Ahmed A.
Abdelwaheb A.
Remi S.

MI-4 tri-K

Rapport C-Wire

Introduction :

Le projet C-WIRE consiste à concevoir un outil performant pour l'analyse des données des stations électriques à l'aide d'un arbre AVL, permettant une gestion optimisée des informations. L'objectif est de lire et traiter efficacement des fichiers CSV, tout en intégrant des fonctionnalités telles que le calcul des consommations spécifiques, la détection des anomalies, et l'automatisation via des script Shell.

Planning :

Nous avons travaillé sur notre projet en utilisant une plateforme en ligne appelée Discord. Cet outil nous a permis de collaborer efficacement en échangeant des idées, en partageant notre code en temps réel et en réfléchissant ensemble pour résoudre les problèmes rencontrés. Grâce à cette organisation, nous avons pu nous corriger mutuellement et identifier les nombreuses erreurs survenues au cours de la préparation du projet.

Avant de commencer, l'idée d'organiser notre travail en plusieurs « blocs » (avec une séparation claire entre les fichiers .h et .c) a été proposée par Ahmed. Par ailleurs, nous avons développé nos propres fonctions de débogage, qui se sont révélées extrêmement utiles tout au long du projet. Ces outils nous ont permis de diagnostiquer rapidement des erreurs, notamment dans la gestion des arbres AVL et le traitement des fichiers CSV, nous faisant ainsi gagner un temps précieux, surtout dans les moments critiques où nous étions bloqués.

En ce qui concerne la répartition des tâches, Rémi et Ahmed ont pris en charge le développement du script Shell destiné à l'automatisation des processus. Ces scripts ont facilité le traitement des fichiers CSV en assurant une interface fluide entre les données et la partie du programme écrite en C qui a été réalisée par Ahmed et Abdelwaheb.

Abdelwaheb a, quant à lui, travaillé sur l'implémentation des fonctions de gestion des arbres AVL, incluant les opérations classiques telles que les rotations simples et doubles (gauche et droite), l'équilibrage, et les calculs nécessaires. Ahmed a développé la fonction insertAndSumAVL, qui combine l'insertion dans l'arbre AVL et la mise à jour des consommations pour les stations existantes, tout en maintenant l'équilibrage de l'arbre. Cette approche a permis de simplifier considérablement la complexité du code.

Enfin, Ahmed et Abdelwaheb ont collaboré sur les fonctions CSVtoAVL et AVLtoCSV. La fonction CSVtoAVL permet d'importer les données d'un fichier CSV dans un arbre AVL, tandis que AVLtoCSV exporte le contenu d'un arbre AVL vers un fichier CSV, assurant ainsi une gestion bidirectionnelle efficace entre les données et la structure de l'arbre.

Limitations et Difficultés Rencontrées

La seule difficulté majeure résidait dans la compréhension globale du sujet, notamment en ce qui concerne les différentes fonctions nécessaires pour relier les deux parties du projet, écrites dans des langages différents. Ce défi a été renforcé par la densité et la complexité du sujet, réparti sur une douzaine de pages riches en informations. Il nous a fallu du temps pour trier les informations pertinentes, définir les priorités, et déterminer une méthode de travail structurée pour savoir par où commencer efficacement.

Le projet repose sur plusieurs fonctionnalités clés :

Lecture et traitement des données CSV *'readCSVtoAVL'* :

- Le programme lit les fichier CSV contenant les informations des stations électriques (ID, Capacité, Charge) ligne par ligne et valide les données extraites et insère les stations dans l'arbre AVL. Ceci est fait grâce à la fonction *'insertAndSumAVL'.*

Celle-ci vérifie chaque entrée pour s'assurer qu'elle respecte les contraintes (ID positif, valeur positive pour la capacité, etc) avant de la traiter. En effet, cela est possible par l'appel de la fonction *'validateStationData'.*

Gestion avec un arbre AVL :

Fonctions de rotation *('LeftRotation', 'RightRotation', 'doubleLeftRotation', 'doubleRightRotation')* :

- Ce sont tout simplement les fonctions usuelles d'un arbre AVL, ces opérations de rotation maintiennent l'équilibre de l'AVL après chaque insertion (ou mise à jour).

Le fonctionnement est basique, les fonctions réalignent les sous-arbres lorsque le facteur d'équilibre est inférieur à '-2' ou supérieur à '2'.

- Encore une fois la fonction *'insertAndSumAVL'* insère les stations dans l'AVL tout en équilibrant l'arbre et en mettant à jour les consommations des nœuds existants.

Elle utilise les rotations pour maintenir l'arbre équilibré et ajoute la charge si un ID est déjà présent.

Gestions de output du programme C :

Fonction *'AVLtoCSV'* :

- Comme dit précédemment, celle-ci exporte le contenu de l'arbre AVL dans un fichier CSV.

Le programme ouvre le fichier en mode écriture, puis, appelle *'writeAVL'* pour écrire les données de l'AVL dans le fichier en respectant le format CSV. Si un problème d'écriture survient, alors l'erreur est signalée.

Fonction *'freeAVL'* :

- Cette fonction libère toute la mémoire allouée à l'arbre AVL.

La fonction va parcourir récursivement l'arbre AVL en libérant les nœuds un par un. Enfin, elle va retourner un pointeur *'NULL'* pour indiquer que la mémoire a été nettoyée.

Shell :

Validation des données

Une des étapes importantes du script est la validation des données d'entrée. Le script vérifie les paramètres d'entrée, que le fichier CSV existe et que les types de stations et de consommateurs spécifiés sont valides. Par exemple, certains types de combinaisons de station et de consommateur sont interdits, comme "hvb all" ou "hva indiv", et le script arrête l'exécution si une de ces combinaisons est détectée. Ce mécanisme assure que les analyses effectuées sont toujours cohérentes avec les règles du projet.

Une fois les paramètres vérifiés et les répertoires créés, le script passe à l'analyse proprement dite. Selon le type de station et de consommateur sélectionné, le script configure des titres spécifiques et prépare les données pour leur traitement. Par exemple, si la station est de type "hvb" ou "hva", le consommateur sera automatiquement considéré comme une entreprise (comp). De même, pour les stations "lv", le consommateur peut être soit une entreprise, soit un individu, soit tout le monde (all).

Traitement des Données CSV

Le cœur du script repose sur la fonction CSV_treatement, qui effectue un traitement détaillé des données extraites du fichier CSV. En fonction du type de station (STATION_TYPE) et du type de consommateur (CONSUMER_TYPE), des filtres spécifiques sont appliqués aux données. Par exemple, si le type de station est hva et que le PLANT_ID est égal à -1 (quand l'id de la centrale n'a pas été spécifié), le script extrait uniquement les lignes où certaines colonnes contiennent des valeurs spécifiques, en remplaçant les caractères "-" par des zéros pour normaliser les données. Le résultat est enregistré dans un fichier temporaire, prêt à être traité par la suite. Ce processus est automatisé et ajusté pour s'adapter à différentes configurations de données.

Après l'exécution du programme C, le script procède à un tri des résultats en fonction de la capacité des stations, puis modifie l'en-tête du fichier de sortie pour y inclure des informations spécifiques au type de station et au type

de consommateur. Si les conditions le requièrent (pour lv all), le script effectue des calculs supplémentaires pour obtenir le lv_all_minmax et génère des fichiers spécifiques pour ces résultats. Enfin, il produit une visualisation graphique des résultats via un graphique généré avec l'outil gnuplot. Le processus de génération des graphiques permet de visualiser l'efficacité électrique des différentes stations et consommateurs de manière claire et intuitive.