' ajouter 'conjuguaison' dans le dictionnaire personnel
Correction (rien pour le garder) : 1
```

Le fichier de corrections donnera des statistiques et indiquera comment a été corrigé un mot (en précisant le numéro de ligne sur lequel le mot apparaît). En voici un exemple (« quid » et « l » n'ont pas été corrigés).

```
34 mots trouvés dans le dictionnaire.
7 mots inconnus :
1. ortografique -> orthographique
3. conjuguaison -> conjugaison
3. acords -> accords
5. poncturation -> ponctuation
6. quid
6. l
6. éfficacité -> efficacité
```

1.5 Correcteur automatique

Le programme `acorrecteur.py` corrigera automatiquement le texte contenu dans un fichier (son nom sera le premier argument de la ligne de commande) en écrivant le texte corrigé dans un nouveau fichier (son nom est le deuxième argument) et en utilisant un dictionnaire principal (son nom est le troisième argument) et un dictionnaire personnel (quatrième argument) ainsi qu'un seuil (dernier paramètre). Le fichier `corrections.txt` contiendra toutes les modifications faites (même format que pour `icorrecteur.py`).

Voici un exemple d'utilisation :

```
python acorrecteur.py original.txt corrige.txt dico.txt perso.txt 3
```

Le module `acorrecteur` devra définir une fonction appelée `main` avec les 5 paramètres décrits ci-avant (dans le même ordre). Il est important de respecter ces consignes car des tests automatiques seront réalisés sur votre programme.

1.6 Étude de performance

Une étude de performance du programme `acorrecteur.py` permettra de comparer les deux stratégies envisagées. Cette étude permettra aussi de voir si les choix de structures de données sont judicieux et les algorithmes efficaces.

Si des changements doivent être faits, il faudra penser à conserver les versions précédentes pour toujours avoir une version fonctionnelle.

1.7 Fichiers fournis

Certains fichiers sont fournis sur Moodle. Le fichier LISEZ-MOI.txt indique à quoi correspondent ces différents fichiers.

1.8 Compléments en Python

1.8.1 Obtenir les mots d'un texte

Pour corriger le texte, il faut retrouver les mots qui le composent mais aussi les éléments du texte qui séparent ces mots. Ceci est nécessaire pour reproduire le texte initial avec les mots

erronés remplacés par les corrections choisies. On utilisera les expressions régulières (module `re`) et sa fonction `split`. Voici un exemple :

```
>>> import re
>>> re.split("(\W+)", "Mots, ponctuation, chiffres, 10...")
['Mots', ', ', 'ponctuation', ', ', 'chiffres', ', ', '10', '...', '']
>>> re.split("(\W+)", "- L'âge du capitaine")
['', '- ', 'L', "'", 'âge', ' ', 'du', ' ', 'capitaine']
```

La fonction `split` prend deux paramètres. Le premier est l'expression régulière qui décrit les séparateurs. Le second est la chaîne à découper suivant les séparateurs. Dans les exemples, l'expression régulière est `"(\W+)"`, une chaîne de caractères. `\W` est un motif qui correspond à tout caractère sauf ceux qui constituent les identifiants en Python (lettre, chiffre ou souligné). Le `+` indique que l'on considère une ou plusieurs fois le motif précédent. Les parenthèses demandent à `split` de fournir aussi les séparateurs. On remarque que `split` produit une liste qui contient alternativement un « mot » et un séparateur et commence toujours par un « mot », éventuellement vide comme dans le deuxième exemple.

1.8.2 Mesurer le temps d'exécution

Pour mesurer le temps d'exécution d'une fonction Python, on pourra utiliser la fonction `timeit` du module `timeit`.

1.9 Extensions possibles

Voici quelques extensions possibles. Elles n'ont pas à être traitées.

1.9.1 Définition du seuil

Prendre un seuil fixe est difficile. Trop petit il ne permettra pas de corriger un nom long qui est susceptible de contenir plusieurs erreurs, trop grand il proposera trop de possibilités pour les mots de petite taille. En conséquence, le seuil devrait être fonction du mot.

1.9.2 Majuscules

Un texte contient des majuscules, en particulier en début de phrase. Dans la correction, il faudrait conserver la majuscule sur le mot remplacé.

Se pose aussi le problème des noms propres. On pourrait les ignorer, utiliser un dictionnaire des noms propres, etc.

1.9.3 Taille du dictionnaire sur le disque

Le dictionnaire contient beaucoup de mots mais il y a des familles de mots qui partagent les mêmes suffixes (ou les mêmes préfixes). Il serait possible de réduire la taille dictionnaire sur le

disque en factorisant les préfixes et les suffixes communs. Par exemple, les verbes du premier groupe ont une conjugaison régulière. On pourrait ainsi noter dans le dictionnaire trouv/1, lav/1, march/1, racl/1, vid/1... avec « 1 » qui correspond aux suffixes des verbes du premier groupe : er, e, es, ons, ez, ent, ais, etc.

2 Organisation

Le projet sera réalisé en équipes de 3 étudiants (éventuellement 2) du même groupe de TD (A1+A2 ou B1+B2 ou C1+C2).

3 Livrables

Les **documents à rendre** dans une archive (projet.zip) sont :

1. Un rapport écrit (rapport.pdf) qui doit contenir les éléments suivants :
   - une page de garde avec titre, année, membre de l'équipe, etc.
   - un résumé qui indique le contenu du document
   - une introduction (1/2 page maximum) qui rappelle de manière concise le sujet (le sujet du projet est connu !) et présente le plan du document
   - les principaux niveaux de raffinages
   - l'architecture en modules
   - les principaux choix faits : structures de données et algorithmes
   - les principales difficultés rencontrées (et les solutions trouvées)
   - l'avancement du projet (ce qui a été fait, ce qui reste à faire, ce qui pourrait être amélioré)
   - l'organisation de l'équipe et le travail réalisé par chaque membre. En particulier, on produira une table avec en ligne les fonctions de votre programme et en colonne les activités spécifications, programmation, test, relecture. Dans les cases figurerons les initiales de l'équipier qui a fait ce travail.
   - un bilan personnel de chaque membre de l'équipe. Chaque membre devra indiquer le temps passé sur le projet (y compris les séances prévues à l'emploi du temps).
2. Un support de présentation pour l'oral (presentation.pdf). Il doit reprendre les principaux élements techniques du rapport : architecture en modules, principales structures de données, principaux algorithmes, avancement du projet, difficultés rencontrées.
3. Les sources de votre projet, y compris les programmes de test et les fichiers d'exemples (texte à corriger, dictionnaires, etc.). Le fichier LISEZ-MOI.txt indiquera à quoi correspondent les différents fichiers.

4 Échéances

Voici les **principales dates** du projet :

— lundi 9 mars 2020, 8h-9h30 : distribution du sujet du projet, constitution des équipes de projet, lecture et comprehension du sujet, réponse aux questions

— vendredi 13 mars : remise du rapport avec les parties « raffinages » et « architecture en modules » traitées (version initiale, par forcément complète)

— jeudi 9 avril 2020 : remise des livrables du projet (projet.zip).

— vendredi 10 avril 2020, 14h-18h : présentation orale et recette (démonstration) du projet (20 minutes par équipe)

Pour la **recette**, il faudra préparer une *démonstration* de 5 minutes maximum. L'objectif est de montrer que le programme répond au cahier des charges.

La **présentation orale** durera 5 minutes.

Les 10 minutes restantes seront utilisées pour des **questions** et, éventuellement, des **tests supplémentaires**.

Attention : Un tirage au sort désignera les membres de l'équipe qui feront respectivement la présentation orale, la démonstration et les réponses aux questions.

5 Évaluation

La table 1 présente les critères qui seront utilisés pour noter ce projet et leur poids. Attention, certains critères ne pourront être évalués que si les consignes données dans ce sujet sont respectées.

TABLE 1 – Principaux critères pour l'évaluation du projet

Raffinages	10
Structuration en modules	5
Sous-programmes	10
Conception (choix des structures de données et algorithmes)	15
Qualité du code	10
Tests unitaires et validation	10
Réalisation des fonctionnalités demandées	15
Robustesse du programme	5
Démonstration	5
Présentation orale	5
Réponse aux questions	2
Rapport écrit	8