AAP - Fil Rouge :
Compte rendu

AAProximation
Maistret James, Thieffry Émile, Chevalier Romain, Feng Yanli & Hong Yutong

11 janvier 2022

Table des matières

Table des figures

1 Introduction

Le fil rouge d'AAP 21-22, visait la création d'arbre équilibrés en language C. Le fil rouge consistait à mise en place de trois programmes, le premier permet la création d'une image d'un arbre AVL à partir d'une liste de mot. Quant au second programme, il sert à indexer un dictionnaire dans un arbre AVL dont le tri des mots est basé sur leur signature. Enfin le dernier programme, qui repose sur le second, permet de déterminer les anagrammes d'un mot dans un dictionnaire.

2 Programme 1 : `displayAVL.exe`

2.1 Développement

2.1.1 Fonction de base pour créer l'arbre

Rotations simples Le programme de la rotation simple à droite était donné pour la rotation de gauche, nous avons raisonné par analogie et le calcul des facteur de déséquilibre est donné ci-dessous. En se basant sur la figure 1, on a $a = h(A_g) - h(A_d), b = h(B_g) - h(A)$ et $h(A) = 1 + \max(h(A_g), h(A_d))$ où $h(x)$ est la hauteur du noeud x. Alors

$$\begin{aligned} b' &= h(B_g) - h(A_g) \\ &= b + h(A) - h(A_g) && \text{en utilisant la définition de } b \\ &= b + 1 + \max(h(A_g), h(A_d)) - h(A_g) && \text{en utilisant la définition de } h(A) \\ &= b + 1 + \max(0, h(A_d) - h(A_g)) \\ &= b + 1 + \max(0, -a) && \text{en utilisant la définition de } a \\ &= 1 + b - \min(a, 0) \end{aligned}$$

Et de même, $a' = 1 + a + \max(b', 0)$

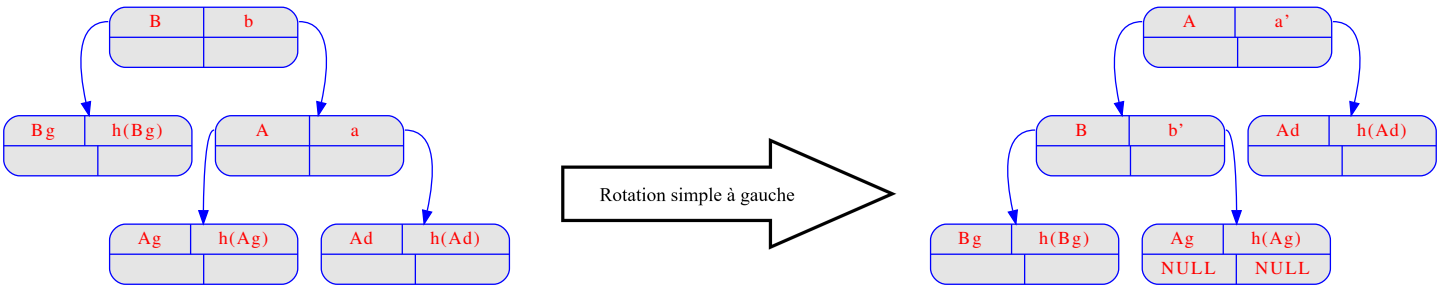


FIGURE 1 – Rotation simple à gauche

BalanceAVL Algorithme donné pour la première partie de la fonction, la seconde partie est l'exacte analogie pour un arbre qui penche de l'autre coté. Lors du développement de cette fonction, nous nous sommes heurté à un problème pour les tests afin de savoir si l'arbre penche, en effet pour faire une rotation double, il faut que le facteur de déséquilibre soit forcément de 2 en valeur absolue et pas seulement non nul, ie l'arbre est bien déséquilibré. Mais une fois rentré dans la rotation double, pour savoir si on fait une rotation double du même coté, ou des deux cotés différents il suffit que l'arbre penche, ie que le facteur de déséquilibre du second nœud a tourner soit non nul.

InsertAVL Cette fonction permet d'insérer, au bon endroit, la maille dans l'arbre AVL. Pour cela, on s'est basé sur l'algorithme donné en séance 5 que nous avons traduit en C.

FreeAVL Cette fonction permet de libérer la mémoire occupé par l'arbre à la fin de l'exécution. Pour cela, elle repose sur la fonction `free`.

2.1.2 Lecture du fichier

En se basant sur [1], nous lisons ligne par ligne le fichier, car il y a un mot par ligne dans le fichier, en ajoutant un compteur de lignes lues afin de respecter le nombre de mots à mettre dans l'arbre renseigné par l'utilisateur.

2.1.3 Affichage avec graphviz

Les fonctions permettant de générer le fichier `.dot` étaient déjà données, néanmoins le programme ne prenait en compte les noms composés (ie les noms avec un tiret dedans). Afin de corriger cette erreur, il a fallut ajouter des guillemets (donc la séquence suivant `\"%s\"`) dans les noms des blocs graphviz à afficher. De plus, l'affichage du facteur de déséquilibre n'était pas configuré, nous l'avons donc ajouté à coté du nom de famille du nœud.

2.2 Jeux de test, exemples d'exécution

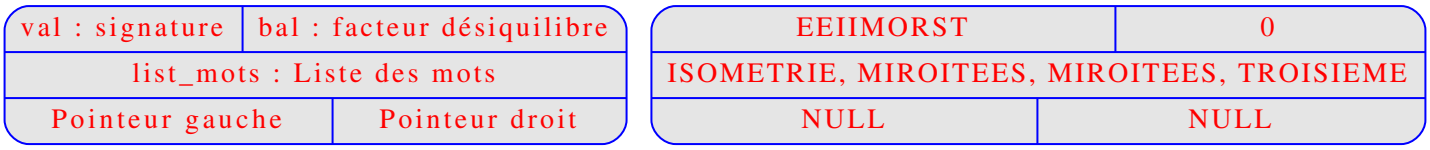
On trouve en figure 2 , les différentes étapes de construction de l'arbre pour 10 premiers mots du fichier `PrenomsV2.txt`. Un exemple d'arbre à 20 mots du fichier `PrenomsV1.txt` en figure 3.

3 Programme 2 : `indexation.exe`

3.1 Développement

3.1.1 Fonction de base pour créer l'arbre

Restructuration des mailles Pour ce programme, nous avons rajouté un champ dans chaque maille de l'arbre. Dans un premier temps, nous avons décidé que le champ `T_avl NodeAVL->val` deviendrai la signature des mots contenus dans le nouveau champ `T_avl NodeAVL->list_mots` qui contient la liste de tout les mots comportant la même signature. Voir illustration des champs d'une maille en figure 4a et un exemple en figure 4b.



(a) Champs de la maille

(b) Exemple de maille

FIGURE 4 – Restructuration des mailles pour le second programme

Calcul de la signature Pour calculer la signature d'un mot, on trie les lettres de ce mot. Dans un premier temps, nous avions utilisé les fonctions de tri fusion du TEA de la semaine 3. Et comme dans ce TEA, nous avions remarqué que le temps de tri de la fonction `qsort` était beaucoup plus faible que le temps du tri fusion. Finalement, nous utilisons la fonction `qsort` pour trier les lettres et donc calculer la signature.

Ajout de mot dans un arbre déjà construit Pour ajouter un mot dans un arbre AVL déjà construit, deux cas de figure se présente :

- Il n'y a pas de maille avec la signature de ce mot, dans ce cas, on crée une nouvelle maille avec la signature de ce mot et ce mot et on l'ajoute au bon endroit, en suivant le même algorithme que dans le programme 1 (cf. 2 - `displayAVL.exe`). Dont la comparaison entre les mailles se fait sur la signature.
- Si il y a déjà une maille avec la signature du mot à ajouter, on ajoute le mot au champ `list_mots`, en prenant garde de réallouer de la mémoire à ce champ et en concaténant le champ `list_mots` et le mot grâce à la fonction `strcat` de la librairie `string.h`.

Remarque On récupère le taille des mots dès la première ligne du fichier ouvert, et on le met en argument de chaque fonction afin de ne pas avoir à le recalculer et ainsi gagner du temps de calcul.

3.1.2 Calcul des paramètres de l'arbre

Pour calculer le nombre de noeud et la hauteur de l'arbre, on utilise la fonction `nbNodesAVL` et `heightAVL`, donnée lors de la séance 4. Pour compter le nombre de mots, noté N , on incrémente un compteur à mot qu'on ajoute à l'arbre. La taille des mots est récupéré dès l'ouverture du ficher grâce à la fonction `strlen`. Pour la durée, on fait la différence entre les heures de début et de fin de la construction. Et enfin pour calculer la hauteur minimale d'un arbre contenant le même nombre de noeud, on calcul $\max\left\{k \in \mathbb{N}, \quad 0 < \left\lfloor \frac{N}{2^k} \right\rfloor \right\}$, ce qui correspond à $\lceil \log_2(N) \rceil$.

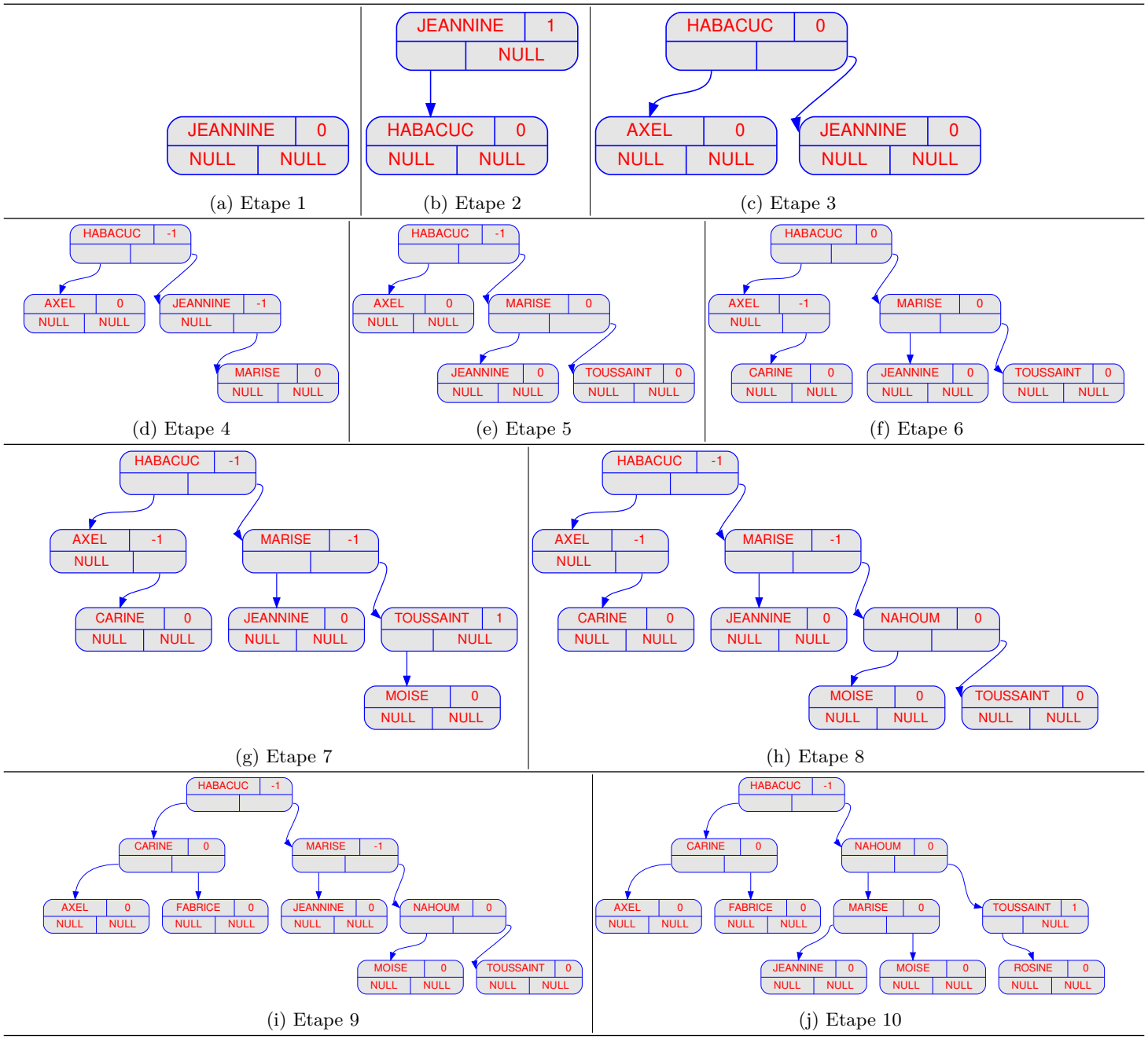


FIGURE 2 – Construction de l'arbre pour les 10 premiers prénoms de `PrenomsV2.txt`

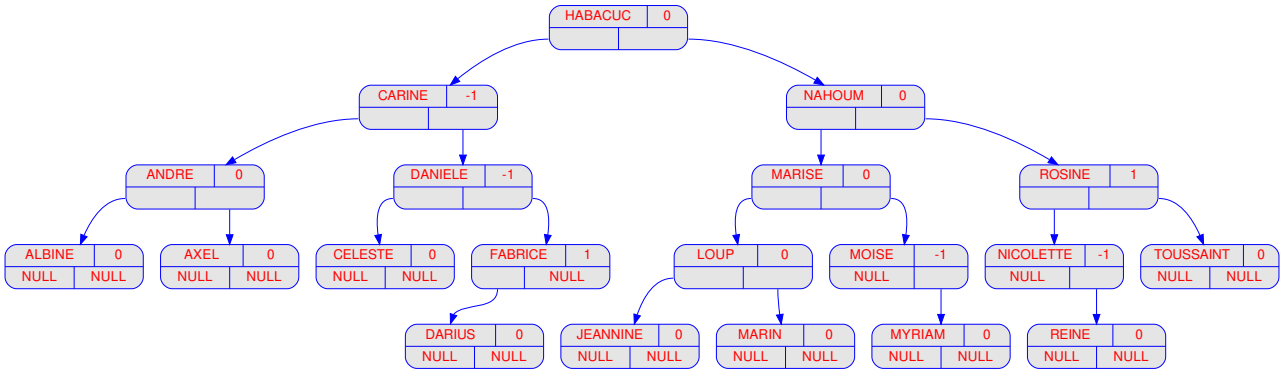


FIGURE 3 – Exemple arbre avec les 20 premiers prénoms de `PrenomsV1.txt`

3.1.3 Recherche de mots

Recherche d'un mot Pour rechercher si un mot est présent dans l'arbre, on récupère le mot entré au clavier par l'utilisateur grâce à la fonction `fgets`, avec l'argument `stdin`, en faisant bien attention de supprimer le retour chariot `\n` qui s'ajoute au mot écrit par l'utilisateur à l'aide de la commande en figure 5. Ensuite, on recherche le mot grâce à la fonction `searchAVL_rec` donnée en séance 4.

```
1    fgets(mot_ecris, 27, stdin) // 27 correspond au nombre maximum de caractères entré par l'utilisateur
     (mot le plus long de la langue française à 26 caractères)
2    mot_cherche = strndup(mot_ecris, strcspn(mot_ecris, "\n"));
3    // la variable mot_ecris correspond au mot entré par l'utilisateur
```

FIGURE 5 – Récupération mot entré par l'utilisateur pour la recherche dans l'arbre

Profondeur du noeud Pour calculer la profondeur, on compte le nombre de fois où l'on fait appel à la fonction récursive `searchAVL_rec`, pour cela on passe ce compteur en argument de la fonction de recherche.

3.2 Jeux de test, exemples d'exécution

On trouve en figure 6 la sortie du programme `indexation.exe` pour le dictionnaire `Dico_09.txt`.

```
1    Taille des mots : 9
2    Nombre de mots: 70039
3    Durée de construction: 273.47
4    Nombre de noeuds: 43444
5    Hauteur: 18
6    Hauteur minimale d un arbre contenant 43444 noeuds: 16
7    Entrer le mot à rechercher (Ctrl+D) pour terminer: RENIPPEES
8    RENIPPEES
9    Profondeur: 17
10   Temps de recherche: 0.03
11   Entrer le mot à rechercher (Ctrl+D) pour terminer: DECHARNEE
12   ADHERENCE, ADHERENCE, DECHARNEE, DECHARNEE
13   Profondeur: 9
14   Temps de recherche: 0.02
15   Entrer le mot à rechercher (Ctrl+D) pour terminer: mot
16   Mot non trouvé
17   Entrer le mot à rechercher (Ctrl+D) pour terminer: // Ctrl+D entré
```

FIGURE 6 – Execution `indexation.exe` avec dictionnaire `Dico_09.txt`

4 Programme 3 : `anagrammes.exe`

4.1 Développement

Pour construire l'arbre, on procède comme dans le programme 2 `indexation.exe` (cf. 3)

Nombre anagrammes Pour compter le nombre d'anagrammes, on utilise la fonction `nb_anagrammes`, voir en figure 7. Pour cela, on parcourt tout l'arbre récursivement et on compte le nombre de fois où le champ `list_mots` (cf. 4a) possède plus de caractères que un mot seul.

Affichage des anagrammes Afin d'afficher les anagrammes, lorsqu'on les compte dans la fonction `nb_anagrammes` on ajoute les anagrammes à un fichier externe afin de garder seulement ce qui nous intéresse de l'arbre (cf. ligne 7 en figure 7). Puis à partir de ce fichier, on crée une liste chaînée, grâce aux fonctions développées en TEA, dont chaque maille contient la liste anagrammes d'une même signature. Pour finir, on trie la liste chaînée en fonction de la longueur de la liste de mots contenue dans chaque maille, par ordre décroissant.

```c
int nb_anagramme(T_avl root, int taille_mots, FILE *fp){
  int compteur = 0; // Vaut 0 si pas d'anagramme pour cette maille et 1 si il y a des anagrammes

  if (root!=NULL){
    if (strlen(root->list_mots)>taille_mots){ // On regarde si la liste de mots de maille contient plus d'
    un mot
      compteur++; //Si c'est le cas, c'est que c'est qu'il y a des anagrammes de ce mot
            fprintf(fp, "%s\n", root->list_mots); // On ajoute les anagrammes au fichier de stockage
    }

    return compteur + nb_anagramme(root->l, taille_mots, fp) + nb_anagramme(root->r, taille_mots, fp); //
    Compte le nombre de mots du dictionnaire disposant d anagrammes
  }

  return 0;
}
```

FIGURE 7 – Fonction : `nb_anagrammes`, programme 3

4.2 Jeux de test, exemples d'exécution

On trouve en figure 8 , la liste chaînée triée par ordre décroissant pour le dictionnaire `Dico_16.txt` et en figure 9 la sortie du programme `anagramme.exe` pour ce même dictionnaire.

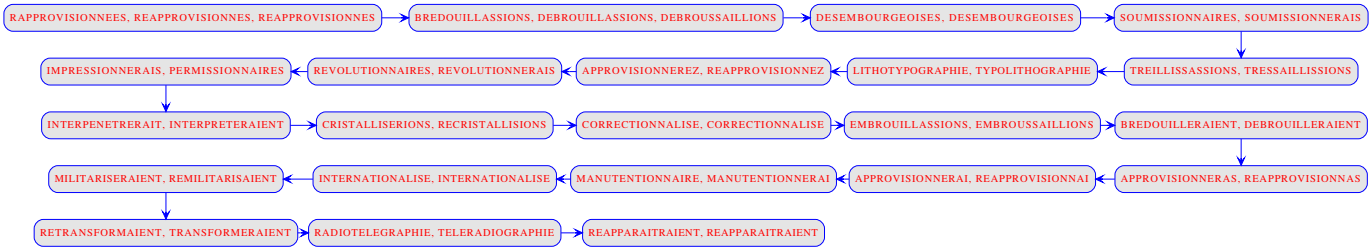


FIGURE 8 – Liste chaînée des anagrammes pour le dictionnaire `Dico_16.txt`

```
Nombre anagrammes: 22
Listes anagrammes:
RAPPROVISIONNEES, REAPPROVISIONNES, REAPPROVISIONNES
BREDOUILLASSIONS, DEBROUILLASSIONS, DEBROUSSAILLIONS
DESEMBOURGEOISES, DESEMBOURGEOISES
SOUMISSIONNAIRES, SOUMISSIONNERAIS
TREILLISSASSIONS, TRESSAILLISSIONS
LITHOTYPOGRAPHIE, TYPOLITHOGRAPHIE
APPROVISIONNEREZ, REAPPROVISIONNEZ
REVOLUTIONNAIRES, REVOLUTIONNERAIS
IMPRESSIONNERAIS, PERMISSIONNAIRES
INTERPENETRERAIT, INTERPRETERAIENT
CRISTALLISERIONS, RECRISTALLISIONS
CORRECTIONNALISE, CORRECTIONNALISE
EMBROUILLASSIONS, EMBROUSSAILLIONS
BREDOUILLERAIENT, DEBROUILLERAIENT
APPROVISIONNERAS, REAPPROVISIONNAS
APPROVISIONNERAI, REAPPROVISIONNAI
MANUTENTIONNAIRE, MANUTENTIONNERAI
INTERNATIONALISE, INTERNATIONALISE
MILITARISERAIENT, REMILITARISAIENT
RETRANSFORMAIENT, TRANSFORMERAIENT
RADIOTELEGRAPHIE, TELERADIOGRAPHIE
REAPPARAITRAIENT, REAPPARAITRAIENT
```

FIGURE 9 – Execution `anagrammes.exe` avec dictionnaire `Dico_16.txt`

5 Gestion de projet

Dans un premier temps, nous avons tous bien lu le sujet et bien compris les enjeux. Nous avons alors utilisé les séances 4 et 5 pour decomposer le projet en différentes lot (dont on retrouve la décomposition en figure 10), de se répartir les tâches et commencer le projet tous ensemble. La répartition des tâches se trouve sur la matrice RACI en figure 11 .Cela nous a permis de finir les 2 premiers programmes rapidement. Nous avons aussi été aidé par M. Slim Hammadi, notamment lorsque nous étions bloqué au programme 2, sur la comprehension du sujet d'un point technique. Nous ne comprenions pas exactement ce qu'il fallait mettre dans les maille de l'arbre (entre autres la liste des mots regroupés par signature). Nous avons chacun fini nos tâches à faire pour la rentrée et nous avons alors pu faire une mise en commun de travail et vérifier que tout fonctionne correctement.

WBS Fil Rouge (Work Breakdown Structure)

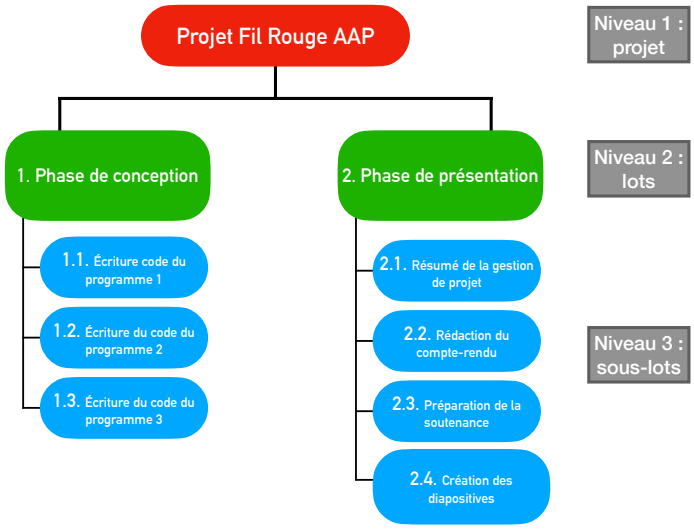


FIGURE 10 – Work Breakdown Structure- Fil rouge AAProximation

Matrice RACI du Fil Rouge (Réalise, Autorité, Conseil, Informe)

	Romain	Yanli	Yutong	James	Emile	S. HAMMADI
Écriture du code du Programme 1	R,A	R	R	R	R	C
Écriture du code du Programme 2	R,A	R	R	I	R	C
Écriture du code du Programme 3	A	R	R	R	I	
Rédaction du compte Rendu	R			I	I,A	
Création des Diapositives	I	R	R,A			
Résumé de la gestion de projet		I	I	I	R,A	
Préparation de la soutenance	R	I	I	R,A	R	

FIGURE 11 – Matrice RACI - Fil rouge AAProximation

6 Conclusion

Pour conclure, nous pouvons dire que ce projet Fil Rouge a été formateur, il nous a permis de vraiment mieux se familiariser avec le language C et de réaliser un projet de nous même avec avec les différents aspect de la gestion de projet. Nous sommes ravi d'avoir pu faire ce projet, c'était l'occasion de travailler en groupe et de mettre en relation nos connaissances apprises durant ce cursus d'algorithme, ce qui a d'ailleurs considérablement approfondi nos connaissances sur le sujet.

Références

[1] Delftstack. Lire le fichier ligne par ligne en utilisant fscanf en c. `https://www.delftstack.com/fr/howto/c/fscanf-line-by-line-in-c/`, Mars 2021.

[2] SARL Infini Software. Koor.fr. `https://koor.fr/C/Index.wp`, 2022.

Annexes : code des programmes

A Code programme 1 : `displayAVL.exe`

```c
#include <assert.h>
#include <sys/stat.h>
#include <string.h>

//#define CLEAR2CONTINUE
#include "../include/traces.h"

//#define DEBUG
#include "../include/check.h"

#include "elt.h" // T_elt
#include "avl.h" // prototypes

char * outputPath = ".";

/*
typedef enum {
    DOUBLE_RIGHT = -2,
    RIGHT = -1,
    BALANCED,
    LEFT,
    DOUBLE_LEFT
} T_bal;

typedef struct aNode{
    T_elt val; // élément contenu dans le noeud
    T_bal bal; // facteur de déséquilibre
    struct aNode *l; // pointeur vers sous-arbre gauche
    struct aNode *r; // pointeur vers sous-arbre droit
} T_avlNode, *T_avl;
*/

static T_avl newNodeAVL(T_elt e);
static T_avlNode * rotateLeftAVL (T_avlNode * A);
static T_avlNode * rotateRightAVL (T_avlNode * B);
static T_avlNode * balanceAVL(T_avlNode * A);

static T_avl newNodeAVL(T_elt e) {

    T_avl nodeAvl;
    nodeAvl = (T_avlNode*) malloc(sizeof(T_avlNode));
    CHECK_IF(nodeAvl, NULL,"erreur malloc dans newNode");
    nodeAvl->val = eltdup(e);
    nodeAvl->bal = 0;
    nodeAvl->l = NULL;
    nodeAvl->r = NULL;

```

```
48      return nodeAvl;
49  }
50
51
52
53  int insertAVL (T_avlNode ** pRoot, T_elt e) {
54
55
56      int deltaH=0;
57      if (*pRoot==NULL)
58      {
59          *pRoot = newNodeAVL(e); // Ajout d'une nouvelle maille
60          return 1;               // Modification de hauteur : on renvoie 1
61      }
62      else if (eltcmp(e, (*pRoot)->val)<=0)
63      {
64          deltaH = insertAVL(&(*pRoot)->l, e);    // insertion dans sous-arbre gauche
65          (*pRoot)->bal += deltaH;                // mise à jour du facteur de déséquilibre
66      }
67      else
68      {
69          deltaH = insertAVL(&(*pRoot)->r, e);    // insertion dans sous-arbre droit
70          (*pRoot)->bal -= deltaH;                // mise à jour du facteur de déséquilibre
71      }
72
73
74      if (deltaH == 0)
75          return 0; // pas de modification de hauteur : on renvoie 0
76      else          // le sous-arbre renvoyé par l appel récursif a grandi
77      {
78          *pRoot = balanceAVL(*pRoot); // on rééquilibre
79      }
80
81      if ((*pRoot)->bal != 0)
82          return 1; // Si l'arbre n'est pas rééquilibré, on renvoie 1 pour qu'il soit lors de l'appel ré
        cursif
83      else
84          return 0;
85
86  }
87
88
89  static T_avlNode *rotateLeftAVL(T_avlNode *B)
90  {
91      // rotation gauche
92
93      T_avlNode *A;
94      T_bal a, b;
95
96      A = B->r;
97
98      a = A->bal; // On récupère les facteurs de déséquilibre avant rotation
99      b = B->bal;
100
101     B->r = A->l; // Rotation simple à gauche
102     A->l = B;
103
104     B->bal = 1 + b - MIN2(0, a); // Mise à jour des facteurs de déséquilibre
105     A->bal = 1 + a + MAX2(B->bal, 0);
106
107     return A;
108 }
109
110
111 static T_avlNode *rotateRightAVL(T_avlNode *A)
112 {
113     // rotation droite
114     T_avlNode *B;
115     T_bal a, b;
116
117     B = A->l;
118
119     a = A->bal; // On récupère les facteurs de déséquilibre avant rotation
120     b = B->bal;
121
```

```
122     A->l = B->r; // Rotation simple à droite
123     B->r = A;
124 
125     A->bal = a - 1 - MAX2(0, b);    // Mise à jour des facteurs de déséquilibre
126     B->bal = b - 1 + MIN2(0, A->bal);
127 
128     return B;
129 }
130 
131  /*
132     A->bal  < 0 \Leftrightarrow arbre penche à droite;
133             = 0 \Leftrightarrow arbre équilibré
134             > 0 \Leftrightarrow arbre penche à gauche
135 */
136 
137 static T_avlNode *balanceAVL(T_avlNode *A)
138 {
139     // rééquilibrage de A
140 
141     if (A->bal == 2) // Penche à gauche
142     {
143         if (A->l->bal <= 0) // Penche à droite
144         {
145             A->l = rotateLeftAVL(A->l); // Rotation double :
146             return rotateRightAVL(A);   // Gauche puis droite
147         }
148         else                // Si ne penche pas
149             return rotateRightAVL(A);   // Rotation simple à droite
150     }
151     else if (A->bal == -2)  // Penche à droite
152     {
153         if (A->r->bal >= 0) // Penche à gauche
154         {
155             A->r = rotateRightAVL(A->r);    // Rotation double :
156             return rotateLeftAVL(A);        // Droite puis Gauche
157         }
158         else                // Si ne penche pas
159             return rotateLeftAVL(A);    // Rotation simple à gauche
160     }
161     else
162         return A;
163     return NULL;
164 }
165 
166 
167 void freeAVL(T_avl root)
168 {
169     // Libérer la mémoire de toutes les mailles de l'arbre
170 
171     if (root != NULL)
172     {
173         freeAVL(root->r);
174         freeAVL(root->l);
175         // printf("Libération de %s\n", root->list_mots);
176         free(root);
177     }
178 }
```

../Programme_1/avl.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <assert.h>
 4 #include <sys/stat.h>
 5 
 6 //#define CLEAR2CONTINUE
 7 #include "../include/traces.h"
 8 
 9 // C'est dans le fichier elt.h qu'on doit choisir l'implémentation des T_elt
10 #include "elt.h"
11 #include "avl.h"
12 
```

```c
int main(int argc, char ** argv) {
    T_avl root = NULL;


    ////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////

    outputPath = "output_Prog1";

    //Ouverture Fichier
    char *filename;

    filename = malloc(sizeof(filename));
    sprintf(filename, "%s",argv[1]);


    FILE *in_file = fopen(filename, "r");
    int nb_ligne = 0;
    int ligne_max = atoi(argv[2]);

    struct stat sb;
    stat(filename, &sb);

    char *file_contents = malloc(sb.st_size);

    // Ajout des mots et création du fichier graphique
    while (fscanf(in_file, "%[^\n] ", file_contents) != EOF && nb_ligne++ < ligne_max) {
        insertAVL(&root, file_contents);
        createDotAVL(root, "displayAVL");

    }

    fclose(in_file);
    printAVL(root,0);
    freeAVL(root);

    return 0;
}
```

../Programme_1/displayAVL.c

B Code programme 2 : `indexation.exe`

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <sys/stat.h>
#include <time.h>

//#define CLEAR2CONTINUE
#include "../include/traces.h"

// C'est dans le fichier elt.h qu'on doit choisir l'implémentation des T_elt
#include "elt.h"
#include "avl.h"

int main(int argc, char ** argv) {
    T_avl root = NULL;

    //Ouverture Fichier
    char *filename;

    filename = malloc(sizeof(filename));
    sprintf(filename, "%s",argv[1]);


    FILE *in_file = fopen(filename, "r");

    struct stat sb;
    stat(filename, &sb);

```

```
30      char *file_contents = malloc(sb.st_size);
31
32      //Initialisation paramètres à afficher
33      long int compteur_mots =0;
34      int taille_mots=0;
35      clock_t start, stop;
36      int hauteur;
37      long int nb_noeud, nb_noeud_temp;
38      int h_min=1; // On initialise h_min pour la hauteur minimale contenant le même nombre de noeud à 1 pour
         compter le premier noeud
39      float duree;
40
41
42      //Ajout des mots à l'arbre
43
44      fscanf(in_file, "%[^\n] ", file_contents);
45      taille_mots = strlen(file_contents);
46
47      start = clock();
48
49      insertAVL(&root, file_contents, taille_mots);
50
51
52      while (fscanf(in_file, "%[^\n] ", file_contents) != EOF){
53          insertAVL(&root, file_contents, taille_mots);
54          compteur_mots++;
55      }
56
57      stop = clock();
58
59      //printAVL(root, 0);
60
61      // Paramètres de l'arbre
62      duree = (stop-start)* 1000.0 / CLOCKS_PER_SEC;
63      nb_noeud = nbNodesAVL(root);
64      hauteur = heightAVL(root);
65
66      nb_noeud_temp = nb_noeud;
67      while ((nb_noeud_temp/=2)>0) h_min++; //La hauteur revient à h_min = round(log_2(nb_noeud)) ou encore
        en comptant le nombre de fois que l'on peut diviser nb_noeud par 2.
68
69      printf("Taille des mots : %d\nNombre de mots: %ld\nDurée de construction: %.2f ms\nNombre de noeuds: %
        ld\nHauteur: %d\nHauteur minimale d un arbre contenant %ld noeuds: %d\n",taille_mots, compteur_mots,
        duree, nb_noeud, hauteur, nb_noeud, h_min);
70
71
72      //Recherche de mots
73
74      T_elt mot_ecris = (T_elt) malloc(27*sizeof(char*)), mot_cherche ; // Taille maximale mot de 26 caractè
        re + 1 pour "\0"
75      clock_t start_rech = clock(), stop_rech;
76      int profondeur=0;
77      T_avlNode * search = NULL;
78
79      printf("Entrer le mot à rechercher (Ctrl+D) pour terminer: ");
80
81
82      while (fgets(mot_ecris, 27, stdin)!=NULL) //Récupération mot donné par l'utilisateur en boucle
83      {
84
85          mot_cherche = strndup(mot_ecris, strcspn(mot_ecris, "\n")); // On garde seulement les caractères
        avec le retour à la ligne pour la recherche
86
87
88          profondeur = 0; // Initialisation de la profondeur pour chaque recherche
89
90          start_rech = clock();
91          search = searchAVL_rec(root, mot_cherche, taille_mots, &profondeur);
92          stop_rech = clock();
93
94          // Affichage résultats de la recherche
95          if (search == NULL) printf("Mot non trouvé\n");
96          else
97          {
98              printf("%s\n", toString(search->list_mots));
```

```
99              printf("Profondeur: %d\n", profondeur);
100             printf("Temps de recherche: %.2f\n", (stop_rech-start_rech)* 1000.0 / CLOCKS_PER_SEC);
101         }
102
103
104         printf("Entrer le mot à rechercher (Ctrl+D) pour terminer: ");
105     }
106
107
108     freeAVL(root);
109     fclose(in_file);
110
111
112     return 0;
113 }
```

../Programme_2/indexation.c

C Code programme 3 : `anagrammes.exe`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <assert.h>
5  #include <sys/stat.h>
6  #include <time.h>
7
8
9  //#define CLEAR2CONTINUE
10 #include "../include/traces.h"
11
12 // C'est dans le fichier elt.h qu'on doit choisir l'implémentation des T_elt
13 #include "elt.h"
14 #include "avl.h"
15 #include "list.h"
16
17
18
19 int main(int argc, char ** argv) {
20     T_avl root = NULL;
21
22     //Ouverture Fichier
23     char *filename;
24
25     filename = malloc(sizeof(filename));
26     sprintf(filename, "%s",argv[1]);
27
28     FILE *in_file = fopen(filename, "r");
29
30     struct stat sb;
31     stat(filename, &sb);
32
33     char *file_contents = malloc(sb.st_size);
34
35     //Paramètres dictionnaire
36     int taille_mots=0;
37
38
39     //Ajout des mots à l'arbre
40
41     fscanf(in_file, "%[^\n] ", file_contents); // On récupère le premier indépendamment pour pouvoir dé
       terminer la taille des mots
42     taille_mots = strlen(file_contents);
43
44     insertAVL(&root, file_contents, taille_mots);
45
46
47     while (fscanf(in_file, "%[^\n] ", file_contents) != EOF ){
48         insertAVL(&root, file_contents, taille_mots); // On insère chaque mot
49     }
50
51     fclose(in_file);
```

```c

    /*
        A cet instant le dictionnaire est crée

        On commence alors à récupérer les anagrammes :
        On va stocker les anagrammes dans un fichier, pour ne pas avoir a reparcourir l'arbre pour les afficher
    */

    // Ouverture fichier stockage
    FILE *fichier = fopen("Liste_anagrammes.txt", "w");
    int nombre_anagrammes=0;

    // Comptage du nombre d'anagramme en les ajoutant au fichier de stockage
    nombre_anagrammes = nb_anagramme(root, taille_mots, fichier);
    printf("Nombre anagrammes: %d\n", nombre_anagrammes);

    fclose(fichier);
    freeAVL(root); // On a plus besoin de l'arbre, on libère la mémoire


    // Affichage des anagrammes, pour cela on les stock temporairement dans une liste chaînée dynamique
    FILE *fichier_lec = fopen("Liste_anagrammes.txt", "r");
    T_list list_anag = NULL;

    struct stat sb_lec;
    stat("Liste_anagrammes.txt", &sb_lec);
    T_elt lignes = malloc(sb_lec.st_size);


    while (fscanf(fichier_lec, "%[^\n] ", lignes) != EOF){
        list_anag = addNode(lignes, list_anag); // On récupère chaque mot avec son / ses anagrammes
    }


    mergesort(&list_anag); // On trie la liste en fonction de la la longueur des listes à l'intérieur en ordre décroissant
    printf("Listes anagrammes:\n"); showList(list_anag); // On affiche la liste par le début


    freeList(list_anag); //Libération mémoire
    fclose(fichier_lec);

    return 0;
}


int nb_anagramme(T_avl root, int taille_mots, FILE *fp){
    int compteur = 0; // Vaut 0 si pas d'anagramme pour cette maille et 1 si il y a des anagrammes

    if (root!=NULL){
        if (strlen(root->list_mots)>taille_mots){ // On regarde si la liste de mots de maille contient plus d'un mot
            compteur++; //Si c'est le cas, c'est que c'est qu'il y a des anagrammes de ce mot
            fprintf(fp, "%s\n", root->list_mots); // On ajoute les anagrammes au fichier de stockage
        }

    return compteur + nb_anagramme(root->l, taille_mots, fp) + nb_anagramme(root->r, taille_mots, fp); // Compte le nombre de mots du dictionnaire disposant d anagrammes

    }

    return 0;
}
```

../Programme_3/anagrammes.c