Sylvain Tran, Kevin Thai et Lesly Jumelle TOUSSAINT

Partie 2

Introduction :

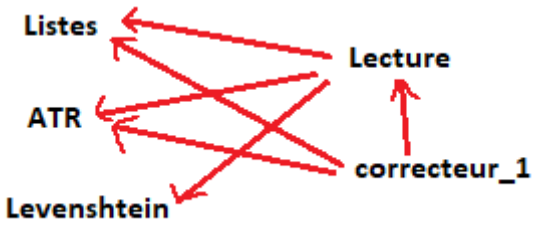
Le but de cette seconde partie est d'implémenter un algorithme qui va proposer des suggestions de correction pour chaque mot mal orthographié d'un texte.

Remarque :

L'effectif du groupe a changé, par rapport aux mauvaises circonstances qui malheureusement a contraint le binôme de Lesly Jumelle TOUSSAINT d'arrêter les cours instantanément. Lesly nous a demandé s'il pouvait intégrer notre groupe et on a accepté, bien entendu avec la permission du chargé de TP. Donc maintenant, on forme un trinôme.

Documentation technique :

Tout d'abord, voici le graphe d'inclusion de notre projet :



Nous avons créé le module **Levenshtein** en y créant la fonction :
`int Levenshtein(char *un, char *deux);` renvoyant la distance de Levenshtein entre la chaîne « un » et « deux ».

Dans le module **Lecture**, nous avons créé 4 nouvelles fonctions :

1) `Liste mots_mal_orthographie(FILE *a_corriger, ATR A);` : elle implémente l'algorithme 1 → Elle renvoie une liste qui contient les mots mal orthographiés du fichier texte « a_corriger ».

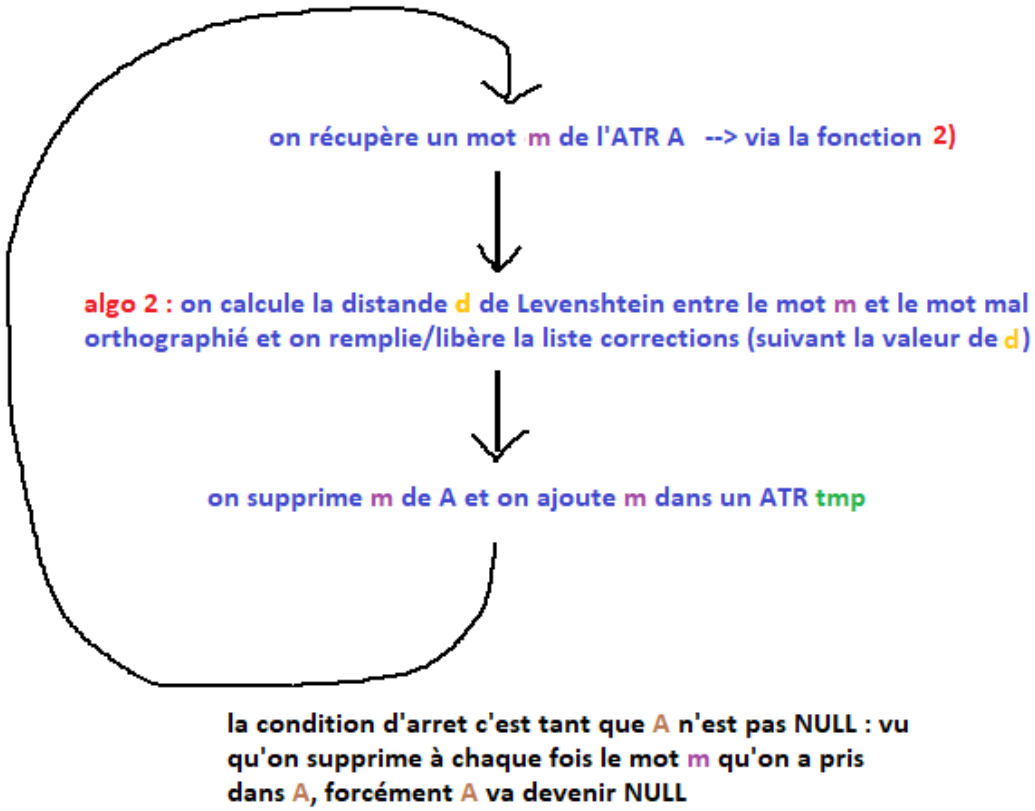
(En fait, dans le rendu 1, on avait écrit l'algorithme 1 directement dans le main() mais on a décidé de faire une fonction pour que ce soit plus compact.)

2) `static void mot_de_ATR(ATR A, char mot[], int i, int *stop)` : elle renvoie un mot de l'ATR A (le plus petit alphabétiquement).

3) `Liste correction_orthographique(ATR *A, char *mot);` : elle implémentre l'algorithme 2 → Elle renvoie une liste qui contient des propositions de corrections pour le mot « mot ».

4) `void afficher_proposition(Liste corrections);` : elle affiche les propositions de corrections (les mots sont sur la même ligne et séparés par des virgules ‘,’).

Pour implémenter la fonction 3), voici l’idée qu’on a eu (en dessin) :



la condition d'arret c'est tant que A n'est pas NULL : vu qu'on supprime à chaque fois le mot m qu'on a pris dans A, forcément A va devenir NULL

quand A est NULL, on fait A = tmp pour récuperer notre ATR et on renvoie la liste corrections

On sait qu’en faisant A = tmp, l’organisation des mots dans l’ATR A ne sera jamais la même mais ce n’est pas gênant.

Au début, on n’avait pas pensé à utiliser une variable temporelle. À chaque fois que A devenait NULL, ce qu’on faisait c’est qu’on appelait la fonction `int construire_ATR(FILE *dico, ATR *A);` qui reparcourait le dictionnaire. Mais on trouvait ça trop lourd de reparcourir à chaque fois le fichier dico pour reconstruire notre ATR (surtout s’il est super grand).

for chaque mot `m` présent dans `dico` **do**

Pour cette partie de l’algorithme 2, on s’est demandé s’il était pas plus simple de faire un simple parcours du dictionnaire avec fscanf() au lieu de parcourir à chaque fois l’ATR A qui code justement celui-ci (ce qu’on veut dire c’est qu’un ATR à ce niveau-là n’est pas très utile et complique surtout l’implémentation).