Sylvain Tran, Kevin Thai et Lesly Jumelle TOUSSAINT

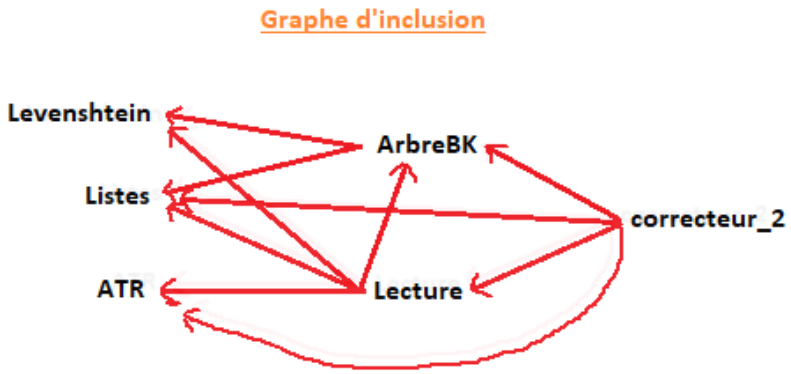
Partie 3 : utilisation d'arbre BK

1) Introduction :

Dans cette troisième et dernière partie du projet, nous allons utiliser les **arbres BK** pour faire à la fois la détection des mots mal orthographiés et les propositions de corrections.

2) Documentation technique :

Tout d'abord, voici le nouveau graphe d'inclusion :



Par rapport à la partie 2, nous avons fait quelques améliorations dans le module **Lecture** :

-dans la fonction : `void verifie_appel(int argc, char **argv)`, nous avons ajouté 2 conditions permettant de vérifier les arguments des noms de fichier : si le 1er fichier n'a pas l'extension « .txt » et le 2nd « .dico », l'exécution du programme s'interrompt.

-nous avons également testé le retour de tous les malloc() pour les mots (ici, dans la fonction `Liste correction_orthographique(ATR *A, char *mot)` par exemple :

```
m = (char*) malloc(sizeof(char) * MAX);

if (m == NULL){
    return NULL;
}
```

Ce que nous n'avions pas fait !

Dans ce même module, nous avons également créé 1 nouvelle fonction :

```
Liste mots_mal_orthographie_avec_ArbreBK(FILE *a_corriger, ArbreBK A);
```

→ Elle renvoie une liste qui contient les mots mal orthographiés du fichier texte « a_corriger » mais en utilisant un arbre BK.

Dans le nouveau module **ArbreBK,** nous avons créé **3** autres fonctions (en dehors de ceux demandées dans le sujet) :

```
NoeudBK* allouer_noeud(char *mot, int val);
NoeudBK* verifier_arete_enfants(ArbreBK parent, int d);
void inserer_nouvel_enfant(ArbreBK *parent, NoeudBK *enfant);
```

(Pour les 2 dernières, vous trouverez une description de ce qu'elles font dans le fichier ArbreBK.c)

De plus, nous avons décidé de modifier le prototype de 2 fonctions :
1)

```
int rechercher_dans_ArbreBK(ArbreBK A, char *mot, Liste *corrections, int *seuil);
```

-Son type de retour est **int** : renvoie 0 si A est vide ou si un problème d'allocation a eu lieu, 1 sinon.
On a ajouté une liste correction et un **seuil**.
Dans cette fonction, on cherche les mots m de A qui sont à distance au plus « **seuil** » de « mot ».

2)

```
void afficher_ArbreBK(ArbreBK A, int nb_espace);
```

Le paramètre « nb_espace » sert à mettre le bon nombre d'espace (là où il faut).

TEMPS EXÉCUTION :

Pour finir, nous avons mesuré le temps d'exécution des 2 méthodes :

Avec le fichier texte donné à la figure 7 et le dictionnaire de la figure 6 du sujet, nous avons obtenue :
1) Pour la méthode par force brute (en parcourant l'intégralité d'un ATR) : 1 seconde.
2) Pour la méthode utilisant les arbres BK : 25 secondes.

Donc, d'après nos résultats, nous pensons que pour proposer de la correction orthographique, l'utilisation d'ATR est plus optimale.