TP4 et TP6: Projet de correction orthographique

Objectifs L'objectif de ce projet est de programmer une application de détection de fautes d'orthographe dans un texte, vis à vis d'un dictionnaire donné. Les mots mal orthographiés sont ceux qui n'appartiennent pas au dictionnaire. Ce projet comporte trois niveaux. Vous devez terminer le niveau 1 avant de traiter le niveau 2 et ainsi de suite.

Niveau 1 Recherche simple (non dichotomique) des mots du texte dans le dictionnaire.

Niveau 2 Tri du dictionnaire et recherche dichotomique des mots du texte dans le dictionnaire.

Niveau 3 Proposition de correction pour les mots mal orthographiés.

Entre le TP4 et le TP6, le TP5 portera sur des algorithmes de tri que vous pourrez utiliser au niveau 2.

Notation Ce projet sera évalué par un **entretien** pendant la séance de TP6 et à l'aide du **compte rendu de TP** à rendre sur Moodle (projet Eclipse et rapport PDF). Les critères d'évaluation de votre projet Eclipse seront l'efficacité et la clarté **pour le code**, la pertinence et la couverture **pour les tests**. Dans le rapport PDF (1 page maximum), indiquez précisément les identifiants des fonctions Java auxquelles vous faites référence et répondez aux questions suivantes :

Question 1 (Niveau 1) Pour un dictionnaire de taille N et un texte à corriger de taille T, quelle est la complexité au pire cas de votre fonction `corriger` ? Justifiez.

Question 2 (Niveau 2) Pour un tableau de taille N, quelle est la complexité au pire cas de votre fonction de tri ? Justifiez.

Question 3 (Niveau 2) Pour un dictionnaire de taille N et un texte à corriger de taille T, quelle est la complexité au pire cas de votre fonction `corrigerDicoRapide` ? Justifiez.

1 Correction orthographique, niveau 1

Procédez comme pour le TP 1 pour **importer** le projet : `/share/l1ie/SI1/ProjetCorrection/ProjetCorrection.zip`

1.1 Fonction de correction d'un texte

Dans le fichier `Main.java` du package `main` du projet Eclipse, définissez une fonction de correction nommée `corriger`. Étant donnés un tableau de mots `texte` (un tableau de `String`) et un dictionnaire (un autre tableau de `String`), `corriger` rend un tableau de booléens de même longueur que `texte`. Dans le tableau de booléens, à l'indice `i`, le booléen est `true` si et seulement si `texte[i]` est un mot figurant dans le dictionnaire. Sa signature sera :

```
public static boolean[] corriger(String[] texte, String[] dico)
```

La fonction `corriger` ne doit pas modifier `texte` et `dico`. Pour programmer cette fonction, vous **utiliserez une fonction de recherche de chaînes** programmée au TP3. Au niveau 1, on demande une recherche simple, non dichotomique.

1.2 Tests unitaires de `corriger` avec JUnit

Dans le répertoire `testsJUnit`, vous trouverez un fichier `CorrecteurTest.java` dans lequel vous devez écrire vos tests. A vous de proposer des tests simples et pertinents. Pour les tests unitaires, utilisez des textes et dictionnaires de petite taille que vous maîtrisez.

1.3 Intégration dans l'interface graphique

Pour l'application de correction orthographique, nous allons utiliser un dictionnaire réaliste. Dans le répertoire `lib` du projet Eclipse, dans le fichier `dico.txt`, vous trouverez le même lexique du français courant qu'au TP3. On rappelle que vous avez accès à la fonction : `String[] ACX.lectureDico(String nomFichier)`. L'instruction `String[] dico= ACX.lectureDico("lib/dico.txt");` permet de créer un tableau de chaînes nommé `dico` contenant les mots du lexique `"lib/dico.txt"`. Définissez la fonction `corrigerDico` dont voici la signature :

```
public static boolean[] corrigerDico(String[] texte)
```

Notez que le dictionnaire utilisé ne doit pas être passé en paramètre de la fonction `corrigerDico`. Cette fonction chargera le dictionnaire `"lib/dico.txt"` dans la variable `dico` et appellera `corriger` avec `texte` et `dico`. On vous fournit une interface graphique pour l'application de correction orthographique. Pour la lancer, dans la fonction `main` du fichier `Main.java` faites un appel à la fonction `ACX.interfaceCorrection`. Par exemple, si le nom de votre fonction de correction est `corrigerDico` faites l'appel suivant :

```
ACX.interfaceCorrection("corrigerDico")
```

Au lancement, une fenêtre apparaît. Vous pouvez entrer du texte au clavier dans cette fenêtre, le texte sera automatiquement analysé avec la fonction de correction dont le nom est passé en paramètre. Les mots détectés comme mal orthographiés apparaissent en rouge, et soulignés. Pensez à utiliser le dictionnaire complet dans la fonction `main`.

1.4 Une première optimisation de la fonction corrigerDico

Si vous lisez le dictionnaire (avec `lectureDico`) à chaque appel de la fonction `corrigerDico`, vous remarquerez que les performances de l'application sont mauvaises. L'idéal est de lire *une seule fois* le dictionnaire et de stocker le tableau correspondant dans une variable globale (elle devra être définie avec `static`), puis d'utiliser le tableau stocké dans cette variable globale dans `corrigerDico`.

1.5 Test de la correction orthographique de niveau 1

Pour tester la correction orthographique, vous pouvez copier-coller dans l'interface graphique le contenu du fichier `germinalExtrait.txt`. Si votre fonction `corrigerDico` fonctionne correctement, les seuls mots soulignés en rouge doivent être, à de très rares exceptions (les mots "tandis", "jusqu", "ci", "parce"), des noms communs et des prénoms. Certains mots figurent bien dans le dictionnaire mais sont écrits avec une majuscule. Pour obtenir une chaîne équivalente à une chaîne `s` mais sans majuscules, vous pouvez utiliser `s.toLowerCase()`. **Attention :** l'interface graphique demande à ce que `corrigerDico` et `corriger` ne modifient pas leurs tableaux en paramètre.

Remarque : vous pouvez remarquer qu'une fois que vous avez copié l'extrait de Germinal, si vous tentez de taper un autre mot dans l'interface graphique, celle-ci est considérablement ralentie. Pour gagner en performance, il faut une recherche de meilleure complexité. Il est temps de passer au niveau 2.

2 Correction orthographique, niveau 2

L'objectif de cette partie est de définir deux autres fonctions `corrigerRapide` et `corrigerDicoRapide` qui utiliseront la recherche dichotomique pour chercher dans le dictionnaire. **Attention**, le lexique `dico.txt` est imparfaitement trié. Pour pouvoir utiliser la recherche dichotomique sur le tableau obtenu à partir de `dico.txt`, vous devez donc le trier au préalable. La signature de ces fonctions sera :

```
public static boolean[] corrigerRapide(String[] texte, String[] dico)
public static boolean[] corrigerDicoRapide(String[] texte)
```

Procédez comme pour la fonction `corriger` pour

- faire des tests unitaires de `corrigerRapide` avec JUnit
- intégrer `corrigerDicoRapide` dans l'interface graphique fournie
- tester l'efficacité de votre solution sur de gros textes

Remarque : suivant le type d'algorithme de tri utilisé, le tri du dictionnaire peut prendre de quelques secondes à plusieurs minutes. Si le tri du dictionnaire est long, vous avez la possibilité d'enregistrer dans un fichier le dictionnaire trié à l'aide de la fonction `ACX.ecritureFichierString(String[] t, String nomFichier)`.

3 Correction orthographique, niveau 3

L'objectif de cette partie est de définir une nouvelle fonction `proposerCorrection` qui pour un mot m donné donnera un tableau de mots **de même longueur** que m, **figurant dans le dictionnaire**, et ne différant de m que par une lettre exactement. Par exemple si on appelle cette fonction avec le mot "rateau", on obtiendra un tableau comportant les mots "**b**ateau", "ra**d**eau", "ra**m**eau", "r**â**teau" qui diffèrent tous de "rateau" par une seule lettre. En revanche le mot "**ca**d**e**au" ne figurera pas dans ce tableau car il diffère de "rateau" par 2 lettres. Il est conseillé de programmer d'abord une fonction intermédiaire calculant le nombre de différences entre deux mots de même longueur (distance de Hamming). Remarque : pour obtenir le caractère à l'indice `i` dans une chaîne `s`, on peut utiliser `s.charAt(i)`. La signature de cette fonction sera : `public static String[] proposerCorrection(String mot)`

Idéalement, la taille du tableau résultat varie avec le nombre de propositions de correction possibles. Procédez comme pour les fonctions précédentes : tester convenablement **toutes** vos nouvelles fonctions avec JUnit, et intégrer `proposerCorrection` dans l'interface graphique fournie à l'aide de la fonction suivante :

`ACX.interfaceCorrection("corrigerDicoRapide","proposerCorrection")`

Dans cette nouvelle interface, si vous cliquez (clic gauche) sur un mot souligné en rouge puis que vous cliquez à nouveau (clic droit), un menu s'ouvrira avec toutes les propositions de corrections fournies par votre fonction `proposerCorrection`.