



# Fil Rouge AAP

Romain Chevalier  
Yanli Feng  
Yutong Hong  
James Maistret  
Émile Thieffry

Groupe : AAPproximation



# Sommaire

- Présentation de chaque exercice
- Gestion de Projet



# Introduction / Présentation du projet



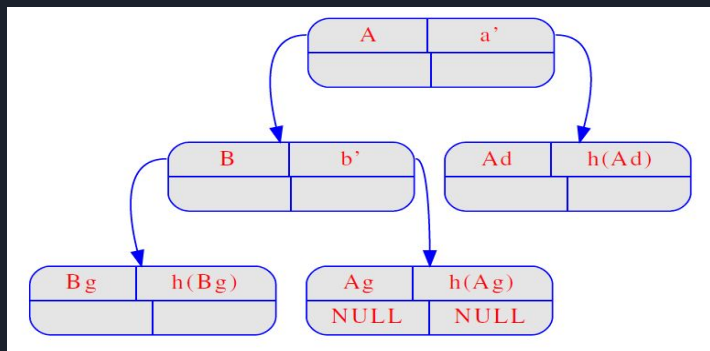
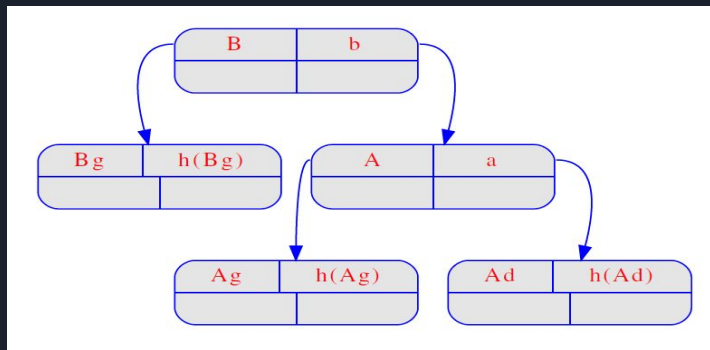
# Programme 1

## Objectifs:

- Créer les fonction de base pour un arbre AVL
- Lecture ligne par ligne d'un fichier
- Affichage avec graphviz

# Création des fonctions de base pour un arbre AVL

## Rotation Gauche



Rotation  
à  
gauche

Facteur de déséquilibre pour la rotation gauche

$$b' = 1 + b - \min(a, 0)$$

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

Ce qui se traduit en C par :

```
static T_avlNode *rotateLeftAVL(T_avlNode *B){ // rotation gauche
    T_avlNode *A;
    T_bal a, b;

    A = B->r;

    a = A->bal; // On récupère les facteurs de
    b = B->bal; // déséquilibre avant rotation

    B->r = A->l; // Rotation simple à gauche
    A->l = B;

    B->bal = 1 + b - MIN2(0, a); // Mise à jour des facteurs
    A->bal = 1 + a + MAX2(B->bal, 0); // de déséquilibre

    return A;
}
```

# BalanceAVL

```
static T_avlNode *balanceAVL(T_avlNode *A){// rééquilibrage de A
    if (A->bal == 2) // Penche à gauche
    {
        if (A->l->bal <= 0) // Penche à droite
        {
            A->l = rotateLeftAVL(A->l); // Rotation double :
            return rotateRightAVL(A); // Gauche puis droite
        }
        else // Si ne penche pas
            return rotateRightAVL(A); // Rotation simple à droite
    }
    else if (A->bal == -2) // Penche à droite
    {
        if (A->r->bal >= 0) // Penche à gauche
        {
            A->r = rotateRightAVL(A->r); // Rotation double :
            return rotateLeftAVL(A); // Droite puis Gauche
        }
        else // Si ne penche pas
            return rotateLeftAVL(A); // Rotation simple à gauche
    }
    else
        return A;
    return NULL;
}
```

# InsertAVL

→ Insertion de la maille

- ◆ NULL
- ◆ Comparaison  $\leq 0$
- ◆ Comparaison  $> 0$

`comparaison = eltcmp(e, (*pRoot)->val)`

→ Rééquilibrage

- ◆ Si la différence de hauteur est non nulle : on rééquilibre.
- ◆ Si le facteur est pas nul : différence de hauteur devient non nulle

# FreeAVL

```
void freeAVL(T_avl root)
{
    // Libérer la mémoire de toutes les mailles de l'arbre

    if (root != NULL)
    {
        freeAVL(root->r);
        freeAVL(root->l);
        // printf("Libération de %s\n", root->list_mots);
        free(root);
    }
}
```

# Lecture ligne par ligne d'un fichier

```
//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");
}
```

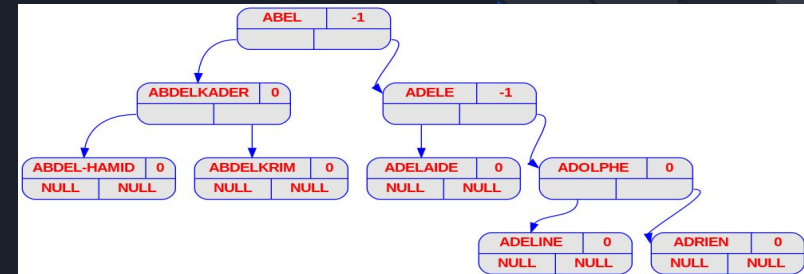
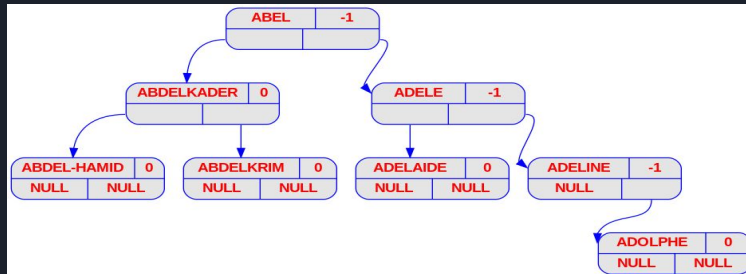
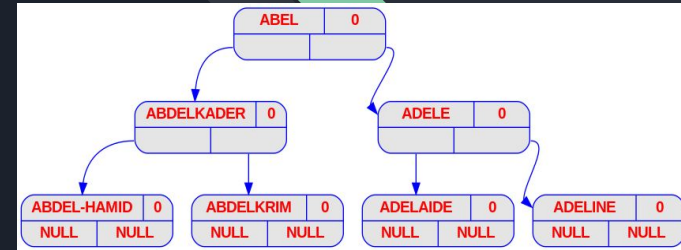
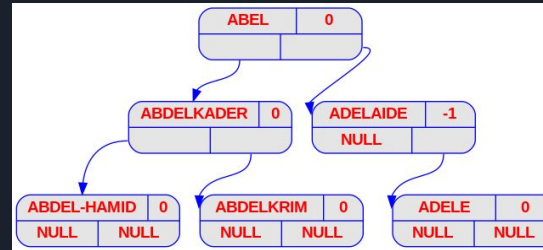
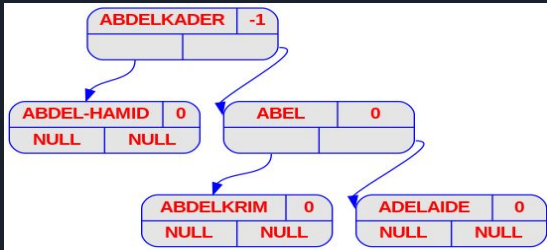
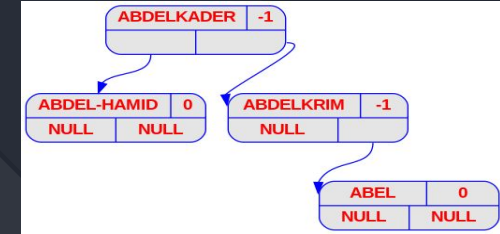
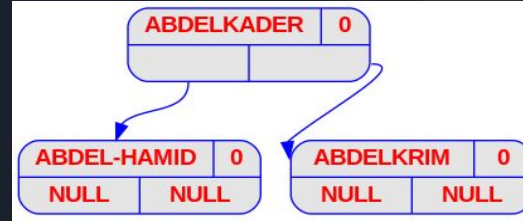
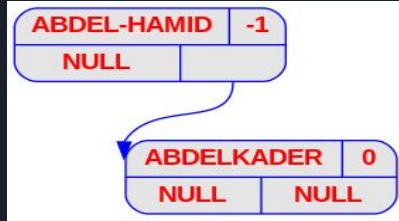
1	ABDEL-HAMID
2	ABDELKADER
3	ABDELKRIM
4	ABEL
5	ADELAIDE
6	ADELE

Extrait du fichier à extraire

Source :  
<https://www.delftstack.com/fr/howto/c/fscanf-line-by-line-in-c/>



## Affichage avec graphviz



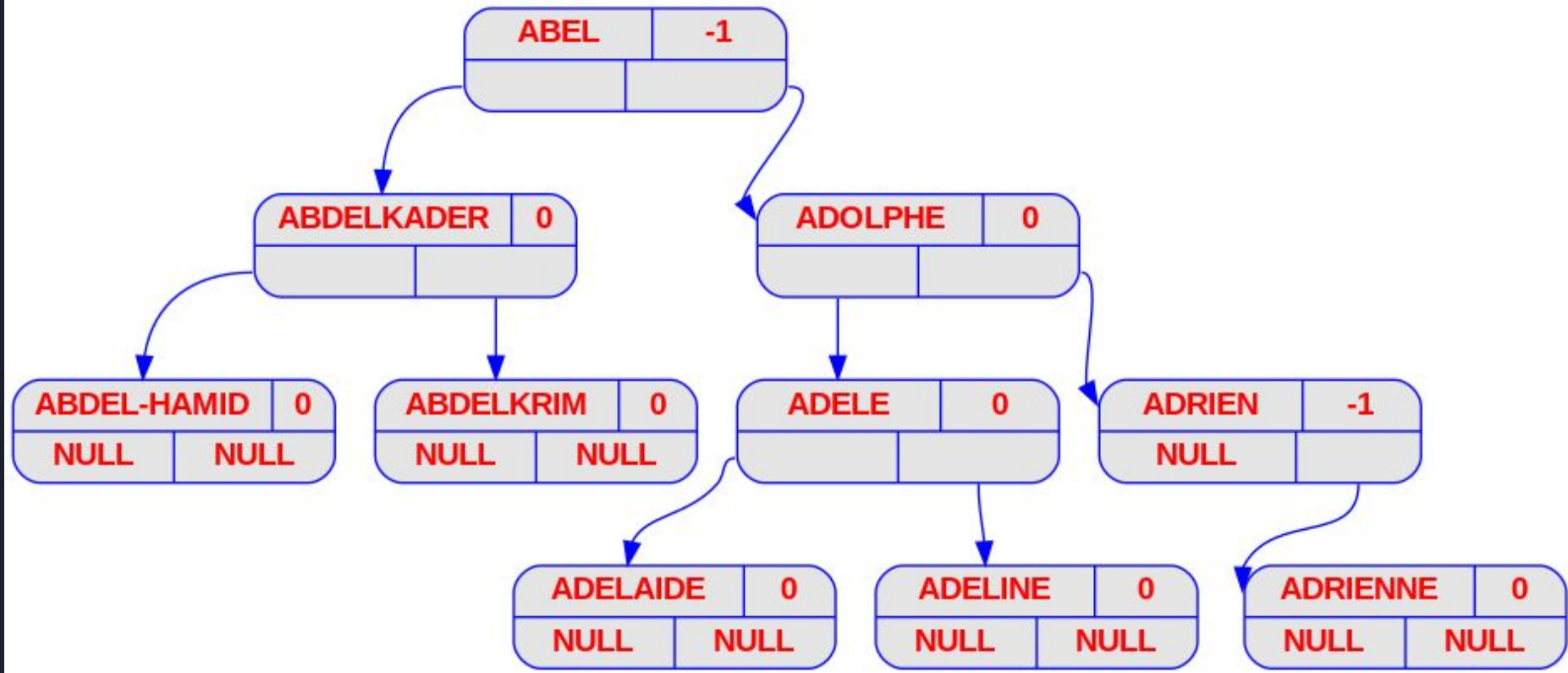
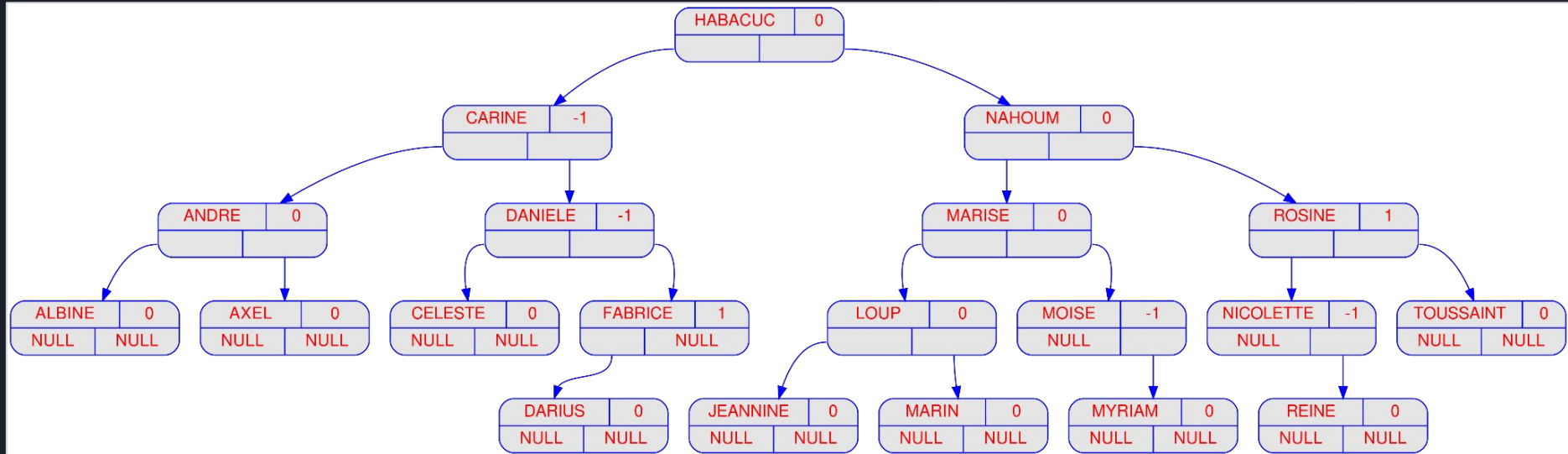


Image finale, arbre construit pour les 10 premiers prénoms de PrenomsV2.txt



Exemple arbre avec les 20 premiers prénoms de PrenomsV1.txt



# Programme 2

Objectifs :

- Développer un programme permettant d'indexer les mots d'un dictionnaire, dont le chemin est fourni en ligne de commande, dans un arbre équilibré.
- Recherche de mot dans ce dictionnaire

## Restructuration du noeud

```
typedef struct aNode{  
    T_elt val; // signature du mot  
    char *list_mot; // liste des mots  
    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;
```

**val : signature**

**bal : facteur déséquilibre**

**list\_mots : Liste des mots**

**Pointeur gauche**

**Pointeur droit**

## Calcul de la signature

maison -> aimnos

```
// Calculer la signature

static int compare (char const *a, char const *b){
    char const *pa = a, *pb = b;
    return *pa-*pb;
}

T_elt cal_signature(T_elt mot, int taille_mots)
{
    // Le calcul de la signature revient à trier la chaîne de caractère par ordre alphabétique
    char *sign;

    sign = (char *) malloc(sizeof(char *) * taille_mots);
    memcpy(sign, mot, taille_mots);

    //mergeSort_tab(sign, 0, taille_mots - 1); // On ne trie pas le caractère de fin \0,

    qsort(sign, taille_mots, sizeof(char), compare); // qsort est plus rapide que mergeSort_tab donc on l'utilise

    return sign;
}
```

# Ajout d'un mot à l'arbre

## Programme 2

```
int insertAVL(T_avlNode **pRoot, T_elt e, int taille_mots)
{
    T_elt sign = cal_signature(e, taille_mots);

    int deltaH = 0;
    if (*pRoot == NULL)
    {
        *pRoot = newNodeAVL(e, taille_mots); // Ajout d'une nouvelle maille
        return 1; // Modification de hauteur : on renvoie 1
    }
    else if (eltcmp(sign, (*pRoot)->val) == 0) // Si e a la même signature qu'un mot déjà présent dans le dictionnaire,
    { // on ajoute e a la liste de mot correspondant à cette signature
        // on rajoute de la place mémoire quand on ajoute des mots a la liste de mots

        (*pRoot)->list_mots = (T_elt)realloc((*pRoot)->list_mots, sizeof(char *) * (strlen((*pRoot)->list_mots) + taille_mots + 2)); // Le +2 corre
        strcat(strcat((*pRoot)->list_mots, " "), toString(e));
    }
    else if (eltcmp(sign, (*pRoot)->val) < 0) // On compare sur la signature
    {
        deltaH = insertAVL(&(*pRoot)->l, e, taille_mots); // insertion dans sous-arbre gauche
        (*pRoot)->bal += deltaH; // mise à jour du facteur de déséquilibre
    }
    else
    {
        deltaH = insertAVL(&(*pRoot)->r, e, taille_mots); // insertion dans sous-arbre droit
        (*pRoot)->bal -= deltaH; // mise à jour du facteur de déséquilibre
    }

    if (deltaH == 0)
        return 0; // pas de modification de hauteur : on renvoie 0
    else // le sous-arbre renvoyé par l'appel récursif a grandi
    {
        *pRoot = balanceAVL(*pRoot); // on rééquilibre
    }

    if ((*pRoot)->bal != 0)
        return 1; // Si l'arbre n'est pas rééquilibré, on renvoie 1 pour qu'il soit lors de l'appel récursif
    else
        return 0;
}
```

# Programme 2

## Calcul des paramètres de l'arbre demandés :

- La taille des mots du dictionnaire
- Le nombre de mots du dictionnaire
- La durée de construction de l'arbre en millisecondes
- Le nombre de noeuds et la hauteur de l'arbre AVL construit
- La hauteur minimale d'un arbre contenant le même nombre de noeuds

```
//Initialisation paramètres à afficher
long int compteur_mots =0;
int taille_mots=0;
clock_t start, stop;
int hauteur;
long int nb_noeud, nb_noeud_temp;
int h_min=1; // On initialise h_min pour la hauteur minimale
float duree;

//Ajout des mots à l'arbre

fscanf(in_file, "%[^\n] ", file_contents);
taille_mots = strlen(file_contents);

start = clock();

insertAVL(&root, file_contents, taille_mots);

while (fscanf(in_file, "%[^\n] ", file_contents) != EOF){
    insertAVL(&root, file_contents, taille_mots);
    compteur_mots++;
}

stop = clock();
```

```
// Paramètres de l'arbre
duree = (stop-start)* 1000.0 / CLOCKS_PER_SEC;
nb_noeud = nbNodesAVL(root);
hauteur = heightAVL(root);

nb_noeud_temp = nb_noeud;
while ((nb_noeud_temp/=2)>0) h_min++; //La hauteur revient à h_min = round(log_2(nb_noeud)) ou enc

printf("Taille des mots : %d\nNombre de mots: %ld\nDurée de construction: %.2f ms\n"
      "Nombre 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);
```

Pour la hauteur,

$$\max \left\{ k \in \mathbb{N}, \quad 0 < \left\lfloor \frac{N}{2^k} \right\rfloor \right\}$$

$$\lceil \log_2(N) \rceil$$



## Recherche d'un mot

## Programme 2

```
//Recherche de mots

T_elt mot_ecris = (T_elt) malloc(27*sizeof(char*)), mot_cherche ; // Taille maximale mot de 26 caractère + 1 pour "\0"
clock_t start_rech = clock(), stop_rech;
int profondeur=0;
T_avlNode * search = NULL;

printf("Entrer le mot à rechercher (Ctrl+D) pour terminer: ");

while (fgets(mot_ecris, 27, stdin)!=NULL) //Récupération mot donné par l'utilisateur en boucle
{

    mot_cherche = strdup(mot_ecris, strchr(mot_ecris, '\n')); // On garde seulement les caractères avec le retour à la ligne pour la recherche

    profondeur = 0; // Initialisation de la profondeur pour chaque recherche

    start_rech = clock();
    search = searchAVL_rec(root, mot_cherche, taille_mots, &profondeur);
    stop_rech = clock();

    // Affichage résultats de la recherche
    if (search == NULL) printf("Mot non trouvé\n");
    else
    {
        printf("%s\n", toString(search->list_mots));
        printf("Profondeur: %d\n", profondeur);
        printf("Temps de recherche: %.2f\n", (stop_rech-start_rech)* 1000.0 / CLOCKS_PER_SEC);
    }

    printf("Entrer le mot à rechercher (Ctrl+D) pour terminer: ");
}
```

On cherche aussi :

- La liste des mots présentant la même signature que le mot saisi

- La profondeur du noeud contenant ce mot dans l'arbre

- Le temps nécessaire pour trouver ce mot dans l'arbre en millisecondes

## Recherche d'un mot - Exemple

```
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
```



# Programme 3

## Objectifs :

- Développer un programme permettant de rechercher tous les anagrammes présents dans le dictionnaire

## Contraintes :

- afficher le nombre de mots ayant des anagrammes
- Afficher les anagrammes en les triant par nombre d'anagrammes décroissant

# Affichage du nombre d'anagrammes

- Création d'une nouvelle fonction "nb\_anagrammes"
- Décompte de l'arbre de façon **récursive**

```
james@DELLMINATOR:~/AAP/Test/Fil_Rouge/Programme_3$ ./anagrammes.exe Dico_nn/Dico_17.txt  
Nombre anagrammes: 12
```

# Fonction nb\_anagramme

```
int nb_anagramme(T_avl root, int taille_mots, FILE *fp){
    //Compte le nb d'anagramme dans un arbre AVL et ajoute les anagrammes au fichier *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;
}
```

Récurtivité

comptage du nombre de fois où le champ list\_mots possède plus de caractères qu'un mot seul


# Affichage des anagrammes

- Dans la fonction nb\_anagramme, on ajoute au fur et à mesure les anagrammes à un fichier externe
- Création d'une liste chaînée, dont chaque maille contient la liste anagrammes d'une même signature
- Puis tri dans l'ordre décroissant

```
james@DELLMINATOR:~/AAP/Test/Fil_Rouge/Programme_3$ ./anagrammes.exe Dico_nn/Dico_18.txt
Nombre anagrammes: 3
Listes anagrammes:
APPROVISIONNERIONS, REAPPROVISIONNIONS
INSTITUTIONNALISES, INSTITUTIONNALISES
CONSTITUTIONNALISE, CONSTITUTIONNALISE
```

# Création des listes d'anagrammes

```
int nb_anagramme(T_avl root, int taille_mots, FILE *fp){
    //Compte le nb d'anagramme dans un arbre AVL et ajoute les anagrammes au fichier *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;
}
```

```

// 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

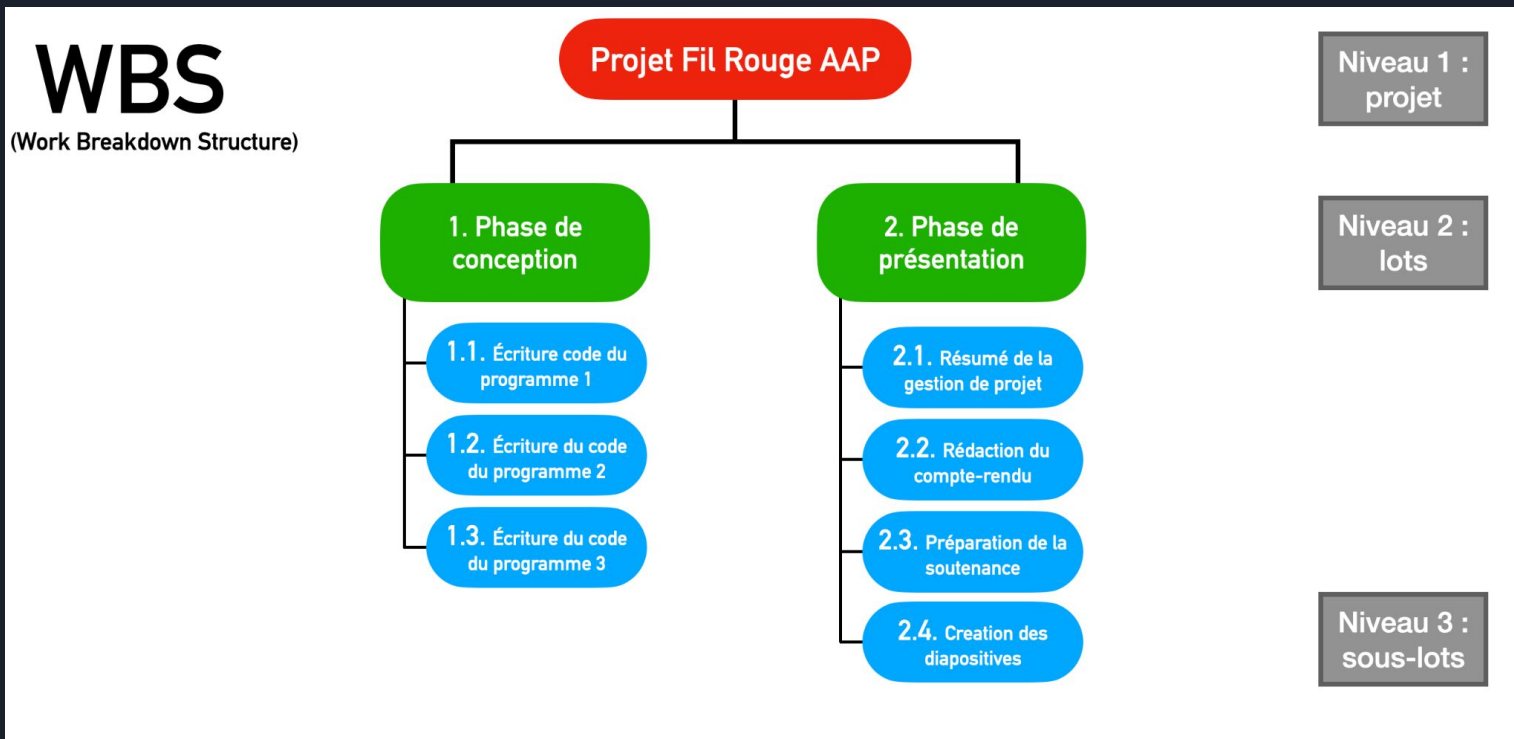
freeList(list_anag); //Libération mémoire
fclose(fichier_lec);

```

Récupération des anagrammes, création et trie de la liste chaînée contenant les anagrammes



# Gestion de projet



# 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	



# Conclusion

- Meilleure familiarisation avec le C
- Aspect de la gestion de projet
- Mise en application des connaissances apprises depuis l'entrée à Centrale