

AAP - Fil Rouge : Compte rendu

AAProximation

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

10 janvier 2022

Table des matières

1	Introduction	2
2	Programme 1 : <code>displayAVL.exe</code>	2
2.1	Développement	2
2.1.1	Fonction de base pour créer l'arbre	2
2.1.2	Lecture du fichier	2
2.1.3	Affichage avec graphviz	3
2.2	Jeux de test, exemples d'exécution	3
3	Programme 2 : <code>indexation.exe</code>	3
3.1	Développement	3
3.1.1	Fonction de base pour créer l'arbre	3
3.1.2	Calcul des paramètres de l'arbre	3
3.1.3	Recherche de mots	5
3.2	Jeux de test, exemples d'exécution	5
4	Programme 3 : <code>anagrammes.exe</code>	5
4.1	Développement	5
4.2	Jeux de test, exemples d'exécution	6
5	Gestion de projet	7
6	Conclusion	8

Table des figures

1	Rotation simple à gauche	2
4	Restructuration des mailles pour le second programme	3
2	Construction de l'arbre pour les 10 premiers prénoms de <code>PrenomsV2.txt</code>	4
3	Exemple arbre avec les 20 premiers prénoms de <code>PrenomsV1.txt</code>	4
5	Récupération mot entré par l'utilisateur pour la recherche dans l'arbre	5
6	Execution <code>indexation.exe</code> avec dictionnaire <code>Dico_09.txt</code>	5
7	Fonction : <code>nb_anagrammes</code> , programme 3	6
8	Liste chaînée des anagrammes pour le dictionnaire <code>Dico_16.txt</code>	6
9	Execution <code>anagrammes.exe</code> avec dictionnaire <code>Dico_16.txt</code>	6
10	Work Breakdown Structure- Fil rouge AAProximation	7
11	Matrice RACI - Fil rouge AAProximation	7

1 Introduction

Le fil rouge d'AAP 21-22, visait la création d'arbre équilibrés en langage 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 est la hauteur du noeud. 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, $a' = 1 + a + \max(b', 0)$

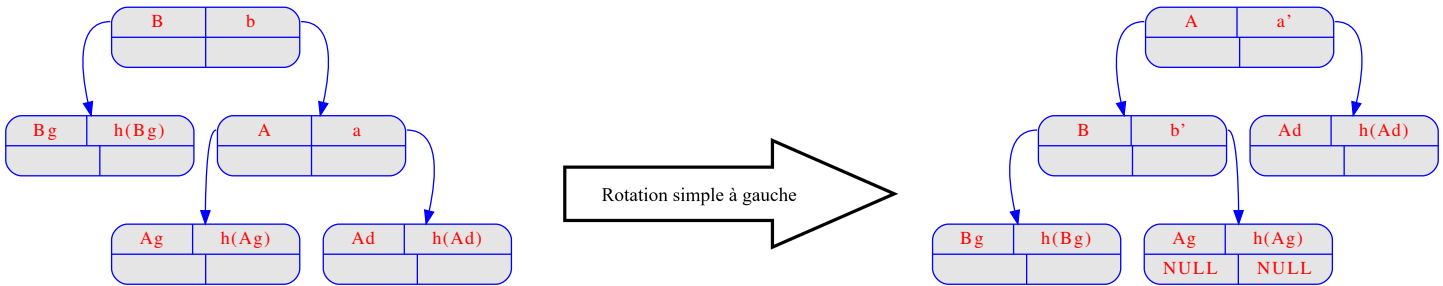


FIGURE 1 – Rotation simple à gauche

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 côté. 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 côté, ou des deux côtés 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 fallu 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 noeud.

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` deviendrait 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.

val : signature	bal : facteur déséquilibre
list_mots : Liste des mots	
Pointeur gauche	Pointeur droit

(a) Champs de la maille

EEIIMORST	0
ISOMETRIE, MIROITEES, MIROITEES, TROISIEME	
NULL	NULL

(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 avons utilisé les fonctions de tri fusion du TEA de la semaine 3. Et comme dans ce TEA, nous avons 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 fichier 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 $\lfloor \log_2(N) \rfloor$.

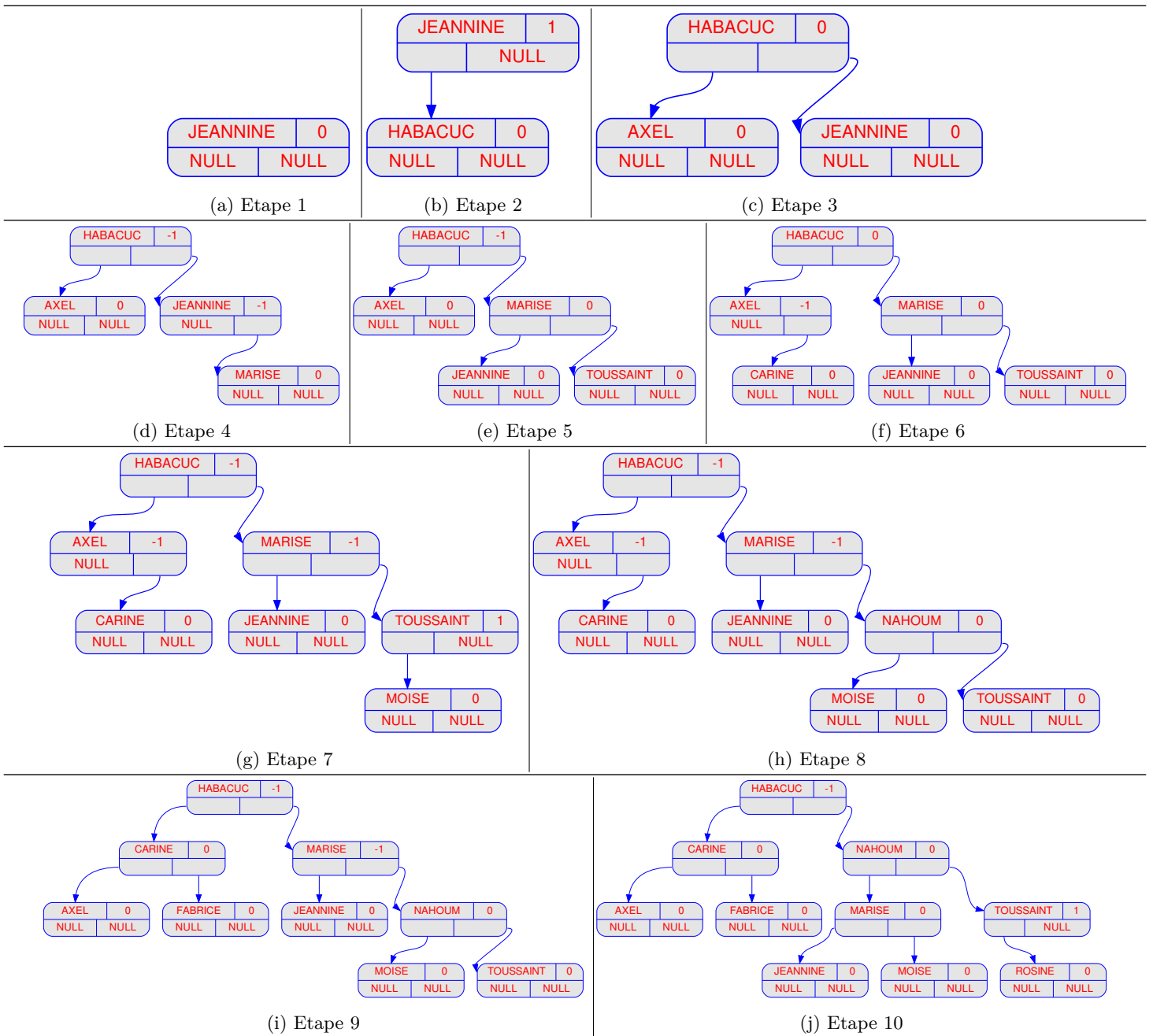


FIGURE 2 – Construction de l'arbre pour les 10 premiers prénoms de PrenomsV2.txt

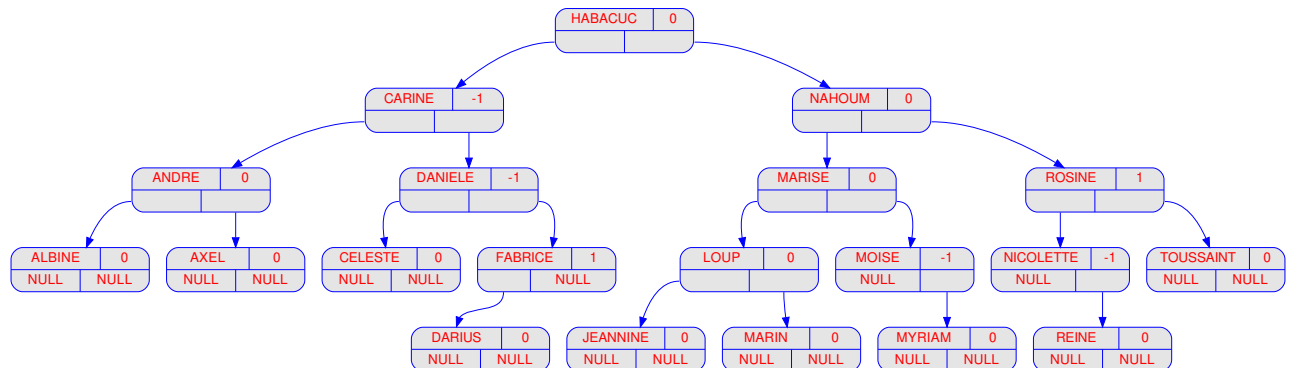


FIGURE 3 – Exemple arbre avec les 20 premiers prénoms de PrenomsV1.txt

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 = strdup(mot_ecris, strchr(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.

```

1 int nb_anagramme(T_avl root, int taille_mots, FILE *fp){
2   int compteur = 0; // Vaut 0 si pas d'anagramme pour cette maille et 1 si il y a des anagrammes
3
4   if (root!=NULL){
5     if (strlen(root->list_mots)>taille_mots){ // On regarde si la liste de mots de maille contient plus d'
        un mot
6       compteur++; //Si c'est le cas, c'est que c'est qu'il y a des anagrammes de ce mot
7       fprintf(fp, "%s\n", root->list_mots); // On ajoute les anagrammes au fichier de stockage
8     }
9
10    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
11  }
12
13  return 0;
14 }

```

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.

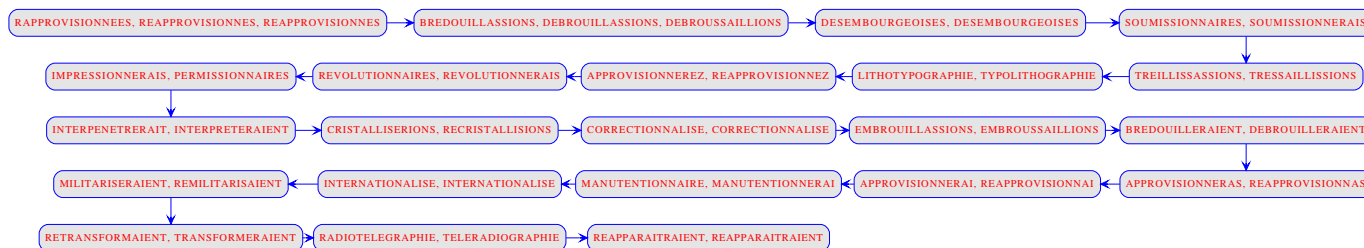


FIGURE 8 – Liste chaînée des anagrammes pour le dictionnaire Dico_16.txt

```

1 Nombre anagrammes: 22
2 Listes anagrammes:
3 RAPPROVISIONNEES, REAPPROVISIONNES, REAPPROVISIONNES
4 BREDOUILLASSIONS, DEBROUILLASSIONS, DEBROUSSAILLIONS
5 DESEMBOURGEOISES, DESEMBOURGEOISES
6 SOUMISSIONNAIRES, SOUMISSIONNERAIS
7 TREILLISSASSIONS, TRESSAILLISSIONS
8 LITHOTYPOGRAPHIE, TYPOLITHOGRAPHIE
9 APPROVISIONNEREZ, REAPPROVISIONNEZ
10 REVOLUTIONNAIRES, REVOLUTIONNERAIS
11 IMPRESSIONNERAIS, PERMISSIONNAIRES
12 INTERPENETRERAIENT, INTERPRETERAIENT
13 CRISTALLISERIONS, RECRISTALLISIONS
14 CORRECTIONNALISE, CORRECTIONNALISE
15 EMBROUILLASSIONS, EMBROUSSAILLIONS
16 BREDOUILLERAIENT, DEBROUILLERAIENT
17 APPROVISIONNERAS, REAPPROVISIONNAS
18 APPROVISIONNERAI, REAPPROVISIONNAI
19 MANUTENTIONNAIRE, MANUTENTIONNERAI
20 INTERNATIONALISE, INTERNATIONALISE
21 MILITARISERAIENT, REMILITARISAIENT
22 RETRANSFORMAIENT, TRANSFORMERAIENT
23 RADIOTELEGRAPHIE, TELERADIOGRAPHIE
24 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 décomposer le projet en différents lots (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 compréhension 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)

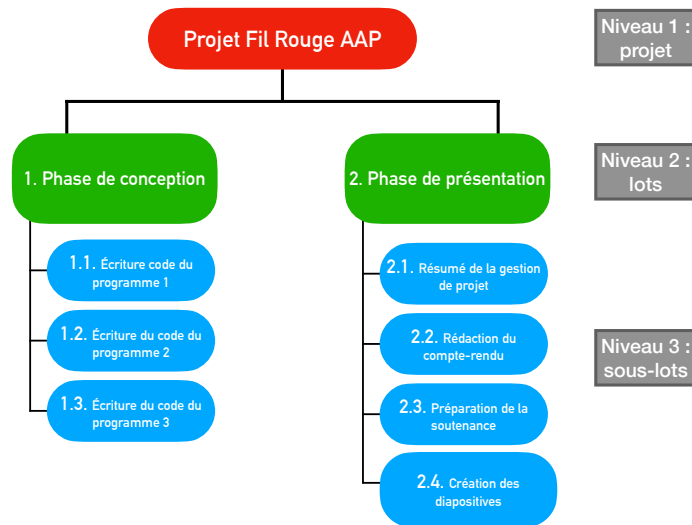


FIGURE 10 – Work Breakdown Structure- Fil rouge AAPproximation

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 AAPproximation

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 langage 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’algorithmie, 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.