DM - Correcteur Orthographique

Rendu 1

MELLOUK Imân – MENSAH ASSIAKOLEY Ako Seer Harley

20/03/2022

Table des matières

OBJECTIF

L'objectif de ce projet est d'écrire une application qui sera enfaite un correcteur orthographique. A cet effet, l'utilisateur pourra charger un dictionnaire servant de référence pour permettre la correction d'un texte(repérage des mots mal orthographiés, puis proposition de correction).

Dans ce premier rendu, l'objectif étant de trouver les mots mal orthographiées (relatif au dictionnaire de référence).

DOCUMENTATION UTILISATEUR

Entrée / Sortie

- Entrée : le programme prend en argument un dictionnaire de référence ainsi que le texte à corriger. Ces fichiers se trouvant dans le répertoire **ressources.**

- Sortie : Sur le terminale tous les mots considérés comme mal orthographié sont affichés

Exécution du programme

Afin de lancer le programme, il faut se situer dans le répertoire principal puis écrire la commande . /make qui créera un exécutable correcteur_0.

Puis taper la commande suivante :

./correcteur_0 ressources/[nom du texte à corriger] ressources/[nom du dictionnaire de ref]

Par exemple, pour lancer le programme avec le texte à corriger a_corriger_0.txt avec pour référence le dictionnaire dico_1.dico. On écrit la commande suivante :

./correcteur_0 ressources/a_corriger_0.txt ressources/dico_1.dico

DOCUMENTATION TECHNIQUE

Structures de données

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

1. Le champ racine de type char, qui contiendra le caractère n-ième d'un mot.
2. Le champ fils, il s'agit d'un arbre contenant les (n + 1)-ieme caractères du mot.
3. Le champ gauche, est arbre contenant les mots dont le caractere n-ieme est inférieur au caractère racine.
4. Le champ droite, est un arbre contenant les mots dont le caractère n-ieme est supérieure au caractère racine.

Pour stocker les mots dit mal orthographiés, on les stockeras dans une liste chainées. Chaque cellule de liste chainée contiennent 2 champs ;

1. Le champ mot de type char *, qui contiendra un mot.
2. Le champ suivant qui est un pointeur permettant le chainage entre les différentes cellules.

Le dictionnaire de référence ainsi que le texte à corriger sont tout les deux des fichiers externes, respectivement d'extension .dico et .txt .

Fonctions

Pour permettre la distinction des mots correctes des non-correctes il requiert plusieurs étapes et fonctions importante.

Implémentation des structures principales.

Dans ce projet, nous allons stocker tous les mot que peux contenir un dictionnaire dans un ATR (arbre ternaire de recherche). C'est sur cette arbre préfixe que nous effectuerons un parcours pour permettre la correction orthographique.

Le module ATR contient les fonctions principales pour manipuler un arbre de type ATR

- ATR creer_ATR_vide(void):

  Cette fonction alloue de la mémoire pour un arbre ternaire de recherche. Si l'allocation est réussie, il initialise la racine au caractère nul et renvoie l'arbre. si non, ils renvoie NULL.

- void liberer_ATR(ATR * arbre):

  Cette fonction permet la libération d'un arbre ternaire de recherche

- int est_vide_ATR(ATR arbre):

  Cette fonction détermine si l'arbre ternaire de recherche est vide ou non. Est considéré comme vide, l'ATR dont la racine est le caractère nul et les champs fils, gauche, droite sont à NULL; selon la construction de notre arbre, racine = '\0' et droite = NULL suffit à conclure. Elle renvoie 1 si oui, ou 0, si non.

- int inserer_dans_ATR(ATR * arbre, char * mot):

  Cette fonction gère l'insertion des mots dans un dictionnaire de la forme d'un arbre ternaire de recherche "arbre". Elle s'aide de **int inserer_aux(ATR * arbre, char * mot, int cmp)** pour chercher la place exacte selon l'ordre lexicographique des mots de l'arbre et insère caractère par caractère le mot grâce à **int ajout(ATR * arbre, char * mot, int cmp).** Cette fonction renvoie 1 si le mot a bien été inséré et 0 si non.

- int recherche(ATR arbre, char * mot, int cmp):

  Cette fonction permet de rechercher un mot donné dans un dictionnaire “arbre”

- void supprimer_dans_ATR(ATR * arbre, char * mot):

  Cette fonction permet de supprimer un mot d’un dictionnaire représenté par un arbre ternaire de recherche arbre si il y appartient, en appelant **int supprimer_dans_ATR_aux(ATR * arbre, char * mot, int cmp)** qui se charge de supprimer caractère par caractère et de **int inserer_arbre(ATR * arbre, ATR droite)** qui raccorde les branches gauche et droite de l’arbre(dans le dico_1, si on devait supprimé “freres” on doit pouvoir raccorder “etait”et “gare”) en n’oubliant de ne pas perdre de l’information (ne pas supprimer trop de caractères de peur de perdre les autres mots du dictionnaire)

- void afficher_ATR(ATR arbre):

  Cette fonction permet d’afficher les mots d’un dictionnaire dont la représentation est l’arbre ternaire de recherche “arbre” en paramètre.

  Elle utilise la fonction **void affiche_atr_aux(ATR arbre, char * mot, int cmp)** afin de remplir un buffer “mot” qui contiendra tout à tour les mots trouvés dans l’arbre et l’affiche sur la sortie standard

- ATR remplis_arbre(FILE * fichier) :

  Cette fonction, à partir d’un fichier ouvert “fichier” qui est un dictionnaire, insère les mots qui s’y trouvent un à un afin de constituer un Arbre Ternaire de Recherche qui est renvoyé

Nous stockons les mots incorrect dans une liste chainée simple, où l’insertion se fait en tête.

Pour l’insertion, si la chaine n’est pas vide, on utilise une liste chainée temporaire tmp qui stocke la tête de lecture de la liste L (pour ne pas perdre le chainage). L pointe désormais vers la nouvelle cellule alloué, son champ suivant pointe vers la liste temporaire.

Pour la libération de la liste, on parcours la liste jusqu'au bout, pour chaque cellule, on rencontrées on la free.

Traitement de fichier

Notre dictionnaire ne contient que des mots en minuscules et sans ponctuation. Ce qui n'est pas forcément le cas du texte à corriger passé en argument. Nous avons donc crée le module fichier, qui contient 3 fonctions permettant d'à partir un fichier, obtenir un nouveau fichier dit traité, c'est-à-dire sans majuscule ni ponctuation.

La fonction **traitement_fichier**, prend en argument le fichier à traité. On effectue une boucle sur le fichier, tant qu'on pas parcourus entièrement le fichier, on récupère le mot avec la fonction fscanf puis on retire les majuscules que peut comporter celui-ci, on stocke ce mot dans une autre variable, puis sur cette même variable on retire les ponctuation s'il y en a. Enfin on écrit le mot final obtenu dans le fichier **traite.txt.** Ainsi ce fichier contient le fichier à corriger de départ sans majuscule ni ponctuation. C'est désormais sur ce celui-ci que l'on va utiliser pour la suite du programme.

Recherche des mots mal orthographiés

Dorénavant, il ne reste plus qu'à rechercher les mots mal orthographié. C'est la fonction **correction** qui le permet. L'algorithme de recherche est le suivant :

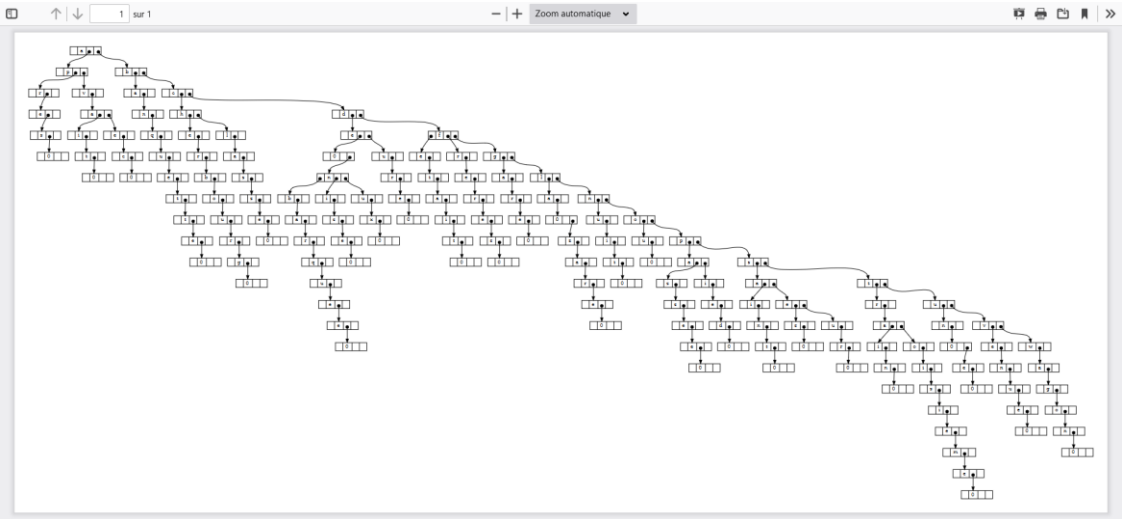
Dans un premier temps on initialise une liste chainée vide, cette liste stockera tout les mots mal orthographié. **Rappel** : un mot est considéré comme mal orthographié s'il n'est pas présent dans le dictionnaire de référence. Il s'ensuit que pour chaque mot du fichier **traite.txt,** on appel la fonction **recherche** qui renvoi un booléen si un mot est présent ou non dans l'arbre (arbre représentant le dictionnaire). Si le retour vaut 0, alors on insère ce mot dans la liste chainée. Finalement on renvoie cette liste.

Puis on l'affiche avec la fonction **affiche_liste**, pour permettre à l'utilisateur de savoir quels mots sont mal orthographié.

Outil de travail

Afin de faciliter et de vérifier l'implémentation de notre arbre lexicographique, nous avons créé un 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.

Un exemple du fichier après création de l'arbre obtenue ayant pour référence le dico_1.dico



Pour l'écriture de certaine fonction nous les avons d'abord imagés en dessinant ce que l'appel de la fonction devait produire, utile aussi pour énumérer tous les cas possible.

Parfois nous avions eu recours à C-tutor, qui permet de visualiser l'exécution de son code pas à pas, pour mieux comprendre certain bug.