Distance de Jaccard

Maysaloon BILAL & Tristan GROULT

13 mai 2025

Sommaire

1 Introduction

Ce projet consiste à développer un programme en langage C permettant de calculer la distance de Jaccard entre plusieurs fichiers texte, afin d'analyser leur similarité lexicale. La distance de Jaccard est une valeur comprise entre 0 et 1 : plus elle est proche de 0, plus les fichiers sont différents, et plus elle est proche de 1, plus ils sont similaires.

La distance de Jaccard entre deux ensembles A et B est définie par la formule :

$$\text{Distance de Jaccard}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Le programme propose plusieurs modes d'affichage ainsi que des calculs personnalisés. Il est possible de mesurer la similarité entre les fichiers à l'aide de l'indice de Jaccard, que nous désignerons par la suite par *Calcul de Jaccard*. Il est également possible d'afficher un graphe représentant l'appartenance des mots à leur(s) fichier(s), que nous nommerons *Graphe de Jaccard*. Nous utiliserons le terme *Calcul de Jaccard* pour faire référence à la réalisation ou la préparation de ces deux opérations.

Pour gérer efficacement les mots extraits des fichiers, nous avons utilisé une table de hachage. Sa gestion — ajout, suppression, affichage du graphe, etc. — est implémentée dans le module `jaccard`. Ce projet s'appuie également sur deux autres modules : le module `word` sous-section 2.1, dédié à la création et la manipulation des mots, et le module `opt` sous-section 2.2, responsable du traitement des options en ligne de commande.

L'ensemble de ces composants est intégré et testé dans le fichier principal `main.c`, qui constitue le point d'entrée du programme.

2 Conception et implémentation

Dans cette section, nous expliquons les différentes étapes et approches utilisées pour implémenter le programme. Nous détaillerons la conception des modules, la gestion des fichiers et l'utilisation de la table de hachage.

2.1 Module word

Le module `word` a pour objectif de construire dynamiquement un mot caractère par caractère, afin de faciliter son traitement sans avoir à connaître sa taille à l'avance. Ce module est utilisé lors de la lecture des fichiers afin d'extraire chaque mot individuellement.

2.1.1 Spécification

Pour cela, nous utilisons une structure de données définie dans la figure 1. Cette structure permet de stocker un mot, d'en gérer dynamiquement la taille, et de le manipuler facilement sans risque de dépassement mémoire.

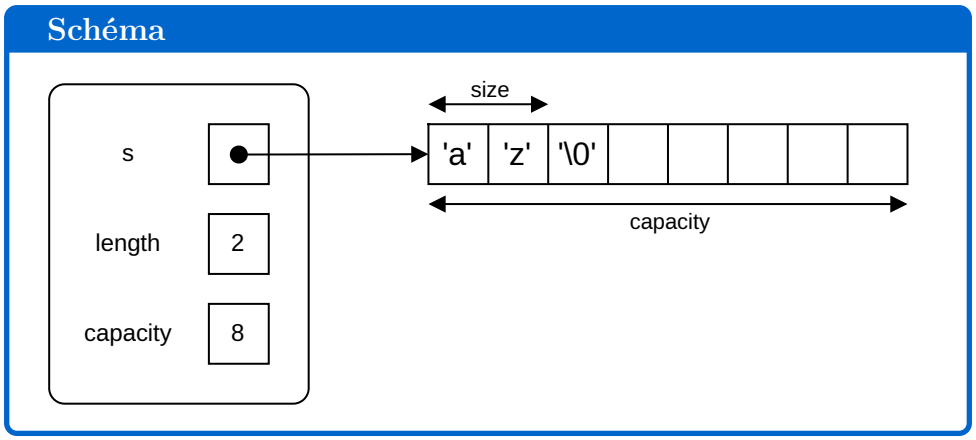

FIGURE 1 – Schéma de la structure word

2.1.2 Implantation

Lors de l'implémentation, nous avons défini une structure `struct word` (figure 2) pour représenter dynamiquement un mot. Elle est composée de trois champs :

— Un champ `s` de type `char *` qui pointe vers un tableau de caractères contenant le mot.
— Un champ `length` de type `size_t` qui représente la longueur actuelle du mot.
— Un champ `capacity` de type `size_t` qui correspond à la taille totale allouée pour le mot.

Nous avons choisi une approche avec allocation mémoire dynamique, ce qui permet de s'adapter à la longueur des mots rencontrés, en multipliant par la constante `CAPACITY_MUL` la capacité du mot lorsque celle ci est atteinte.

```
struct word {
    char *s;
    size_t length;
    size_t capacity;
};
```

FIGURE 2 – Structure word

Nous avons développé plusieurs fonctions pour interagir avec ce module :

— `word_init()` : Initialise un mot vide en allouant une capacité minimale définie par la constante `CAPACITY_MIN`, et met la longueur à zéro.
— `word_add()` : Ajoute un caractère à la fin du mot. Si la capacité est atteinte, elle est doublée avec `realloc()` pour permettre l'ajout de nouveaux caractères.
— `word_reinit()` : Réinitialise le mot en mettant sa longueur à zéro, ce qui permet de le réutiliser sans réallouer de mémoire.
— `word_get()` : Renvoie un pointeur vers la chaîne de caractères actuelle.
— `word_get_clean()` : Copie le contenu réel du mot dans une destination passée en paramètre, sans les éventuels caractères résiduels.

2.2 Module opt

Le module `opt` a pour objectif de gérer les options de la ligne de commande, en les analysant puis en les restituant sous une forme exploitable par le reste du programme. Dans le cadre de ce projet, les options suivantes sont nécessaires :

- `-g` : Affiche le graphe de Jaccard.
- `-p` : Ignore les caractères de ponctuation lors de l'analyse.
- `-iVALUE` : Spécifie la longueur maximale des mots.
- `-` : Utilise l'entrée standard comme source.
- `--` : Indique que l'argument suivant est un nom de fichier.
- `-u` : Affiche les informations d'utilisation de l'exécutable.
- `-v` : Affiche la version de l'exécutable.
- `-?` : Affiche l'aide de l'exécutable.

2.2.1 Spécification

Pour représenter les options analysées, nous utilisons une structure de données illustrée dans la figure 3. Nous avons choisi de limiter le nombre de fichiers grâce à une constante nommée `MAX_FILES`. Ce choix permet d'assurer une consommation mémoire bornée et facilite l'intégration avec les contraintes d'implémentation du module `jaccard`.

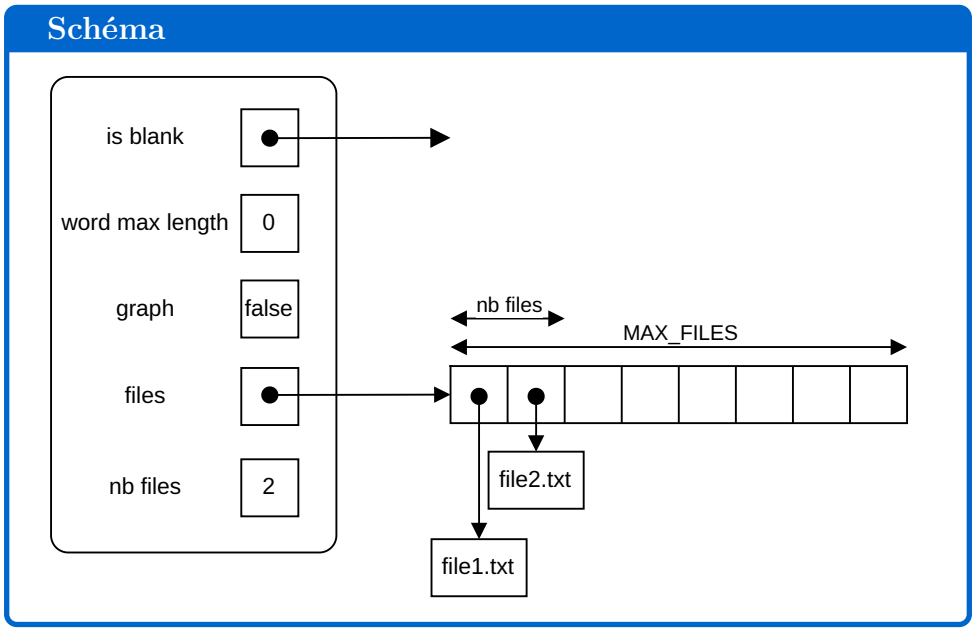

FIGURE 3 – Schéma de la structure opt

2.2.2 Implémentation

L'implémentation repose sur l'utilisation d'une structure `struct opt`, définie comme suit (figure 4) :

- `isBlank` : Pointeur vers une fonction `int(int)` déterminant les caractères séparateurs de mots.
- `word_max_length` : Entier définissant la longueur maximale d'un mot (0 signifie aucune limite ; valeur par défaut : 0).
- `graph` : Booléen indiquant si le graphe de Jaccard doit être affiché.

— `files` : Tableau de chaînes de caractères représentant les noms des fichiers à analyser.
— `nb_files` : Nombre de fichiers fournis en argument.

```
struct opt {
    int (*isBlank)(int);
    int word_max_length;
    bool graph;
    char const **files;
    int nb_files;
};
```

FIGURE 4 – Structure opt

Afin de permettre une gestion homogène de l'entrée standard, nous la représentons sous la forme d'une chaîne vide (`""`), en la définissant via une macro `STDIN`. Cette chaîne est utilisée comme nom de fichier, en tirant parti du fait qu'aucun système courant ne l'autorise comme nom valide.

Seules les options courtes ont été implémentées, comme listé plus haut. Pour l'option `-iVALUE`, la valeur doit être directement collée à l'option (par exemple `-i3`), l'espacement (`-i 3`) n'étant pas pris en charge.

Enfin, toutes les options sont définies via des macros constantes. Le préfixe des options courtes est également paramétrable, ce qui rend l'ensemble du module facilement adaptable à d'autres conventions si nécessaire.

2.3 Module Jaccard

Le module `jaccard` a pour objectif de réaliser le calcule de Jaccard en donnant la possibilité d'ajouter des mots au calcule ainsi que de gérer l'affichage du graphe ou le calcule de la distance.

2.3.1 Spécification

Pour réaliser cella, nous utilisons une structure de données définie dans la figure 5. Cette structure permet de sauvgarder les éléments utilisé pour réaliser le calcule de la distance ansi que les autres strucure donné que nous avons décidé d'utiliser.

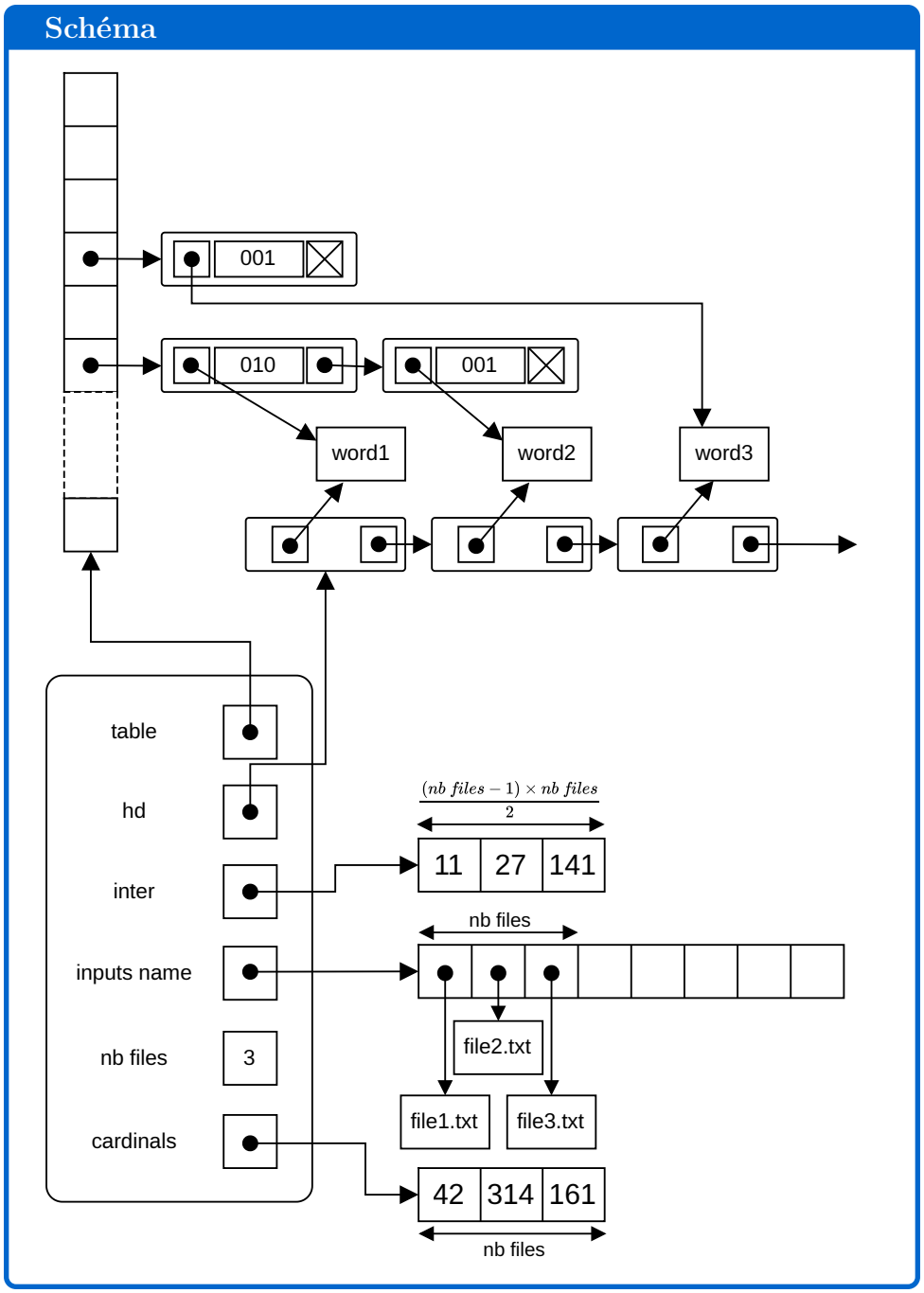

FIGURE 5 – Schéma de la structure jaccard

Pour réaliser le calcule de Jaccard, nous avons besoin :

— D'un champ $nb\ files$ permettant de conaitre le nombre de fichier utilisé pour le calcule de Jaccard courrant.

— D'un tableau *inputs name* contenant le nom des fichiers utilé lors de l'affiche du graph de Jaccard ou du calcule de Jaccard. L'ordre des fichiers est important car plusieur autres endroit utilise ce même ordre pour représenter ces même fichiers. On considère que ce tableau de nom de fichier est de taille $nb\ files$.

— D'un tableau consttitué des cardinaux de la forme ensembliste de chaque fichier. Le cardianle à l'indice i est cellui du fichier d'indice i dans inputs name. Ce tableau est donc de taille $nb\ files$.

— (*Explication complète de l'intersection (Maysaloon). Le principe du calcule de la taille et de chaque case liès à qu'elle intersection mais pas le systheme de byte derrière*)
— Un champ *table* conteant une table de hashage dans laquelle les mots vont pouvoir être recherché et ajouté durant le calcule de Jaccard
— Le champ *hd* contenant une liste dynamique en mesure d'acceuillir tout les mots qui sont ajouté au calcule de Jaccard. C'est sur cette liste dynamique que doit être réalisé le trie si neccessaire car une table de hashage n'est pas trié.

2.3.2 Implémentation

Pour la réalisation de ce module nous avons développer de nombreuse fonction afin de créer un calcule de Jaccard, d'ajouter des mots au calcule et finalement de procéder à l'affichage du calcule de Jaccard et du graph de Jaccard.

Nous avons rencontré un problème sur le calcule de l'appartenance de chaque mot à son/ses fichiers. (*Explication du calcule durant l'ajout de l'intersection de chaque mots et problème de pointeur et pas nombre puis résolution. (Peut être cour exemple pour expliqué la détection d'un certain bit à 1 ou 0)*).

— `jaccard_init()` Initialise un nouveau calcule de Jaccard pour le nombre de fichier et noms passé en paramètre.
— `jaccard_add()` Ajoute le mots liès à l'indice du fichier passé en paramètre. (*Explication lors de l'ajout d'un nouveau mots ou mot déja trouvé avec mise à jour de l'intersection et du cardinal*).
— `jcrd_print_graph()` (*ToDo : Tristan*)
— `jcrd_print_distance()` (*ToDo : Maysaloon*)