Thomas Mertz IMA3

Projet Programmation Avancée

Réalisation d’un correcteur orthographique

Polytech’ Lille

Informatique Micro-électronique Automatique
2018/2019

Sommaire

I. Présentation du projet

Le but du projet est de réaliser un programme qui détecter les erreurs dans un texte à partir d'un dictionnaire. Ce dernier pouvant être une liste de mots, un texte ou tout autre document.

Pour commencer, il fallait imaginer une structure de données minimisant l'usage de la mémoire. De plus, il fallait proposer une solution rapide que ce soit pour charger le dictionnaire ou analyser le texte.

Enfin, il fallait proposer un programme ergonomique. Le dictionnaire et le fichier à analyser doivent être facilement choisi par l'utilisateur. Avec également l'utilisation d'une interface claire pour l'affiche des résultats d'analyse.

II. Explication du programme

Le programme d'analyse de fichier accepte tous les caractères de l'alphabet latin (accents compris) mais également les caractères d'autres alphabets (cyrillique, kanji, …). Il est possible de modifier les caractères séparateurs d'un texte (tirets, accolades, apostrophes) pour que le dictionnaire soit correctement enregistré et que le fichier soit correctement analysé.

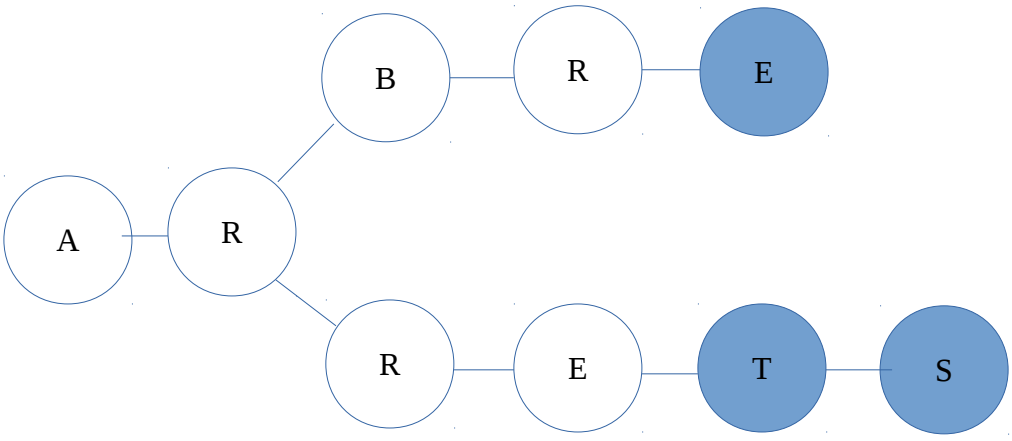
L'utilisation précise du programme est expliquée dans le fichier « READme.md ». Il est possible d'entrer les noms des fichiers en argument du programme ou directement pendant son utilisation (dans ce cas les nom du ne devront pas contenir d'espaces).

Le programme indiquera le nombre de mots non reconnus mais les également. Cependant il ne prend pas en compte la casse des caractères.

1. Structure de données

Le programme stock les mots du dictionnaire sous la forme d'arbres, le but étant de limiter l'utilisation de mémoire en ne stockant pas les lettres communes entre plusieurs mots.

Exemple en enregistrant les mots « arbre », « arrets » et « arret » :



Les lettres en bleue étant des lettres de terminaison de mots.

Chaque nœud de l'arbre (les cercles sur le schéma précédent) peuvent avoir autant de « fils » (lien avec d'autres nœuds) que nécessaires à l'enregistrement du dictionnaire en mémoire. Chaque nœud connaît ainsi l'adresse de tous ses fils, qui ne sont pas ordonnés.

Pour augmenter la vitesse du programme, un arbre est créé pour chaque lettre de début de mots. Ceux-ci étant stocké dans un tableau dynamique : rapidité pour trouver les mots de « a » à « z ».

2. Structuration du programme

On peut simplifier le programme en deux parties. La première étant celle de création du dictionnaire, la seconde celle de recherche des mots du fichier à analyser.

a) Création du dictionnaire

Initialement on a un tableau de 26 pointeurs d'arbres vides (un pour chaque lettre de l'alphabet latin). Lors de la lecture du dictionnaire entrée en paramètre on va commencer par lire chaque ligne et la passer en minuscule.

Chaque ligne va être divisée suivant les séparateurs donnés par l'utilisateur et dans tous les cas par chaque espace.

Chaque mot pourra ainsi être ajouté au dictionnaire. S'il ne commence pas par une lettre de « a » à « z » on modifie le tableau de pointeurs d'arbres pour l'ajouter si nécessaire.

Lorsque l'on ajoute un mot on va regarder si la lettre que l'on souhaite ajoutée n'existe pas déjà. Si c'est le cas on regarde si l'un des « fils » correspond à la lettre suivante et ainsi de suite jusqu'à la fin du mot. Par contre, si la lettre n'existe pas on crée un nouveau « nœud » et on modifie le nœud précédent pour qu'il puisse également pointer sur cette nouvelle lettre.

Lorsque l'on arrive à la fin d'un mot on modifie le nœud pour que l'on sache qu'un mot peut se terminer ici.

Il est également indispensable de libérer la mémoire avant la fin du programme. Pour cela on va libérer la mémoire de chaque arbre : Si un nœud à un fils on va d'abord voir si ceux-ci n'en n'ont pas aussi. Si ce n'est pas le cas, on peut libérer cette place mémoire, et remonter de cette manière dans l'arbre.

b) Recherche dans le dictionnaire

La recherche dans le dictionnaire est assez similaire dans la forme à la création de celui-ci. On va également lire chaque ligne du fichier, passer tous les caractères en minuscules et les diviser suivant les séparateurs et les espaces.

On va ainsi regarder pour chaque mot si la première est bien présente dans le dictionnaire. Ensuite on regardera si les « fils » de cet arbre contiennent la lettre suite et ainsi de suite jusqu'à la fin du mot.

Si chaque lettre du mot est bien présent et que la dernière lettre est bien notée comme une lettre de fin de mot alors le mot est bien juste. Sinon on affichera ce mot et notre compteur d'erreurs augmentera.

Le programme principal va simplement demander à l'utilisateur son dictionnaire et le fichier à tester, mais également les caractères de séparation. Il affichera aussi le nombre total de mots non reconnus. L'utilisateur pourra tester plusieurs fichiers avec le même dictionnaire sans relancer le programme.

III. Limites du programme

Le programme suit le cahier des charges donné par le client. Cependant il existe quelques limites qui sont expliquées ci-dessous :

Une ligne est limitée à 3000 caractères, au-delà de ce chiffre le programme plantera. Ce nombre correspondant à une page de texte complète ce qui semble être suffisant.

Il n'existe pas d'interface graphique pour améliorer l'ergonomie du programme. (Cela aurait été plus simple pour sélectionner les fichiers, cependant 3 dictionnaires sont disponibles facilement pour l'utilisateur.) Et les noms des fichiers ne peuvent pas contenir des espaces sauf s'ils sont utilisées en argument à l'appel du programme.

Le programme n'est pas sensible à la casse. Cette fonctionnalité pourrait être désactivée très simplement en supprimant l'appel à la fonction « toLowerCase ». Cependant la gestion de casse pourrait entraîner un plus grand nombre de mots non reconnus dans le dictionnaire car ayant/n'ayant pas de majuscule (par exemple en début de phrase pour un mot). Un autre intérêt est que cela limite l'utilisation de la mémoire.

On pourrait augmenter la vitesse du programme en trillant les lettres lorsqu'on les ajoute (suivant les codes ASCII) pour améliorer la recherche des mots.