Projet C-Wildwater

BEN ABDESSELAM Youcef
MARCHAND Maxence
VANDEL Kylian

2025/2026 **Pré-Ing2 MI2**

1. Introduction

Ce projet s'inscrit dans le cadre de notre formation en programmation. L'objectif était de concevoir une application modulaire capable de simuler un réseau de distribution d'eau et de détecter des fuites, le tout piloté par une interface en ligne de commande (Shell).

La complexité du projet résidait moins dans le code final que dans l'architecture des données : nous devions faire cohabiter un Shell interactif, des arbres AVL pour un stockage optimisé, et des arbres n-iaires pour la modélisation physique du réseau. Ce rapport détaille nos choix techniques, notre organisation, et surtout les difficultés que nous avons surmontées pour arriver au rendu final.

2. Organisation et Méthodologie de travail

2.1 Gestion du temps et répartition

Dès le lancement du projet début décembre, nous avons compris que la clé serait la séparation des tâches pour ne pas modifier les mêmes fichiers en même temps.

- **Maxence** a pris le rôle de chef d'orchestre : création du squelette, gestion des fichiers d'en-tête (.h) et développement du Shell.
- **Kylian** s'est concentré sur la logique métier : la simulation hydraulique et la structure en arbres n-iaires.
- **Youcef** s'est chargé de la "salle des machines" : la structure de données AVL pour garantir des performances optimales lors des recherches.

La répartition définitive des rôles nous a permis de travailler de manière autonome.

2.2 Une approche basée sur l'auto-formation

En analysant l'historique de notre dépôt, vous noterez peut-être une absence de "commits" réguliers durant les premières séances de TD. Nous tenons à expliquer ce point en toute transparence. En début de projet, nous nous sommes rendu compte que nos bases en manipulation de pointeurs complexes et en récursivité étaient trop fragiles pour coder directement en séance. Plutôt que de produire du code “brouillon" ou de copier sans comprendre, nous avons pris le parti de travailler chacun de notre côté nos cours pour nous mettre à niveau.

Cette phase de "l'ombre" a duré jusqu'à mi-décembre. C'est pourquoi l'avancement semble s'accélérer brusquement vers le 18 décembre (finalisation des AVL par Youcef). Ce n'est pas du travail de dernière minute, mais l'aboutissement d'une longue phase de tests et d'apprentissage en local.

3. Le Shell et l'Interaction Utilisateur

(Module développé par Maxence Marchand)

Le Shell est la porte d'entrée de notre programme. C'est lui qui orchestre les appels aux différentes bibliothèques que nous avons créées. L'enjeu ici n'était pas algorithmique, mais ergonomique : il fallait que l'utilisateur puisse interagir avec le réseau sans connaître le code source. En réalité, le code du shell en lui-même n'est pas très compliqué. On vérifie les conditions (nb d'arguments, que ce soit les bons etc), et selon ce qui nous est donné. On prend le .dat et on fait un premier tri juste pour garder ce qui nous intéresse, donc par exemple juste les lignes qui définissent l'usine directement. Bref, rien d'horrible d'un point de vue uniquement code.

Les difficultés rencontrées

Le plus dur a été de coder sur un langage que je ne connaissais pas. La première étape était de faire les TD pour avoir un début de connaissance du langage. Et ensuite d'apprendre comment il fonctionne pour pouvoir recoder derrière ce dont j'ai besoin. Il faut aussi trouver les fonctions existantes qui font ce qui m'intéresse, mais avec tous les lettres qui existent, donc le nombre de possibilités… Finalement, on trouve ces fonctions. Par exemple, awk qui est la fonction parfaite pour ce qu'on nous demande, pour faire la première épuration pour limiter le nombre de lignes qui rentre dans l'AVL, trap pour enclencher une dernière fonction même s'il y a un exit, c'était parfait. Après il faut comprendre ce qui différencie les lignes qui nous intéresse de celle qui ne nous intéresse pas, heureusement c'est décrit dans le PDF de base donc ça peut aller. Et enfin, écrire pour gnuplot, qui est plutôt simple dans son fonctionnement, mais ça reste autre chose que ce qu'on fait d'habitude, donc c'est ennuyeux. Mais sinon, de manière générale, j'ai jamais été bloqué donc ça allait.

4. Structure de Données : Les Arbres AVL

(Module développé par Youcef Ben Abdesselam)

Cette partie constitue le cœur du stockage de notre application. Pour gérer les données du réseau de manière efficace, nous ne pouvions pas nous contenter de listes chaînées ou de tableaux statiques qui auraient ralenti le programme avec de gros volumes de données.

4.1 Le choix des AVL face aux ABR

Au départ, l'option la plus simple semblait être l'Arbre Binaire de Recherche (ABR) standard. Cependant, nous avons identifié une faille majeure : si les données arrivent déjà triées (exemple : ID 1, ID 2, ID 3...), un ABR se comporte comme une

simple liste chaînée. La complexité de recherche passe alors de O(log n) (très rapide) à O(n) (lent).

Youcef a donc implémenté des arbres AVL. C'est une structure auto-équilibrante qui garantit que la différence de hauteur entre le sous-arbre gauche et le sous-arbre droit ne dépasse jamais 1 (en valeur absolue) (wikipédia). Cela assure une performance constante, peu importe l'ordre d'insertion.

4.2 Structure et Logique d'Agrégation

La structure Station est définie dans avl.h. Contrairement à un AVL d'entiers simple, j'ai dû gérer des identifiants textuels et des cumuls de données :

```
typedef struct Station {
    char* id;
    long capacite;
    long volume;
    int hauteur;
    struct Station* gauche;
    struct Station* droit;
} Station;
```

Une particularité importante de mon implémentation se trouve dans la fonction insererStation. Contrairement à un arbre classique qui rejetterait un doublon, mon programme gère l'agrégation :

- Si l'ID n'existe pas : on crée un nouveau nœud.
- Si l'ID existe déjà (détecté par strcmp) : on ne crée pas de nœud, mais on additionne le nouveau volume au volume existant (nœud->volume += volume).

4.3 Difficultés techniques surmontées

Ce module a été finalisé le 18 décembre après une longue phase de débogage. Voici les obstacles rencontrés :

- **Gestion de la mémoire et chaînes de caractères :** Contrairement à des entiers, manipuler des IDs textuels (char*) demande de la rigueur. J'utilise strdup pour dupliquer l'ID lors de la création d'une station. Le piège était d'oublier de libérer cette mémoire spécifique. J'ai dû écrire une fonction libererArbre qui effectue un parcours postfixe pour free(station->id) avant de free(station), sinon cela créait des fuites mémoires invisibles.

- **Le cauchemar des rotations :** Lors des doubles rotations (Gauche-Droite ou Droite-Gauche), la réassignation des pointeurs est complexe. Au début, je perdais souvent des sous-arbres entiers, causant des erreurs de segmentation. J'ai dû schématiser chaque étape sur papier pour m'assurer que l'arbre gardait sa cohérence.

- **Le Parcours de sortie :** Pour générer le fichier final, j'ai utilisé une fonction récursive traiterSortie. La difficulté était de choisir le bon ordre de parcours pour respecter le format attendu, tout en écrivant les données accumulées (Capacité et Volume total) dans le fichier CSV via fprintf.

5. Module 3 : Simulation du Réseau (Arbres N-iaires)

(Module développé par Kylian Vandel)

Si les AVL sont destinés à conserver des données organisées, la représentation physique d'un réseau hydraulique nécessite une contrainte distincte de celle d'un arbre binaire : un tuyau ne se sépare pas nécessairement en deux. Pour modéliser un réseau hydraulique, on utilise des arbres n-aires où chaque nœud détient une liste chaînée de descendants. Chaque nœud symbolise un participant du réseau et renferme son identifiant, le taux de fuite du segment entrant ainsi que ses descendants. L'élaboration du réseau se base sur un AVL auxiliaire qui facilite la recherche du parent d'un nœud lors de son insertion, assurant ainsi une complexité logarithmique. L'objectif principal de ce module est de calculer les pertes d'eau. L'algorithme, à partir d'un identifiant d'usine fourni par l'utilisateur, effectue une exploration récursive de l'ensemble du réseau aval. Ce calcul repose fortement sur la récursivité, chaque appel représentant une descente dans l’arbre. Le résultat final correspond au volume total d’eau perdu après traitement par l’usine, conformément aux exigences du projet.

6. Protocole de Tests et Validation

Compte tenu de la complexité des structures manipulées, nous avons adopté une stratégie de **tests unitaires**. Chaque module a été validé isolément avant l'intégration finale.

6.1 Validation du Shell

```
maxen@TRES-TRES-CONTENT-J-ADORE:~/Projet$ ./Projet.sh c-wildwater_v3.dat histo src
La durée totale du traitement a été de 3626 millisecondes.
```

6.2 Validation du module AVL

Pour vérifier la robustesse de l'équilibrage, j'ai créé un programme de test dédié (avl_test_main), séparé du projet principal.

Test 1 : Exécution et traitement des données L'image ci-dessous montre l'exécution de mon programme de test sur le jeu de données complet. Le programme lit le fichier source et génère un fichier de sortie sans erreur d'exécution.

```
PS D:\VSC codes\projet> ./test_avl c-wildwater_v3.dat resultat.txt
```

Test 2 : Vérification du tri L'analyse du fichier de sortie resultat.txt confirme que les données sont correctement triées par identifiant croissant, preuve que la structure AVL fonctionne correctement malgré les milliers d'insertions.

```
Unit #ZY000149H;2016;1848
Unit #ZX0014110;4798;4344
Unit #ZX001045V;5815;5009
Unit #ZX000822V;6754;6077
Unit #ZX000643Q;5135;4395
Unit #ZX000612S;3718;3401
Unit #ZW001305C;6525;6089
Unit #ZW000760D;7876;7650
```

6.3 Validation de la Simulation Réseau

leaks.dat

```
1   Station(ID);Volume fuité(M.m^3/an)
2   Plant #JA200000I;-1.000000
3
```

7. Limitations Fonctionnelles

Conformément aux demandes, voici l'état du projet rendu :

Tout fonctionne, tout du moins les consignes de base. On n'a cependant pas fait les bonus.

Et la seule chose dérangeante, c'est dans 'vol_min50.png' où nous n'arrivons pas à avoir les ID des 50 usines bien visibles car ça prend trop de places sur l'image dans tous les sens possibles.

8. Conclusion

Finalement, ce projet nous a permis à chacun de devenir meilleur sur chacun des domaines sur lesquels nous avons travaillé, que ce soit par des connaissances ou des méthodes. Ça nous a aussi permis d'apprendre à combiner le C avec le shell. Et nous nous sommes rendu compte que les AVL sont plutôt utiles pour faire des arbres triés. De plus, quand on a un grand nombre d'arbres, être au pire à O(log n) est très pratique en termes de temps. Ça nous a aussi permis d'expérimenter le travail en groupe qui est chose rare pour nous normalement en code. Finalement, avec plus de temps pris, nous aurions pu essayer de faire les bonus mais on ressort quand même avec de nouveaux acquis personnels non négligeables.

Voici le planning de réalisation et la répartition des tâches au sein du groupe sous forme de tableau :

Répartition des tâches

Maxence	Kylian	Youcef
Création du MindMap	Création du MindMap	Réalisation de la partie AVL qui servira
Création des fichiers nécessaires à rendre	Partie C - fuites	Rédaction du PDF
Répartition des rôles et tu travail de chacun	Finit partie C	Ajout des tests
Partie Shell	Finalisation du PDF	Finalisation README.md
Liaison + correction de tous les fichiers pour q		Réalisation du test final
Ajout des historigrammes en image		Correction du code global
Ajout des tests du bon fonctionnement final de		Agencement du dépôt Git
Avancement sur la rédaction du PDF		Finalisation du PDF

Planning de réalisation

Date	Réalisation	Personne
02/12/2025	Création du MindMap	Maxence
02/12/2025	Création du MindMap	Kylian
02/12/2025	Création des fichiers nécessaires à rendre	Maxence
08/12/2025	Répartition des rôles et tu travail de chacun	Maxence
15/12/2025	Avance sur sa partie C du projet	Kylian
18/12/2025	Finalisation de la partie AVL	Youcef
18/12/2025	Rédaction du PDF	Youcef
19/12/2025	Ajout des tests	Youcef
20/12/2025	Finit partie C	Kylian
20/12/2025	Finit partie SHELL	Maxence
21/12/2025	Liaison + correction de tous les fichiers pour que ça puisse fonctionner	Maxence
21/12/2025	Ajout des historigrammes en image	Maxence
21/12/2025	Ajout des tests du bon fonctionnement final de notre programme	Maxence
21/12/2025	Avancement sur la rédaction du PDF	Maxence
21/12/2025	Finalisation du PDF	Kylian
21/12/2025	Finalisation README.md	Youcef
21/12/2025	Réalisation du test final	Youcef
21/12/2025	Correction du code global	Youcef
21/12/2025	Agencement du dépôt Git	Youcef
21/12/2025	Finalisation du PDF	Youcef