# TP 11 – Analyse de texte
Année 2019-2020

Ce TP aborde quelques algorithmes d'analyse de textes. Vous spécifierez vos structures de données en suivant les consignes données.

## 1 Recherche de motif

On s'intéresse dans cet exercice aux algorithmes de recherche exacte d'un motif (mot) $M$ dans un texte $T$. Ces algorithmes parcourent séquentiellement le texte et retournent l'indice (index) du motif dans le texte.

Par exemple, si l'on considère le texte `T="hello world!!!"`, pour le motif `M="ello"`, l'algorithme retourne l'indice 1, alors que pour le motif `M="elo"` l'algorithme renvoie -1 (le motif n'est pas trouvé dans le texte).

On considère que le texte $T$ prend ses caractères dans un alphabet donné $A$. Dans cet exercice vous serez amenés à comparer l'algorithme de force brute avec l'algorithme de Boyer-Moore qui prétraite le motif. Ces algorithmes ont été vus dans le cours.

1. Soit l'algorithme de force brute `brutForce(T, M)` qui retourne l'indice du début d'une sous-séquence du texte $T$ qui est égale au motif $M$ si cette sous-séquence est trouvée dans $T$ et qui retourne -1 si la sous-séquence n'est pas trouvée. Le texte et le motif seront donnés par des listes de caractères.
   (a) Implémentez l'algorithme `brutForce(T, M)`.
   (b) Donnez la complexité de l'algorithme en fonction des longueurs $m$ du motif $M$ et $n$ du texte $T$.
   (c) Testez l'algorithme dans des cas particulier de votre choix.
2. L'algorithme de Boyer-Moore permet d'accélérer la recherche. Il repose sur deux heuristiques :
   - On compare le motif $M$ avec le texte $T$ en partant de la fin du motif.
   - Lorsqu'une divergence entre $M$ et $T$ se produit pour le caractère $T[i] = c$, on effectue un saut :
     – si $M$ contient $c$, on décale $M$ en alignant la dernière occurrence de $c$ dans $M$ avec $T[i]$ (caractère courant)
     – Sinon, on décale $M$ de $m$ à partir de la position courante de $M$, $m$ étant la longueur du motif.

(a) Écrivez l'algorithme `traiter_motif(M, A)` qui permet de prétraiter le motif en construisant le dictionnaire $L$ correspondant au prétraitement du motif $M$, pour un alphabet donné.
**Indications** On choisit une structure de données de type dictionnaire (vous prendrez le type prédéfini `dict`), car on associe à chaque caractère de l'alphabet l'index de la dernière occurence de ce caractère dans le motif quand il est présent, ou -1 quand il n'est pas présent.
Par exemple, si l'on choisit $A$ et $M$ de la façon suivante :

```
A = ['a','b','c','d','h'] ; M = ['a','a','a','b','a']
d = traiter_motif(M, A)
```

retourne un dictionnaire $d$ qui contient :
`{'a': 4, 'b': 3, 'c': -1, 'd': -1, 'h': -1}`

(b) Écrivez la fonction `onstruitalphabet(T)`

(c) Écrivez l'algorithme `boyer_moore(T, M)` vu en cours qui permet d'effectuer des sauts lorsqu'on détecte qu'un caractère du texte n'est pas présent dans le motif recherché.
**Indications** Si l'alphabet n'est pas donné, vous pourrez le construire à partir du texte

(d) Donnez la complexité de l'algorithme de Boyer-Moore.

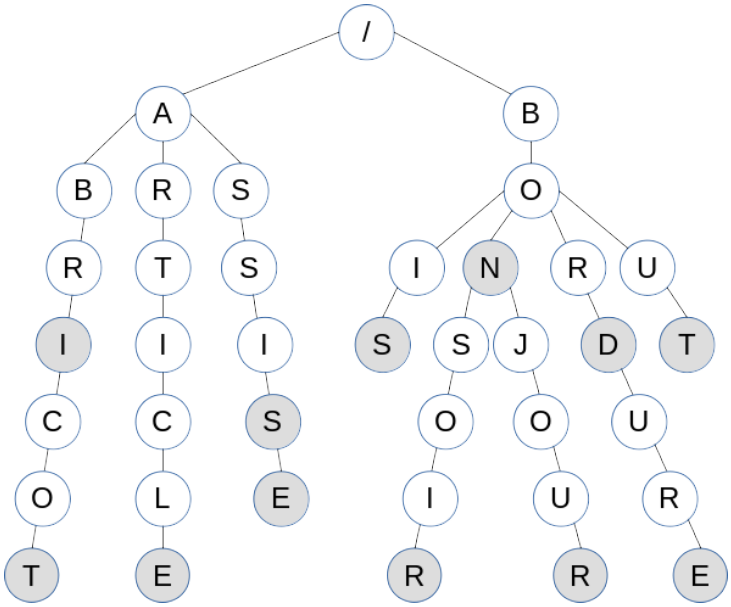
(e) Testez l'algorithme sur des exemples judicieusement choisis.

## 2 Vérificateur orthographique à l'aide d'un arbre de préfixes

### Arbre de préfixes

Un arbre de préfixes est un arbre contenant des mots : chaque noeud de l'arbre contient une lettre et un booléen indiquant si ce noeud est la fin d'un mot (noeud terminal). Un mot est représenté dans l'arbre par un chemin partant du noeud racine de l'arbre et se terminant à un noeud terminal. Le noeud racine contient le caractère "/".

Par exemple, l'arbre représenté ici (les noeuds terminaux sont grisés) contient les mots suivants: abri, abricot, article, assis, assise, bois, bon, bonsoir, bonjour, bord, bordure, bout.



La classe **ArbrePrefixe** est donnée ci-dessous. Vous devrez la compléter.

```
class ArbrePrefixe():
    """
    Arbre prefixe contenant des mots ayant la meme premiere lettre
    """

    def __init__(self, lettre):
        """
        Initialise l'arbre
        Entree: la lettre prefixe de l'arbre
        """
        self.prefixe = lettre # la lettre a la racine de l'arbre
        self.derniere = False # vrai si cette lettre est la derniere d'un mot
        self.fils = []        # la liste des arbres prefixes de cet arbre
```

1. Ecrivez une méthode `donner_fils(self, lettre)` pour la classe **ArbrePrefixe** retournant l'arbre fils dont la lettre racine est un caractère donné. Cette méthode prendra en entrée un caractère et renverra un arbre de préfixe (qui peut être `None`).

2. Ecrivez une méthode `ajouter_mot(self, mot)` pour la classe **ArbrePrefixe** permettant d'ajouter un mot dans l'arbre. Cette méthode prendra en entrée un mot.

3. Ecrivez une méthode `verifier_mot(self, mot)` pour la classe **ArbrePrefixe** permettant de vérifier si un mot est présent dans l'arbre. Cette méthode prendra en entrée un mot et renverra un booléen vrai si le mot se trouve dans l'arbre.

### Vérificateur orthographique

On veut maintenant écrire un vérificateur orthographique. Le programme devra construire un arbre de préfixes à partir d'un dictionnaire, puis vérifier que les mots d'un texte donné sont bien présents dans ce dictionnaire en affichant les mots qui ne s'y trouvent pas.

Dans cette partie on a besoin de travailler avec des fichiers en python. Pour lire ou écrire dans des fichiers, il faut d'abord "ouvrir" ces fichiers avec la fonction `open()`. Cette fonction prend obligatoirement deux arguments : le chemin du fichier (relatif ou absolu), et le mode d'ouverture qui défini si on va lire le contenu du fichier ou écrire dans le fichier.

Exemple : `fichier = open('/chemin vers le fichierfichier.txt', 'r')` ». Pour être en mode « lecture », on utilise « r » pour du texte, « rb » pour une lecture binaire ; pour être en mode écriture on utilise « w » pour écrire du texte, « wb » pour une écriture binaire. Lorsque le fichier est ouvert, pour lire son contenu on utilise la méthode `read()` qui retourne le contenu du fichier. Exemple : `contenu = fichier.read()`.
Pour écrire dans le fichier on utilisera la méthode `write()`. Exemple : `fichier.write(content)`.
Le fichier ouvert étant verrouillé, il ne faut pas oublier de le fermer avec la méthode `close()`.

On définit un dictionnaire comme étant un fichier contenant un ensemble de mots usuels (un mot par ligne). Le texte est contenu dans un fichier et composé de mots séparés par

des espaces. Par souci de simplification, ce texte ne contient ni majuscules ni ponctuation. Un dictionnaire et quelques exemples de textes sont fournis.

En utilisant les méthodes précédemment écrites, vous écrirez les fonctions suivantes qui prendront en paramètres deux noms de fichiers : celui contenant le dictionnaire et celui contenant le texte.

4. Dans le programme principal, écrivez la fonction `construire_arbre_prefixe(dico)` qui construit un arbre de préfixes contenant les mots du dictionnaire `dico`. Vous utiliserez la méthode `ajouter_mot(self, mot)` définie précédemment.

5. Écrivez la fonction `verifier_mots_texte(dico,text)`, qui, pour chaque mot du texte `text`, vérifie si ce mot est dans l'arbre de préfixe défini par le dictionnaire `dico` et affiche ce mot s'il n'est pas présent. Vous utiliserez la méthode `verifier_mot(self, mot)` définie précédemment.

Par exemple, avec les fichiers fournis, on obtiendra :

```
verifie_dictionnaire(en.dic, text1.txt)
 holliday
 thi

verifie_dictionnaire(en.dic, text2.txt)
 ares

verifie_dictionnaire(en.dic, text3.txt)
 wich
```