v1.0.2 Projet CY-Trucks

FILIERE préING2 • 2023-2024
AUTEURS E.ANSERMIN – R.GRIGNON
E-MAILS eva.ansermin@cyu.fr – romuald.grignon@cyu.fr

DESCRIPTION GENERALE

- Vous êtes le gérant d'une société nationale de transport routier. Vous disposez d'un outil gérant l'ensemble de votre logistique, mais les données recueillies sont bien trop volumineuses et mélangées pour être traitées par un humain. Vous décidez donc de développer un programme qui vous permettra d'afficher des données consolidées sur votre activité.
- En d'autres termes, vous devez réaliser un programme qui va analyser le contenu du fichier de données et générer des graphiques résumant le contenu de ce fichier. Lisez attentivement ce cahier des charges vant de vous lancer dans la réalisation du projet.
- Techniquement vous devez réaliser un script Shell qui parfois, pour des raisons évidentes de performances, devra appeler un autre programme codé en langage C pour traiter les données d'après le cahier des charges. Puis le script Shell devra créer des graphiques à partir des résultats obtenus du programme C.
- Le fichier de données d'entrée devra être copié dans un dossier '*data*'
- Le programme C et tout ce qui s'y rapporte (makefile, exécutable, ...) devront être situés dans un dossier '*progc*'
- Les graphiques seront stockés dans des images sur le disque dur dans un dossier '*images*'
- Les fichiers intermédiaires nécessaires à votre application seront placés dans un dossier '*temp*'.
- Les résultats d'exécutions précédentes seront dans le dossier '*demo*'.
- Le script Shell sera quand à lui placé à la racine de votre projet.

SCRIPT SHELL

- Votre script Shell va prendre en paramètres le chemin du fichier CSV d'entrée contenant les données des trajets routiers effectués. Il prendra également d'autres paramètres qui seront les choix des traitements à faire (voir plus bas).
- Le chemin du fichier de données sera toujours le premier argument et est obligatoire. Ensuite parmi les autres arguments, il en faut au moins un qui soit fourni, mais vous pouvez en fournir plusieurs si vous souhaitez effectuer plusieurs traitements d'affilée.
- Si un argument vaut -h, alors tous les autres arguments sont ignorés, et votre programme devra afficher un message d'aide expliquant les options pouvant être utilisées ou non.

- Le script Shell devra vérifier la présence de l'exécutable C sur le disque dur, si ce dernier n'est pas présent, le script devra lancer la compilation et vérifier que cette dernière s'est bien déroulée. Si ce n'est pas le cas, un message d'erreur doit être affiché. Une fois la compilation effectuée il pourra effectuer le traitement demandé en argument.
- Le script Shell vérifiera la présence du dossier temp et images : si ces derniers n'existent pas il devra les créer. Si le dossier temp existe déjà, il devra le vider avant l'exécution des traitements.
- La durée de chaque traitement à effectuer devra être affichée en secondes à la fin. Peu importe si le script s'est déroulé correctement ou a rencontré une erreur, les durées devront être affichées systématiquement. Les durées ne comprendront pas la phase de compilation du programme C ou la création des dossiers effectuée au départ. Les temps affichés doivent être des temps utiles de traitement des données.
- Une fois les traitements de données terminés, le script Shell devra créer un graphique (une image) qui contiendra les données de sortie du traitement. Pour cela vous utiliserez le programme ***GnuPlot***.
- Les différentes options à passer en argument à votre script Shell dépendent des traitements à effectuer dont la liste se trouve ci-dessous :

- **TRAITEMENT [D1] : conducteurs avec le plus de trajets : option -d1**
  Vous devez récupérer la liste des conducteurs avec le nombre de trajets différents qu'ils ont effectués. Depuis cette liste, triée par ordre décroissant de nombre de trajets, vous ne devez garder que les 10 premiers conducteurs.
  Votre programme devra créer un graphique de type ***histogramme horizontal*** avec en ordonnée les noms complets des conducteurs, et en abscisse le nombre de trajets effectués. Le conducteur avec le plus de trajet sera placé en haut du graphique.
  Ce traitement pourra être réalisé uniquement à l'aide d'un script Shell et de commandes Unix du moment que le temps d'exécution ne dépasse pas l'ordre de grandeur de 8 secondes environ.

- **TRAITEMENT [D2] : conducteurs et la plus grande distance: option -d2**
  Pour ce traitement, vous devez récupérer la distance totale parcourue par chaque conducteur, c'est à dire faire la somme des distances de toutes les étapes qui leurs sont associées. Vous ne garderez que les 10 plus grandes distances classées par ordre décroissant.
  Votre programme devra créer un graphique de type ***histogramme horizontal*** avec en ordonnée les noms complets des conducteurs, et en abscisse la distance totale parcourue. Le conducteur avec le plus de distance sera placé en haut du graphique.
  Ce traitement pourra être réalisé uniquement à l'aide d'un script Shell et de commandes Unix du moment que le temps d'exécution ne dépasse pas l'ordre de grandeur de 7 secondes environ.

- **TRAITEMENT [L] : les 10 trajets les plus longs : option -l**
  Pour ce traitement, vous devez récupérer la distance totale de chaque trajet, c'est à dire la somme de chaque étape et ce, pour chaque trajet. Vous ne garderez que les 10 distances les plus grandes, et ordonnerez ce résultat par numéro d'identifiant de trajet croissant.

Votre programme devra créer un graphique de type ***histogramme vertical*** avec en abscisse l'identifiant du trajet, et en ordonnée la distance en km.
Ce traitement pourra être réalisé uniquement à l'aide d'un script Shell et de commandes Unix du moment que le temps d'exécution ne dépasse pas l'ordre de grandeur de 8 secondes environ.

- **TRAITEMENT [T] : les 10 villes les plus traversées : option -t**

Pour ce traitement, vous compterez le nombre de trajets qui parcourent chaque ville, ainsi que le nombre de fois où ces villes ont été des villes de départ de trajets.
Une ville n'est traversée qu'une seule fois par trajet, mais forcément elle peut apparaître deux fois dans les données, une fois comme ville de départ d'une étape, et une autre fois comme ville d'arrivée de l'étape précédente. Il vous faudra gérer ce cas un peu particulier.
A la fin du traitement, vous ne conserverez que les 10 premières villes avec le plus de trajets, par ordre alphabétique.
Votre programme devra créer un graphique de type ***histogramme regroupé***. On retrouvera en abscisse les noms des villes, par ordre alphabétique, et en ordonnée le nombre de trajets.
Pour chaque ville, il y aura 2 barres verticales, celle de gauche indiquera le nombre de trajets total qui traversent cette ville, et celle de droite le nombre de fois où cette ville est la ville de départ d'un trajet.

Attention :

Si vous tentez de réaliser ce traitement uniquement à l'aide d'un script Shell et de commandes Unix, vous ne recevrez aucun point pour le traitement lui-même (mais vous pourrez avoir les points sur la partie graphique si l'image est générée correctement malgré tout). Il est donc obligatoire de passer par un programme C pour effectuer le tri des données : l'utilisation d'une structure de type AVL est obligatoire. Aucune commande Unix de type '*sort*' ne doit être appelée ici.

- **Traitement [S] : statistiques sur les étapes : option -s**

Pour ce traitement, on cherche à récupérer les distances minimales, maximales et moyennes des étapes, et ce, pour chaque trajet.
Votre programme devra créer un graphique de type courbes min-max-moyenne, c'est à dire 2 courbes indiquant pour chaque abscisse les valeurs min et max des distances, et une troisième entre les deux pour indiquer la moyenne.
En abscisse il y aura les identifiants des trajets, et en ordonnée les distances en km (mini, moyenne, maxi).
Ces données seront triées par «***distance_maxi – distance_mini***» par ordre décroissant, et on ne conservera que les 50 premières valeurs.

Attention :

Si vous tentez de réaliser ce traitement uniquement à l'aide d'un script Shell et de commandes Unix, vous ne recevrez aucun point pour le traitement lui-même. Il est donc obligatoire de passer par un programme C pour effectuer le tri des données : l'utilisation d'une structure de type AVL est obligatoire. Aucune commande Unix de type '*sort*' ne doit être appelée ici.

PROGRAMME C

- La compilation du programme C devra se faire avec l'utilitaire '*make*' en utilisant un '*Makefile*'.
- Le programme C à réaliser ne possède pas de critères bien précis au niveau fonctionnel puisqu'il s'agira pour vous justement de les définir. Ce programme C devra effectuer des opérations de filtrage, de tri et/ou de calcul en fonction de ce dont le script Shell a besoin.
- Les seuls arguments qui pourraient évenutellement être demandés sont les noms des fichiers d'entrée et de sortie. Le format du fichier d'entrée est déjà fourni, mais pour ce qui est du fichier de sortie le format est laissé libre.
  Il est possible de ne passer aucun argument et laisser votre programme C récupérer les données depuis l'entrée standard et fournir la sortie attendue directement sur la sortie standard : cela vous évite la gestion des fichiers (et simplifie aussi la gestion des arguments) mais cela change la manière de coder le programme.
- Le programme C devra retourner un code d'erreur avec une valeur strictement positive si un problème est rencontré, et 0 sinon. Ce programme ne devra jamais s'arrêter de manière inattendue : c'est à vous de bien vérifier que toutes les données que vous traitez sont correctes. Si vous détectez un problème, vous devez stopper le programme et renvoyer un code d'erreur.
- Le code du programme C devra être donc robuste, et surtout devra libérer les allocations mémoires avant de se terminer dans le cas nominal. Il n'est pas demandé de libérer toute la mémoire dans le cas où le programme rencontre une erreur.
- Il vous est également demandé de limiter au mieux la taille mémoire utilisée. Pour se faire, vous devrez définir des variables dans vos structures ayant le moins d'empreinte mémoire possible, tout en garantissant le fonctionnel demandé bien sûr.

BONUS

- Toute amélioration du projet conduira à des points bonus si elles sont fonctionnelles. Vous êtes libres d'améliorer le fonctionnement de ce projet à partir du moment où l'ensemble du cahier des charges est déjà rempli.

INFORMATIONS SUPPL.

- **Le fichier de données CSV**

Le fichier « data.csv » fourni contient l'ensemble des données de trajets routiers des camions.
Pour chaque ligne du fichier, il y a plusieurs colonnes avec l'identifiant unique du trajet routier, l'identifiant de l'étape pour ce trajet spécifiquement, les noms des villes de départ et d'arrivée pour cette étape, la distance parcourue en km, et enfin le nom complet du conducteur sur ce trajet.

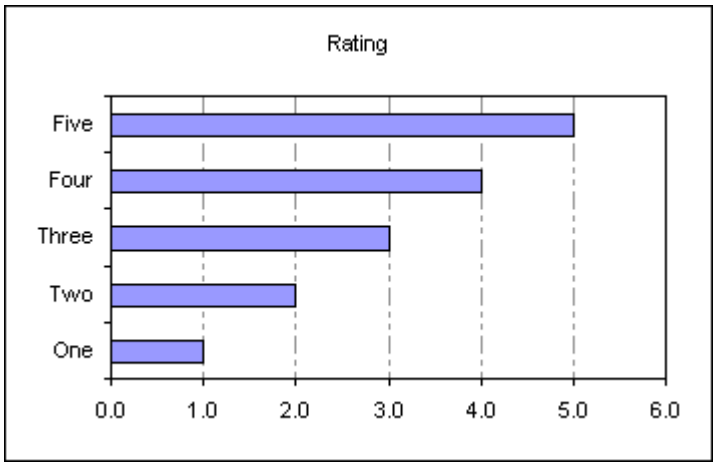
C'est un fichier volumineux (+300Mo) avec plus de 6 millions de lignes avec 1 ligne = 1 étape d'1 trajet, et presque 300k trajets en tout.
Pour chaque trajet, il peut y avoir jusqu'à +50 étapes avec une moyenne de 20 étapes par trajet, et un total de +6k conducteurs qui se sont répartis l'ensemble de ces trajets.

Les étapes relient des villes parmi une liste de +30k communes (l'ensemble des communes de France, Métropole + Corse).
Il va donc être très difficile voire impossible d'avoir une vision d'ensemble précise de ce fichier avec des outils de bureautique classique. Il va vous falloir passer par un programme informatique pour extraire les données dont vous aurez besoin.

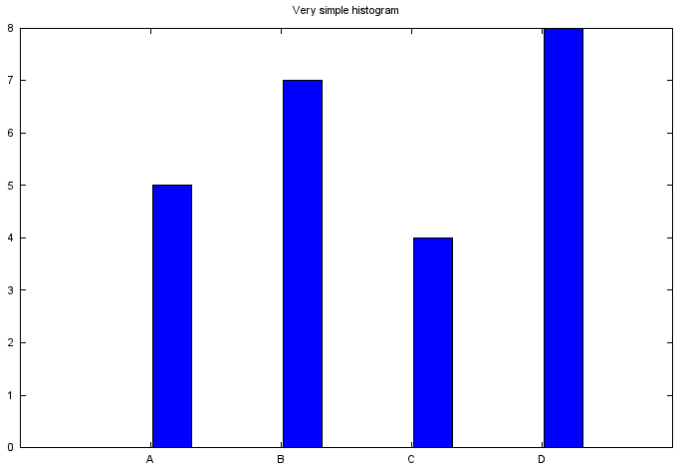
- **Les types de graphiques à fournir**

Le script Shell devra fournir plusieurs types de graphiques.
Chaque graphique sera stockée dans une image au format JPG ou PNG.
Chaque graphique devra contenir une légende pour chaque axe, des graduations claires, ainsi qu'un titre.
Faites en sorte que les graphiques soient centrés sur les données utiles (exemple : n'afficher que les abscisses utiles pour 'zoomer' sur les données intéressantes dans le cas où la plage d'abscisse utile serait entre 100k et 110k, ne pas afficher le graphique en partant en dessous de 100k et ne pas afficher au dessus de 110k)
Bien entendu vous êtes libres du choix des couleurs, polices de caractères, …
Faites simplement en sorte que les graphique soit lisibles et contiennent l'ensemble des données demandées (valeurs, légendes, titres, …)

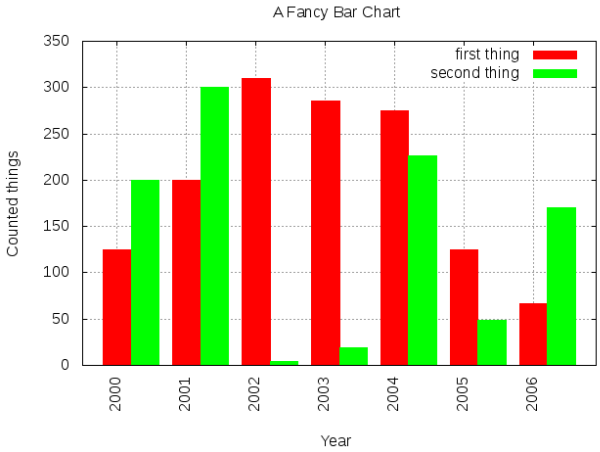
Le graphique en barre type histogramme horizontal devra etre similaire à l'image suivante (**option -d1**):



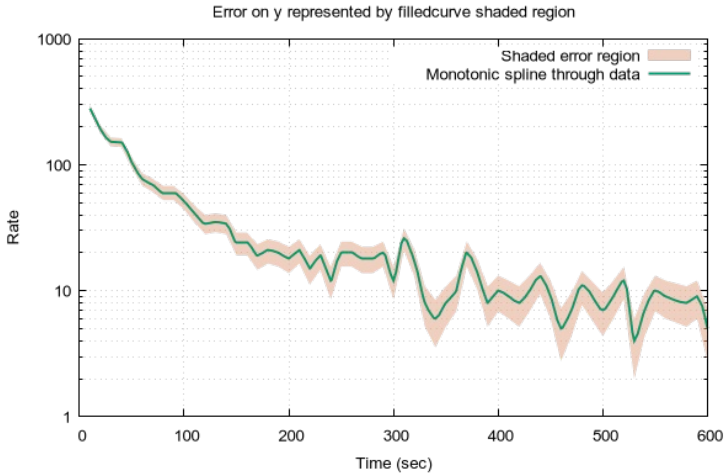
Le graphique en barre type histogramme devra etre similaire à l'image suivante (**option -l**):



Le graphique en barres type histogramme groupé devra être simillaire à l'image suivante (**option -t**) :



Le graphique de courbes de type errorbars devra être similaire à l'image suivante (**option -s**):



CRITERES DE NOTATION

- Le rendu du travail sera un **lien github** menant au projet. Inutile d'envoyer les fichiers par email : le seul livrable attendu et qui sera évalué sera le lien du dépôt git dans lequel doivent se trouver tous les fichiers de votre projet. Avant la date de rendu vous pouvez configurer ce dépôt en « privé » pour ne pas laisser d'autres personnes vous plagier. A la date de rendu, ce dépôt devra être visible publiquement pour que vos chargés de projet puissent y accéder librement. Tout retard d'accès à ce dépôt public aura un impact négatif sur votre note finale.
- Le dépôt de code contiendra en plus des fichiers de code, un fichier ReadMe contenant les instructions pour compiler ET pour utiliser votre application. Il contiendra aussi un document au format PDF présentant la répartition des tâches au sein du groupe, le planning de réalisation, et les limitations fonctionnelles de votre application (la liste de ce qui n'est pas implémenté, et/ou de ce qui est implémenté mais qui ne fonctionne pas correctement/totalement).
- Il vous est demandé d'effectuer une livraison sur le dépôt 2 fois par semaine. Si vous n'avez aucune évolution au niveau de votre code pendant toute une semaine, vous ferez tout de même une mise a jour de votre dépôt par exemple en mettant à jour un fichier texte dans lequel vous indiquerez qu'il n'y a pas eu de modifications.

- Votre rendu contiendra des exemples d'exécution de votre application. Vous mettrez dans le dossier '*demo*', les images, fichiers intermédiaires et finaux, images, et vous présenterez ces résultats dans le document PDF cité précédemment. Ces éléments doivent donc être reproductibles en utilisant votre programme.
- Le rendu est un travail de groupe : si des similitudes entre groupes sont trouvées, et/ou si des exemples disponibles sur Internet sont découverts sans être sourcés, une procédure de fraude à un examen pourra être envisagée. Le but pédagogique de ce projet est que vous réalisiez par vous-même ce programme.
- Le code sera séparé en **modules** (fichiers .c et .h dans des sous-dossiers).
- Un fichier **Makefile** sera présent et il permettra de compiler l'exécutable. La première cible permettra de compiler le projet. Ce fichier inclura entre autres une cible '**clean**' qui permet d'effacer les fichiers générés.
- Votre code sera **commenté** (modules, fonctions, structures, constantes, …) et correctement **indenté**.
- Les **symboles** du code (variables, fonctions, types, fichiers, …) seront dans la **même langue** que les **commentaires** (soit tout en anglais, soit tout en français, mais pas de mélange entre les langues utilisées).

- Le programme C doit respecter toutes les consignes décrites plus haut.
- Le script Shell doit respecter toutes les consignes décrites plus haut.
- Le programme C et le script Shell ne doivent en aucun cas générer d'erreur inattendue. Il ne doit y avoir aucune **erreur de segmentation**, **erreur de syntaxe**, ou de nom de **commande inconnue**, etc…
- Si une erreur est détectée par votre programme/script, un message d'erreur doit s'afficher pour indiquer la cause à l'utilisateur, et un code retour avec une valeur strictement positive doit être retournée.

- Pour certaines fonctionnalités, l'utilisation du script Shell seul sera possible. Cela veut dire que vous pouvez effectuer cette partie du projet en script Shell sans appeler de programme C, mais en fonction des opérations à réaliser, votre script prendra trop de temps à s'exécuter. Si cela arrive, vous ne pourrez pas obtenir tous les points car votre programme donnera l'impression d'être bloqué dans une boucle infinie.
  Dans le cas ou votre programme fourni un résultat au bout d'un temps relativement long mais acceptable, vu que le fonctionnel sera présent et testable, vous aurez tout de même une partie des points alloués. Il est donc de votre responsabilité de réaliser un programme à la fois fonctionnel et performant en terme de temps d'exécution.
  Ce dernier point passera forcément par l'appel d'un programme C pour effectuer les traitements plus rapidement que le script Shell ne pourrait le faire. Il est plus que conseillé de réaliser des traitements à l'aide d'un programme C pour être certain de marquer l'ensemble des points (le code C avec les arbres binaires étant un très gros aspect du contenu pédagogique de ce semestre).

- Les structures allouées temporairement dans votre programme doivent être désallouées explicitement avant la fin. La quantité de mémoire non libérée à la fin de l'exécution, sera évaluée.

- De plus, la quantité de mémoire vive consommée par votre programme, sera également notée : pensez donc à limiter votre empreinte mémoire. Attention tout de même à faire en sorte que votre programme fonctionne : l'optimisation mémoire reste un plus. La première étape étant de faire quelque chose de fonctionnel.

- Vous disposez d'un fichier de données CSV qui est unique. Il est donc possible pour un groupe d'étudiants de « coder en dur » les résultats attendus. Pour éviter ce cas de triche, il est possible que l'évaluation de votre programme se fasse avec un fichier de données CSV différent du votre (mais similaire en terme de structure, de taille, ...). Pensez donc à faire un programme véritablement générique pour éviter une mauvaise surprise lors de l'évaluation.

RESSOURCES UTILES

GitHub
- site Web : https://github.com/

Format CSV
- site Web : https://fr.wikipedia.org/wiki/Comma-separated_values

GnuPlot
- site Web : http://gnuplot.info/