Anagrammes des mots du dictionnaire

Votre projet est à rendre avant le lundi 8 janvier 2007 minuit, par courier électronique à `Luc.Maranget@inria.fr`. Trop tard ! Le corrigé.

Objectif

Écrire un programme `Ana` qui trouve toutes les anagrames présentes dans un dictionnaire.

Plus précisément, si le dictionnaire `petit.txt` est constitué des mots `apéros`, `broutera`, `chenil`, `lichen`, `narval`, `obturera`, `opéras`, `outre`, `paréos`, `raboteur`, `rabouter`, et `raton-laveur`, alors la commande « `java Ana petit.txt` » produit l'affichage suivant :

```
rabouter raboteur obturera broutera
lichen chenil
paréos opéras apéros
```

On observe que chaque ligne regroupe les anagrammes, et que les mots isolés (ici `outre`, `narval` et `raton-laveur`) ont disparu.

Ce qui est demandé

Le défi est de traiter des dictionnaires comportant jusqu'à cent mille mots. Bien que cela puisse paraître difficile à première vue, c'est tout à fait faisable (et c'est aussi assez classique).

La suite décrit deux techniques possibles de production des anagrammes. Il est demandé d'implémenter au moins une des deux techniques, voire les deux si vous en avez le courage.

Dans tous les cas, il faut tester votre programme sur les dictionnaires fournis plus bas.

- Pour vérifier que votre programme trouve bien toutes les anagrammes.
- Pour établir un graphique de performance et estimer la complexité en pratique.

Il est inutile de rédiger un rapport, je suis prêt à me contenter du graphique légèrement commenté.

Signatures

À chaque mot, on associe une signature qui est telle que deux mots ont la même signature si et seulement si ils sonts anagrammes l'un de l'autre (ou encore un mot s'obtient par permutation des lettres de l'autre).

Un telle signature s'obtient en triant les lettres des mots. Par exemple, `lichen` et `chenil` sont des anagrammes parce que leur signature `cehiln` est la même.

Pour fabriquer les signatures le plus simple est de produire un tableau de **char** (méthode **char** `[] toCharArray()` de la classe `String`), de trier le tableau (à vous de jouer !), puis de fabriquer une chaîne avec le tableau (constructeur `String (`**char** `[] value)`)).

Trouver les anagrammes, en triant

Trouver les anagrammes revient donc à regrouper les mots qui ont la même signature. On y arrive en triant et en trois temps.

1. Fabriquer une liste de paires (mot × signature).
2. Trier cette liste, selon les signatures, on compare deux `String` par s_1`.compareTo(`s_2`)` qui renvoie -1 si s_1 est plus petit que s_2, 0 si s_1 est égal à s_2, et 1 si s_1 est plus grand que s_2 (cf. la méthode `compareTo`).
3. Maintenant, les anagrammes se suivent dans la liste triée, et on peut les afficher en un parcours de liste.

Avec une table de hachage

On se donne une table de hachage dont les clés sont les signatures (donc des `String`) et les valeurs des listes de mots (donc des listes de `String`). Tous les mots d'une liste donnée ont la même signature et la table associe la liste à cette signature commune.

L'algorithme procède en deux temps.

1. Pour chaque mot du dictionnaire, calculer sa signature et ranger le mot dans la bonne liste.
2. Parcourir toutes les listes de la table et afficher les listes de taille supérieure ou égale à deux.

Détails techniques

Interface

Votre programme doit prendre le nom du fichier-dictionnaire en argument et afficher les anagrammes. Rassurez vous, tout le code nécessaire pour lire les fichiers dictionnaires est donné ci-dessous.

Un exemple d'exécution est donc

```
% java Ana /usr/share/dict/words
cleansing cleanings
ushers rushes
smitten mittens
racing caring arcing
diuretics crudities
breaks brakes bakers
...
```

L'ordre de présentation des anagrammes n'est pas spécifié et votre programme peut très bien avoir un affichage différent. Pour savoir si votre programme fonctionne comme le mien, vous pouvez compter les lignes et les mots du fichier produit ainsi :

```
% java Ana ./petit.txt | wc
      3       9      71
% java Ana ./moyen.txt | wc
    448    1000    9090
% java Ana ./francais.txt | wc
   8378   18515  168057
```

Le tuyau (*pipe*, noté « `|` ») connecte la sortie standard d'une commande Unix avec l'entrée standard de la suivante. La commande `wc` compte les lignes, les mots et les caractères d'un fichier.

Des exemples de dictionnaires

Voici quelques dictionnaires.

1. Le petit dictionnaire `petit.txt`.
2. Un dictionnaire de mille mots `moyen.txt`.
3. Il y a aussi un dictionnaire anglais sur vos machines en `/usr/share/dict/words` (45427 mots normalement). Notez qu'il est inutile de copier ce dictionnaire dans votre répertoire. Il suffit de faire :

   ```
   % java Ana /usr/share/dict/words
   ```

4. Un dictionnaire français plutot complet, car il comprend plus de cent mille mots. Ce dictionnaire `francais.txt` est donné sous forme compressé `francais.txt.gz`, de sorte que, une fois récupéré `francais.txt.gz`, on produit `francais.txt` par la commande « `gunzip francais.txt.gz` ».

Lecture des dictionnaires

Les dictionnaires sont des fichiers dont chaque ligne est un mot. Je fournis le code d'une classe des lecteurs de mots (`WordReader`) suffisante pour lire ce genre de dictionnaires.

Cette classe est conforme au modèle décrit lors de l'amphi 04. Par ailleurs, la classe elle-même comporte une méthode `main` qui donne un exemple d'utilisation (recopier le dictionnaire dans la sortie standard).

```
public static void main(String [] arg) {
  /* Fabrication du lecteur des mots (des lignes en fait) */
  WordReader in = null ;
  if (arg.length > 0) {
    in = new WordReader (arg[0]) ;
  } else {
    in = new WordReader (new java.io.InputStreamReader (System.in)) ;
  }
  /* Lecture de tous les mots de l'entrée */
  String w = in.read() ;
  while (w != null) {
    System.out.println(w) ;
    w = in.read() ;
  }
  /* Fermer l'entrée (c'est plus propre, ça libère des ressources) */
  in.close() ;
}
```