

# DM - CORRECTEUR ORTHOGRAPHIQUE

*Rendu 3*

MELLOUK Imân – MENSAH ASSIAKOLEY Ako Seer Harley  
23/04/2022

## Table des matières

Table des matières .....	1
Objectif.....	2
Documentation utilisateur .....	2
Entrée / Sortie.....	2
Exécution du programme.....	2
Documentation technique.....	3
Implémentation de la structure Arbre BK .....	3
ATR ou ArbreBK ? .....	4
Extension .....	4
Outils de travaux.....	5
Conclusion .....	5
Répartition des tâches.....	6
Ce que nous avons appris .....	6

## OBJECTIF

Dés à présent, l'objectif de ce troisième rendu est d'implémenter un correcteur orthographique, en utilisant comme structure de données principale, un arbre BK.

## DOCUMENTATION UTILISATEUR

### Entrée / Sortie

- Entrée : le programme prend en premier argument un entier compris entre 0 et 2, représentant le correcteur que l'on souhaite lancé, un dictionnaire de référence ainsi que le texte à corriger. Ces fichiers se trouvant dans le répertoire **ressources**.
- Sortie : Sur le terminale, on y retrouve l'exécution du correcteur lancé. C'est-à-dire que si l'on lance le correcteur 0, on obtiendra uniquement les mots dit mal orthographié tandis qu'avec le correcteur 1 ou 2, on obtiendra tous les mots considérés comme mal orthographié sont affichés suivie d'une proposition de correction

### Exécution du programme

Afin de lancer le programme, il faut se situer dans le répertoire `Correcteur_Part3` puis écrire la commande `./make` qui créera un exécutable correcteur.

Puis taper la commande suivante :

```
./correcteur [entier entre 0 et 2] ressources/[nom du texte à corriger] ressources/[nom du dictionnaire de réf]
```

Par exemple, pour lancer le correcteur 2 avec le texte à corriger `a_corriger_1.txt` avec pour référence le dictionnaire `dico_2.dico`. On écrit la commande suivante :

```
./correcteur 2 ressources/a_corriger_1.txt ressources/dico_2.dico
```

NB :

- Le correcteur 0, correspond à la recherche de mot mal orthographié dans un ATR.
- Le correcteur 1, correspond à la correction orthographique d'un texte en utilisant un arbre ATR.
- Le correcteur 2, correspond à la correction orthographique d'un texte en utilisant un arbre BK..

## DOCUMENTATION TECHNIQUE

Dans un premier le temps, le programme va distinguer les mots non correctes des mots corrects, puis les stockera dans une liste chaînée. C'est à partir de cette liste que nous allons proposer des corrections possibles. Cette fois-ci la structure de donnée principales sera un arbreBK.

### Implémentation de la structure Arbre BK

Pour stocker les mots du dictionnaire de référence, on utilisera un arbre BK. Chaque nœuds de cette arbre contiennent 4 champs ;

1. Le champ mot de type char \*, qui contiendra le mot que stocke ce nœud
2. Le champ val de type int. Stockant la distance de Levenshtein entre ce nœud et le nœud parent de celui-ci.
3. Le champ frereD, est un sous-arbre contenant les mots dont la distance de Levenshtein entre le parent et ce mot est supérieure au champs val.
4. Le champ filsG, est un sous-arbre contenant les mots dont la distance de Levenshtein entre le parent et ce mot est inférieur ou égal au champs val.

L'insertion se fait de tel façon : pour trouver où placer un nœud, on calcule la distance de Levenshtein entre le mot à insérer et le nœud parent (pour débiter ça sera la racine). Si sa distance Levenshtein est égale ou inférieur à celle d'un autre fils du même parents, alors on devient le fils du fils et on se place dans le sous arbre du filsG. Au cas contraire on se place de le sous arbre frereD.

Pour effectuer la recherche d'un mot dans un tel arbre, nous n'allons pas effectuer un parcours préfixe parcourant tout l'arbre jusqu'à rencontrer le mot. Nous allons nous déplacer dans l'arbre stratégiquement, en utilisant le calcul de la distance de Levenshtein. Ce calcul permettra de savoir s'il faut se déplacer le sous-arbre filsG ou dans le sous-arbre frereD.

Quant à l'affichage de l'arbre Bk, nous effectuons un parcours préfixe de l'arbre, afin de rencontrer tout les mots le composant.

Une fois la structure écrite il ne nous resté plus qu'à implémenter les algorithmes permettant la correction orthographique.

## ATR ou ArbreBK ?

En comparant les deux implémentation des structures de données nous nous sommes rendu compte qu'un arbre BK était plus efficace qu'un ATR. Son efficacité réside dans la recherche. Dans un arbre BK grâce au calcul de la distance de Levenshtein on sait à peu près où le mot se situe et dans quel sous-arbre recherché. Il n'est pas nécessaire de parcourir l'arbre de façon préfixe jusqu'à la rencontre du mot qui serait de complexité  $O(n)$  (comme pour l'arbre ATR). Dans un arbre BK la complexité serait plus liée à la hauteur, si en effet le mot rechercher serait une feuille. Mais comme on sait dans quel branche rechercher alors la complexité se réduit à  $O(h)$ , ce qui est nettement plus efficace.

- Pour vérifier le temps mis par chaque méthode on appelle la fonction du système 'time' sur chaque exécution :

```
time ./correcteur [entier entre 0 et 2] ressources/[nom du texte à corriger] ressources/[nom du dictionnaire de réf]
```

## EXTENSION

On a eu l'idée d'ajouter une amélioration à notre correcteur\_2 afin de permettre à l'utilisateur de corriger un texte en temps réel grâce au dictionnaire qu'il aura choisi. Mais on a pas pu aboutir notre travail.

Problèmes rencontrés :

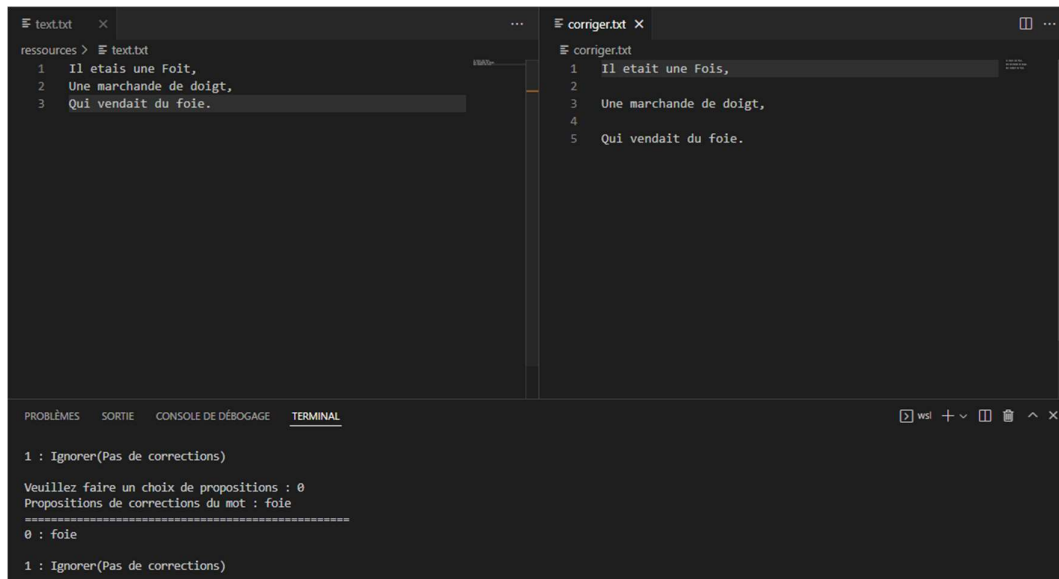
Le programme stagne après un certain nombre de temps qu'on n'arrive pas à déterminer. Mais il arrive qu'en même à corriger de petit texte.

On rend l'ensemble de ces compléments dans le dossier **Extension** qui sera en plus du dossier Correcteur\_Part3 contenant le projet sans amélioration mais tout de même fonctionnel. (cette extension conserve les majuscules)

Après avoir lancé le makefile, pour exécuter le programme, la commande ci est nécessaire :

**`./correcteur_2 [nom de fichier à corriger] [nom du fichier contenant un dictionnaire]`**

Exemple sur le fichier « text.txt » : (cette extension conserve les majuscules)



```

ressources > F text.txt
1 Il etais une Foit,
2 Une marchande de doigt,
3 Qui vendait du foie.

F corriger.txt
1 Il etait une Fois,
2
3 Une marchande de doigt,
4
5 Qui vendait du foie.

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL
1 : Ignorer(Pas de corrections)
Veuillez faire un choix de propositions : 0
Propositions de corrections du mot : foie
0 : foie
1 : Ignorer(Pas de corrections)

```

## OUTILS DE TRAVAIS

Afin de faciliter et de vérifier l'implémentation de notre arbre lexicographique, nous avons utilisé et adapté notre module Visualise, qui nous permet de générer un fichier PDF représentant l'arbre que l'on manipule. Ainsi on pouvait vérifier si nos insertion/suppressions avait réellement fonctionné ou non.

Nous avons également utilisé des dessins, ainsi que le cours pour mieux comprendre cette structures.

## CONCLUSION

## Répartition des tâches

Ce projet a été réalisé en binôme, nous avons organisé quelques appels sur discord ensemble. Nous avons eu en pratique des temps de travail en commun. Mais parfois chacun de notre côté, nous permettant d'avancer à notre rythme que nous mettions en commun sur github ou sur discord. Pour toutes décisions : idées générale d'implémentation, bonne compréhension du sujet etc... Nous nous consultons à chaque fois. De plus nous étions plutôt complémentaires, ce qui nous a davantage permis d'avancer de manière fluide et efficace.

## Ce que nous avons appris

Ce projet nous a permis d'apprendre à manipuler et comprendre plus finement la structure qu'est un arbre, en implémentant deux types d'arbre, les arbres BK et ATR. Cela nous a appris à user une structure de données plus complexe mais efficace dans un contexte concret tel que la correction orthographique.