3/2/2023

A blue and white logo

Description automatically generated with low confidence

Emma Féneau & Théo Rousselet – Groupe MI 2 – Préing2

Rapport de projet

**Projet Météo**

Organisation :

Concernant notre organisation, nous avons élaboré la structure générale du programme ensemble. Puis, nous avons divisé le programme en plusieurs parties à coder que nous nous sommes distribuées.

Au début du projet, nous avons programmé quelques réunions sur Teams pour se rendre compte mutuellement de nos avancées. Nous y partagions les programmes effectués entre deux réunions. Nous pouvions ainsi éclaircir certains points de nos programmes respectifs, expliquer de nouvelles fonctions ou revoir ensemble certaines parties du code qui pouvaient être optimisées ou changées.

Tout au long du projet, les idées ont été mises sur un Powerpoint. Ce Powerpoint nous a aidé à mieux comprendre le sujet, faire des schémas d’arbres, de listes chaînées ou des graphiques. Ce travail nous a permis de mieux comprendre le sujet, à détailler certaines parties plus difficiles que d’autres et à préparer en amont le rapport.

Les structures des arbres binaires, des AVL et des listes chaînées et des fonctions de bases de ces dernières a été fait par Théo Rousselet et le codage en Shell, la partie Gnuplot et les fonctions du code C ont été fait par Emma Féneau.

Cinématique de l’algorithme :

En ce qui concerne nos choix technologiques, nous avons utilisé la bibliothèque standard soit stdlib.h, string.h, time.h, stdio.h et map.h. Nos environnements de développement sont visualStudioCode et Xcode.

La structure de notre programme est composée de 3 fichiers pour le Shell et de 10 fichiers pour le C :

1. Pour le shell :

* meteo.sh : programme principal.
* config.meteo.sh : fichier contenant toutes les constantes pour le fichier functions.sh.
* functions.sh : fichier contenant toutes les définitions des fonctions.

1. Pour le code en C :

* main.c : programme principal
* functions\_avl.c : fichier contenant les définitions des fonctions des AVL.
* functions\_avl.h : fichier contenant les prototypes des fonctions des AVL.
* functions\_common.c : fichier contenant les définitions des fonctions communes aux trois méthodes de tris ;
* functions\_common.h : fichier contenant les prototypes des fonctions communes aux trois méthodes de tris.
* functions\_list.c : fichier contenant les définitions des fonctions des listes chaînées.
* functions\_list.h : fichier contenant les prototypes des fonctions des listes chaînées.
* consts.h : fichier contenant les constantes.
* functions\_abr.h : fichier contenant les prototypes des fonctions des ABR.
* functions\_abr.c : fichier contenant les définitions des fonctions des ABR.

Tous ces fichiers se trouvent dans un répertoire nommé « Code », composé de 8 fichiers :

* include : répertoire composé des fichiers .h et de config.meteo.sh.
* lib : répertoire composé des fonctions ainsi que tous les codes pour générer les graphiques sur gnuplot.
* source : répertoire composé du main.c et du makefile
* obj : répertoire composé des fichiers .o
* input : répertoire composé du fichier de bases des relevés météo.
* output : fichier temporaires initialement vide
* temp : répertoire composé de tous les fichiers intermédiaires.
* results : répertoire composé des fichiers .png généré par le programme.

Dans cet algorithme nous devons pouvoir trier de trois manières différentes, par ABR, AVL ou par liste chaînée. Ci-dessous un exemple de la structure pour chaque type de tri. Nous avons essayé de généraliser le code.

Nous décrivons ci-dessous chacun des membres de cette structure :

Structure de la liste chaînée

Text

Description automatically generated

Structure de l’ABR

Text

Description automatically generated

Structure de l’AVL

Text

Description automatically generated

*Les fonctions du C :*

Pour une question de performance, nous avons utilisé la librairie map.h qui permet de mettre en mémoire une partie du fichier. Ce qui respecte la consigne de ne pas mettre l’ensemble du fichier en mémoire.

Comme pour le code Shell, nous avons dû faire des vérifications sur l’envoi de la commande entrée par l’utilisateur dans le Shell par le C. Nous voulions éviter toutes erreurs ou segmentation fault. Normalement, le code Shell doit arrêter toutes les erreurs entrées par l’utilisateur et donc se stopper. Mais cette fonction nous sert de deuxième filet de vérification.

Text

Description automatically generated

Grâce à cette capture d’écran, nous voulions mettre en avant notre choix pour la création d’un arbre grâce aux données que nous recevons du Shell (la création des listes est inspirée du même code). Nous pouvons voir que nous avons scindé tous les modes (dans cette capture il n’y a que p1 et p2 mais il y a cela pour tous les modes) dans un « if ». Grâce à l’extraction des données par le Shell, nous avons associé un nouveau nœud au champ requis dans le triage de données. Nous avons évalué que cette méthode était relativement intéressante. De plus, grâce à cela nous avons pu génériser ce code aux AVL et aux listes.

Text

Description automatically generated

*Les fonctions du Shell :*

Pour plus de rapidité, nous avons utilisé le matching pattern.

Dans le cahier des charges, il était précisé de répertorier toutes les erreurs que l’utilisateur pourrait entrer. Grâce aux 1), 2), 3) etc, nous voyons que nous arrivons à gérer les erreurs. Les constantes sont codées dans le fichier consts.c pour avoir plus de détails.

*Text

Description automatically generated*

Pour découper le code de la manière la plus petite et facile pour nous, nous avons créés des fonctions qui vérifient si l’option rentrée est une option longue dans ce cas-là (nous avons aussi une fonction pour les options courtes). Cette fonction vérifie aussi le nombre de méthodes de tries que l’utilisateur met dans la ligne de commande. Pour rappel il n’a le droit qu’à une seule méthode de tries ou par défaut, s’il n’en met pas, il n’y en a pas.

Text

Description automatically generated

*Gnuplot :*

Nous arrivons à sortir tous les graphiques sauf celui en 3D car nous nous sommes rendu compte que pour les graphiques 3D, Gnuplot attendait un style de données bien précis qui est celui-ci-dessous :

A screenshot of a computer

Description automatically generated with medium confidence

On dira que la première colonne contient les x qui sont la longitude. Pour créer un bon graphique 3D, comme celui-ci-dessous, nous aurons besoin de plusieurs mêmes longitudes en créant une matrice. Nous sommes arrivés à créer la matrice mais dans notre exercice avec les altitudes et les humidités, nous devons en avoir 60 différentes pour un seul jeu de coordonnées. Or c’est incompatible avec ce que veut Gnuplot.

Chart

Description automatically generated

Voici ce que donne le graphique quand nous mettons les données du fichier de la database. Alors qu’on devrait avoir un graphique comme celui-ci :

Chart, surface chart

Description automatically generated

De plus, nous avons ce message d’erreur qui s’affiche :

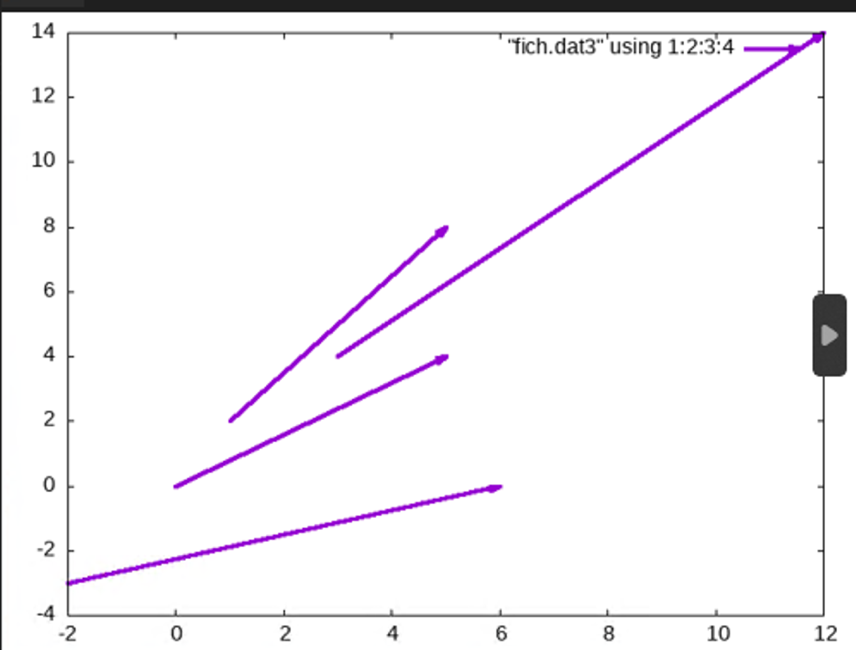
Warning: empty x range [0:0], adjusting to [-1:1]

Warning: empty y range [0:0], adjusting to [-1:1]

Warning: Single isoline (scan) is not enough for a pm3d plot.

Hint: Missing blank lines in the data file?

Les essais graphiques :



Essai graphique des vecteurs pour le vent.

Chart, line chart

Description automatically generated

Essai graphique des température et pressions en fonction des dates.

Chart, surface chart

Description automatically generated

Essai du graphique en 3D.

Les limites et les avantages de notre programme :

Nous avons essayé de répondre à toutes les attentes du cahier des charges. Nous avons essayé d’optimiser un maximum notre code en Shell et en C pour qu’il soit le plus optimal. Tout au long du projet, nous avons tenu un powerpoint de travail dans lequel nous avons mis toutes nos pistes pour mener à bien ce projet. Ce powerpoint apporte beaucoup de détails sur la construction de notre code. Vous pouvez le consulter si vous le voulez.

Nous avons laissé les fichiers temporaires extraient sont localisés dans « temp »et les fichiers d’entrées pour Gnuplot sont dans output. Ils ne sont pas effacés à la fin du processus pour que vous les vérifiez.

Nous avons réussi à implémenter complètement les ABR et les listes chaînées. Notre code arrive à sortir des graphiques avec ces deux méthodes de tris.

Quant aux limites, notre programme a été partiellement développé sur Mac sur Xcode. De plus, nous n’avons pas réussi à faire le makefile. Nous n’avons pas non réussi à finir à temps le code des AVL. Nous l’avons quand même mis pour que vous voyez notre avancée mais le code ne sort pas de graphique et n’est pas relié au Shell.

De plus, pour les graphiques, nous n’avons pas réussi à prendre en compte la TimeZone. Nous n’avons pas su s’il y a vraiment une conséquence à cela car nous prenons déjà en compte cette TimeZone dans les tris.

Au cours du développement, nous avons effectué des recherches sur des sites tels que : StackOverflow.com ou encore Koor.fr.

Il n’y aura aucun plagiat dans nos programmes.