

Compte rendu de projet CY-Truck



VOISIN Anthony

HARROIS Killian

LABBE-NOBILI Thomas

Informatique 3

ANSERMIN Eva

Organisation initiale

Après avoir formé notre groupe, nous avons décidé de nous répartir les tâches en fonction des points forts/préférences de chacun. L'organisation initiale était la suivante :

- Anthony : écriture de la partie shell et du gnuplot
- Killian : écriture de la partie C du traitement T et makefile
- Thomas : écriture de la partie C du traitement S et fonctions AVL

Vous verrez plus tard que cette dernière a été légèrement modifiée durant l'évolution du projet.

Pour l'avancement du projet, grâce aux commits github nous pouvons avoir la date exacte de certaines avancées, celles qui n'ont pas de date exacte ont été réalisées environ dans la semaine dans lesquelles elles sont situées.

Avancement du projet

1^{ère} semaine (27/11 au 01/12) Début du projet

Création du github et partage du travail.

2^{ème} semaine (02/12 au 08/12)

Les parties shell des traitements D1, D2 et L sont terminées.

3^{ème} semaine (09/12 au 15/12)

Pour les traitements S et T, premiers codes d'une fonction qui récupère les données d'un fichier txt car nous avons oublié comment ça marche depuis l'année dernière.

Réflexion pour comment réaliser l'ancien traitement T (il fallait dire pour chaque ville combien de conducteurs passaient en plus). Nous avons pensé à une technique qui utiliserait 3 AVL mais ce n'était pas du tout optimisé selon nous. Création des graphiques des traitements D1 et D2 avec gnuplot.

4^{ème} semaine (16/12 au 22/12)

20/12 : L'énoncé du traitement T est modifié et nous sommes débloqués. Ajout de toutes les fonctions AVL et tests pour voir si l'AVL marche bien.

21/12 : Nous avons fait la partie shell pour le traitement T. Notre première méthode d'approche pour le traitement T était de générer 2 fichiers temporaires: un qui contenait combien de fois chaque ville est en Town A seulement si $\text{step id} = 1$ (c'est à dire elle est ville de départ), et l'autre qui contenait combien de fois chaque ville est en Town B peu importe son step id . Le problème avec cette approche est que quand on traitait le 2^{ème} txt, la ville pouvait déjà être dans l'AVL si elle était dans le 1^{er} txt et il fallait vérifier dans tout l'AVL si la ville était déjà enregistrée dedans ou pas, ce qui n'était pas optimisé (vérifier tout l'AVL pour chaque ville...).

5^{ème} semaine (23/12 au 29/12) Vacances de Noël

Pendant les vacances nous avons un peu moins travaillé.

24/12 : Notre première méthode pour le traitement T marche (sur des fichiers tests de quelques milliers de lignes). Il faut cependant ajouter des sécurités, et libérer l'espace mémoire (il n'y a aucun free dans le code pour l'instant).

6^{ème} semaine (30/12 au 05/01) Vacances de Noël

30/12 : Nous avons appris que nous pouvions utiliser AWK pour le traitement T et donc désormais notre shell génère un seul fichier txt temporaire beaucoup plus efficace que le précédent, celui ci nous affiche pour chaque ligne une ville, le nombre de fois qu'elle est visitée et le nombre de fois qu'elle est ville de départ. Cela rend la partie C beaucoup plus simple car nous n'avons plus qu'à envoyer chaque ville dans l'AVL qui sera trié en fonction du nombre de visite. Le traitement T marche et pour l'instant print les 10 villes les plus visitées mais par ordre de nombre de visite, pas ordre alphabétique des villes.

Le graphique du traitement L est fait.

7^{ème} semaine (06/01 au 12/01)

07/01 : Le shell génère un nouveau fichier txt qui ne prend pas en compte les doublons de ville suite à la remarque d'un élève sur teams, rien à changer pour la partie C.

Les graphiques S et T ont été réalisés même si nous n'avions pas encore fini la partie C, à l'aide de fichiers txt qui contenaient les bons résultats.

8^{ème} semaine (13/01 au 19/01)

13/01 : Ajout du code du traitement S qui marche, et qui comme le T print les résultats. En Shell nous commençons à mettre en commun tous les traitements dans un seul fichier shell (depuis le début chaque traitement était réalisé dans un fichier à part).

16/01 : Changement pour la gestion des résultats, les S et T ne printent plus les résultats mais maintenant écrasent leurs txt temporaires respectifs envoyé par le shell, et ensuite placent les résultats dans ces mêmes fichiers. Le shell récupère ensuite les données dans ces fichiers pour faire les graphiques.

Du 16 au 24 janvier nous n'avons presque rien fait car nous devons réviser pour nos partiels (du 18 au 24 janvier).

9^{ème} semaine (20/01 au 26/01)

24/01 : Reflexion sur quelle serait la manière la plus optimisée pour le T de renvoyer les villes par ordre alphabétique (pour l'instant les résultats étaient triés par nombre de visites). Nous hésitons entre faire un simple ABR car il n'y a que 10 villes à trier, faire un tableau qui récupère les 10 premières valeurs et les trie avec un algorithme de tri classique type tri à bulle, ou utiliser un 2ème AVL. Création d'une sécurité qui vérifie les arguments en shell.

Au final, nous décidons tout simplement de mettre en place un deuxième AVL trié par ordre alphabétique et qui récupérera les 10 premières valeurs du 1^{er} AVL. Cela prend beaucoup de place dans le code (il faut redéfinir toutes les fonctions AVL) mais nous sommes sûrs que c'est optimisé.

25/01 : Implémentation du fichier header et ajout de fonctions.h et d'un main.c. Ce jour nous avons perdu beaucoup de temps à cause du problème réseau à l'école, nous avons perdu tout le travail de la journée et avons donc du le refaire le lendemain. Creation du Makefile et implementation du make dans le shell.

10^{ème} semaine (27/01 au 02/02)

27/01 : Test allocation dynamique dans les codes C dans les fonctions 'formeAVL' pour allouer dynamiquement de l'espace pour les chaînes de caractère 'ligne' et 'villetemp' cependant après avoir réalisé plusieurs tests nous avons remarqué qu'avec les malloc le temps d'exécution était plus long. Donc nous avons laissé une allocation statique. Creation du repertoire demo et optimisation de celui ci et creation de sécurités dur les dossiers image, temp et prog.

28/01 : Ajout des sécurités sur les T et S, c'est à dire vérifier si les données traitées sont correctes (du genre nbVisite est bien un nombre et pas le nom d'une ville, etapeMin est bien supérieur à 0 etc ...)

31/01 : Ajout des commentaires sur tout les fichier.

02/02 : Réalisation du compte rendu du projet.

Repartition des tâches

Au final, Anthony s'est occupé de toute la partie shell du projet comme prévu. Killian a réussi à faire le traitement T assez rapidement grâce aux fonctions AVL fournies par Thomas. Cependant Thomas avait un peu de mal à faire le traitement S. Comme Killian avait fini le traitement T il a pris le relais et a fait le traitement S. Thomas a donc fait le makefile pour les fichiers c et il a aussi fait le read me. Anthony s'est aussi occupé de générer les graphiques avec gnuplot.

Au final voici comment le travail a été partagé :

- Anthony : partie shell / graphiques gnuplot
- Killian : partie C sauf makefile
- Thomas : fonctions AVL / makefile / readme

Le compte rendu a été réalisé par toute l'équipe.

Informations supplémentaire

Notre projet vérifie l'ensemble du cahier des charges de la dernière version de l'énoncé.

Aucun ajout bonus n'a été fait.

Seuls les traitements T et S utilisent du c, la première fois que vous lancerez l'un deux, l'exécutable devra se créer et le message "Compilation terminée" s'affichera.

Si vous lancez plusieurs fois le même traitement sur une ligne, un message indiquant qu'il a déjà été réalisé s'affichera autant de fois que vous avez écrit en trop le traitement.

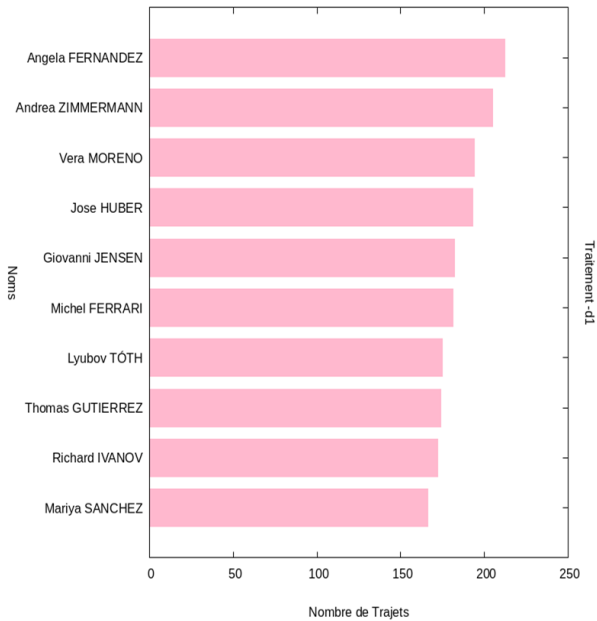
Si les 5 différents traitements ont été réalisés en une seule ligne, le programme s'arrête et ne lit pas les autres traitements demandés (ils ont forcément déjà été fait).

Seules les 20 premières demandes seront traitées.

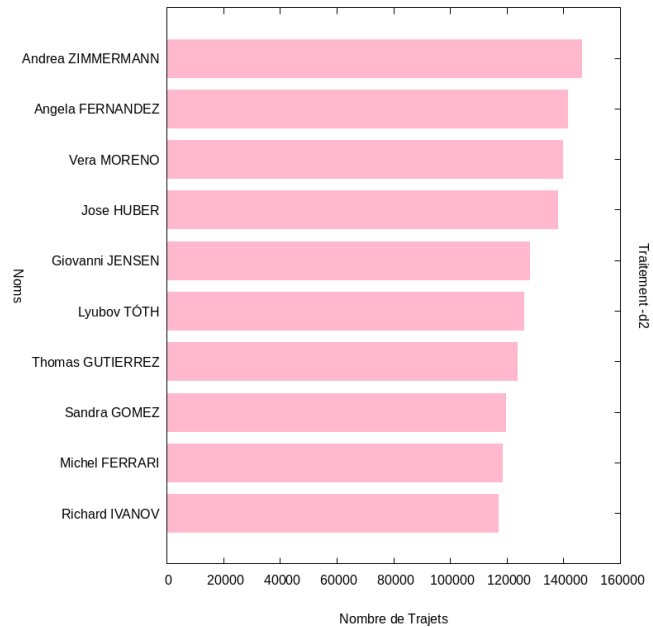
Si vous demandez l’option h, toutes les autres options seront ignorées et cela affichera les aides.

Exemple des résultats

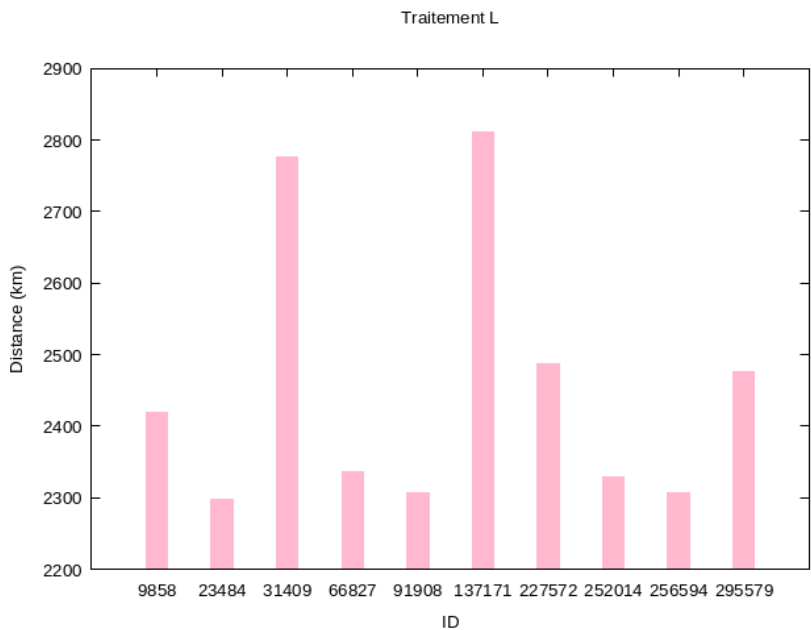
Les temps indiqués sont ceux obtenus sur les PC de l’école.



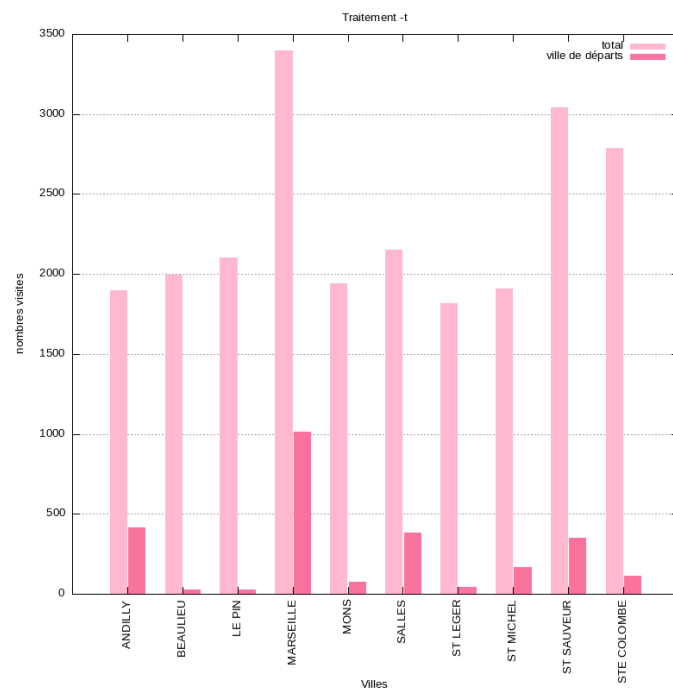
Graphique D1 (~ 2.1 s)



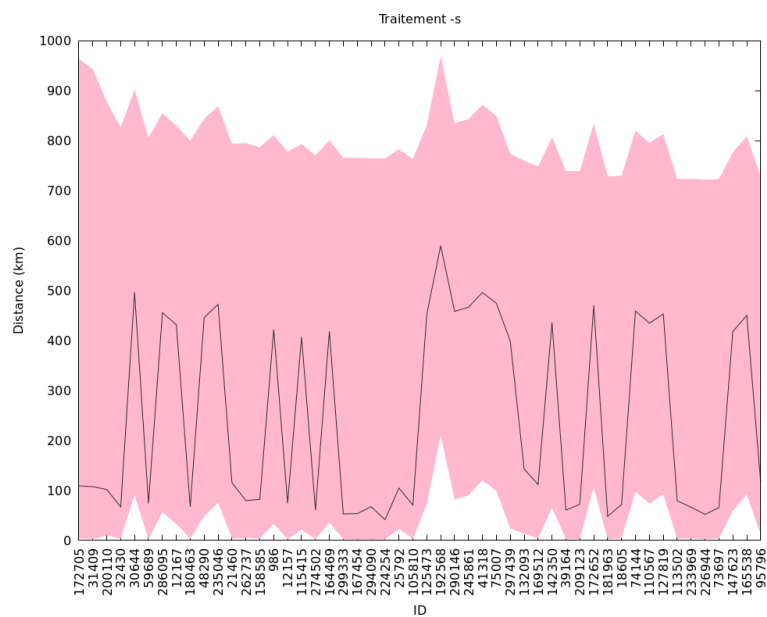
Graphique D2 (~ 2.7 s)



Graphique L (~ 5.3 s)



Graphique T (≈ 18 s)



Graphique S (≈ 17 s)

Codes erreurs pour les fichiers c

Main.c

1 : le traitement demandé n'est pas valide

traitementT.c

2 : dans le txt il manque au moins la donnée de la ville

3 : dans le txt il manque au moins la donnée nbVisite

4 : dans cette ligne, nbVisite n'est pas un nombre ou est égal à 0

5 : dans cette ligne nbDepart n'est pas un nombre ou est égal à 0

6 : erreur d'allocation mémoire dans concatenerDonneesT

7 : erreur d'allocation mémoire pour AVL_T

8 : erreur d'allocation mémoire pour AVL_T2

19 : il y a moins de 10 villes dans l'AVL

traitementS.c

9 : dans cette ligne il manque au moins la donnée id

10 : dans cette , id n'est pas un nombre ou id est ≤ 0

11 : dans cette ligne il manque au moins la moyenne

12 : dans cette ligne la moyenne n'est pas un nombre ou elle est ≤ 0

13 : dans cette ligne il manque au moins le minimum

14 : dans cette ligne le minimum n'est pas un nombre ou elle est ≤ 0

15 : dans cette ligne il manque au moins le maximum

16 : dans cette ligne le maximum n'est pas un nombre ou elle est ≤ 0

17 : erreur d'allocation mémoire dans concatenerDonneesS

18 : erreur d'allocation mémoire pour AVL_S

20 : il y a moins de 50 villes dans l'AVL