tp_l8dn003_21_02_anag

April 18, 2021

0.1 TP n°2 : recherche des anagrammes

0.1.1 Première partie: mise en place

1. On part d'un fichier texte qui est chargé en mémoire sous la forme d'une liste de lignes.
2. A partir de ce fichier, il est d'abord demandé de construire un *lexique* : la liste de tous les mots apparaissant dans le fichier. On peut utiliser la fonction `split()` ou essayer d'améliorer le découpage.
3. Ecrire une fonction booléenne `anagramme(x,y)` qui détermine si les deux mots x et y sont des anagrammes l'un de l'autre. On peut utiliser la combinaison de fonctions `sorted(list(x))`.
4. Au moyen d'un double parcours du lexique, afficher toutes les paires de mots qui sont des anagrammes.
5. Construire un dictionnaire indiquant, pour chaque mot du lexique, la liste de ses anagrammes

0.1.2 Deuxième partie: amélioration

1. Le lexique a été obtenu par un brutal `split()`. En utilisant d'autres ressources (tokenizers existants ou simples regex) on peut beaucoup améliorer la qualité des résultats obtenus dans la première partie.
2. La plupart du temps, les anagrammes sont définies sans tenir compte des diacritiques (par exemple *grenades* est considérée comme une anagramme de *déranges*). Mettre en place une stratégie pour générer les anagrammes correspondant à cette définition.
3. Il arrive que l'on admette aussi la présence d'espace (ou d'autres signes comme les apostrophes). Par exemple on peut associer *Le commandant Cousteau* à *Tout commença dans l'eau* (dans cet exemple, il faut non seulement ignorer les espaces et l'apostrophe, mais aussi la cédille, et les majuscules). Pour permettre la recherche de ces anagrammes plus étendues, on fait quelques hypothèses:
   - les anagrammes impliquent des suites de mots, mais les mots doivent être contigus et entièrement pris en compte (pas de portions de mots)
   - le nombre de mots considérés est limité à 4.