Manar Nssair & Nguyen Sarah

DM - Correcteur orthographique

Étape 2 : Correction orthographique par force brute

Mode d'emploi :

- Dans le terminal, utiliser la commande `make` pour compiler le projet.
- Pour lancer le jeu avec le fichier texte à corriger (exemple : a_corriger_1.txt) et le dictionnaire de support (exemple : dico_3.dico),

  écrivez `./main a_corriger_1.txt dico_3.dico` .

Modules

Listes

- **Cellule * allouer_Cellule(char * mot)** : Alloue une cellule où l'on y met mot et la retourne.
- **int inserer_en_tete(Liste * L, char * mot)** : Insère en entête une nouvelle cellule dans Liste.
- **void liberer_Liste(Liste * L)** : Libère les nœuds alloués dans Liste.
- **void afficher_Liste(Liste L)** : Affiche les mots présents dans Liste.

ATR

- **ATR creer_ATR_vide()** : Alloue un arbre ternaire de recherche vide et la renvoie.
- **void liberer_ATR(ATR * A)** : Libère les _ATR alloués dans ATR.
- **ajoute_branche(ATR * A, char * mot) :** Ajoute une nouvelle branche à ATR contenant le nouveau mot.
- **int inserer_dans_ATR(ATR * A, char * mot)** : Insère dans ATR un nouveau mot.
- **void supprimer_dans_ATR(ATR * A, char * mot)** : Supprime le mot entré en paramètre dans ATR.
- **void afficher_aux(ATR A, char buffer[], int i)** : Affiche les mots présents dans ATR.

- **void afficher_ATR(ATR A)** : Initialise les valeurs buffer et i pour l'appel de la fonction afficher_aux().
- **int remplir_ATR(FILE * df, ATR * A)** : Rempli l'ATR A avec les mots du fichier.
- **int appartient_dico(ATR t, char * mot)** : Verifie si le mot appartient au dico (s'il est présent dans les mots pouvant être formés dans l'ATR t). S'il y appartient il renverra 1, et dans le cas contraire 0 .

Levenshtein

- **int min(int a, int b)** : Renvoie le plus petit entier entre a et b.
- **int max(int a, int b)** : Renvoie le plus grand entier entre a et b.
- **int Levenshtein(char * un, char * deux)** : calcule la distance de Levenshtein entre les mots un et deux.

correcteur_0

- **FILE * ouvre_fichier(const char * chemin)** : Ouvre le ficher avec le mode d'accès choisi.
- **Liste algo_1(FILE * df, ATR * A)** : (Implémente l'algorithme 1 de détection de mots mal orthographiés) Renvoie la liste des mots contenant des erreurs orthographiques (mots non présents dans le dictionnaire).

correcteur_1

- **FILE * ouvre_fichier(const char * chemin)** : Ouvre le ficher avec le mode d'accès choisi.
- **Liste algo_1(FILE * df, ATR * A)** : (Implémente l'algorithme 1 de détection de mots mal orthographiés) Renvoie la liste des mots contenant des erreurs orthographiques (mots non présents dans le dictionnaire).
- **Liste algo_2(FILE * df, char * mot)** : (Implémente l'algorithme 2 de détection de mots mal orthographiés par un texte de force brute) Renvoie la liste des mots de corrections proposés pour le mot entré en paramètre.

Conclusion

Étape 1 : Nous avons rencontré certaines difficultés lors du codage de la fonction supprimer. En effet comme nous ne possédons pas l'algorithme nécessaire pour celle ci, cela nous a pris plus de temps que prévu puisque nous avons du tester et prendre en compte les différents cas possibles des mots à supprimer en testant plusieurs algorithmes différents.

Pour la répartition des tâches nous avons répartis les différentes fonctions à coder, excepté pour la fonction supprimer où l'on a travaillé dessus ensemble. Pour les autres fonctions, nous nous sommes entraidées quand nous étions bloquées.

Étape 2 :

L’étape n°2 était bien rapide et nous n’avons pas rencontré de difficulté à la faire. En effet nous avons simplement appliqué les deux algorithmes de l’énoncé. Pour la répartition du travail, une personne s’est occupé de la fonction Levenshtein et l’autre de celle de l’algorithme 2.