Option Informatique

TP $n°04$ – Tries (arbres lexicographiques préfixes)

Pour pouvoir copier coller : énoncé disponible à l'adresse http://mpsi.daudet.free.fr/info/TD/TD04.pdf.

Les *tries* (ou *arbres binaires préfixes*, ou *arbres lexicographiques*) sont une structure arborescente permettant de stocker un ensemble de mots.

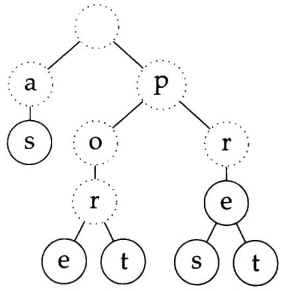
On définit le type `trie` comme suit :

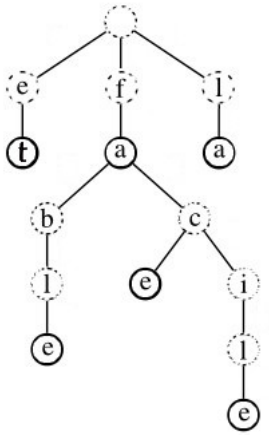
```
type trie = Noeud of (bool * char * trie) list;;
```

La racine n'a pas d'étiquette, mais a une liste d'enfants, éventuellement vide. Les autres nœuds sont étiquetés par un caractère et un booléen. On peut lire des mots (des suites de caractères) en se plaçant à la racine et en descendant dans l'arbre en lisant systématiquement le caractère qui étiquette chaque nœud exploré. Les booléens qui étiquettent les nœuds indique si le mot qu'on a lu en descendant de la racine à cette étiquette appartient, ou pas, à l'ensemble de mot que l'on souhaite représenter par la trie.

De plus, on dira qu'une trie est *valide* lorsque pour tous ses nœuds, la liste de ses fils est triée par ordre croissant des caractères étiquettes et ne comporte pas de répétition.

Exemples : À gauche, une trie valide permettant de stocker l'ensemble de mots {as, port, pore, pré, près, prêt} (sans les accents). À droite, une trie valide permettant de stocker l'ensemble de mots {et, fa, fable, facile, la}. Les nœuds sont en trait plein lorsqu'ils sont étiquetés par le booléen `true`, et en trait pointillé sinon.





L'ensemble vide de mots peut être représenté par la trie vide. On pourra utiliser la fonction suivante :

```
let trie_vide () = Noeud [];;
```

Les mots seront codés ici par des listes de caractères et non pas par des chaînes de caractères.

```
type mot = char list;;
```

Q1 Écrire en Caml une fonction `appartient : trie → mot → bool` permettant de déterminer si un mot appartient ou pas à un ensemble de mots représenté par une trie valide. Cette fonction devra explorer au plus une branche de la trie.

Q2 Quelle est la complexité temporelle de la fonction précédente en nombre de comparaisons de caractères ? Quelle serait la complexité temporelle d'une fonction qui teste l'appartenance d'un mot à une liste de mots en nombres de comparaisons de caractères ? En quoi cela explique-t-il l'intérêt des tries par rapport aux listes pour la représentation d'un ensemble de mots ? Donner un autre intérêt de cette représentation. Préciser pour quels types d'ensembles de mots l'intérêt est particulièrement notable.

Q3 Écrire en Caml une fonction `ajouter : trie → mot → trie` permettant, à partir d'une trie valide et d'un mot, de retourner une trie valide ayant pour mot supplémentaire le mot donné en argument.

Q4 Écrire en Caml une fonction `initialiser : mot list → trie` permettant de convertir une liste de mots en trie valide représentant le même ensemble de mots. L'utiliser pour définir la trie valide correspondant à l'ensemble de mots {as, port, pore, pré, près, prêt} (sans les accents).

Quelques autres fonctions intéressantes sur les tries :

Q5 Écrire en Caml une fonction `compter : trie → int` permettant de déterminer le nombre de mots contenus dans une trie (valide ou pas).

Q6 Écrire en Caml une fonction `extraire : trie → mot list` permettant de déterminer la liste des mots contenus dans une trie, de sorte que cette liste soit triée dans l'ordre lexicographique si la trie est valide.

Q7 Écrire en Caml une fonction `valider : trie → bool` permettant de déterminer si une trie est, ou pas, valide.