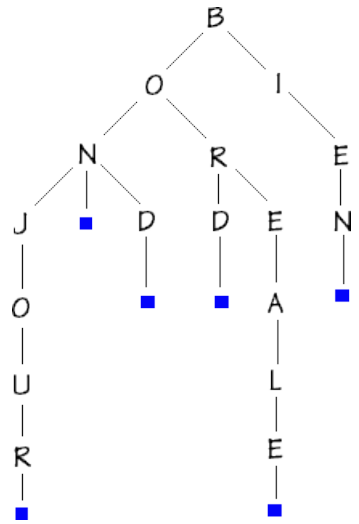
Exercice 1

L'objet de ce sujet est l'utilisation de la structure d'arbre lexicographique en tant que correcteur orthographique. Nous utilisons ici cette structure pour construire un dictionnaire. Nous voulons être capable d'y rechercher un mot de manière efficace.
Soit le dictionnaire représenté par les mots : bonjour, bon, bond, bord, boreale et bien. Celui-ci peut être représenté par l'arbre suivant



Un arbre est un arbre lexicographique si les conditions suivantes sont vérifiées : à chaque noeud est associée une lettre, chaque chemin de la racine vers une feuille correspond à un mot, un préfixe commun à plusieurs mots de l'arbre n'apparaît qu'une fois dans l'arbre.
Un arbre lexicographique ainsi définit ne pourra représenter qu'un ensemble de mots ayant une même première lettre.

1. Construisez la classe `Noeud` possédant un caractère et un ArrayList indiquant ses fils. Le caractère sera soit une lettre, soit un espace marquant une fin de mot (les carrés bleus de la figure).
2. Réalisez un constructeur correspondant au dictionnaire n'ayant aucun mot.
3. Ecrivez une méthode `public Noeud aPourEnfant(Char a)` qui renvoie le noeud fils étiqueté par le caractère `a`.
4. Ecrivez une méthode `public boolean appartient(String w)` qui indique si le mot `w` fait partie du dictionnaire représenté par l'arbre lexicographique.
5. Ecrivez une méthode permettant d'ajouter un mot donné en entrée. Si le mot ajouté n'a pas une première lettre compatible, la méthode ne fait rien.
6. Ecrivez une méthode affichant l'ensemble des mots d'un arbre lexicographique.
7. Ecrivez une méthode permettant d'afficher l'ensemble des mots qui sont préfixes d'autres mots. Par exemple dans l'exemple donné, bon est un préfixe de bonjour.
8. Ecrivez une méthode permettant de supprimer un mot donné en entrée.

Exercice 2

Nous définissons maintenant une distance entre deux mots. Si un mot est préfixe d'un autre, la distance entre ces deux mots est le nombre de lettres les différenciant. "Bon" et "Bonjour" sont à distance 4. Pour le moment on ne définit pas de distance si aucun mot n'est préfixe de l'autre, donc la distance entre "Bon" et "Bien" n'est pas définie.

1. Ecrivez une méthode `public String procheDe(String w)` renvoyant le mot dont la distance par rapport à `w` est la plus faible parmi les mots de l'arbre et `null` si aucun mot n'a de distance à `w` de définie.
2. Ecrivez une méthode prenant un mot `w` et un entier `n` et affichant l'ensemble des mots dont la distance à `w` est inférieure à `n`.

Exercice 3 (facultatif)

La distance de Levenshtein entre deux mots est définie par le minimum d'opérations à réaliser sur ces mots pour les rendre identiques. Les opérations autorisées étant l'insertion d'une lettre, la suppression d'une lettre et la modification d'une lettre. Ainsi la distance entre "Bond" et "Bord" est de 1 (modification du "n" en "r"), la distance entre "Abcd" et "Dabc" est 2 (Ajout du "d" au départ et suppresion de la dernière lettre).
Ecrivez une méthode calculant la distance de Levenshtein entre deux mots.