Plus long complément

On se propose ici de résoudre le problème suivant : étant donnés un ensemble de mots $M = \{m_1, m_2, \ldots, m_n\}$ et un mot p, existe-t-il des mots dans M dont p soit un préfixe, et le cas échéant quel est le plus grand préfixe commun à tous ces mots?

Ce problème a des applications pratiques : certains environnements informatiques proposent ce que l'on appelle en anglais "*the filename completion*", c'est-à-dire la possibilité de compléter les noms de fichiers autant que possible tant qu'il n'y a pas d'ambiguïté. Si par exemple, j'ai dans le répertoire courant les trois fichiers

`recursivite.ml` `recurrence.ml` `arbres.ml`

alors la chaîne de caractères `re` peut être complété en `recur` (qui est le plus grand préfixe commun à tous les noms de fichiers commençant par `re`), `recurs` complété directement en `recursivite.ml` et `a` en `arbres.ml`, par exemple.

1 Solution naïve

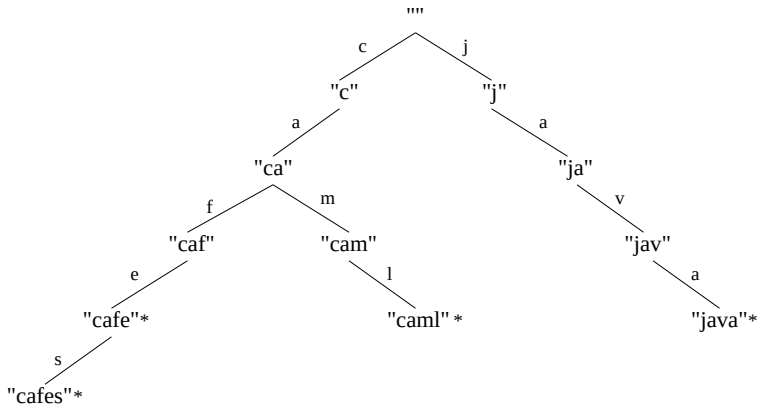
Imaginez une solution naïve (mais sans la réaliser) et évaluez sa complexité en fonction du nombre de mots n et de la longueur maximale L des mots.

2 Une solution efficace

On propose ici une solution plus efficace consistant à construire un arbre contenant les mots de M tel que la recherche se fera ensuite en temps L.

L'idée est que les lettres du mot p doivent permettre d'éliminer successivement les mots dont p n'est pas un préfixe, pour ne garder que les autres. Pour cela, on construit à partir des mots de M un arbre dont les branches sont étiquetées avec des lettres et les nœuds avec des mots, qui sont soit des préfixes de mots de M, soit des mots de M.

Exemple : prenons $n = 4$ et $M = \{caml, cafe, cafes, java\}$. Alors l'arbre est le suivant :



où l'on a marqué les mots de M par une astérisque.

2.1 Préliminaires

• Ecrire deux fonctions transformant les chaînes de caractères en listes de caractères et réciproquement :

```
char_list_of_string : string -> char list
string_of_char_list : char list -> string
```

On prendra pour le type des arbres le type CAML défini ci-dessous. Les nœuds de l'arbre sont étiquettés soit par un préfixe d'un mot de M, soit par un mot de M. Les différentes branches partant d'un nœud sont représentées par une liste d'association.

```
type etiquette = Mot    of string
               | Prefixe of string ;;

type arbre = Noeud of etiquette * (char * arbre) list ;;
```

De manière générale, une liste d'association est une liste de type `('a * 'b) list`, associant des objets de type `'b` à des objets de type `'a`. L'utilisation des listes d'association en CAML est fréquente et il existe dans la bibliothèque standard les fonctions suivantes : `(mem_assoc` a l`)` teste si l'object a est dans le domaine de l, et `(assoc` a l`)` renvoie l'objet associé à a dans l (lève une exception si a n'est pas dans le domaine de l).

```
mem_assoc : 'a -> ('a * 'b) list -> bool
assoc     : 'a -> ('a * 'b) list -> 'b
```

• Ecrire une fonction `change_assoc` modifiant une entrée dans une liste d'association :

```
change_assoc : ('a * 'b) list -> 'a -> 'b -> ('a * 'b) list
```

2.2 Construction de l'arbre

On va construire l'arbre correspondant à l'ensemble de mots M en partant d'un arbre initialement vide (`Noeud (Prefixe "",[])`) et en y insérant les mots de M un à un.

• Ecrire une fonction `insere_mot : arbre -> string -> arbre` qui insére un mot dans un arbre. (Il est conseillé d'écrire une fonction prenant un arbre et la liste des caractères du mot, puis de l'appeler en utilisant `char_list_of_string`).

• En déduire une fonction `construit_arbre` prenant une liste de mots (`string list`) et construisant l'arbre associé. La tester sur l'exemple donné ci-dessus.

2.3 Recherche du plus grand complément

• Ecrire une fonction `complete : arbre -> string -> string` recherchant le plus complément possible d'un mot p, c'est-à-dire le plus grand préfixe commun à tous les mots de M dont p est un préfixe.

Quelle est la complexité de cette recherche, en fonction de n et de la longueur maximale L des mots de M ? Comparer avec la méthode naïve.

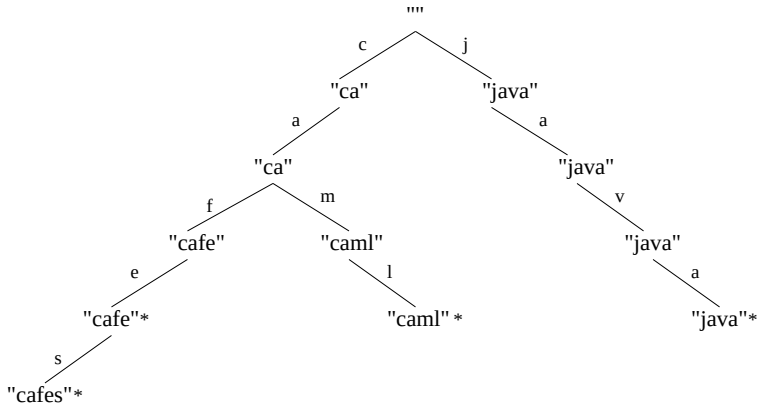
2.4 Recherche de tous les compléments possibles

• Ecrire une fonction `trouve_complements : arbre -> string -> string list` qui renvoie, pour un mot p, tous les mots de M dont p est un préfixe.

3 Optimisation

On peut améliorer la recherche du plus grand complément de la façon suivante : au lieu de stocker sur les nœuds de l'arbre le mot correspondant au chemin qui va de la racine au nœud, on y stocke directement le plus grand complément du mot correspondant au chemin depuis la racine.

Exemple : pour le même ensemble $M = \{caml, cafe, cafes, java\}$ on a maintenant l'arbre suivant :



• Ecrire une fonction `optimise : arbre -> arbre` prenant un arbre (construit avec la première méthode) et rendant un arbre *optimisé*.

• Ecrire une fonction `complete_opt` effectuant la recherche du plus grand complément dans un arbre optimisé. Quelle est maintenant la complexité de la recherche? Est-elle optimale?

S'il vous reste du temps...

• Ecrire une fonction `insere_opt` permettant d'insérer un mot dans un arbre optimisé. Quelle est maintenant la complexité de la construction de l'arbre?

Pour les curieux...

Les implantations pratiques de telles méthodes se font d'une manière similaire, mais utilisent parfois des arbres équilibrés (AVL) pour stocker les mots.