

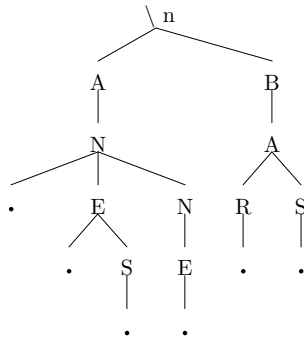
## TD d'Algorithmique 4

### Tri de mots en parties communes

On se propose de ranger une famille de mots par ordre alphabétique en construisant un arbre dans lequel chaque nœud porte une lettre. Cet arbre est défini par les propriétés suivantes :

- le sous-arbre issu d'un nœud donné est formé de tous les mots qui commencent par les lettres que l'on rencontre quand on va de la racine à ce nœud,
- les différents fils d'un même nœud sont ordonnés par ordre alphabétique.

Par exemple, l'arbre correspondant à la suite AN, ANE, ANES, ANNE, BAR et BAS est l'arbre :



Les fins de mots sont indiquées par le caractère '.', dont on pourra utiliser sa propriété d'être inférieure à toutes les lettres. La racine de l'arbre porte le caractère de fin de ligne. Tous les arbres de ce type seront appelés dictionnaires. On se propose d'écrire un programme JAVA de gestion d'un tel dictionnaire.

1. Définir les champs de la classe *Dictionnaire* nécessaires pour représenter la structure de données relative à ce type de dictionnaire, c'est-à-dire les arbres quelconques étiquetés par des caractères.
2. Définir le constructeur correspondant au dictionnaire vide.
3. Définir le constructeur permettant de créer un nouvel arbre dont les champs sont passés en paramètres.

4. Ecrire une méthode *insérer* qui prend en paramètres *c* de type *char* et *a* de type *Dictionnaire*. Cette méthode recherche *c* parmi les racines des arbres de la liste des fils de *a*. Elle renvoie le fils *t* de *a* dont la racine porte *c*, si cet arbre existe. S'il n'existe pas, elle insère un nouveau fils portant ce caractère à l'endroit convenable de la liste des fils, et le renvoie.
5. Ecrire une méthode *traiter* qui insère un nouveau mot (de type *String*) dans un dictionnaire.
6. Ecrire une méthode *imprime* qui étant donné un dictionnaire, affiche à l'écran tous les mots par ordre alphabétique à raison d'un mot par ligne.

On se propose maintenant d'écrire une méthode *liste\_des\_mots* qui renvoie la liste chaînée rangée par ordre alphabétique des mots d'un dictionnaire.

7. Définir une classe *Liste* implémentant les listes chaînées dont les maillons portent des chaînes de caractères. Elle comportera outre un constructeur pour la liste vide et un constructeur permettant de créer un nouveau maillon, une méthode pour insérer un nouveau maillon en fin de liste et une méthode permettant d'imprimer les mots portés par la liste par ordre d'apparition. Veiller à ce que les insertions se fassent en temps constant.
8. Ecrire une méthode *position* qui prend en paramètres *c* de type *char* et *a* de type *Dictionnaire*. Cette méthode recherche *c* parmi les racines des arbres de la liste des fils de *a*. Elle renvoie le fils *t* de *a* dont la racine porte *c*, si cet arbre existe. S'il n'existe pas, elle renvoie *null*.
9. Ecrire la méthode *liste\_des\_mots*.
10. Ecrire une fonction *complement* qui prend un dictionnaire en paramètre et un mot *p*, et renvoie la liste chaînée, rangée par ordre alphabétique, de tous les mots du dictionnaire débutant par *p*.
11. Ecrire la méthode *main* assurant au programme le comportement suivant illustré par la trace ci-dessous. On affichera notamment le dictionnaire de deux façons différentes : soit en utilisant la méthode *imprime*, soit en créant puis affichant la liste des mots du dictionnaire.

```
% java PartiesCom essai ane annee anes essayer an essieu
```

```
an
ane
anes
annee
essai
essieu
essuyer
```

```
impression avec la liste
```

```
an
ane
anes
```

annee  
essai  
essieu  
essuyer

entrer un prefixe : ess

Voila le resultat

essai  
essieu  
essuyer

%