Ilhan ACHERCHOUR / Hakim AOUDIA
Semestre 4 - 2ème année de Licence

20/03/2022

Algorithmique des arbres

Rapport de l'étape 1

Sommaire :

Sommaire : **1**

Introduction **2**

Compilation **2**

Difficulté **3**

Liste chaînée 3

Insertion 3

Suppression 3

Accents 4

Documentation utilisateur **4**

Conclusion **4**

Introduction

Pour ce projet final du semestre nous avons travaillé sur les **arbres ternaires de recherches**. La première étape consiste à introduire un dictionnaire complet dans un **ATR** (arbre ternaire de recherche) puis de voir si chaque mot d'un texte est correctement orthographié selon le dictionnaire donné. Si les mots sont mal orthographiés on les insère dans une liste chainée et on affiche cette même liste a la fin.

Pour cela nous devions créer 3 fichiers, **Listes.c** et **ATR.c**.

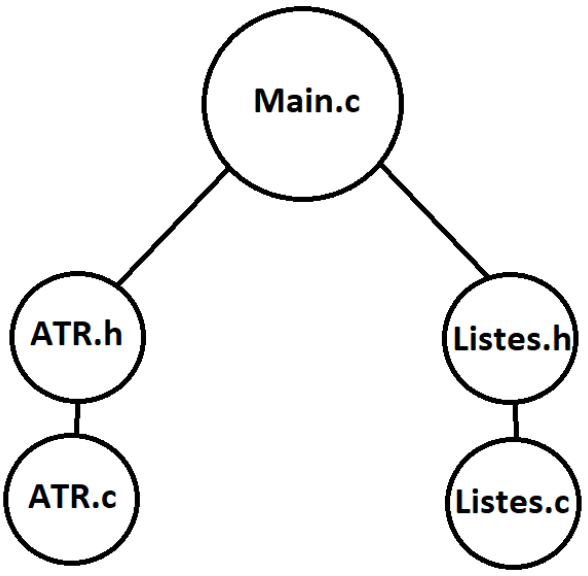
- Listes.c contient les fonctions permettant de gérer une liste chaînée.
- ATR.c nous sert à créer les fonctions de base pour utiliser un arbre ternaire de recherche.

Il fallait ensuite créer un fichier **Main.c** qui gère le programme principal, c'est-à- dire la recherche de mots mal orthographiés dans un texte à l'aide des 2 autres fichiers précédemment présentés.

Compilation

On compilera à l'aide d'un makefile de cette manière :
makefile Projet
./correcteur_0 texte_a_corriger.txt dictionnaire.dico

On peut schématiser la relation des fichiers entres eux de cette manières :



Difficulté

Liste chaînée

Les fonctions de traitement des listes chaînées ont été rapides à implémenter. En revanche, la partie concernant les arbres ternaires de recherche s'est avérée plus compliquée que prévu.

Insertion

Au début de l'écriture de la fonction d'insertion, les idées ne manquaient pas. Les premiers tests étaient concluants puis nous avons découvert que la fonction ne prenait pas en compte tous les cas.
Nous devions donc revoir nos idées. Au final on s'est rendu compte que ce qui posait problème c'était notre compréhension des arbres ternaires de recherche, nous croyons comprendre leur structures mais ça n'était pas vraiment le cas. Après lecture attentive de la définition d'un arbre ternaire, nous avons repris le travail sur l'insertion et avons réussi son implémentation.

Suppression

Après l'insertion, nous avions compris que la suppression n'allait pas être facile mais le travail fourni précédemment nous permettait de mieux appréhender la suite surtout grâce à notre meilleure compréhension de la structure des arbres ternaires de recherche. Les premières idées consistaient à compter le nombre de "\0" pour connaître le nombre de mots d'un arbre c'est pourquoi nous avons créé une fonction annexe **nb_mots_restants** qui s'en charge.

Les tests fonctionnaient, mais encore une fois, seulement pour certains cas. Nous devions préciser notre manière de compter les '\0', quels sous arbres était concerné par ce comptage, c'est pourquoi il y a **trois cas distincts** de comptage des "\0" dans notre fonction.

Un cas où l'on compte pour le fils du nœud du milieu, un autre pour son frère droit et un dernier pour son frère gauche. Ainsi après un parcours de recherche dans l'arbre lorsque l'on trouve un sous arbre qui ne possède qu'un seul "\0"c'est lui qui est supprimé. En effet, si le parcours de recherche nous à amené jusqu'à ce nœud c'est que c'est lui qu'il faut supprimer.
Maintenant que nous savons comment trouver les nœuds à supprimer il faut veiller à garder ses frères c'est pourquoi nous avons implémenté une deuxième fonction annexe **inserer_ATR_dans_ATR** qui fonctionne un peu comme l'insertion et qui sert à rebrancher ces frères perdus avec le reste de l'arbre. Avec tout ça et beaucoup de tests, la fonction de suppression était terminée.

Accents

Dans le sujet il est écrit qu'on travaillera seulement sur des fichiers avec des lettres minuscules a-z et des points. Cependant dans certains fichiers à corriger il y avait des accents. Comme par exemple “débarquées” ou “à”. Ce qui nous a posé problème au début car nous avons cherché un moyen de s'en débarrasser. On s'est rendu compte que cela est compliqué de travailler avec des accents en langage C et avons déduit que cela devait seulement être une erreur de sujet étant donnée que d'autre mot comme “était” n'avait pas d'accent dans les textes à corriger.

Documentation utilisateur

Cette partie est assez rapide car pour l'utilisateur il suffit seulement de lancer l'exécutable correcteur_0 avec en premier paramètre son texte à corriger et en deuxième paramètre un dictionnaire. Le programme va ensuite afficher sur la sortie standard la liste chaînée contenant tous les mots qui ne sont pas présents dans le dictionnaire donnée précédemment.

Exemple :
./correcteur_0 a_corriger_0.txt dico_3.dico0

Conclusion

Le niveau de travail fourni était presque exponentiel. Les fonctions de listes chaînées n'ont pas poser trop de difficultés. L'insertion dans l'arbre ternaire de recherche montait tout d'un cran mais ça n'était pas grand chose à côté de la suppression qui nécessitait une grande compréhension de l'insertion pour au moins commencé à être écrite. Nous avons travaillé ensemble sur toutes les fonctions en appelle vocal sur discord à part pour certaines comme la suppression mais nous avons toujours veillé à avoir un suivi du travail de l'autre au cours de notre travail à l'aide de partage d'écran régulier.