Projet :Codage de Huffman

KIEGAIN DJOKO Yvan Charles
BELMAKHFI Nihal

Département Sciences du Numérique - Première année
2021-2022

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet était la compression et la décompression d'un fichier texte en utilisant un système de codage très connu qui est le codage de HUFFMAN et pour cela nous avons décider de nous reposer sur deux modules qui sont :
- Un module **LCA** (liste chainées)
- Un module **ABR_HUFFMAN** (arbre binaires)

2 Raffinages

2.1 Raffinage de COMPRESSER

R0 : Compresser le fichier texte

R1 : Comment "Compresser le fichier à compresser" ?
Compter les fréquences de caractères Fichier : in ; char : in out ; Créer une feuille pour chaque caractère ListAbr : in out ; Abr, freq, Char : in out ; Générer l arbre d'Huffman ListAbr : in out ; Abr,abr_g,abr_d : in out
Générer le fichier compresser Fichier_initiale : in ; Fichier_compresser : out

R2 : Comment Compter les fréquences de caractères ?
Tanque non fin du fichier à compresser Fichier faire lire un caractère char ; si Char dans la liste freq freq(Char) = freq(Char) + 1 sinon freq(Char) = 0 finsi fin tanque

R2 : comment "Créer une feuille pour chaque caractère" ?
Creer une liste listAbr qui à chaque fréquence dans freq associe un arbre pour chaque Char dans freq
creer un arbre Abr avec Clé = freq(Char) et Donnée = Char
ajouter Abr dans listAbr_ fin pour

R2 : Comment "Générer larbre de Huffman" ?
Tant Que length(listAbr) = 1
abr_g = Rechercher_minimum(listAbr)

abr_d = Rechercher_2ème_minimum(listAbr)
creer un arbre Abr avec Donnee = abr_g.Donnee + abr_d.Donnee
Abr.arbre_gauche = abr_g
Abr.arbre_droit = abr_d
Ajouter Abr dans listAbr
Supprimer abr_g et abr_d dans listAbr
Fin Tant Que

R2 : Comment "Générer le code de l arbre d huffman" ?
CodeHuffman -> Chaine de Caractère de vide
Si Abr est une Feuille alors
Ajouter 1 à CodeHuffman
Sinon
Ajouter 0 à CodeHuffman
Générer le Code de L arbre D Huffman avec en entrée fils Gauche de Abr - -appel récursif
Générer le Code de L arbre D Huffman avec en entrée fils Droit de Abr - -appel récursif

R0 : Compresser le fichier texte

R2 : Comment "Générer le fichier Compresser" ?
Ecrire chaque caractère présent dans la liste freq dans le fichier_compresse
Réecrire le dernier cractère de la liste freq dans le fichier_compresse
Générer le code de l arbre d'huffman CodeHuffman : out ; Abr : in
Ecrire le code D'huffman de l arbre dans le fichier compresse
Générer la table dHuffman de l ?arbre tableHuffman : out ; Abr : in ; SuiteBit : in out
ecrire le texte transcrit avec huffman dans le fichier compresser en faisant la correspondance entre un caractère et son code d huffman grace à la table d huffman
Fichier_Compresser : out ;
tableHuffman : in

R3 : Comment "Générer le code de l arbre d huffman" ?
CodeHuffman -> Chaine de Caractère de vide

Si Abr est une Feuille alors
Ajouter 1 à CodeHuffman
Sinon
Ajouter 0 à CodeHuffman
Générer le Code de L arbre D'Huffman avec en entrée fils Gauche de Abr - -appel récursif
Générer le Code de L arbre D'Huffman avec en entrée fils Droit de Abr - -appel récursif

R3 : Comment "Générer la table d'Huffman de l'arbre" ?
tableHuffman -> liste de couple caractère - code Huffman du caractère
Si Abr est une Feuille alors
Ajouter dans tableHuffman le couple SuiteBit et Donnée de Abr
Sinon
Ajouter 0 à SuiteBit1
Générer la table d'Huffman de l'arbre avec en entrée fils Gauche de Abr et SuiteBit1
- -appel récursif
Ajouter 1 à SuiteBit2
Générer la table d'Huffman de l'arbre avec en entrée fils Gauche de Abr et SuiteBit2
- -appel récursif

2.2 Raffinage de DECOMPRESSER

R0 : Décompresser le fichier binaire

R1 : comment "Décompresser le fichier binaire" ?

retrouver tous les caractères du texte open_file : File in,
liste_caractere : liste out ;
posCaracTerminal : entier out ;
S, S_Next : T_Octet in out ;

reconstruire l'arbre d'huffman nombreDe1 : entier in,

Suite_de_bit , bit : chaine de caractere in out ;
liste_caractere : liste in ;
open_file : Stream_Access out ;

reconstruire le texte originale caractere,suite_de_bit : chaine
de caractere in out ;
caractere_terminal : chaine de
caractere in ;
open_file : Stream_Access out ;

R2 : comment "Retrouver tous les caractères du texte" ?

S -> lire un octet dans le fichier compresser
S_Next -> lire un octet dans le fichier compresser

Répéter jusqu à S = S_Next

S -> lire un octet dans le fichier compresser
S_Next -> lire l octet suivant dans le fichier compresser
Enregistrer le caractère associé à l octet S dans liste_caractere

Fin répéter

R2 : comment "Reconstruire l'arbre d'huffman" ?
lire un bit B dans le fichier compresser
si B = 1 alors
Mettre fils gauche et droit de Abr Null
Donnée de Abr -> le caractère dans liste_caractère.. ..avec pour clé le
nb1
nb1 -> nb1 + 1 - - le nombre de 1 déja rencontré
sinon
creer un arbre abr_g
Donnez en fils gauche de Abr abr_g
Reconstruire l arbre d'Huffman avec abr_g en entrée - -appel récursif
creer un arbre abr_d
Donnez en fils droit de Abr abr_d

Reconstruire l arbre d Huffman avec abr_d en entrée - -appel récursif finsi

R2 : comment "Reconstruire le texte originale" ?
Abr -> racine de l arbre reconstitué
Tant que Donnée de Abr /= caractere_terminal faire
Si Abr est une feuille
écrire la donnée de Abr en octet dans le fichier décompressé
Abr -> racine de l'arbre reconstitué
Sinon
lire un bit B dans le fichier compressé
si B = 1 alors
Abr -> Fils Droit de Abr
sinon
Abr -> Fils gauche de Abr
Fin tant que

3 Modules

Pour réaliser la compression et la décompression de notre fichier text par la méthode de huffman on s'est servi de deux structures de données que nous avons considéré les mieux adapté à la réalisation de notre projet

3.1 Module LCA

Le module LCA a été conçu pour fonctionner comme un dictionnaire ; il associe à chaque clé une donnée. Dans notre projet, on l'a utilisé à plusieurs reprise par exemple lorsque nous devions initier une liste qui à chaque caractère lui associe la fréquence correspondante, mais également pour générer une liste qui à chaque caractère lui associe son codage dans l'arbre d'huffman, nous permettant d'une part d'afficher cette table et d'autres part de traduire les caractères de notre fichier initiales en leur correspondance dans le fichier compressé. la petite spécificité de notre LCA est qu'elle possède une procédure qui permet d'enregistrer

les éléments de façon triée par rapport à une fonction de tri qui est fournie au package.

3.2 Module ABR_HUFFMAN

Le module ABR_HUFFMAN a été conçu pour fonctionner comme un arbre binaire ; chaque noeud est composé d'une clé et une donnée, et chacun des noeuds est susceptible d'avoir un arbre gauche et un arbre droit. les noeuds sont enregistrés dans l'arbre à gauche ou à droite suivant que leur clé soit plus petite ou *égale* à la clé de l'arbre en lui même ; cela nous autorise d'enregistrer dans l'arbre des éléments aux clés identiques. Dans ce dernier nous implantons hors mis les fonctions classique de manipulation de l'arbre comme accesseur, mutateur, etc.. ; de nombreuses procédures propres aux codage d'huffman qui sont :

- **Affichage Huffman** : qui nous permet d'afficher l'arbre d'huffman sous une forme arborescente.
- **Table_Huffman** : qui nous donne en sortie une LCA qui associe à chaque Clé sa représentation dans l'arbre d'Huffman.
- **Code_Huffman** : qui nous fournit le code de l'arbre Huffman nécessaire à sa reconstruction .
- **ReConsAbr** : qui permet de reconstruire un arbre d'huffman à partir de son code généré par la procédure précédente.

3.3 Module de Test LCA et ABR

Pour nous assurer de la fonctionnalité des fonctions et procédures unitaires utilisées dans les modules LCA et ABR_HUFFMAN, on les a testé dans deux modules tests appelé test_module_lca.adb et test_module_abr.adb

4 Programme principale

Nous Présentons maintenant ce qui aura été un peu le but et l'accomplissement de ce projet l'implémentation de Compression et Decompression.

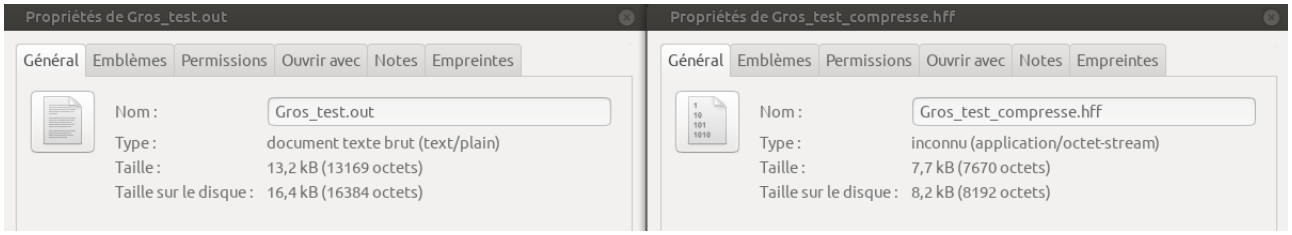
4.1 COMPRESSION

La compression dans notre cas de fichiers texte grace à l'arbre d'huffman commence par l'analyse du fichier en question afin d'en extraine une LCA dans laquelle à chaque caractère du texte est associé son nombre d'apparitions dans le texte (fréquence).
Ensuite à partir de cette liste nous créons encore une LCA mais qui cette fois associe à chaque fréquence, une feuille d'arbre ayant pour couple clé-donnée la fréquence et le caractère cité ci-haut. ensuite par un processus itératif d'extraction de deux éléments minimaux de la LCA suivis de la création d'un arbre donc la fréquence est la somme des deux extraits et lui attribuant en fils gauche et droit ces deux derniers. Sachant que l'extracton des minimaux de la LCA est facilitée par le faite que nous l'avons enregistré de façon triée directement grace à la procedure enregistrer_trie cité dans le module de LCA, nous demandant juste à retirer le premier élément à chaque fois.

Une fois notre arbre D'Huffman générée nous en déduisant sa table d'Huffman grace à la procédure **Table_Huffman**, ainsi que son codage d'huffman obtenu par parcours infixe de ce dernier grace à la procédure **Code_Huffman** toutes deux implémenter dans le module ABR_Huffman. Toutes les cartes en main pour rédiger notre fichier compresser, nous décidons tout d'abord d'écrire dans ce fichier chanqe octet représentant nos caractères en doublons le dernier pour marquer la fin de cette suite de caractères.

Nous décidons ensuite pour le reste d'opter pour une manipulation de Unbounded_String(Chaines de caractères à taille variables) en insérant dans cette dernière tout d'abord le code huffman de notre arbre suivis de chacun des caractères du texte bien sur remplacé par son codage Huffmann et le caractère terminal que nous avons choisi. Enfin nous finissons par lire cette chaine de caractères (de 0 et de 1) par intervalle de 8 caractères représentant un Octet que après traitement nous écrivons dans notre fichier compressé jusqu'à ce qu'il ne reste plus qu'un chiffre non multiple de 8 de caractères saisissant ces derniers et les complétent par des 0 pour ecrire le dernier Octet de notre fichier. Voici notre fichier

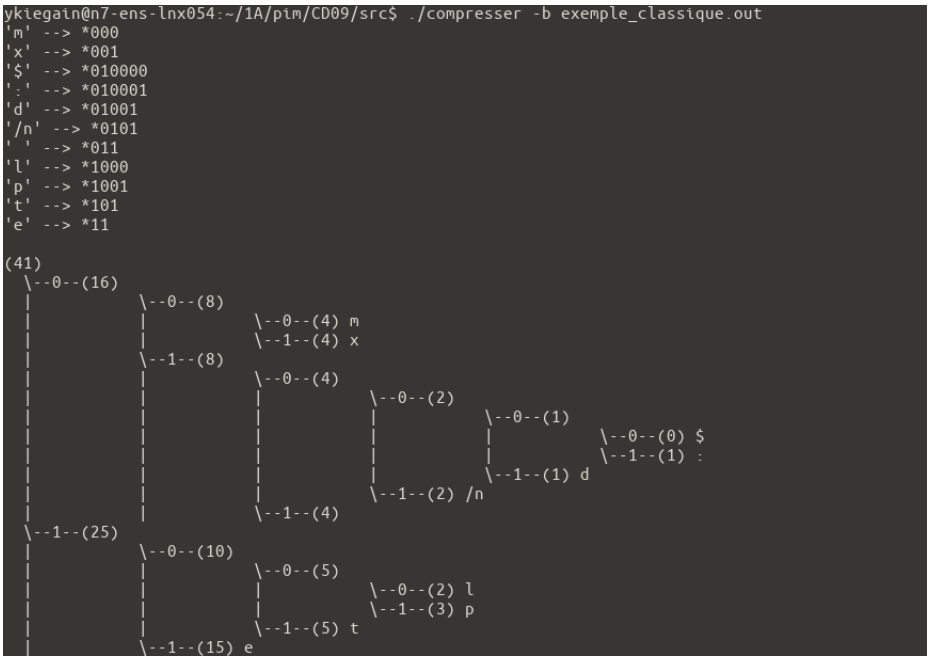
compresser généré.
Notre programme nous assure de retouver un fichier de plus petite taille que le fichier originale et également grace à l'outil valgrind qu'il n'y a pas de fuite de mémoire de nos utilisations de pointeur mais possiblement quelques fuites dû à l'utilisation de la fonction prédéfinie de ADA TO_String() dont nous ne connaissons pas la cause.




```
ykiegain@n7-ens-lnx029:~/1A/pim/CD09/src$ valgrind --leak-check=full ./compresser Gros_test.out
==45913== Memcheck, a memory error detector
==45913== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==45913== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==45913== Command: ./compresser Gros_test.out
==45913==
==45913==
==45913== HEAP SUMMARY:
==45913==     in use at exit: 71,088 bytes in 2 blocks
==45913==   total heap usage: 14,692 allocs, 14,690 frees, 399,915,420 bytes allocated
==45913==
==45913== 10,312 bytes in 1 blocks are possibly lost in loss record 1 of 2
==45913==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==45913==    by 0x42A8FE: __gnat_malloc (in /home/ykiegain/1A/pim/CD09/src/compresser)
==45913==    by 0x433ACC: system__secondary_stack__allocate_dynamic (in /home/ykiegain/1A/pim/CD09/src/compresser)
==45913==    by 0x406501: compresser__compression.2 (compresser.adb:204)
==45913==    by 0x4050CB: _ada_compresser (compresser.adb:238)
==45913==    by 0x40D4C4: main (b__compresser.adb:312)
==45913==
==45913== 60,776 bytes in 1 blocks are possibly lost in loss record 2 of 2
==45913==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==45913==    by 0x42A8FE: __gnat_malloc (in /home/ykiegain/1A/pim/CD09/src/compresser)
==45913==    by 0x433ACC: system__secondary_stack__allocate_dynamic (in /home/ykiegain/1A/pim/CD09/src/compresser)
==45913==    by 0x413E5E: ada__strings__unbounded__to_string (in /home/ykiegain/1A/pim/CD09/src/compresser)
==45913==    by 0x40646D: compresser__compression.2 (compresser.adb:204)
==45913==    by 0x4050CB: _ada_compresser (compresser.adb:238)
==45913==    by 0x40D4C4: main (b__compresser.adb:312)
==45913==
==45913== LEAK SUMMARY:
==45913==    definitely lost: 0 bytes in 0 blocks
==45913==    indirectly lost: 0 bytes in 0 blocks
==45913==      possibly lost: 71,088 bytes in 2 blocks
==45913==    still reachable: 0 bytes in 0 blocks
==45913==         suppressed: 0 bytes in 0 blocks
==45913==
==45913== For lists of detected and suppressed errors, rerun with: -s
==45913== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

4.2 Terminal

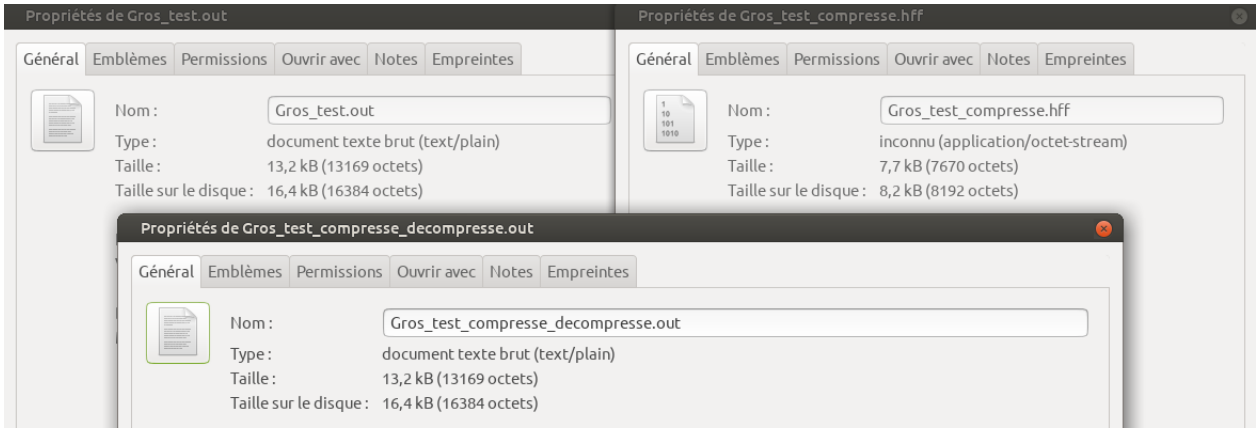
Nous avons implanter la manipulation de notre progamme via un terminal permettant non seulement de compresser et décompresser celui ci mais aussi de nous afficher la table de huffman et la représentation de l'arbre utilisé lors de ces processus je vous en donne un aperçu ci-dessus :



4.3 DECOMPRESSION

S'agissant de la décompression, la première chose à faire ici est de récupérer les caractères qui vont constituer notre texte lisant les octets du fichiers compresser un à un jusqu'à remarquer un doublon d'octet qui signale que nous avons lu le dernier caractère nous stockons ces derniers dans une LCA qui accocie un entier à chaque caractère entier correspondant à sa position dans l'arbre d'Huffman initiale.
Ensuite en adoptant la même méthode de manipulation de Unbounded_String que dans compression, nous lisons à la suite chacun des octets jusqu'à la fin du fichier et pour chacun de ces octets nous le décomposons en bit que nous insérons au fur et à mesure dans la chaine précédente. Vient le moment de reconstruire notre arbre initiale en lisant les bits et en créant des fils gauche et droit au fur et à mesure de la lecture de bits et insérant les caractères dans l'arbre à chaque rencontre d'un 1 le caractère choisi dans la LCA cité ci-haut suivant le nombre de 1 déja rencontré dans la lecture auparavant. cette algorithme étant selon notre constat optiminal, il s'arretera après avoir entièrement reconstruit l'arbre.
L'arbre une fois reconstruit plus qu'à reconstitué le texte initial en lisant un à un pour le reste de bit de notre Unboubded_String et en vérifiant à chaque fois s'il n'existe pas un caractère correspondant à ce

dernier. L'algorithme s'arrete lorsqu'il rencontre le caractère terminal choisi. Voilà notre Fichier texte initiale reconstruit à partir du fichier compressé au bit près.
Notre programme nous assure de retouver un fichier de la même taille que le fichier originale et également grace à l'outil valgrind qu'il n'y a pas de fuite de mémoire.




```
ykiegain@n7-ens-lnx029:~/1A/pim/CD09/src$ valgrind --leak-check=full ./decompresser Gros_test_compresse.hff
==47452== Memcheck, a memory error detector
==47452== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==47452== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==47452== Command: ./decompresser Gros_test_compresse.hff
==47452==
==47452== Conditional jump or move depends on uninitialised value(s)
==47452==    at 0x405712: decompresser__decompression.0 (decompresser.adb:94)
==47452==    by 0x404D5C: _ada_decompresser (decompresser.adb:124)
==47452==    by 0x40755A: main (b__decompresser.adb:312)
==47452==
==47452==
==47452== HEAP SUMMARY:
==47452==     in use at exit: 0 bytes in 0 blocks
==47452==   total heap usage: 242,978 allocs, 242,978 frees, 3,690,277,164 bytes allocated
==47452==
==47452== All heap blocks were freed -- no leaks are possible
==47452==
==47452== Use --track-origins=yes to see where uninitialised values come from
==47452== For lists of detected and suppressed errors, rerun with: -s
```

5 Retour sur Expérience

5.1 Difficultés rencontrées

5.2 Répartition des Taches du Projet

Dans l'ensemble chacun a fournie à son niveau ce dont l'autre attendait de lui .
Yvan s'est occupé de l'implémentation du code en majeur partie, les modules lca, abr_huffman, compresser et decompresser, le raffinage de décompresser et une partie de la rédaction du rapport.

Nihal c'est elle occupé de l'implémentation des fichiers de test, du choix des modules que nous allions utilisés pour le projet, le raffinage de compresser et la rédaction de l'autre partie du rapport .
Nous pensons que nous aurions pu mieux répartir le travail, ce qui aurait influencer sur le temps passés sur le projet en positif.

5.2.1 1er problème

On a pensé à utiliser une LCA où les clés etaient des féquences et les données étaient des caractères(les caractères composant notre fichier text). Pourtant, on a remarqué que les clés ne seront pas uniques vu que deux caractères différents peuvent avoir même fréquence. Ceci étant en contradiction avec la propriété d'unicité des clés.

Ensuite pour reméder à ce problème, on a pensé à inverser les types les clès et les données.Mais dans ce cas, la logique de l'arbre ne serait plus respecté car par exemple pour un noeud parent de deux feuilles il ne serait pas identifiables dans l'arbre par sa clé mais plutôt par sa donnée.

Mais au final on a decidé de rester sur l'idée de base en considérant que notre arbre ne vérifie tout simplement pas la propriété d'unicité des clés, tout ceci nous ayant fais perdre du temps car au début nous n'avons pas voulu être plus flexible au niveau de la conception de notre arbre.

5.2.2 2ème problème

Le second problème que nous avons rencontré durant notre implémentation à été la manipulation des Octets plus précisement notre capacité les bits dans le fichier les uns après les autres.
Et après de nombreuses réflexions, nous avons opter pour la manipulation d'un stucture beaucoup moins contrénuante qui est la chaines de caractères. Decidé de généralement faire passer les grandes suites d'octet de notre fichier qui ne devait cependant ne pas etre lu comme des octets mais comme bloc de 1,2,3 voir 4 bits et plus dans des unbounde_String ce qui nous grandement facilité la vie.

5.3 Etat D'avancement et Amélioration possible

Nous avons réussi à terminer le projet, c'est -à-dire que celui remplie toutes les exigences du cahier de charge. Concernant l'algorithme de compression en lui même nous n'avons pas pensé à d'éventuelle amélioration car ce dernier est optimal pour ce qu'il fait par contre, nous avons pensé à améliorer l'interface de retour pour l'utilisateur en lui donnant par exemple les informations comme le taille de son fichier avant compression puis après compression mais également le taux de compression de ce dernier, lui disant si oui ou non il aura été utile de compresser son fichier ; dans les cas où par exemple le fichier possède en moyenne pour chaque caractère le même nombre de fréquence

6 Conclusion

Chaque étape de ce projet a nécessité énormément de temps de travail, l'implémentation aura pris plus de temps à cause du fait que le travail de conception et les raffinages n'était pas bon dès le début ; en cela nous comprenons en temps que jeune programmeur excité par l'implémentation du code, la conception de ce dernier est une étape tout aussi importante à ne pas négliger.
Il en va de soi que ce projet a été des plus enrichissant pour nous, il nous a permis d'énormément apprendre sur le language ADA en quelques semaines mais d'en plus améliorer notre rigueur dans le travail, découvrir le travail d'équipe ainsi que ces avantages et ces inconvénients dù à la divergence d'idées et bien d'autres soucis.
Nous en ressortons grandement muris et feront mieux pour les projets avenir autant sur le plan individuel que collectif.