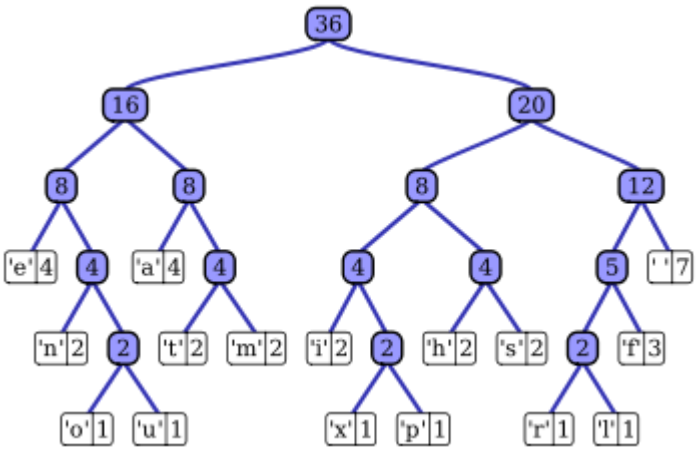
Rapport Décodage de Huffman

Andrew Mary Huet de Barochez



Polytech Annecy, 2021 - 2022
PROJ631 - Projet algorithmique

I. Introduction

Dans ce rapport, je vais aborder les différents points de développement concernant mon projet de réalisation de l'algorithme de décodage de Huffman. Il s'agit d'un algorithme de décompression de données sans perte. J'ai choisi ce sujet car j'ai trouvé le concept très intéressant et ingénieux.

Concernant le choix du langage, étant déjà à l'aise avec le python ainsi que le java, j'ai décidé de sortir de ma zone de confort. Ayant réalisé mon dernier projet algorithmique en C, j'ai donc choisi le C++ pour ce projet.

Premièrement je vais parler de la façon dont je me suis organisé pour réaliser le projet, ainsi que des différents outils qui m'ont servi durant le développement. Par la suite, j'aborderai les différents choix de structure de mon projet.

Je vais ensuite développer les parties plus techniques, comme le fonctionnement de l'algorithme en lui-même. Finalement, j'énnoncerai les différentes difficultés auxquelles j'ai fait face, avant de conclure.

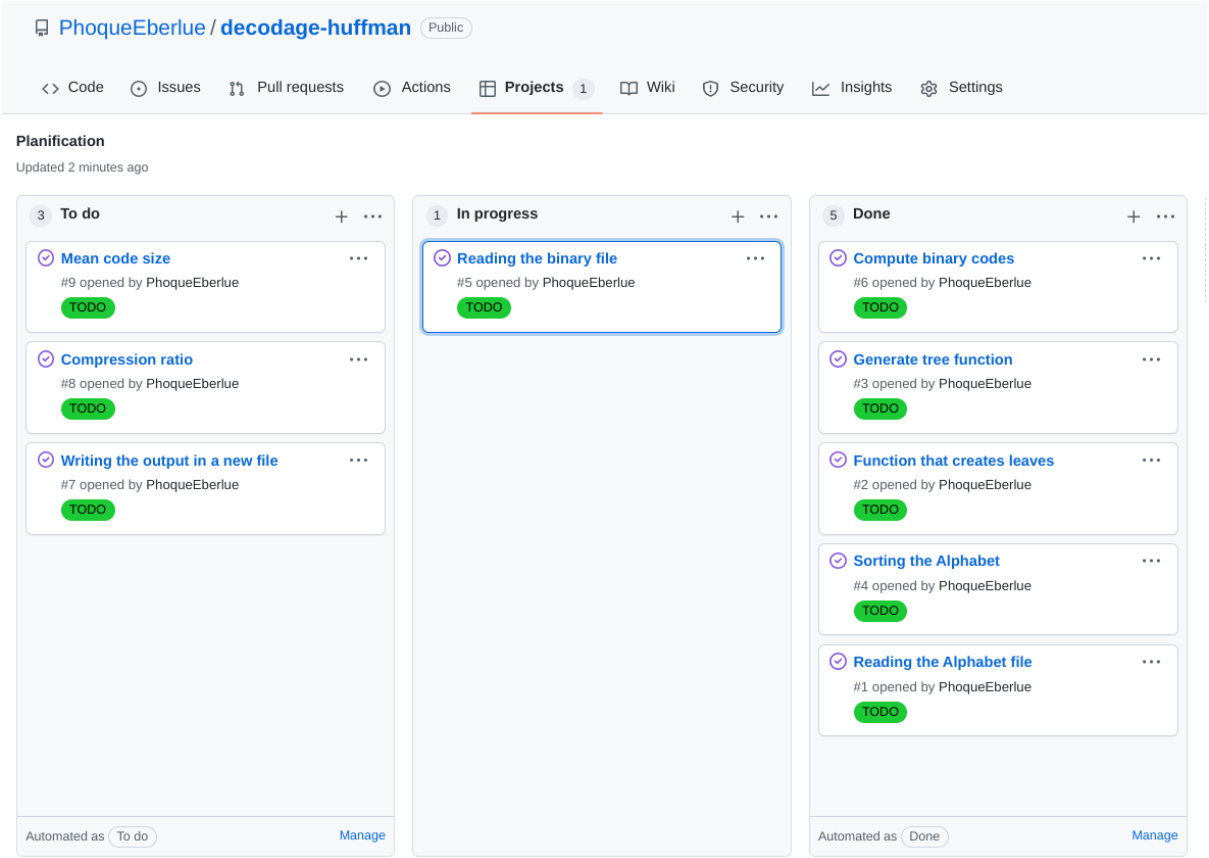
II. Organisation / Outils

Git / Github

Pour mieux m'organiser dans ce projet, j'ai utilisé une fonctionnalité de GitHub appelé 'Projects'. Il s'agit du tableau style Trello où l'on vient ajouter les différentes tâches à réaliser. Ces tâches sont disposées dans la colonne "To Do" et lorsqu'on commence l'une d'entre elles, on la met dans "In Progress", et une fois fini dans "Done". Cela permet de mieux s'organiser et de bien visualiser l'avancée du projet. C'est aussi utile pour se coordonner en équipe, même s'il s'agit d'un projet solo, cela me sera pratique pour de futurs projets.

Une différence importante entre Trello et GitHub projects c'est que les tableaux de GitHub sont directement liés au repository, et donc au code. Cela nous permet donc de créer ce qu'on appelle des 'issues', qu'on pourrait comparer aux 'user stories' des méthodes agiles. Ces issues servent à décrire plusieurs demandes dans un répertoire de travail partagé : la description d'une fonctionnalité à développer, un bug dans l'application, un retravaillement du code etc.

J'ai donc commencé par créer une issue par fonctionnalité demandée dans le sujet en essayant d'au mieux les décomposer. Ensuite, pour chaque issue, je crée une nouvelle branche, j'implémente les / la fonctionnalité(s), puis j'effectue un pull request vers la branche main en indiquant à quel Issue ce dernier répond. Une fois accepté, le pull request et l'Issue sont fermés et l'Issue est automatiquement déplacé dans la colonne "Done".

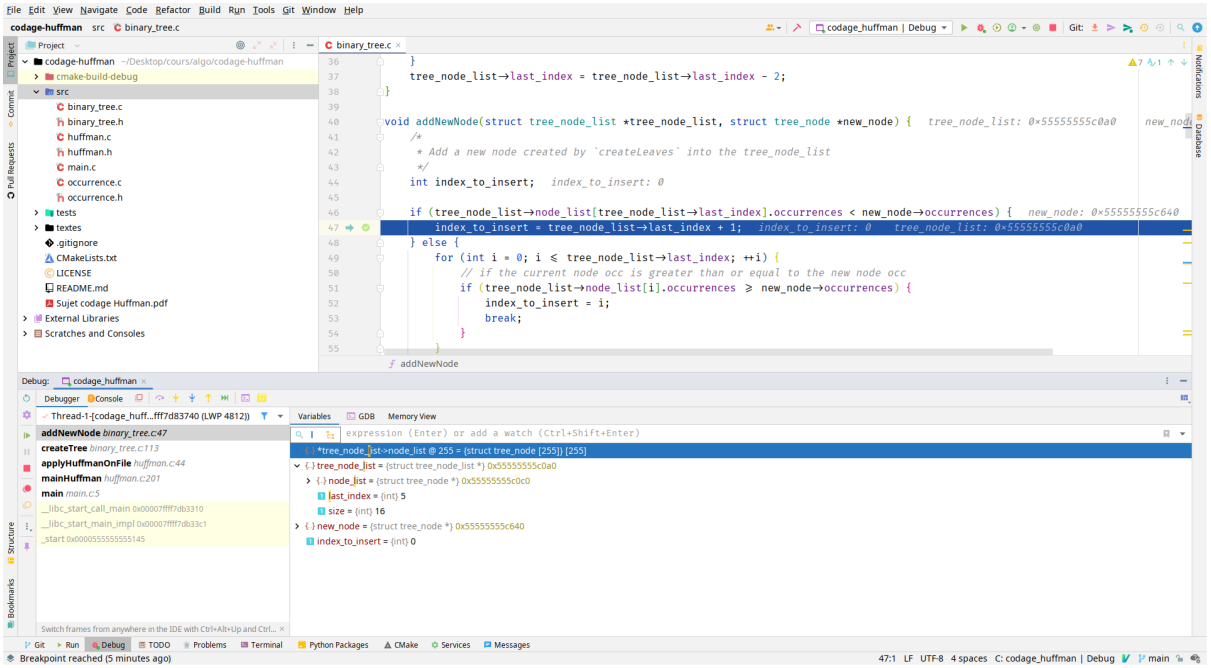


Capture d'écran 1. Page projects sur mon repository GitHub

CLion et CMake

Pour ce qui est de l'IDE, je suis depuis quelques années un grand fan de la suite Jetbrains, c'est pourquoi CLion fut un choix solide dans mon opinion. Il s'agit d'un IDE spécialement conçu pour le C et le C++, accueillant différents outils très puissants et pratiques lorsqu'on développe dans ces langages.

Par exemple, le debugger (Capture d'écran 1) intégré permet de visualiser les différentes variables ainsi que les tableaux alloués dynamiquement. Il y a également Valgrind qui permet de trouver les potentielles fuites de mémoire dans le code, cela m'a été très utile pour déboguer les problèmes liés à l'allocation de mémoire.



Capture d'écran 1. L'IDE CLion avec le debugger en bas de l'écran.

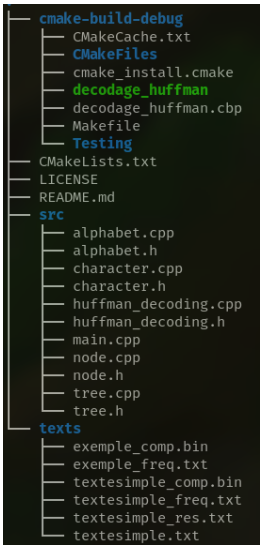
Finalement j'ai utilisé CMake pour compiler le projet, car j'avais déjà des bases avec Make et que CMake est beaucoup plus récent et facile d'utilisation.

III. Structure du projet

Le projet est structuré par 4 différents dossiers (Capture 2) :

- **cmake-build-debug**, contenant les fichiers générés par CMake ainsi que l'exécutable du main et de tests unitaires.
- **src**, le dossier regroupant le code source du projet.
- **texts**, qui sert à stocker les fichiers binaire et de fréquence donnés avec le sujet, mais également les fichiers de résultats.

Un point important à noter concernant la structure du code en elle-même, c'est l'utilisation des headers permettant une meilleure lisibilité et organisation du code. En effet, chaque fichier .cpp contenant des fonctions utilisées par l'algorithme de Huffman a un header en .h, qui est en fait un fichier listant les différentes fonctions et structures définies.



Capture d'écran 2. Arbre de mon projet

IV. Aspects techniques

En faisant la transition du C au C++, je me suis forcé à apprendre et à utiliser quand c'était possible les nouvelles fonctionnalités du C++ 20. Un aspect très intéressant que j'ai utilisé tout le long du projet est les smarts pointers. Lors du dernier projet lorsqu'on instanciait des structures et tableaux avec malloc, on était obligé de gérer la mémoire à la main en appelant la fonction free() permettant de libérer un bloc de mémoire. En C++ on peut utiliser des smarts pointers afin de ne plus avoir à gérer la mémoire à la main.

Il s'agit d'un type de pointeur qui sauvegarde le nombre de fois ou son adresse est référencée dans le code afin de libérer l'objet pointé de la mémoire quand il n'est plus accessible par aucune variable dans le code.

Il existe plusieurs types de smarts pointers, et les deux principaux que j'ai utilisés sont les unique pointers et les shared pointers. Le premier permet de créer un pointeur intelligent unique, et le deuxième de créer un pointeur qui pourra être référencé à plusieurs endroits du code. Malgré la syntaxe qui change, les smarts pointers étaient très pratiques et agréables à utiliser.

Également, le cpp nous permet l'utilisation de classes, d'une façon très intéressante dans la mesure où comme en C, on peut écrire la signature des différentes fonctions (pour les classes méthodes) dans un fichier .h, et ensuite les implémenter dans le fichier .cpp. De ce que j'ai compris, la norme est d'écrire toutes les petites méthodes d'une ligne dans le fichier .h et de mettre les autres dans le .cpp.

J'ai beaucoup apprécié cette façon de faire de la programmation orientée objet, cela ressemble à un mélange entre du C et du Java, mêlant l'aspect agréable du C pour gérer ses variables d'une manière proche de la machine, avec l'aspect conceptuel et objet de Java. De plus, je trouve que le Cpp permet beaucoup plus de libertés par rapport aux classes dans Java car il permet de créer plusieurs constructeurs, des destructeurs (pour gérer la suppression dans la mémoire de l'objet), et de notations beaucoup moins strictes et flexibles en général.

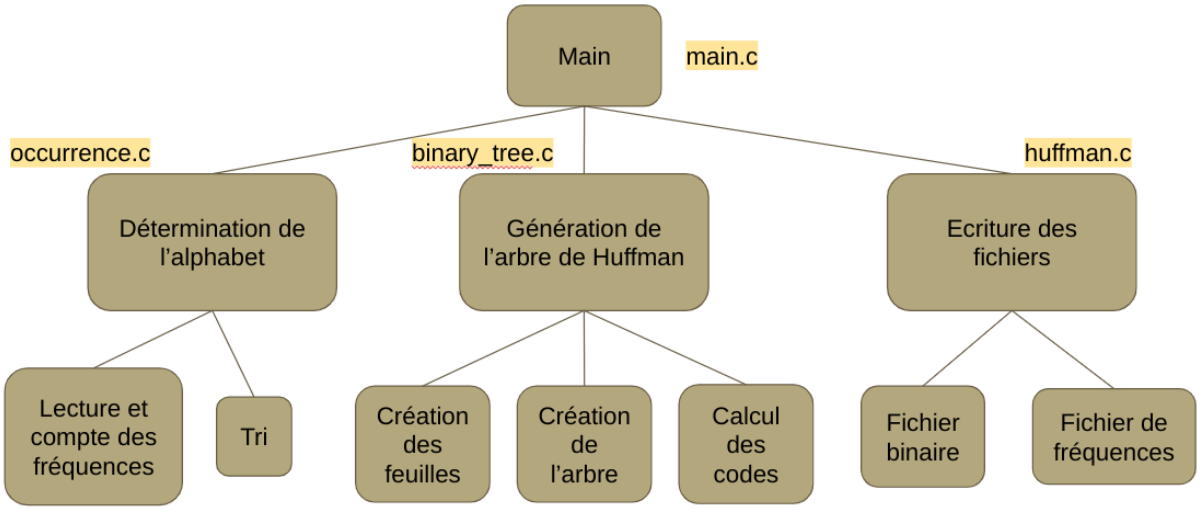
V. Difficultés rencontrées

Concernant les difficultés rencontrées, elles portaient majoritairement sur le fait de transitionner entre le langage C et C++.
En effet, le C offre très peu de types différents et oblige à créer soi-même ses structures de données. Le C++ lui, met à notre disposition de très nombreux types permettant de répondre à diverses problématiques. L'enjeu est ici de trouver le bon type pour chaque cas d'utilisation. C'est ce qui rends d'après moi ce langage complexe : la variété de fonction et de type dans le langage.

VI. Conclusion

Pour conclure ce projet fût très enrichissant, j’ai réussi à décoder le texte que j’avais encodé avec mon autre projet de codage de Huffman. J’ai pu mieux m’organiser au niveau de la gestion de projet et du git. Et finalement j’ai apprécié utiliser un langage avec plus de fonctionnalités que le C.

VII. Annexes



Annexe 1. Décomposition fonctionnelle de l'algorithme de Huffman.