Programmation avancée **Problème**

Stockage d'un lexique dans un arbre lexico-graphique

L'objet de ce problème est d'étudier les performances du stockage d'un lexique dans un arbre lexicographique.

Votre travail se découpera en au moins 5 fichiers :

- apc.cpp et apc.h qui contiendront toutes les données et fonctions de gestion de l'arbre lexicographique,
- principal.cpp, programme principal lançant la création du dictionnaire dans l'arbre et du dictionnaire dans le tableau,
- makefile qui servira à compiler tout ça,
- un fichier de commentaires.

Nous vous fournissons 2 fichiers de mots, un petit et un gros.

1 - Arbre lexicographique (arbre en partie commune)

Un arbre lexicographique permet de stocker, dans une structure d'arbre, un ensemble de mots en factorisant au maximum les débuts de mots de manière à utiliser le moins d'espace mémoire possible, tout en permettant d'accéder de manière efficace à ces mots. Une définition à peu prés formelle :

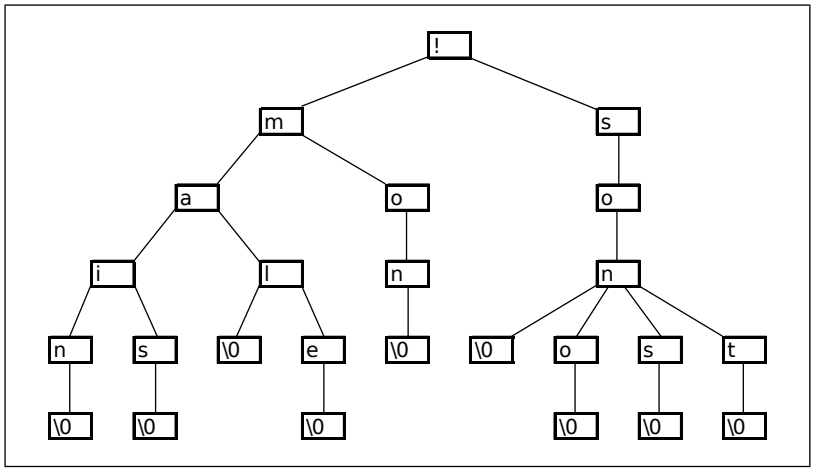
- les nœuds portent des caractères comme information,
- la racine est conventionnelle et ne porte aucune information significative,
- les fils de la racine sont des nœuds qui portent les premières lettres des mots, rangées par ordre alphabétique,
- soit r un des fils de la racine, c la lettre qu'il porte. Les fils de r sont des nœuds qui portent les deuxièmes lettres, rangées par ordre alphabétique, des mots commençant par c,
- plus généralement, soit $r_0r_1r_2\ldots r_n$ un chemin depuis la racine (r_0 est la racine), $c_1c_2\ldots c_n$ les lettres respectivement rangées dans ces nœuds. Les fils de r_n sont des nœuds qui portent les $(n+1)_{\text{èmes}}$ lettres, rangées par ordre alphabétique, des mots commençant par $c_1c_2\ldots c_n$,
- tous les mots sont terminés par un caractère spécial : '\0'.

Voici par exemple l'arbre correspondant aux mots : *main*, *mais*, *mal*, *male*, *mon*, *son*, *sono*, *sons* et *sont.*

Dans ces arbres, chaque mot est stocké dans un chemin de la racine vers une feuille.

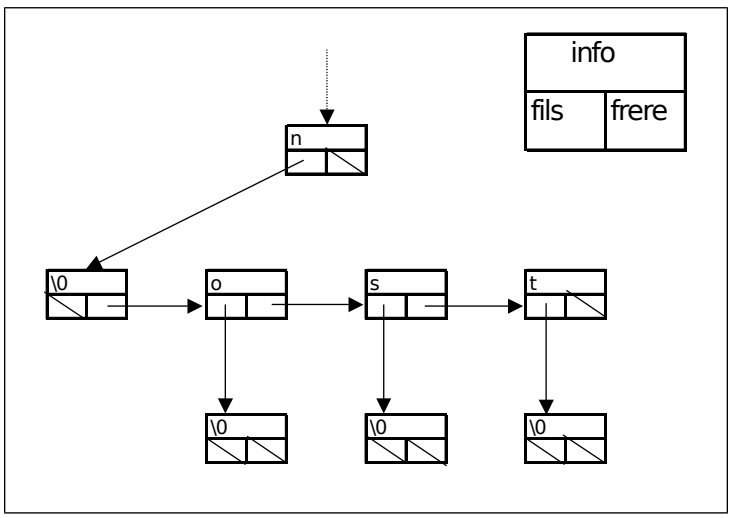
Structures de données :

Il est à noter que nous avoncs affaire ici à des arbres dont le nombre de fils de chaque nœud est quelconque. Vous utiliserez, pour stocker ces arbres, des structures proches de celles vues pour les arbres binaires, qui conviennent tout à fait moyennant un petit changement des noms des membres données de nœud :



`struct noeud`
`{`
`  char info;`
`  noeud * fils, * frere;`
`  public:`
`...`
	`class arbre`
	`{`
	`  noeud * racine;`
	`  public:`
	`...`

Chaque nœud possède deux pointeurs, un pour descendre dans l'arbre, et un pour parcourir un niveau. Ainsi, le sous-arbre complètement à droite de l'arbre précédent (où le nœud 'n' possède 4 fils) sera représenté ainsi :



Marche à suivre :

- lire tous les mots du « gros » lexique un par un et les charger dans l'arbre,
- estimer le temps de chargement et l'espace mémoire de l'arbre plein,
- lire tous les mots du « petit » lexique et tester leur présence dans l'arbre,
- estimer le temps de test de tous les mots.

4 – Rendu

Respecter ABSOLUMENT les modalités décrites sur la plate-forme !