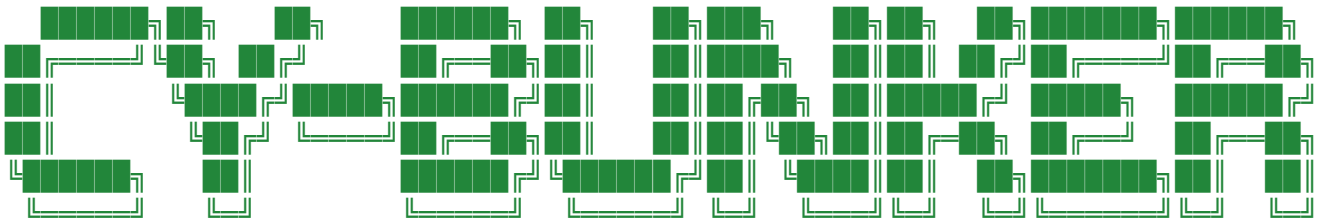
Informatique
PréIng 2
MI01
Trinome_MI-1_D

RAPPORT DE PROJET INFORMATIQUE



Sofyane ABAJOU
Hicham EL JARJINI
Nassim JAMHOUR

Sommaire

I/ Description du sujet

II/ Gestion de l’équipe

III/ Problèmes rencontrés et les solutions

IV/ Résultats obtenus

I/ Description du sujet

Cy Bunker est un logiciel conçu pour gérer efficacement l'électricité dans un bunker en contexte post-apocalyptique, tel qu'une guerre nucléaire. Ce programme assure la collecte, le filtrage, et le traitement des données du système de distribution électrique afin de garantir une gestion optimisée et fiable des ressources énergétiques.

Il se compose de deux modules principaux :

- **La collecte et le filtrage des données** : effectué par un script Shell pour éliminer les anomalies et organiser les informations.
- **Traitement et calculs** : réalisé via un programme en langage C pour analyser les données et ajuster la distribution énergétique.

Instructions d'utilisation :

- **Exécutez le projet avec la commande suivante** :

```
bash c-wire.sh <Fichier CSV> <Type de Station> <Type de Consommateur> [ID Centrale]
```

- **Options** :

<Fichier CSV> : Chemin vers le fichier CSV.
<Type de Station> : Type de station (hva, hvb, lv).
<Type de Consommateur> : Type de consommateur (comp, indiv, all).
[ID Centrale] : (Optionnel) ID de la centrale électrique.
-h : Pour afficher le manuel d'aide, utilisez l'option

Pour toute informations supplémentaires consulter le README.MD (en anglais)

II/ Gestion de l’équipe

Notre équipe, composée de Sofyane, Nassim et Hicham, a travaillé de manière collaborative pour concevoir et développer ce programme. Chacun avait un rôle bien défini : Nassim a pris en charge le développement de la partie en langage C, Sofyane s'est concentré sur la création et l'intégration du script Shell, tandis que Hicham s'est occupé de l'aspect esthétique et de l'interface visuelle.

Pour garantir une bonne organisation et une communication fluide, nous avons utilisé Discord comme principal outil de coordination, ce qui nous a permis d’échanger et de partager nos idées. Nous avons également centralisé notre code et nos contributions sur GitHub, assurant ainsi un suivi clair des modifications et une collaboration efficace. Enfin, pour le développement, nous nous sommes appuyés sur l’éditeur VS Code Online et l’extension Live Share, qui nous a permis de travailler simultanément, même à distance.

III/ Problèmes rencontrés et les solutions

Nous avions utilisé des fonctions en C à partir de l'AVL pour créer le fichier lv_allminmax.csv ce qui nous obligeait à trouver la valeur minimale de dans l'AVL en fonction de la consommation puis de supprimer la valeur de l'AVL, de l'insérer dans le fichier lv_allminmax.csv et de faire de même pour le maximum ce qui n'était pas efficace et qui nous avait demandé de rajouter cinq fonctions à notre programme : maxConsAVL, minConsAVL, removeMin, removeAVL et write_csv_minmax.

De plus, le programme ne marchait pas pour des AVL de moins de 20 valeurs car il y avait souvent une intersection entre les 10 minimums et 10 maximums ce qui entraînait la création de doublons dans le fichier final. Pour remédier à cela nous avons décidé d'enlever cette partie de programme C, de ne pas manipuler l'AVL mais le fichier lv_all.csv dans la partie Shell afin de traiter correctement et beaucoup plus facilement grâce aux commandes dont on dispose (head, tail, awk). Il suffisait alors de trier en fonction de la consommation puis de prendre les 10 premières et 10 dernières lignes grâce à tail et head puis de trier en fonction de la différence entre la consommation et la production en utilisant une colonne temporaire tout en veillant à bien traiter les fichiers sans les headers.

```
{
  head -n 2 "tmp/lv_allminmax.csv"
  awk -F":" '{diff = $2 - $3; print $1 ":" $2 ":" $3 ":" diff}'
"tmp/minmax.csv.tmp" | sort -t":" -k4 -n | cut -d":" -f1-3
} > "tmp/lv_allminmax.csv.tmp"
```

c-wire.sh:201-204

L'un des autres points qui a causé des difficultés significatives, ou du moins entraîné une perte de temps considérable, a été le test de notre programme avec un fichier CSV volumineux sur les ordinateurs de la faculté. Ces machines se sont révélées peu performantes en termes de temps de traitement, ce qui a ralenti nos essais et analyses.
Un autre problème auquel nous avons été confrontés concerne l'ajout de fonctionnalités bonus, comme l'intégration d'une musique de fond ou l'affichage de la date du jour écrite en russe.
`TZ='Europe/Moscow' date --date="$(date +'%m-%d %H:%M:%S' | sed 's/^/1971-/') +2 hours"` On a finalement utilisé la date avec la langue de l'utilisateur.
Les autres ajouts, bien que non essentiels, auraient apporté une touche d'originalité et de finition à notre projet.
Cependant, leur mise en œuvre s'est avérée compliquée en raison de l'impossibilité d'obtenir les droits nécessaires sur linux pour utiliser certaines commandes (sudo impossible). Cette restriction s'applique aux ordinateurs de la faculté mais aussi à nos environnements virtuels personnels. Cette contrainte technique nous a empêchés de finaliser ces ajouts, malgré notre volonté de proposer un rendu plus abouti. Pour contrebalancer cela nous avons essayé d'opter pour une interface originale dans le terminal que l'on vous laisse découvrir.

IV/ Résultats obtenus

Tous les fichiers CSV sont disponibles dans le dossier “tests/” ils ont été générés à partir du document input/c-wire_v00.dat en testant avec les paramètres obligatoires et avec “hvb comp 1”, les lignes sont triées par capacité croissante de la plus petite à la plus grande. Le fichier temp.csv correspond au fichier que le programme Shell génère pour le programme C avec les paramètres “lv all”.

Les documents ont le bon header en fonction de leurs paramètres respectifs. Le fichier lv_allminmax.csv est bien trié par différence entre la consommation et la capacité et cela a été fait avec une colonne temporaire qui est supprimée après le tri.