Objectifs

Concevoir, implémenter en C et évaluer différents algorithmes de base sur des arbres binaires de recherche et les arbres lexicographiques. Comprendre l'intérêt de ces structures de données via l'évaluation des complexités temporelle et spatiale. Présenter et discuter les choix d'implémentation. Gérer ce TP4 projet.

But

Ce TP est inspiré de l'exercice 2 du final NF16 A2016. Le but final est d'implémenter un correcteur orthographique. Le correcteur orthographique aura 3 principales fonctionnalités : (1) Corriger l'orthographe d'un ensemble de mots dans le fichier file.txt ou ajouter certains mots dans un dictionnaire depuis file.txt. (2) Afficher un sous ensemble (ou la totalité) de l'ensemble des mots du dictionnaire dans l'ordre lexicographique (3) et proposer une correction, une suppression ou une validation d'un mot.

Travail à faire

A la fin de ce projet, vous déposerez votre code source et votre rapport sur Moodle dans l'espace dédié au rendu du TP4. L'organisation **minimale** du code sera la suivante:

- Fichier d'en-tête tp4.h, contenant la déclaration des structures et fonctions
- Fichier source tp4.c, contenant la définition de chaque fonction
- Fichier source main.c, contenant le programme principal

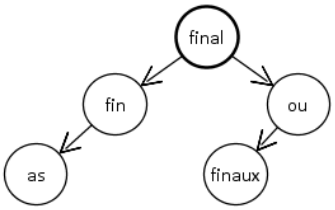
On n'hésitera pas à définir des déclarations, fonctions et fichiers (source et d'en-tête) supplémentaires

Votre rapport de **4 pages maximum** contiendra :

- Une description précise et concise de chaque structure de données et chaque fonction implémentées.
- Un exposé succinct de la complexité des **fonctions principales**. Le code n'est pas autorisé sauf pour mentionner une instruction importante.
- Les difficultés rencontrées, et comment vous les avez surmontées.
- Tout autre élément que vous jugerez utile, tout en expliquant.

Partie I : Représentation 0 – ABR

Dans cette partie on représente le dictionnaire avec un ABR. On suppose qu'un mot est une chaîne de caractères. Le dictionnaire obtenu après les insertions successives des mots "final", "ou", "fin", "as" et "finaux", est alors :

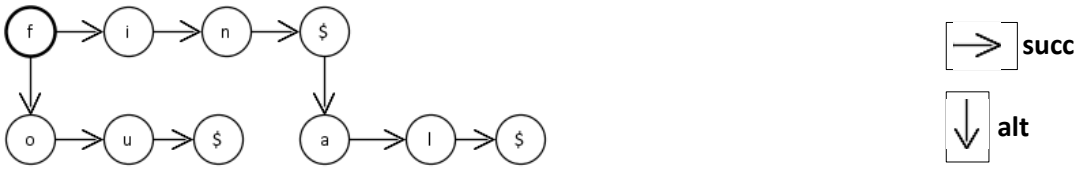


1. Implémenter les structures de données permettant de définir le type **DicoABR** qui représente un dictionnaire comme un ABR ; sachant l'ordre dans l'ABR est donné par l'ordre lexicographique.
2. Implémenter et tester les fonctions de base **initDico, ajoutMot, rechercheMot, supprimeMot** permettant de manipuler le dictionnaire.
3. Implémenter et tester une fonction **suggestionMots**; suggestionMots prend en entrées: **k** un entier positif, **dico** un dictionnaire et **souschaine** une chaîne de caractères. La fonction retourne les k mots plus proches de souschaine se trouvant dans dico.
   Soient deux mots x et y qui commencent respectivement par les n1 et n2 premières lettres de souschaine. x est plus proche de souschaine que y si n1>n2 ou bien n1=n2 et x est plus petit que y dans l'ordre lexicographique.

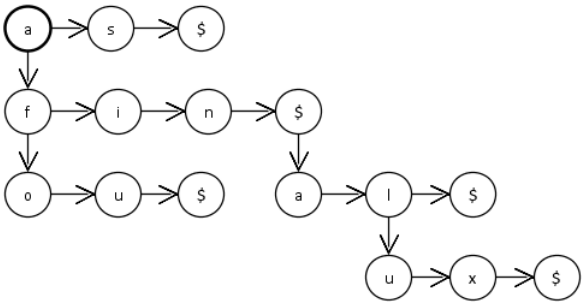
Partie II : Représentation plus compacte – Arbre Lexicographique (AL)

Lorsque le dictionnaire est dense c'est-à-dire contient plusieurs de mots parmi lesquels plusieurs partagent le même préfixe, il est intéressant de ne pas répéter ces préfixes partagés. Une structure d'AL permet de gérer cette situation. De plus, elle permettra d'opérer plus rapidement sur le dictionnaire.

Le dictionnaire contenant les trois mots : "ou", "fin" et "final" sera toujours représenté par l'AL ci-dessous, ce quel que soit l'ordre d'insertion des trois mots. La racine courante est la Cellule contenant 'f'.



Après des insertions des mots "as" (a s $) puis "finaux" (f i n a u x $) le dictionnaire devient :



La racine courante devient alors la Cellule contenant 'a' de "as".

Un mot est un pointeur sur une structure **Car**. Car contient les informations suivantes :

- un caractère **c** qui représente un caractère d'un mot
- un pointeur **suiv** sur Car qui contiendrait le caractère suivant **c**

Le caractère '$' marque la fin de tout mot. Le dictionnaire est un pointeur sur une structure Cellule. Une Cellule contient les informations suivantes :

- un caractère **c** qui représente un caractère d'un mot
- un pointeur **succ** sur Cellule qui contiendrait un caractère suivant **c** dans un mot
- un pointeur **alt** sur Cellule qui contiendrait un caractère strictement supérieur et alternatif à **c**

Pour tout d, défini par Cellule *d, si on suppose que le caractère '$' est plus petit que tous les autres caractères, alors on a:

d <> NIL et d->alt <> NIL si et seulement si d->c < d->alt->c.

1. Représenter le dictionnaire donné en exemple après la suppression du mot "final" (f i n a l $)
2. Définir les types **Mot** et **Dico** pour représenter un mot et un dictionnaire. Dico est-il un ABR? Justifiez
3. Implémenter et tester les fonctions suivantes
    a. **Dico initDico2(Dico dico, Mot mot)** initialise dico avec mot et retourne dico mis à jour
    b. **Dico prefixeMot(Dico dico, Mot mot)** retourne un pointeur sur la cellule contenant le dernier caractère du plus long préfixe du mot **mot.** Si ce caractère n'existe pas retourne NULL.
    c. **int rechercheMot2(Mot mot, Dico dico)** retourne 1 si mot existe dans dico 0 sinon
    d. **Dico ajoutMot2(Mot mot, Dico dico)** insère mot dans dico et retourne dico mis à jour
    e. **Dico supprimeMot2(Mot mot, Dico dico)** supprime mot dans dico et retourne dico mis à jour
4. Refaire la question 3 de **Partie I** avec les types Mot et Dico

Partie III : Intégration et fonctionnement

On cherche maintenant à tirer profit des fonctions implémentées pour réaliser un correcteur orthographique. Au lancement du programme principal, la liste des mots (un mot par ligne) du dictionnaire se trouvant dans le fichier **dictionnaire.txt** est chargée dans un objet dictionnaire de type Dico. Cet objet est également sauvegardé lorsqu'on quitte le programme. Le programme devra permettre la sélection de la représentation ABR ou AL.

Ecrire un programme via le main.c qui implémente les trois fonctionnalités en offrant 3 options principales à un utilisateur u1:

1. **print** : le programme affiche un sous ensemble (ou la totalité) de mots du dictionnaire dans l'ordre lexicographique. On proposera différentes stratégies pour la sélection d'un sous ensemble.

2. **verimot** : le programme lit le fichier **file.txt**, puis pour chaque mot (un mot par ligne) contenu dans le fichier, l'affiche, si ce mot n'existe pas dans le dictionnaire, le programme propose à u1 de
    a. remplacer ce mot dans le fichier avec un mot du dictionnaire choisi par u1. Des mots devront être suggérés à u1 par le programme.
    b. ajouter le mot dans le dictionnaire
3. **veridico** : le programme affiche un sous ensemble de mots du dictionnaire et pour chaque mot, propose à u1 une
    a. correction de ce mot dans le dictionnaire. u1 devra rentrer le mot corrigé.
    b. suppression de ce mot dans le dictionnaire
    c. validation

Lorsque l'utilisateur choisit une option, le programme lui demande toujours de confirmer l'action qui sera effectuée avant de l'appliquer.

Partie IV (Bonus) Comparer de manière expérimentale les complexités des opérations (ajout, recherche, suppression, et modification) sur l'ABR (et un arbre AVL) et l'AL. On ne demande pas nécessairement d'implémenter ces opérations pour l'arbre AVL; elles sont toutes en $O(\log_2(n))$.