Rapport de Projet

Rendu 2

Algorithmique des Arbres

Gaël Paquette
Hervé Nguyen
2 avril 2022
L2 Informatique

Table de matières

Introduction

Ce second rendu consiste à implémenter une correction orthographique avec l'utilisation de la distance de Levenshtein.

Il nous est alors requis d'écrire un module Levenshtein afin de pouvoir l'utiliser pour calculer des distances pour déterminer des mots qui peuvent être des solutions possibles.

Pour ce second rendu , nous avons repris notre rendu 1 et puis nous avons crée un module Levenshtein pour implémenter une correction orthographique.

Module Levenshtein

Dans ce module, la fonction la plus importante est la fonction « int Levenshtein(char * un, char *deux) ». Celle-ci est permet de calculer la distance de Levenshtein entre les mots « un » et « deux ».

Cette fonction reprends l'algorithme de calcul de la distance de Levenshtein avec des tableaux alloués sur le tas.

Il y a alors quelques fonctions qui servent à initialiser ces tableaux, ces fonctions prennent en paramètre les dimensions du tableau voulu car la taille ces tableaux varient en fonction des mots à calculer.

Pour alors obtenir la distance de Levenshtein, il suffit donc de calculer la distance de Levenshtein les « sous-suites » de lettes deux mots en s'approchant au fil des itérations (dans la boucle) vers les deux mots entiers.

Algorithme 2

Pour l'algorithme 2 décris dans le sujet, cet algorithme se trouve dans la fonction « static void suggestion() » dans le fichier source du second rendu Main_1.c.

Pour faire une boucle avec une itération pour chaque mot du dictionnaire nous avons utilisé un while avec fscanf sur le pointeur FILE* sur le fichier du dictionnaire.

Et le calcul de la distance se fait avec l'appel de la fonction Levenshtein sur le mot du dictionnaire de l'itération actuelle et le mot à corriger.

La liste de suggestion/correction sera mise à jour au fur et à mesure des itérations.

Après cela il suffit alors d'afficher le contenu de la liste de correction pour montrer des potentiels candidats de correction.

Structuration du rendu

Le rendu contient les fichiers en-têtes des modules dans le sous répertoire « ../include »

Les fichiers sources sont dans le sous répertoire « ../src ».

Le makefile, les fichiers régénérables, le rapport se trouvent au répertoire courant du rendu.

Voici la liste des modules :

- ATR (Permet d'avoir une implémentation d'un dictionnaire)
- Listes (Implémentation de listes chainées simples)
- Levenshtein (Permet de calculer des distances de Levenshtein)

Voici les fichiers « main » :

- Main_0.c (Fichier main du rendu 1 pour le programme correcteur_0)
- Main_1.c (Fichier main du rendu 2 pour le programme correcteur_1, affiche des suggestions de correction en plus du comportement habituel de correcteur_0)

Deux fichiers textes sont fournis :

- a_corriger_0.txt
- a_corriger_1.txt

Trois fichiers dictionnaires sont fournis :

- dico_1.dico
- dico_2.dico
- dico_3.dico

Addendum

Lors de l'élaboration du module Levenshtein, nous nous somme heurté à un problème de programmation où lors de l'exécution il y aurait eu une « corruption » du tas (HEAP CORRUPTION DETECTED).

Le problème s'avère être dû à une mauvaise manipulation des tableaux dynamiques à deux dimensions, où dans les boucles imbriqués pour calculer les distances des « sous-suites » nous avons itéré de 1 à len_un dans la boucle principale et aussi de 1 à len_un dans la sous-boucle.

Nous avons également corrigé l'affichage du programme du rendu 1. Où le programme n'affichait que les mots erronés mais pas :

« Mot(s) mal orthographié(s) : \n»

Comme requis dans le sujet.

Comment compiler les programmes ?

Nous avons fourni un makefile qui permet de compiler les deux programmes de ce rendu.

Il suffit de faire :

```
make
```

Pour supprimer les fichiers objets :

```
make clean
```

Pour supprimer la totalité des fichiers régénérables :

```
make mrproper
```

Comment utiliser les programmes ?

Le premier programme est correcteur_0.

Celui-ci permet d'afficher les erreurs contenu dans le fichier texte passé en argument selon le dictionnaire aussi passé en argument.

Pour le lancer, il suffit de faire :

./correcteur_0 [fichier_texte] [fichier_dictionnaire]

Le second affiche les erreurs et aussi des suggestions de correction pour chaque mot erronés

Pour le lancer il suffit de faire :

./correcteur_1 [fichier_texte] [fichier_dictionnaire]

Conclusion

Hormis ce qui a été dit précédemment, il était nécéssaire d'avoir une vision d'ensemble sur les objectifs requis. La compréhension des relations entre chaque module et leur utilité est aussi un point important.