Projet programmation 2022

Le but du projet est d'utiliser la base de données des arbres de Paris, stockée au format CSV, afin d'en extraire un sous-graphe sans cycle (un arbre) de poids minimum (le poids d'un lien étant ça longueur ici).

Imaginez que vous vouliez relier tous les arbres de Paris avec des cordes, de telle sorte que les écureils puissent se balader partout. Cependant vous voulez utiliser le moins de corde possible.

Votre programme doit avoir plusieurs actions possible. La première est la conversion de la base de données pour la stocker dans un fichier **binaire** dont vous définirez le format. Ce fichier contiendra uniquement les informations nécessaire à la résolution du problème. La deuxième action consiste à prendre la base de données stockée en binaire et d'effectuer le calcul de l'arbre de poids minimum. Chaque action acceptera plusieurs arguments de ligne de commande afin de modifier le fonctionnement (comme choisir le chemin de sortie, customiser l'apparence, filtrer, etc ...).

Notamment chaque action doit accepter au moins l'argument `-i BASE_DE_DONNEES`, indiquant le chemin vers le fichier csv de la base de données ou vers le fichier binaire que vous aurez généré à partir de la base de données dans votre propre format (le programme doit detecter quel fichier est donné en entrée).

Le projet doit s'effectuer seul ou en binôme. Les fichiers d'entêtes doivent tous être documenté. Une documentation globale devra être générée à partir des fichier source (génération à l'aide de `make doc`) et la page d'accueil de la documentation doit expliquer rapidement la structure de votre programme, comment il doit être utilisé, et ce qui ne fonctionne pas!! Le code source doit être déposés sur un projet git sur le git de l'unistra (l'enseignant doit être ajouté dans les membres "maintainer" du projet pour y avoir accès). Le code source doivent être déposés sur le git du projet avant le vendredi 20 Janvier 2022, 23h59.

Attention, si vous utilisez du code source trouvé sur internet, vous devez le citer clairement dans le code source et vous devez comprendre ce que vous avez copié (je me réserve le droit de vous poser des questions dessus). La difficulté du projet dépend de la quantité de choses que vous aurez implémentée vous-même et est prise en compte dans la notation.

La grille de notation tient compte de la clareté du code, de la structure, des choix d'implémentation, des tests et de la quantité de choses réalisées.

Notamment vous devez:

- organiser vos fichiers correctement (dans plusieurs dossiers).
- Ecrire une documentation (dans les fichiers d'entête).
- rendre un projet qui compile sans erreurs ni warning (en activant tous les warning), et sans erreur valgrind.
- écrire des tests (unitaires avec un programme `tests.c` et/ou d'exécution avec un `test.sh`) séparés qui permettent de vérifier que vos fonctions sont correctes.
- écrire un makefile avec des règles appropriées (all, doc, tests).
- prendre en charge le signal SIGINT (qui se produit lors d'un "Ctrl+C").

Détail sur le projet

Votre programme doit parser le fichier csv disponible sur le site `https://parisdata.opendatasoft.com/explore/dataset/les-arbres/export/`. Le fichier étant relativement volumineux, il est fortement conseillé de se "fabriquer" un fichier plus petit possédant les mêmes caractéristiques afin de tester le fonctionnement de votre programme. **Attention, pensez à mettre un fichier .gitignore dans votre projet qui contient le chemin vers le fichier .csv, afin qu'il ne soit pas ajouté par git. Si le fichier .csv apparait lorsque vous faites un `git status`, c'est qu'il n'est pas correctement ignoré**.

L'étape de parsage du fichier doit permettre de stocker les informations pertinentes du fichier dans des structures (on peut ainsi ignorer beaucoup d'informations). Le parsage peut être spécifique au fichier mais il serait préférable que cette étape puisse tolérer des petites variations dans le format, ou en tout cas, afficher des erreurs lisibles en cas de problème de format (s'il manque un champs par exemple).

Une fois le parsage terminé, votre programme doit être capable de sauvegarder la structure dans un fichier binaire (indiqué avec une option `-o OUTPUT_FILE`) ayant un format bien défini que vous devrez documenter. Cette étape permettra notamment de recharger la structure en mémoire beaucoup plus rapidement que de le faire depuis le fichier csv.

Une fois la structure en mémoire, vous devez implémenter des algorithmes de graphes sur le graphe des arbres de Paris. Chaque algorithme est exécuté en donnant une option de ligne de commande. Par exemple (évidement beaucoup d'exemple ci-dessous sont des extensions possibles, vous devez commencer par résoudre le problème de base avant de vous lancer dans des extensions):

- l'option `-t FICHIER` pourra correspondre à la construction de l'arbre de poids minimum dont le résultat sera sauvegarder dans le fichier `FICHIER`.
- l'option `-l LIMITE` permettra d'interdire de relier des arbres si leur distance est supérieur à `LIMITE` (dans ce cas le graphe de sortie n'est pas forcément couvrant, vous devez alors récupérer la plus grande composante connexe qu'il est possible de relier par de telles cordes).
- l'option `-s` donnera des statistiques sur le graphe (diamètre, degré moyen, ...). Cette option est surtout utile avec l'option `-l`
- l'option `-f X1,Y1,X2,Y2` pourrait permettre de ne prendre en compte que les arbres situé dans une zone géographique délimité part les points `X1,Y1` et `X2,Y2`.
- l'option `-g GENRE` permettra de ne prendre en compte que les arbres du genre donnée.
- l'option `-h HAUTEUR` permettra de ne prendre en compte que les arbres d'une hauteur donnée.

D'autres options peuvent être ajoutées. Vous pourrez aussi implémenter un affichage graphique afin de visualiser la positions des arbres de Paris (juste avec des points sur fond blanc, ou plus...).

Tests

Le projet doit avoir un `Makefile` avec une règle `clean` et dont la première règle permet de compiler l'exécutable principal.

Un dossier `tests` doit contenir un Makefile permettant de compiler un programme `tests.c` qui teste les différentes fonctions que vous avez implémentées. Il pourra aussi y avoir un script `test.sh` qui exécute le programme sur des fichiers de tests afin de vérifier que la sortie est correcte.