

# Projet Huffman



Projet commun aux modules TI301-Fondamentaux de l'algorithmique 3 et TI  
304-Introduction au génie logiciel

Rapport TI 304-Introduction au génie logiciel

Membres de l'équipe :

- Gianluca ANNICHARICO
- Valère GOMEZ
- Lisa SANGLAR
- Quentin VINCENT

## Table des matières

Introduction.....	3
Contexte et module.....	3
Le logiciel.....	3
Planification des étapes du projet (Gantt) .....	3
Répartition des tâches.....	4
Architecture Logicielle .....	5
Schéma .....	5
Prototypes de fonctions et structures.....	6
Tests Unitaires .....	7
Conclusion .....	8

# Introduction

## Contexte et module

Au sein du module Introduction au Génie Logiciel, qui a pour but de nous initier aux bonnes pratiques de conception d'un logiciel, il nous a été demandé de faire un projet de programmation nommé « Projet Huffman ». L'objectif étant, ici, de mettre en application les notions vues en cours et en TP lors de ce semestre 3.

Ainsi, nous nous sommes répartis en équipe de 4 ou 5 étudiants. Afin de débiter ce projet sur une base solide et à un niveau égal, ces équipes de projet seront inchangées lors des TP d'Introduction au génie logiciel et de Fondamentaux de l'algorithmique 3, le sachant commun aux deux modules.

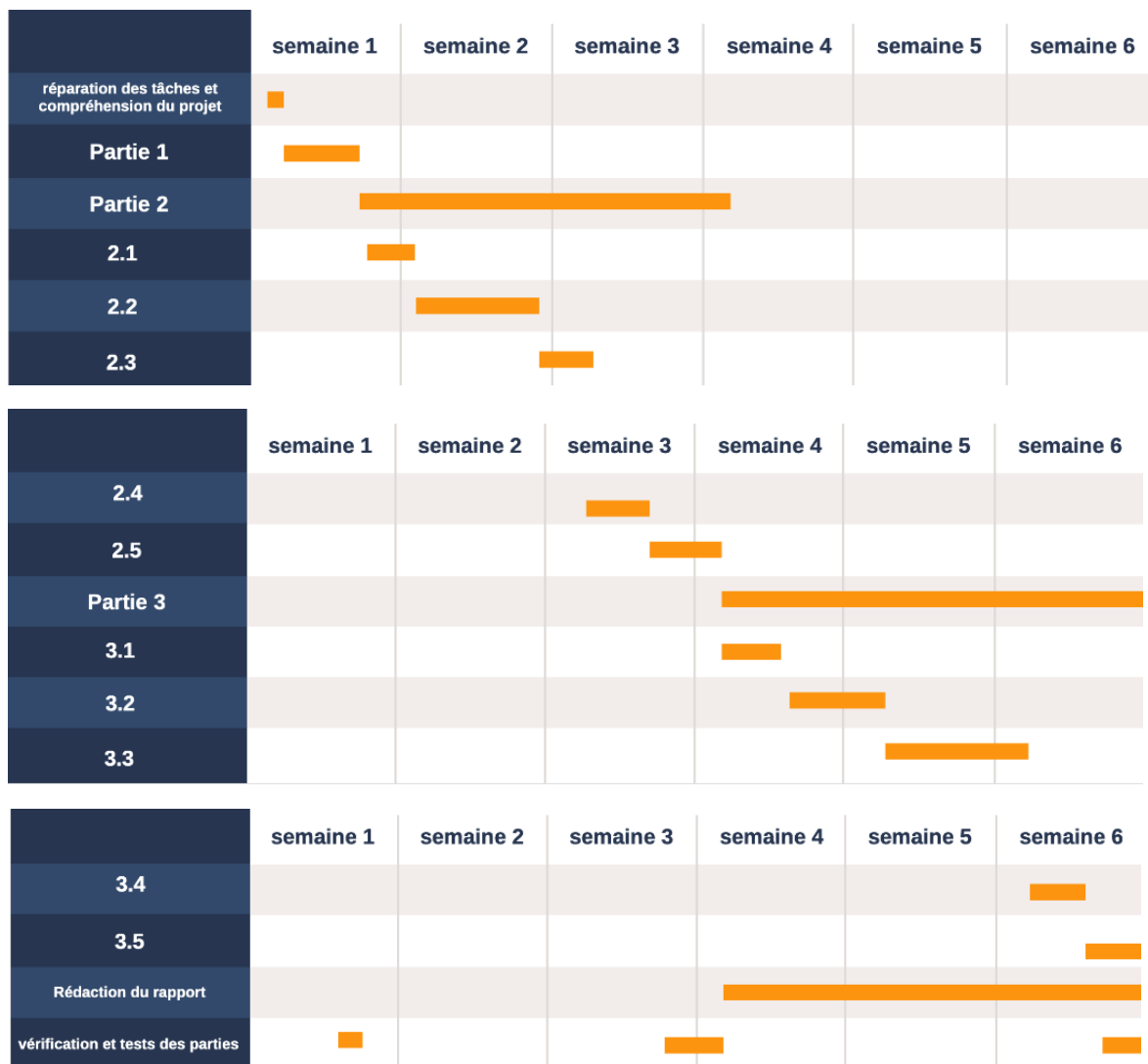
## Le logiciel

Le projet correspond à un logiciel de compression de texte sans perte de données, se basant sur le principe de l'arbre de Huffman. Il s'agit de compresser le contenu d'un fichier texte converti en bit, afin de réécrire un code binaire encodé, qui prend moins de place que l'original. L'application devra, sur une interface console, demander à l'utilisateur de lui donner le chemin d'accès à un fichier d'origine, pour lui en créer un nouveau compressé, ainsi qu'un fichier dictionnaire permettant la décompression. Il faudra penser à l'utilisateur, le taux de réussite de la compression (si des caractères n'ont pas réussi à être compressés), ainsi que le taux de compréhension (rapport du nombre de bits d'origine sur nombre de bits d'arrivé). Ce logiciel a donc besoin de pouvoir interagir avec des fichiers extérieurs, tout en utilisant des structures de données adaptées à son bon fonctionnement. La première étape était donc, de créer une architecture logicielle adaptée aux différents modules de ce logiciel.

## Planification des étapes du projet (Gantt)

Grâce au module d'Introduction au Génie Logiciel, nous avons pris conscience que la gestion et la mise en place d'une planification étaient indispensables à la réalisation dans de bonne condition d'une programmation fluide. En effet, l'organisation est le point de départ essentiel d'un projet et permet de construire des bases solides pour la suite.

De ce fait, après concertation lors de notre première réunion sur teams, nous avons mis en place un diagramme de Gantt. Le but étant de planifier de façon optimale le déroulement du projet, nous avons déterminé les dates de finalisation potentiel de chaque partie. La durée d'une tâche est ajustée selon sa difficulté (notée par des étoiles sur le sujet).



## Répartition des tâches

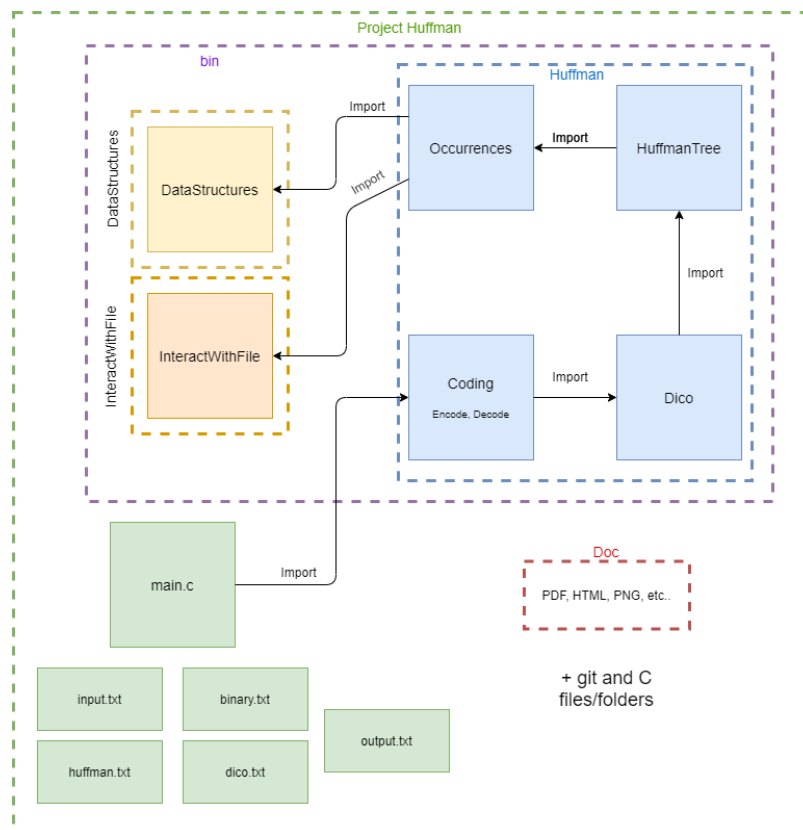
Le fait de mettre autant de personnes pour un projet de cette ampleur permet de séparer grandement les tâches, et de répartir des rôles à chacun, de cette manière (en ne prenant en compte que les tâches liées directement à ce module) :

- Gianluca A. : Documentation Doxygen.
- Valère G. : Programmation, débogage.
- Lisa S. : Guide d'utilisation.
- Quentin V. : Propriétaire du GitHub, architecture logicielle, refactoring, programmation.

## Architecture Logicielle

### Schéma

Après étude du sujet, nous avons créé un schéma représentant une architecture logicielle, que voici :



Etant donné les 4 grandes parties de l'algorithme de Huffman, chacune a son propre fichier à son nom, regroupées dans un dossier Huffman. Les fonctions permettant les interactions avec les fichiers, et tout ce qui est structure de données, ont un dossier et un fichier respectivement. Tous les dossiers regroupant le code des fonctions à été placé dans un dossier bin, afin de bien séparer toutes les parties du logiciel. Les fichiers main.c et .txt sont à la racine, pour faciliter

leurs accès utilisateurs, ainsi que pour simplifier l'accès à la machine, selon le fichier d'exécution (CMake ou autre). Enfin, un dossier Doc contenant toute la documentation html, PDF et images se trouve à la racine.

## Prototypes de fonctions et structures

Nous allons vous présenter nos différents prototypes de fonctions :

/DataStructures

DataStructures.h

```
typedef struct Noeud {
    char ch;
    int occ;
    char* bin;
    struct Noeud* sad;
    struct Noeud* sag;
}Noeud;
typedef Noeud* Arbre;

typedef struct ElementNode{
    Arbre data;
    struct ElementNode* suivant;
}ElementNode;

typedef struct Queue{
    struct ElementNode* last;
}Queue;

extern Arbre    creerNoeud      (char ch, size_t occ, char* bin);
extern void     afficherArbreOcc(Arbre a);
extern void     afficherArbreBin(Arbre a);
extern void     freeArbre      (Arbre a);

extern void     addNodeAVLch    (Arbre* AVL, Arbre tmp);
extern void     addNodeBSTch    (Arbre* AVL, Arbre tmp);
extern void     addNodeBSTocc   (Arbre* AVL, Arbre tmp);

extern size_t   depth          (Arbre a);
extern int      getBF          (Arbre a);
extern void     leftRotation    (Arbre* a);
extern void     rightRotation   (Arbre* a);
extern void     balance         (Arbre* a);

extern Queue*   createQueue     (void);
extern int      isEmptyQueue    (Queue* q);
extern void     pushQueue       (Queue* q, Arbre val);
extern Arbre    popQueue        (Queue* q);
extern int      sizeQueue       (ElementNode* q);
extern Arbre    getMinQueues    (Queue* q1, Queue* q2);
```

/InteractWithFile

## InteractWithFile.h

```
extern void error1 (void);
extern char* loadFile (FILE* file);
extern size_t countCharFile (FILE* file);
extern void printFile (FILE* file, char* content);
extern void emptyFile (char* name);
```

## /Huffman

### Coding.h

```
extern char* codeFromChar (char ch, Arbre dico);
extern void int2bin (int n, char* bin);
extern void textFileToBinFile (FILE* file, char* fBinName);
extern void zipFile (char* toZipName, char* zippedName);
extern void addNodeDico (Arbre* a, Arbre tmp, int index);
extern int nbrCaractere (const char* ch);
extern void createHuffmanFromDico (char* dicoName, Arbre* arb);
extern char chercheArbreCh (Arbre arb, const char* bin);
extern void unzipFile (char* dicoName, char* unzipName);
extern float calculateRatio (void);
```

### Dico.h

```
extern void createAVLDico (Arbre* a, Arbre add);
extern void createBinCode (Arbre huffmanTree, char* binCode, int index);
extern void printDicoFile (Arbre dico, FILE* fDico);
```

### HuffmanTree.h

```
extern Arbre createHuffmanTree (Arbre AVLocc);
```

### Occurrences.h

```
extern void createAVLcaractere (Arbre* AVL, char* content, size_t taille);
extern void addNodeAVLocc (Arbre* AVL, Noeud * tmp);
extern void createAVLoccurrence (Arbre* AVL, Arbre a);
extern void creerOccQueue (Arbre AVL, Queue* q);
```

## Tests Unitaires

Pour les tests unitaires nous avons essayé, dans la mesure du possible, de tester au moins 3 cas pour les fonctions :

- Un cas **vide**, ou « **NULL** », afin de vérifier que cela ne crash pas
- Un cas **petit**, afin de vérifier facilement que cela marche
- Un cas **complet**, afin de vérifier que cela marche en condition réelles

Par exemple, pour le test du calcul et rangement des occurrences dans un fichier, voici à quoi cela ressemble :

```
int main(void) {
    printf(« \n\nTEST FULL :\n\n ») ;
    FILE * fInput = fopen(« ../input.txt », « r+ ») ;
    if ( !fInput) error1() ;

    char* content = loadFile(fInput) ;
    size_t size = countCharFile(fInput) ;
    Nœud* AVLocc = NULL ;

    createAVLcaractere(&AVLocc, content, size) ;
    afficherArbre(AVLocc) ;

    fclose(fInput) ;
    freeArbre(AVLocc) ;
    free(content) ;

    printf(« \n\nTEST LITTLE :\n\n ») ;
    char content2[] = {'A', 'A', 'B', 'C', 'C', 'C', 'C', 'Z', '\0'} ;
    int size2 = 8 ;
    Nœud* AVLocc2 = NULL ;

    createAVLcaractere(&AVLocc2, content2, size2) ;
    afficherArbre(AVLocc2) ;
    freeArbre(AVLocc2) ;
    printf(« \n\nTEST NULL :\n\n ») ;

    char* content3 = NULL ;
    int size3 = 0 ;
    Nœud* AVLocc3 = NULL ;

    createAVLcaractere(&AVLocc3, content3, size3) ;
    afficherArbre(AVLocc3) ;
    freeArbre(AVLocc3) ;
    return EXIT_SUCCESS ;
}
```

Les trois cas sont séparés et écrits dans des printf. L'objectif ici, est de tester que cette suite de fonction est bien fonctionnelle pour un fichier texte rempli (fichier 'input.txt' fourni en annexe de sujet), une petite suite de caractères, pour vérifier que les fonctions font bien leur travail, et un test à vide, pour vérifier que cela ne dysfonctionne pas. Nous effectuons ces tests tant que nous n'avions pas un résultat correct et satisfaisant.

## Conclusion

Ainsi, ce projet fut extrêmement riche en apprentissage et pleins de surprise. L'intérêt n'était pas seulement lié à une programmation en langage C, il s'agissait également de comprendre et d'exploiter les outils mis à notre disposition pour la création et la gestion de projet de programmation.



Nous avons donc appris à élaborer une architecture de code ordonnée en la structurant et en adoptant une syntaxe la plus cohérente possible. Pour se faire, il a fallu mettre entre place une architecture modulaire avec l'utilisation de headers, prototypes, import et #include.

Quant à la programmation pure en langage C du projet, elle n'a pas été de tout repos. Les premières fonctions à réaliser sur la conversion binaire et l'écriture dans des fichiers txt furent relativement simple, mais cela ne correspondait pas réellement aux points sur lesquels nous étions attendus. En effet, le cœur du projet introduit l'utilisation d'arbres AVL. Ces derniers sont utilisés pour le codage d'une lettre, il s'agit ici de la clé de notre compression. Les files et tableaux seront plus perçus tel des aide-mémoires bénéfiques à l'arbre. La création de l'arbre d'Huffman et du dico fut selon nous les parties les plus complexes du projet, notamment celles qui ont demandé plus de temps. En effet, avant tout code, il fallait assimiler et intégrer le concept pas toujours évident de cet arbre, comprendre et visualiser la façon dont il était construit et comment le traverser pour obtenir le code de 1 et de 0 voulu. Mais à force de détermination et de documentation sur ce dernier, nous en sommes venus à bout. Nous avons essayé de déboguer vers la fin du projet, notamment les fuites mémoires. Ces dernières nous ont posés beaucoup de soucis, et nous en posent encore car nous avons un certain nombre de fuites que nous n'avons pas réussi à corriger, sinon sans faire fonctionner le programme. Ce sont donc les derniers problèmes qui persistent au terme de ce projet.

Il est important de souligner que le logiciel Git mis en relation avec le service GitHub, qui sont des outils indispensables pour tout programmeur et équipe de développement. En effet, un projet de groupe peut parfois s'avérer fastidieux lorsqu'il s'agit de mettre en commun les travaux. Or grâce à son utilisation, nous avons pu partager facilement nos programmes, protéger nos codes et modifications, contrôler les différentes versions du projet, et avancer étape par étape en faisant bien attention de tout tester à chaque fois. Le travail collaboratif fut fluide sans réel problème.

L'utilisation d'un outil de documentation automatique comme Doxygen facilite vraiment la compréhension du code pour des personnes extérieurs au projet, et permet de simplifier l'évolution du logiciel, puisqu'une trace écrite décrivant toutes les fonctions existe. Ainsi, il est facile de comprendre de quoi est composé le code, comment il fonctionne, et donc de plus simplement le reprendre pour l'améliorer.

## Commentaires personnels :

### Quentin Vincent :

Pour ma part c'est un projet qui a été très intéressant dans son ensemble, quoique loin d'être reposant.

Etant déjà un peu familier avec GitHub, je n'ai pas rencontré beaucoup de soucis avec l'utilisation de ce service, mais j'ai tout de même approfondi ma connaissance de ce dernier. J'ai programmé bon nombre de fonctions et ai aussi aidé mes collègues dans leur

programmation, mais ce qui a demandé le plus d'effort a été de bien organiser les tâches de chacun, qu'elles soient faites à temps et efficacement. Nous avons pris du retard, certainement à cause d'une sous-estimation de la charge de travail que nécessitait ce projet. Je tiens à rajouter aussi que travailler entièrement à distance, avec des personnes que nous n'avons pas forcément eu l'occasion d'apprendre à connaître à cause des conditions de ce semestre, n'a vraiment pas facilité la fluidité de travail, et donc la progression du projet.

Je reste tout de même satisfait du rendu que nous faisons, et des choses que nous avons pu apprendre et approfondir tout au long de ces semaines.

#### Valère Gomez :

Pour ce projet, j'ai travaillé sur une grande partie de la programmation des fonctions qui touchent aux listes, tableaux et arbres ainsi qu'à leur amélioration.

Les enseignements du cours de Génie Logiciel sont nécessaires au bon déroulement des projets, de manière à être efficace. Les applications vues en cours de TD nous permettent d'exploiter chaque méthode l'une après l'autre. Ces exercices très bien construits viennent peut-être trop tardivement dans le semestre. Les divers projets de programmation devraient faire suite aux exercices et non se confondre. Cela permettrait d'avoir une vue d'ensemble des méthodes à exploiter et ainsi savoir les combiner pour une meilleure productivité.

Le travail d'équipe est toujours difficile à distance, le re-confinement est venu perturber la dynamique mise en place lors de la rentrée en présentiel.

#### Gianluca Annichiario :

Ce projet aura été bénéfique tant sur le plan logiciel qu'esprit d'équipe, en effet l'apprentissage et l'utilisation de GitHub nous aura permis de travailler ensemble en nous répartissant les tâches. Utilisant un AGL Visual Studio Code j'ai dû m'adapter et configurer mon logiciel afin que tout fonctionne correctement, aujourd'hui grâce à ce projet j'ai pu mettre en pratique tout ce que j'ai appris et je me sens d'autant plus productif et performant.

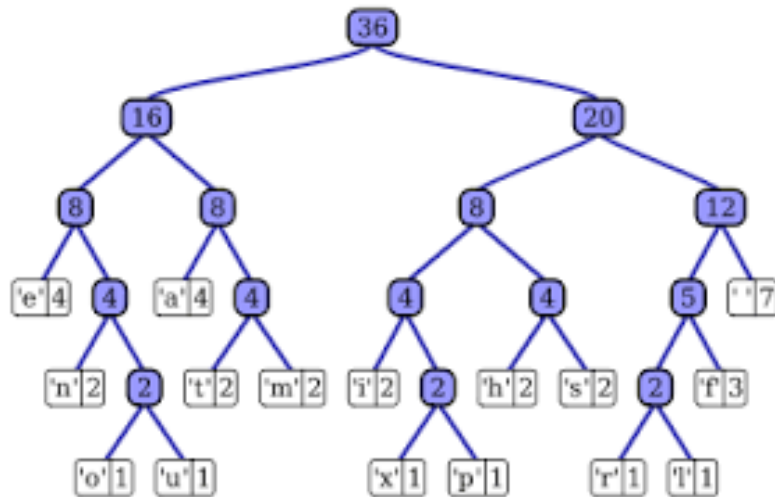
#### Lisa Sanglar :

Le travail d'équipe à distance ne fut pas des plus approprié, rendant la réalisation de ce projet moins agréable. Mais cela ne nous a pas empêché de communiquer et d'échanger via certaines plateformes. De plus, j'ai rencontré quelques difficultés avec l'utilisation de certains logiciels de TP (étant sur Mac) ce qui n'a cependant pas atténué mon envie d'être utile et de m'impliquer pour ce projet. L'équipe reste bienveillante et toujours disposée à l'entraide. J'ai énormément appris, notamment avec le logiciel GitHub.

Dans l'ensemble, le projet m'a permis de visualiser et de comprendre l'utilisation et l'utilité de ces outils de programmation indispensable au travail d'équipe de programmation ou de projet.

# Guide d'utilisation

## Compresseur de texte Huffman



### I- Lancement du logiciel

Voici ce que vous obtenez lors de l'ouverture du dossier Huffman :

bin	Dossier de fichiers
Doc	Dossier de fichiers
Doxyfile	Fichier
main.c	Fichier C
binary.txt	Fichier TXT
dico.txt	Fichier TXT
huffman.txt	Fichier TXT
input.txt	Fichier TXT
output.txt	Fichier TXT
documentation	Raccourci
Huffman.exe	Raccourci

Dans un premier temps, veuillez choisir le texte que vous souhaitez compresser. Ouvrez ensuite le fichier input (présenté ci-dessus) et inscrivez-y votre texte. Ici, nous avons pris comme exemple “Efrei” :



Enfin, pour lancer le logiciel, il vous suffit de lancez « Huffman.exe », ou bien d’exécuter le fichier main.c dans votre IDE préféré.

Après ouverture, le logiciel se présente comme ceci :

```

C:\Users\lalla\OneDrive - Efrei\Bureau\Huffman-release_beta\Huffman-release_beta\main.exe
-----Compresseur de fichier Huffman-----

Que desirez-vous faire ?
.1 -> Compresser mon fichier 'input.txt'
.2 -> Décompresser mon fichier 'huffman.txt'
.3 -> Fermer le programme

ATTENTION :
- Tous vos fichiers seront écrasés par les nouveaux, assurez-vous de faire les sauvegardes nécessaires si besoin.
- Assurez-vous que vos fichiers soient placés dans le dossier parent de l'exécutable !

Votre choix ->

```

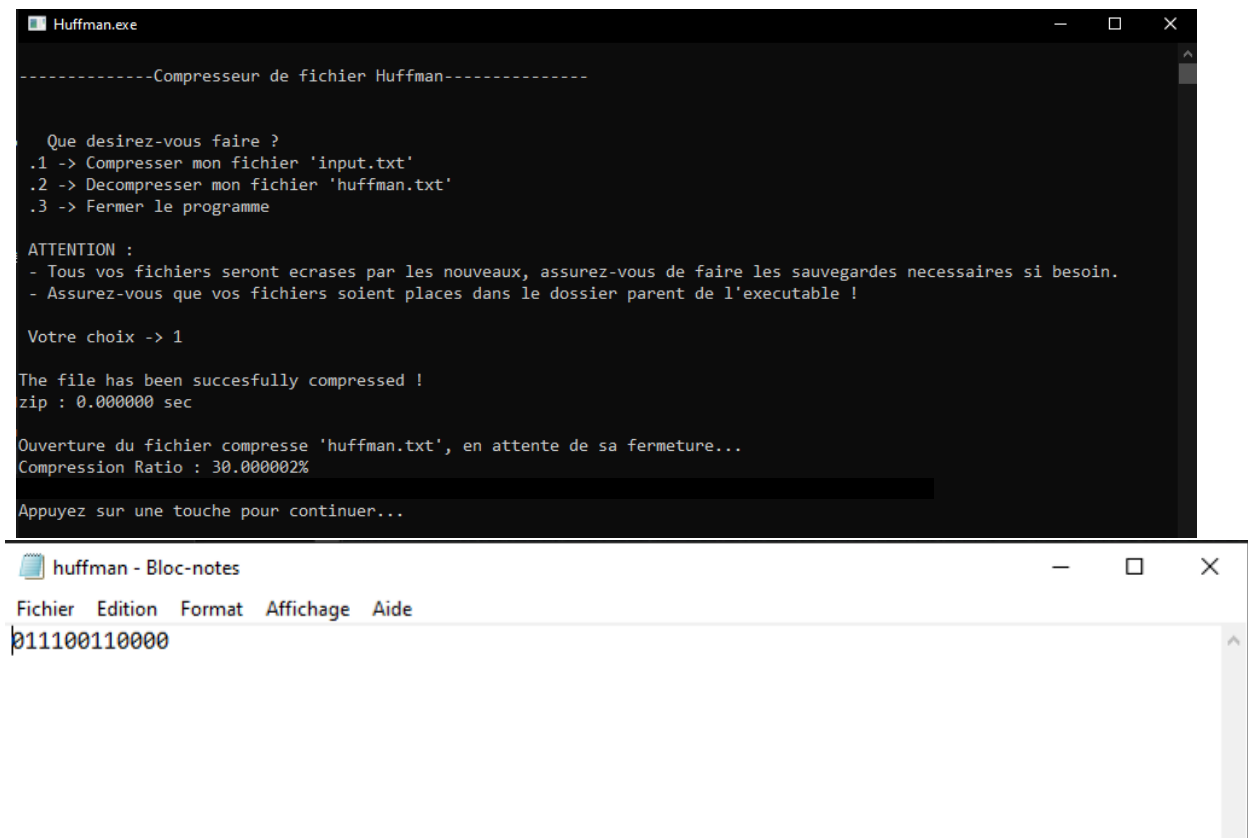
## II- Utilisations

Trois choix d’utilisation se présenteront à vous. Selon vos besoins saisissez la commande 1, 2 ou 3 :

- Pour une compression : 1
- Pour une décompression : 2
- Pour fermer le programme : 3

## 1- Compression d'un fichier texte choisi par l'utilisateur :

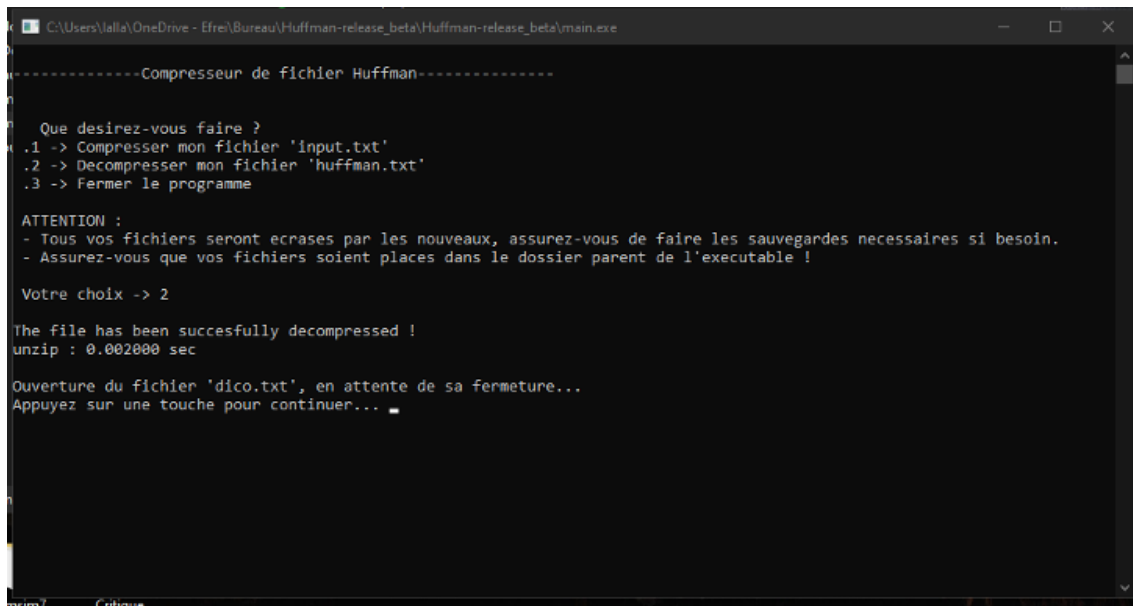
En choisissant la commande 1, vous obtiendrez alors votre texte compressé, provenant du fichier "input.txt" rempli au préalable, encodé dans le fichier nommé "Huffman.txt" :



Votre texte est à ce stade compressé, son taux de compression s'affiche en % (la place que prend le code binaire du fichier par rapport à avant, ici 30%, soit 70% de gain de place), son code binaire l'est en tout cas, et un fichier "dico.txt" a été généré. Ce dernier va permettre la décompression, comme ci-dessous :

## 2 – Décompression d'un fichier texte

La commande 2 vous permettra d'obtenir la clé de la décompression de votre fichier. Bien sûr, si vous n'avez pas compressé votre fichier au préalable, ou si vous avez effectué une quelconque modification dans un des fichiers .txt, cela ne marchera pas correctement.



```
-----Compresseur de fichier Huffman-----

Que desirez-vous faire ?
.1 -> Compresser mon fichier 'input.txt'
.2 -> Décompresser mon fichier 'huffman.txt'
.3 -> Fermer le programme

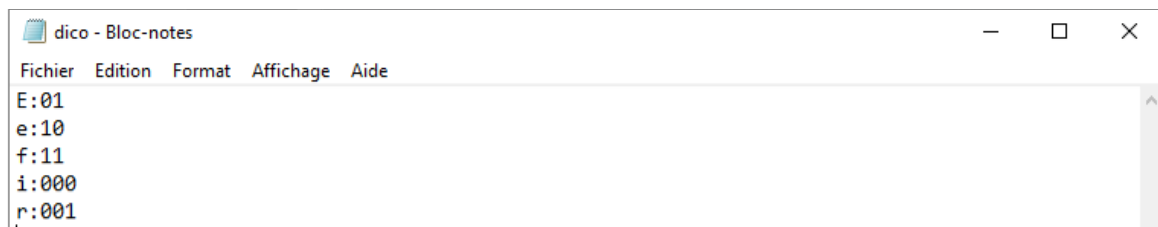
ATTENTION :
- Tous vos fichiers seront écrasés par les nouveaux, assurez-vous de faire les sauvegardes nécessaires si besoin.
- Assurez-vous que vos fichiers soient placés dans le dossier parent de l'exécutable !

Votre choix -> 2

The file has been successfully decompressed !
unzip : 0.002000 sec

Ouverture du fichier 'dico.txt', en attente de sa fermeture...
Appuyez sur une touche pour continuer... █
```

Ainsi, vous obtiendrez le fichier décompressé dans "output.txt", et le logiciel vous rouvre le dico comme ceci :



```
dico - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
E:01
e:10
f:11
i:000
r:001
```

Pour vous ouvrir le fichier décompresser, il suffit de suivre l'étape suivante :

### 3- fermeture du logiciel

En choisissant la commande 3, votre fichier décompressé apparaîtra et le logiciel se fermera :

```
C:\Users\lalla\OneDrive - Efrei\Bureau\Huffman-release_beta\Huffman-release_beta\main.exe

-----Compresseur de fichier Huffman-----

Que desirez-vous faire ?
.1 -> Compresser mon fichier 'input.txt'
.2 -> Décompresser mon fichier 'huffman.txt'
.3 -> Fermer le programme

ATTENTION :
- Tous vos fichiers seront écrasés par les nouveaux, assurez-vous de faire les sauvegardes nécessaires si besoin.
- Assurez-vous que vos fichiers soient placés dans le dossier parent de l'exécutable !

Votre choix -> 3

Fermeture du programme...

Ouverture du fichier décompressé 'output.txt', en attente de sa fermeture...
```

```
output - Bloc-notes
Fichier Edition Format Affichage Aide
Efrei
```

Par ailleurs, si vous souhaitez obtenir le code binaire de votre texte (non-compressé), il vous suffit de vous rendre dans le fichier "binary.txt" :

```
binary - Bloc-notes
Fichier Edition Format Affichage Aide
0100010101100110011100100110010101101001
```

Ainsi, résumons :

- Le fichier **input** doit comprendre votre texte à compresser
- Le fichier **binary** correspond au code binaire du texte
- Le fichier **dico** stock le dictionnaire permettant la décompression
- Le fichier **huffman** présente le code compressé du texte selon le principe Huffman
- Le fichier **output** comprend le texte décompressé

Nous vous souhaitons une bonne utilisation !