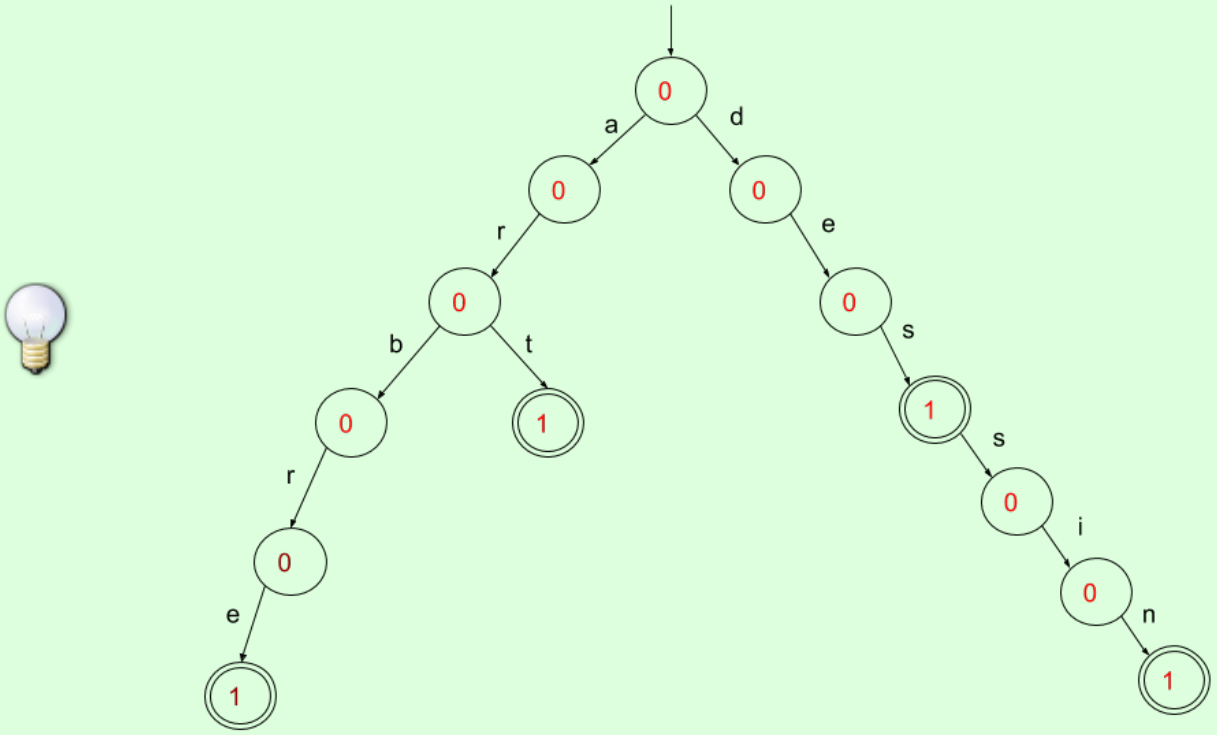
# Arbre de complétion

Un algorithme de complétion est un mécanisme logique permettant d'anticiper la saisie et de proposer des mots automatiquement pour faciliter les recherches dans un formulaire sur une page web par exemple.

On utilise pour cela une structure de données arborescente, où chaque nœud de l'arbre est une étape de lecture et chaque arête correspond à la lecture d'une lettre. Les nœuds sont indexés par les lettres suivantes possibles du mot, avec un compteur par nœud pour savoir si celui-ci est final ou non (le nœud est final si le compteur est >0).

Le but de cet exercice est de construire un arbre de complétion à partir de mots de vocabulaire, puis de l'utiliser pour compléter un début de mot proposé par l'utilisateur.

V = {art, arbre, des, dessin}




### 2.1 tests

Un arbre de complétion sera défini de manière récursive. Un nœud de l'arbre contient 2 éléments :

- fils : un dictionnaire de nœuds indexés par des caractères
- compteur : un entier valant :
  - 0 si le mot n'est pas dans la base
  - une valeur >0 indiquant le nombre d'occurrences du mot dans la base sinon.

Vous devez :

1. Créer un nœud vide sous la forme d'un dictionnaire à 2 entrées:

- `"fils"` : un dictionnaire vide `{}`
- `"compteur"` : la valeur `0`

2. Définir la fonction :

```
def insere(A, mot):
   ...
```

qui insère un mot dans un nœud `A`.

Cette fonction récursive teste si la première lettre du mot est présente dans le dictionnaire des fils. Si non, elle crée l'entrée correspondante. Elle vérifie ensuite si le mot est un caractère isolé. Si oui, elle incrémente le compteur du nœud fils, sinon, elle insère la suite du mot dans le nœud fils.

3. Écrivez et testez la fonction :

```
def affiche(A, s):
    ...
```

qui affiche l'arbre de complétion de manière récursive. Pour chaque fils de `A`, on fait un appel récursif en ajoutant le caractère correspondant à la chaîne `s`. Si le compteur est >0, on affiche `s`.

Créez un arbre de complétion. Insérez les mots `"bonjour"`, `"bonsoir"`, `"bon"`, `"jour"`, `"soir"` et vérifiez grâce à l'affichage que les mots sont correctement insérés.

4. Écrivez et testez la fonction :

```
def suivant(A, debut):
...
```

qui, à partir de la chaîne `debut` et de l'arbre de complétion `A`, retourne la liste de mots qui complète le début de mot.

Améliorez votre fonction pour que les mots les plus courants apparaissent en premier dans la liste.

5. Écrivez et testez la fonction :

```
def appartient(A, mot):
...
```

qui retourne `True` si le mot est présent dans l'arbre de complétion `A`, et `False` sinon.

### 2.2 Vocabulaire complet

Une fois que tout fonctionne, vous devez créer un arbre contenant l'intégralité du vocabulaire français. Pour cela, vous devez télécharger le fichier : Lexique382.zip

- ouvrez le fichier
- créez une liste de vocabulaire vide
- pour chaque ligne du fichier, ajouter à la liste de vocabulaire le premier mot de la ligne
- ajouter chaque mot de la liste de vocabulaire dans l'arbre de complétion
- vérifiez que l'arbre reconnaît correctement les mots de vocabulaire rentrés par l'utilisateur
- affichez les listes de mots proposés pour différents débuts de mots (exemples : 'ga', 'bu', 'zo', 'meu', etc.)

## Exercice 2 : format xml / librairie etree

** Fichiers XML**

Les fichiers xml permettent de stocker ou transmettre des données de type texte organisées de manière hiérarchique, à la façon d'un document contenant des chapitres, sous-chapitres, sections... Les balises sont des délimiteurs <balise> ... </balise>, qui encadrent et caractérisent une portion de texte :

Exemple :

```
<titre> Bilan d'activité 2009-2010 </titre>
```

Chaque portion de texte peut-elle-même contenir des portions de textes encadrées par des balises et ainsi de suite... Les balises définissent ainsi la mise en forme du document, selon une organisation de type hiérarchique.

Exemple :



```
<chiffre_d_affaires> 12456
    <premier_semestre> 7866 </premier_semestre>
    <second_semestre> 4590 </second_semestre>
</chiffre_d_affaires>
```

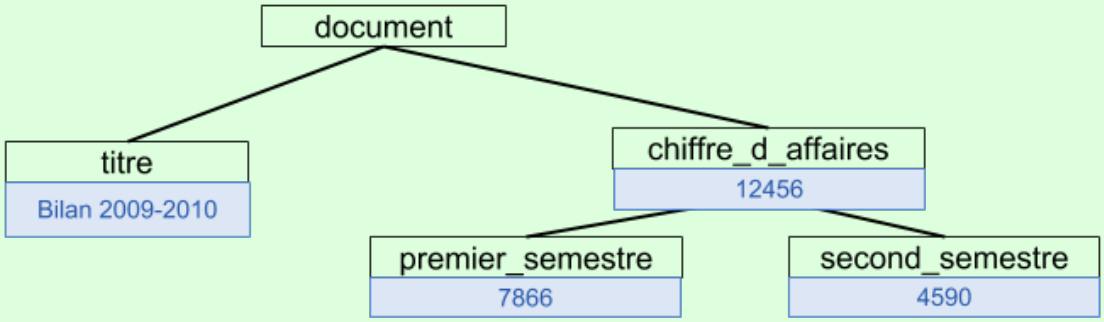
Le traitement des fichiers xml est facilité en Python à l'aide de la librairie etree :

```
import xml.etree.cElementTree as etree
```

qui construit une sructure de données arborescente à partir d'un fichier xml:

```
f = open ('bilan.xml')
doc = etree.parse(f)
```

l'objet doc (représentant le document) contient la structure de données :






- `r = doc.getroot()` : la variable r représente la racine de l'arbre.
- `r.tag` donne le nom de la balise
- `r.text` donne la portion texte correspondante (éventuellement nulle)
- `r.attrib` est un dictionnaire contenant des valeurs d'attributs de la balise
- Enfin, `r` est un objet de type liste. Ainsi : `r[0]` représente le premier fils, `r[1]` le second fils etc...

La boucle :

```
for s in r :
    ...
```

permet de parcourir la liste des fils de `r`.

**A faire**

On souhaite réaliser un convertisseur de monnaies à partir d'un fichier contenant des taux de conversion actualisés de l'euro vers d'autres monnaies (dollar, yen, etc...). Le fichier http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml contient les taux de conversions mis à jour régulièrement, sous la forme d'un fichier au format xml.

Pour ouvrir ce fichier, situé sur le web, on utilisera la librairie `urllib` :

```
from urllib.request import urlopen
f = urlopen("http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml")
```

Affichez le contenu de ce document.

Vous remarquerez qu'il est organisé en 3 sections : `subject`, `sender`, `cube`

- 1 - Écrivez une fonction affiche de manière récursive le contenu du document, c'est à dire qui, pour chaque nœud, affiche le nom de balise, le texte et le dictionnaire contenant les attributs.

- 2 - Selon la norme adoptée ici, les balises de type cube servent à stocker les données sous forme de listes : ''<cube attrib1 = valeur1 attrib2 = valeur2....> </cube> ''

Le premier sous-niveau représente la date (time), `<cube time="2011-09-23"> … </cube>`

et le second sous-niveau contient les valeurs de taux de change sous forme d'attributs currency (devise) et rate (taux de change).

```
  ''<cube currency="USD" rate="1.3430"/>''
```

Ainsi, pour obtenir les taux de change, il faut extraire les attributs de chaque fils du premier sous-niveau : si a est un fils du premier sous-niveau, alors `a.attrib['currency']` contient le nom de la devise et `a.attrib['rate']` contient le taux de change.

Écrivez une fonction `taux_de_change` qui reçoit le nom d'une devise et retourne le taux de change.

- 3 - Écrire une fonction convertisseur qui reçoit un montant en euro et un nom de devise et retourne un montant dans cette devise.

Toutes ces fonctions seront testées au niveau du programme principal

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
**https://wiki.centrale-med.fr/informatique/tc_info:tp7_old**

Last update: **2019/09/28 16:17**