Devoir 2 : Correcteur orthographique.

Dans ce devoir, nous vous demandons d'implémenter un correcteur orthographique. En utilisant un dictionnaire des mots français, contenant plus de 400.000 entrées, vous devez décider pour chaque mot s'il est dans le dictionnaire, et sinon trouver un mot le plus proche.

Bien sûr, cela représente beaucoup de mots, et le nombre de mots dans le texte à tester est aussi grand. Il nous faut donc des algorithmes assez rapides pour travailler sur autant de données. Dans un premier temps, vous implémenterez un algorithme de calcul de distance entre deux mots. Celui-ci, bien qu'assez rapide sur une paire de mots, ne peut pas en temps raisonnable être utilisé sur toutes les paires d'un mot à tester et d'un mot du dictionnaire. Nous allons donc ajouter une heuristique qui, étant donné un mot mal orthographié, va trouver rapidement une petite liste de mots qui nous semblent être de bons candidats à être la bonne orthographe.

1 Rendu

Nous consacrerons 2 séances pour ce devoir, que **vous réaliserez en Java**. Vous pouvez réutiliser les tables coucous et les arbres préfixes du devoir précédent. Cependant, si vous n'avez pas eu le temps suffisant pour finir leur mise au point, ou simplement si vos implémentations ne sont pas assez efficaces, vous pouvez à la place utiliser les classes de la librairie `java.util` de Java.

Nous vous demandons de rendre votre devoir terminé sur Ametice. L'ensemble du code que vous avez écrit devra être archivé dans un unique fichier, au format `gz` ou `tar.gz` **exclusivement**. Si nécessaire, vous pouvez y joindre un bref rapport expliquant les difficultés que vous avez rencontrées.

Si votre projet contient du code source que vous n'avez pas écrit vous-même, **vous indiquerez précisément sa nature et sa provenance** (internet, les amis, ...). Tout projet contenant du code non-correctement attribué à ses véritables auteurs s'expose à recevoir une note nulle. Vous êtes priés de ne pas fournir votre code à d'autres étudiants, sous peine de vous exposer aussi à recevoir une note nulle.

La date de rendu dépend de votre campus, demandez-la à votre enseignant de TP.

2 Distance de Levenshtein

Pour mesurer la qualité d'une correction orthographique, nous allons utiliser une distance sur l'ensemble des mots. Nous devons donc définir une fonction de distance, qui à toute paire de mots, nous donne une valeur positive, la distance entre ces deux mots. Nous voulons que cette distance soit d'autant plus basseque les mots se ressemblent, du point de vue orthographique. La distance souvent utilisée pour cet objectif est la distance d'édition, ou distance de Levenshtein, entre deux mots.

Une *édition* dans un mot consiste en une opération locale, qui peut être :

— l'insertion d'une lettre supplémentaire,
— la suppression d'une lettre,
— le remplacement d'une lettre par une autre.

La *distance d'édition* entre deux mots est le nombre minimum d'opérations d'édition nécessaires pour transformer le premier mot en le deuxième mot. Par exemple, la distance entre les mots `logarytmique` (incorrect) et `algorithmique` (correct) est de 5 :

1. ajouter un 'a' au début `alogarytmique`,
2. supprimer le 'o' en 3^{e} position `algarytmique`,
3. changer le 'a' en 'o' en 4^{e} position `algorytmique`,
4. changer le 'y' en 'i' en 6^{e} position `algoritmique`,
5. ajouter un 'h' en 8^{e} position `algorithmique`.

Pour calculer cette distance, le principe est de partir à une extrémité des deux mots, par exemple à la fin. Si la dernière lettre est la même dans les deux mots, la distance d'édition est la même qu'entre les deux mots sans leur dernière lettre. Si les deux dernières lettres sont distinctes, il faudra faire soit une opération d'insertion, soit une opération de suppression, soit une opération de remplacement sur la dernière lettre du premier mot. Cela donne trois choix, par récurrence on détermine celui des trois qui donne la plus petite distance. Enfin, on utilise un technique de programmation dynamique pour éviter que la récurrence explose exponentiellement.

Un fois implémenté, n'oubliez pas de tester votre algorithme, et de mesurer son temps d'exécution (utilisez `System.nanoTime()`).

Il est ensuite possible d'ajouter des opérations d'éditions supplémentaires. Pour les plus avancés, vous pourrez ajouter la transposition de deux lettres successives (clea arriev fréquemmnet quadn on

écrit au calvier), la répétition de lettres ou son absence (une vérritable ereur d'orthoggraphe), les erreurs d'accents, ou bien d'autres règles spéciales ("aie" ou "aille", *etc.*). Dans ce cas, on ne veut pas compter le nombre d'opérations, mais attribuer à chaque opération un coût, et minimiser la somme des coûts d'édition.

3 Les trigrammes

Pour trouver la bonne orthographe d'un mot, on pourrait calculer la distance d'édition avec chaque mot du dictionnaire et prendre le plus proche. Sachant que le dictionnaire contient plus de 400.000 mots, calculez le temps que cela va prendre. Même une implémentation très rapide du calcul de distance d'édition prendra quelques dizièmes de seconde, au mieux.

Nous avons donc besoin d'utiliser une heuristique pour restreindre le nombre de mots à tester avec le calcul de distance d'édition. Pour cela, nous allons repérer les mots qui possède au moins un triplet de lettres consécutifs en commun. Par exemple, `logarytmique` et `algorithmique` ont en commun les triplets `miq`, `iqu`, `que`. Un tel triplet de lettres est appelé *trigramme*.

Pour trouver les mots possèdant des trigrammes communs avec le mot à corriger, nous allons associer à chaque trigramme possible, la liste de tous les mots contenant ce trigramme. Plus exactement, pour tout mot du dictionnaire, nous allons ajouter un caractère spécial au début, et un à la fin. Par exemple, pour `algorithmique` cela donne `<algorithmique>`. Ensuite on construit la liste des trigrammes de ce mot allongé : `<al`, `alg`, `lgo`, ..., `que`, `ue>`. Puis à chacun de ces trigrammes, on ajoute dans sa liste associé le mot `algorithmique`.

Une fois construit les associations trigrammes – liste de mots (quelle structure de données utiliser ?), en utilisant tous les mots du dictionnaire, on peut commencer la correction orthographique. On procède ainsi pour chaque mot.

1. on teste si le mot est dans le dictionnaire (quelle structure de données utiliser ?). Dans ce cas son orthographe est correcte et on s'arrête. Sinon :
2. on construit la liste de ces trigrammes,
3. pour chaque trigramme, on récupère la liste des mots contenant ce trigramme,
4. on trie ces mots, et on garde ceux avec le plus de trigrammes en commun avec le mot à corriger (on peut en garder une centaine).
5. on calcule la distance d'édition de chacun avec le mot à corriger, on garde les meilleurs, triés dans l'ordre de distance d'édition. Le premier est notre proposition, mais peut-être la bonne solution n'est placée que 2^e ou 3^e, donc on donne une liste des meilleurs choix à l'utilisateur pour qu'il choisisse celle qu'il préfère.

4 Suppléments

Vous réutiliserez le dictionnaire donné au premier TP pour le correcteur orthographique. En plus, nous vous fournissons une liste de mots mal-orthographiés ; il s'agit d'une liste de fautes courantes en français. Un autre fichier donne les corrections pour chacun de ces mots, dans le même ordre, ce qui vous permet de vérifier la justesse de votre correcteur, et éventuellement de repérer ses failles pour pouvoir l'améliorer. Ce deuxième fichier peut aussi vous servir de petit dictionnaire pour vos tests.

Vous mesurerez le temps de calcul pour corriger toutes les fautes d'orthographes. Pensez-vous que votre algorithme est suffisamment rapide et efficace pour être utile ? Êtes-vous capable de l'améliorer, tant pour sa célérité que pour la pertinence de ses propositions ?