Anagrammes

Luc Maranget*

Important, il existe une *page de suivi*[1].

Présentation

Comme vous le savez certainement, un anagramme traditionnel est une permutation d'un mot. Ainsi, `orant` est un anagramme de `raton`. La production de tous les anagrammes d'un mot donné est un exercice très classique et de nombreuse solutions sont possibles.

Mais ici on s'intéresse à la question généralisée de produire des anagrammes composés de plusieurs mots. C'est à dire que, étant donné un stock de lettres (qui peuvent être répétées) on veut produire une suite de mots formés exactement à l'aide de ces lettres. Par exemple, à partir des lettres de `ratonlaveur` on peut produire, entre autres, les « *anagrammes* » `loutre varan`, `outre narval` et `rat ou verlan`. Les mots pouvant entrer dans la composition des anagrammes sont donnés par un dictionnaire. Dans l'exemple du raton-laveur, il s'agit d'un dictionnaire français.

Le but de ce projet est l'écriture d'un programme `Anagramme` qui prend en arguments un nom de fichier-dictionnaire et un stock de lettres et affiche *tous* les multi-ensembles de mots du dictionnaire épuisant le stock de lettres. Par multi-ensemble on entend qu'un mot donné peut apparaître plus d'une fois dans un anagramme et que l'ordre des mots n'est pas significatif. Ainsi, si le dictionnaire est composé des mots `cc`, `coco`, `cocu`, `cou`, `coucou`, `oc` et `ou`, alors les anagrammes de `coucou` sont :

`cc ou ou`, `cocu ou`, `cou cou`, `coucou`

On constate que le premier anagramme comprend deux fois le mot `ou` et qu'aucun anagramme n'est répété (afficher l'anagramme supplémentaire `ou ou cc` serait une erreur).

Représentation des dictionnaires

Le dictionnaire est fourni sous la forme d'un fichier texte, à raison d'un mot par ligne. Mais en machine, on le représentera par un arbre.

Commençons par une description un peu abstraite de cet arbre. Chaque nœud de l'arbre contient un booléen et possède un nombre arbitraire de fils étiquetés par des caractères. Les mots du dictionnaire sont représentés par des chemins dans l'arbre aboutissant en un nœud dont le booléen vaut *true*. Par exemple, soit le petit dictionnaire suivant

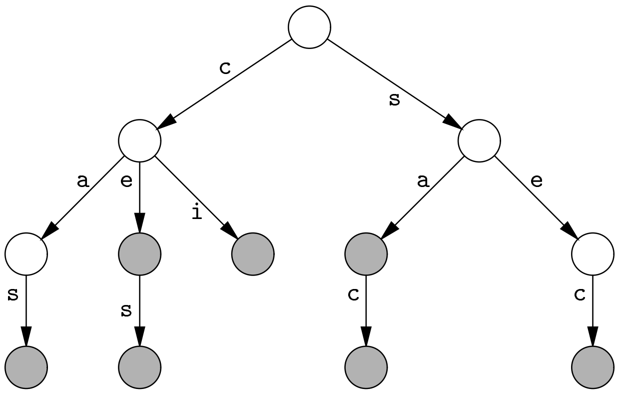
`cas`, `ce`, `ces`, `ci`, `sa`, `sac`, `sec`

Alors on aura l'arbre suivant (un champ booléen égal à *true* est représenté par un cercle grisé) :

---

*`Luc.Maranget@inria.fr`

[1] `http://www.enseignement.polytechnique.fr/profs/informatique/Luc.Maranget/IF/projet/suivi.html`



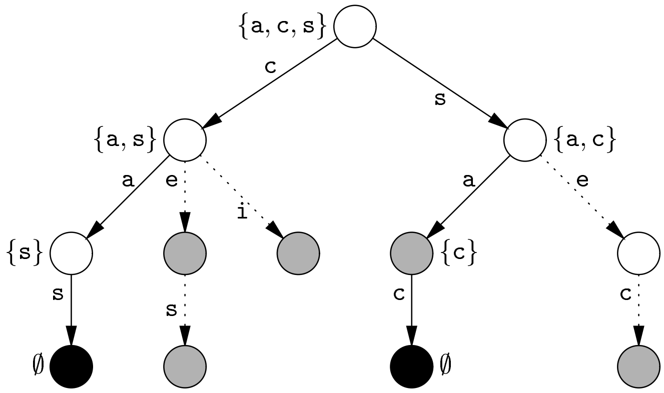
La représentation en arbre permet un partage maximal des préfixes des mots du dictionnaire. L'exemple de `ce` et de `ces`, montre bien la nécessité du champ booléen pour tous les nœuds et pas seulement pour les feuilles.

Ce codage des dictionnaire est très flexible, dans le sens qu'il permet la réalisation efficace de nombreuses opérations. Ainsi, on vérifie très simplement si un mot donné appartient au dictionnaire en suivant le chemin défini par les lettres du mot. Il est aussi très facile de lister les mots du dictionnaire dans l'ordre, ou encore de lister tous les mots compris entre deux mots.

En pratique, la réalisation des arbres n-aires doit être effectuée selon un compromis adapté entre la taille de la structure et l'efficacité des opérations. Je vois au moins quatre façons de réaliser effectivement de tels arbres, à l'aide d'un tableau de fils de taille le nombre de lettres, à l'aide de listes d'associations, comme des arbres binaires [2], et à l'aide de *ternary search trees* [1]. C'est à vous de faire un choix.

Anagrammes d'un mot

On peut maintenant trouver tous les anagrammes de, par exemple, `sac`, en un parcours incomplet de l'arbre.



Sur ce dessin, les arcs non-parcourus sont pointillés, chaque nœud parcouru est décoré par le stock de lettres pertinent à chaque étape, et les nœuds correspondant à des anagrammes sont en noir (on trouve les anagrammes eux-mêmes en « lisant » les chemins qui mènent aux nœuds noirs).

Étant donné un nœud et un stock de lettres, le parcours ne suit que les arcs dont l'étiquette appartient au stock. Ce caractère est alors consommé comme il apparaît dans le stock du nœud cible. Les nœuds noirs sont alors les anciens nœuds gris avec un stock vide.

Anagrammes formés de plusieurs mots

Il est conceptuellement facile de modifier le parcours décrit précédemment pour qu'il trouve les anagrammes partiels (n'utilisant qu'une partie du stock de lettres). Il suffit de noircir tous les nœuds gris vus lors du parcours. En répétant ce procédé on devrait réaliser le projet.

Notons quand même que la contrainte de ne pas afficher deux fois le même multi-ensemble de mots complique un peu les choses. Pour ce faire, je vois deux idées (il peut y en avoir d'autres). On peut soit trouver les anagrammes dans l'ordre croissant des mots qui les composent, ou encore supprimer un mot du dictionnaire quand tous les anagrammes qui le contiennent ont étés trouvés. Ici encore il faut choisir.

Prolongement

On considère un problème un peu différent : sélectionner, parmi les anagrammes traditionnels d'un mot, ceux qui peuvent se prononcer facilement, ou encore qui sont des mots qui auraient pu exister. Par exemple l'état civil de Marguerite *Yourcenar* est Marguerite de *Crayencour(t)*. Marguerite Yourcenar a donc (à une lettre près) choisi un anagramme de son nom de famille comme pseudonyme.

Il est demandé un programme qui produit ce style d'anagrammes. L'idée est cette fois de choisir un entier w ni trop grand ni trop petit et de décréter qu'un anagramme est *vraisemblable* quand tous ses sous-mots de longueur w sont des sous-mots de mots d'un dictionnaire donné.

Ainsi par exemple, en choisissant $w = 4$, on peut dire que `yourcenar` ressemble bien à un mot français, puisque l'on a les sous-mots suivants de mots français. `yourte`, `sourcil`, `pourceau`, `percent`, `mercenaire`, et `renard`.

On peut renforcer cette conviction de vraisemblance en considérant le préfixe et le suffixe de trois lettres `you` et `nar`. En effet, ils sont bien respectivement un préfixe et un suffixe de mots français, (`youyou` et `dinar`). Cela revient simplement à se donner deux lettres supplémentaires signifiant début et fin de mot. En outre, on peut, en considérant le nombre d'apparitions d'un sous-mot donné dans le dictionnaire français, produire un indice de vraisemblance permettant de présenter les anagrammes en commençant par les plus raisonnables et finissant par les plus étranges.

Références

[1] Jon Bentley and Bob Segdgewick, "*Fast Algorithms for Sorting and Searching*", Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, Juanary 1997. `http://www.cs.princeton.edu/~rs/strings/`.

[2] Paul Zimmerman, "*Epelle, un logiciel de détection de fautes d'orthographe*", rapport de recherche 2030 de l'Inria Rocquencourt, septembre 1993.