Aix*Marseille université

Domaine Sciences et Technologies
L3 Info, parcours Maths-info

Algo. avancée et th. des graphes : TP 2
Code UE : S04IN5B2

Année 2020-21

Correcteur orthographique

Dans ce devoir, vous allez implémenter un correcteur orthographique. A partir d'un dictionnaire des mots français, votre correcteur devra déterminer si un mot donné est dans le dictionnaire, et s'il n'y est pas proposer une sélection de mots qui lui sont très proches. Du fait de la taille des données (le dictionnaire contient plus de 400.000 entrées et on peut avoir à tester un grand nombre de mots), l'algorithme de correction devra être très efficace.

Dans un premier temps, vous implémenterez un algorithme qui calcule la distance d'édition entre deux mots. Cette distance caractérise leur *proximité orthographique*. Ce calcul, bien que assez rapide, ne peut pas en temps raisonnable être utilisé pour trouver les mots les plus proches d'un mot donné M en le comparant avec tous les mots du dictionnaire. Il faut donc, dans un premier temps, sélectionner les mots du dictionnaire susceptibles d'être proches de M. On se limitera ensuite au calcul des distances sur les mots de la sélection. Pour faire la sélection nous vous proposons une heuristique simple basée sur les *trigrammes* que contiennent les mots.

1 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 basse que 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.

2 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 que les associations trigrammes – liste de mots sont construites (quelle structure de données utiliser ?), la correction orthographique d'un mot donné M se fait de la façon suivante :

1. tester si M est dans le dictionnaire (quelle structure de données utiliser ?). Si c'est le cas M est correctement orthographié et le processus s'arrête. Sinon :
2. construire la liste des trigrammes du mot M,
3. sélectionner les mots du dictionnaire qui ont le plus de trigrammes communs avec M (on pourra se limiter à une centaine de mots par exemple),
4. déterminer les cinq mots de la sélection les plus proches de M au sens de la distance d'édition. L'utilisateur choisira parmis ces 5 mots celui qui lui convient.

3 Ce que vous devez faire :

1. construire le dictionnaire à partir des mots du fichier `dico.txt`,
2. implémenter le calcul de la distance entre deux mots,
3. implémenter la phase de sélection basée sur les trigrammes communs,
4. évaluer le temps de calcul nécessaire pour corriger tous les mots du fichier `fautes.txt`.

Pour ce TP on vous demande de bien réfléchir en avance à la structure du projet. En particulier, on vous donne dans le répertoire `src` du `git` de classes abstraites/interfaces que l'on vous demande d'implémenter. Chaque déviation de la structure du projet sera amplement justifiée dans le fichier README.

Choisissez avec soin les structures de données que vous allez utiliser pour stocker le dictionnaire et les associations trigrammes/mots du dictionnaire. Vous pouvez par exemple vous servir des arbres de préfixe (vu en L2) et de tables de hachage. Pour les structures plus élémentaires (e.g. tables de hachage ou dictionnaires), vous pouvez vous servir de classes fournies avec le langage `java` ou utiliser du code disponible librement. Dans ce dernier cas, indiquez clairement la source du code dans le fichier README.

Dans un deuxième temps (début de novembre) vous pourrez aussi donner des implémentations des tables de hachage plus performantes, comme sera expliqué en cours. Si cette implémentations est présente, le projet sera mieux évalué.

Vous avez environ 4 semaines pour rendre le TP (voir date limite du rendu sur AMETICE).