Rapport de projet d'algorithmique 2 : structures récursives, linéaires et binaires

KLAK Karolina et MASSIAS Antoine [1]

Janvier 2022

[1]Rapport réalisé avec laTex

Contents

Chapter 1

Généralités

Ce projet constitue l'aboutissement du projet d'algorithmique 2, du semestre 3 de la licence d'informatique à l'UFR Sciences et Techniques de Rouen.

Le projet a été réalisé par A. Massias et K. Klak dans le cadre de leurs études. Ce projet consiste en la réalisation, en C, d'un exécutable produisant sur la sortie standard une liste de mots partagés. En effet, cet exécutable doit, au terme de plusieurs noms de fichiers inscrits sur la ligne de commande de trier les mots figurant dans chacun d'eux, et d'en conclure une liste de "mots" dits "partagés" entre les fichiers, tout en spécifiant leur motif et leur nombre total d'occurences. A titre informatif, le motif fait référence aux fichiers dans lesquels ledit mot apparaît. Il se compose d'une suite de 'x' ou '-' en focntion de la présence ou non du mot dans chacun des fichiers.

1.1 Spécifications

La liste dont il est question est triée selon trois filtres :
-ordre décroissant du nombre de fichiers dans lesquels le mot figure,
-ordre décroissant des nombres totaux d'occurrences,
-ordre lexicographique croissant des mots.

Une ligne est réalisé pour chaque mot, avec le descriptif ci-dessus. Chaque liste est composée de dix lignes, limite fixée par défaut qu'il est possible de modifier par le biais d'options (cf. 1.2 Options). Une autre limitation fixée par défaut : le nombre de caractères de chacun des mots étant pris en compte. En effet, par défaut seuls les 63 premeirs caractères d'un mot sont pris en compte. Au-delà de cette limite, le mot est coupé.

Exemple : L'exécutable produit par défaut les lignes de textes suivantes :
$ ./ws textes/tartuffe.txt textes/lesmiserables.txt
./ws: Word from 'textes/lesmiserables.txt' at line 33239 cut: 'paroisse-meurs-de-faim-si-tu-as-du-feu-meurs-de-froid-si-tu-as-...'.
./ws: Word from 'textes/lesmiserables.txt' at line 62928 cut: '–L'agent-de-police-Ja-vert-a-ete-trouve-noye-sous-un-bateau-du...'.
xx 22021 de
xx 15256 la
xx 13062 et
xx 11685 a
xx 11194 le
xx 7252 les
xx 6114 un
xx 5667 que
xx 5444 il
xx 5375 qui

Il y a précision si des mots ont été coupés. Ainsi comme convenu seules 10 lignes de textes sont affichées.
Le motif ici est 'xx' pour chacun des mots de la liste : cela signifie que les mots apparaissent dans les deux fichiers.
Le nombre qui suit indique le nombre d'occurrences totales, dans tous les fichiers.
Enfin, en toute logique, la dernière chaine de caractères de la ligne correspond au mot, entier ou coupé.

1.2 Options

En totalité, cinq options coexistent : deux d'entre elles concernent le nombre de lignes de mots, une concernant la longueur maximale du préfixe du mot, et enfin deux sur la notion du mot.
En outre :
- '-t' : limite du nombre de lignes de textes (limite du nombre de mots à produire)
- '-s' : production de lignes supplémentaires de tous les possibles mots dont le motif et le nombre d'occurrences sont égaux à ceux du dernier des mots associés à la limite
- '-i' : fixer la longueur maximale du préfixe de chaque mot
- '-p' : permet de ne plus considérer une marque de ponctuation comme un "mot"
- '-u' : permet de convertir un caractère correspondant à une lettre minuscule en un caractère de lettre majuscule

Exemple : ci-dessous les lignes de textes en sortie avec l'utilisation des 5 options.

$ ./ws -t 12 -s -i 5 -p -u textes/toujoursetjamais.txt textes/lecid.txt textes/lesmiserables.txt textes/lavare.txt
...
./ws: Word from 'textes/lecid.txt' at line 34 cut: 'TEMPS...'.
./ws: Word from 'textes/lecid.txt' at line 34 cut: 'VAILL...'.
xxxx 18020 LA
xxxx 14920 ET
xxxx 13995 A
xxxx 13407 LE
xxxx 11824 L
xxxx 8742 UN
xxxx 8396 LES
xxxx 6531 D
xxxx 6246 UNE
xxxx 6169 EST
xxxx 5943 QUI
xxxx 5671 DANS

Chapter 2

L'algorithme

Dans ce chapitre, nous exprimerons l'algorithme général et son fonctionnement.

2.1 Général

Notre projet se base sur le module hashtable présentée t travaillé en cours et en tp, lors de ce semestre, auquel il était interdit d'apporter des modifications. Des modules supplémentaires ont été développés : lang context et linked list. Ces modules seront présentés dans une section prochaine.

Prenons deux points de vue. D'un point de vue utilisateur, ce dernier écrit en ligne de commande dans le bash, en faisant attention d'y inclure l'exécutble, les options s'il le veut, ainsi que les fichiers textes. Il en résulte une liste, à ligne à 3 éléments, dans laquelle on peut remarquer des mots figurant dans les fichiers , triés selon les 3 conditions expliquées dans la *section options.*. L'utilisateur peut également choisir d'écrire sa propre suite de mots en entrée standard.

Le point de vue développeur est quant à lui plus intéressant à développer.

En effet, derrière cet algorithme, se trouvel'usage de plusieurs structures de données, chacune ayant son interèt, de modules, étudiés ou non en cours. Lorsque l'utlisateur inscrit sa ligne de co;;ande, une fonction la 'parse', et vérifie chaque caractère, s'il y en a, de la ligne, pour en tirer la fonction, savoir de quelle potentielle option il peut s'agir, en triant d'avance selon option longue courte, fichier. Ces caractères ou suite de caractères selon les cas, sont stockés. Lorsqu'un nom de fichier est détecté, celui-ci est immédiatement enregistré, et ouvert. Il y a lecture de ce dernier, et tous les mots vérifiant les conditions (en fonction des optios données), sont stockés.

2.2 Structure de données

Nous avons utilisé plusieurs types de structures de données dans le but d'utiliser au mieux certaines de nos fonctions.

Nous avons notamment utilisé les tables de hachage, afin de gérer notre mot et les caractéristiques qui lui sont affiliées; c'est-à-dire le mot en lui-même (chaine de caractères), le nombre d'occurences et le motif. Nous avons donc une table pour nos mots. Cela permet de gérer au mieux notre temps, et la logique de l'algorithme.

Une autre structure intéressante utilisée : les listes chaïnées. Leur utilisation est bénéfique pour les fichiers, garder leur nom en mémoire, après une saisie de l'utilisateur.

Chapter 3

Le descriptif

Dans ce chapitre, les différents modules seront expliqués, et leur but sera fondé.

3.1 Les modules

Il y a eu utilisation de plusieurs modules, dont l'idée de développement est parfois survenue en plein projet, suite à une volonté d'améliorer ce dernier

Dans un premier temps, vous remarquerez l'utlisation des modules hashtable et holdall, modules travaillés en cours qui permettent une gestion très performante des mots, grâce à la table de hachage et la fonction. Cela ayant déjà été travaillé, intéressons-nous aux autres modules.

Le *context* est un sorte de moyen de stockage de toutes les variables, dans un seul et même endroit donc. Ce context offre une oportunité de ne paas avoir à passer des apramètres dont on pourrait se passer, lorsque l'on gère des données, mais à n'avoir qu'à passer le context. D'un point de vue logique, il est donc beaucoup plus simple de s'y retrouver donc et éventuellement penser à des optimisations.

Bien que les gestions fondamentales, se passent dans le *main*, c'est dans le context qu'il y a tout le processing, toute cette gestion. Ainsi, en l'incluant dans le context, on permet de cacher la machinerie, de découper le code à notre aise, et le rendre bien plus lisible et clair.

Opt est également dans le lot. Ce module permet la egstion des différentes options et des messages d'erreurs et d'avertissement si besoin est. De même, une telle découpe permet un code plus clair et surtou une meilleure gestion s'en concluant.

Un autre module intéressant : le linked list. On reprend ici un simple principe de

liste chaïnée, que l'on a étudié et on en crée un module pour l'utiliser à notre aise dansle projet. C'est ce module qui nous permet de garder les fichiers en mémoire, les gérer, par le biais de liste donc chaïnée.

Enfin, lang, un module permettant plusieurs idées que l'on affectionne. En effet, on peut grâce à celui-ci de choisir la langue des différents messages qui s'affichent en sortie, anglas, français... Le tout étant donc une amélioration du projet, pour agrandir la visée. Par ailleurs, on y stocke donc les messages, afin de créer un code clair, lisible, et aéré.

Chapter 4

Les limites

Nous approcherons la question des différentes contraintes auxquelles on a fait face, ainsi que les problèmes que l'on a pu rencontrer.

4.1 Contraintes/limites

Nous avons rencontré une limite, si telle est-elle. Il s'agit de la gestion de lecture de mots en entrée standard.

Par ailleurs, on ne recense pas d'autre contrainte à l'heure actuelle. Nous avons réussi à nous organiser, et permettre on codage soutenu, grâce à l'utilisation de GIT, ce qui nous permettait de partager et mettre à jour quasi instantanément nos avancées respectives sans créer de problémes, de retards chez l'autre.

4.2 Les problèmes rencontrés

A ce jour, on recense deux problèmes assez conséquents : l'optimisation et la lecture desmots sur l'entrée standard au lieu de les lire sur fichier. En effet, ainsi le projet ne remplit pas toutes les conditions requises pour qu'il soit complet. Néanmoins nous nous sommes concentrés sur la gestion de l'espace, et tout optimisation possible afin de vous délivrer un code des plus propres et lisibles que l'on puisse faire en ce temps imparti.

Par ailleurs, lorsque l'on compare les espaces utilisés avec l'exécutable de référence, on s'aperçoit qu'il y a un facteur 4.

C'est un cas que nous jugeons important car nous portons un intérèt majeur quant aux différentes améliorations que nous pourrions apporter au projet.