Projet : Arbre couvrant de poids minimal

L'objectif de ce projet est de développer un code optimisé pour calculer *le réseau le plus court*[1] *reliant des villes françaises*, ainsi que la taille de ce réseau. Pour ce faire, on utilisera l'algorithme de Prim sur un graphe initial dont les nœuds sont les villes, les arêtes sont des connexions entre les villes, et le poids associé à chaque arrête est la distance entre les villes adjacentes. Initialement, on considère que toutes les villes sont reliées *(i.e. le graphe initial est complet)*.

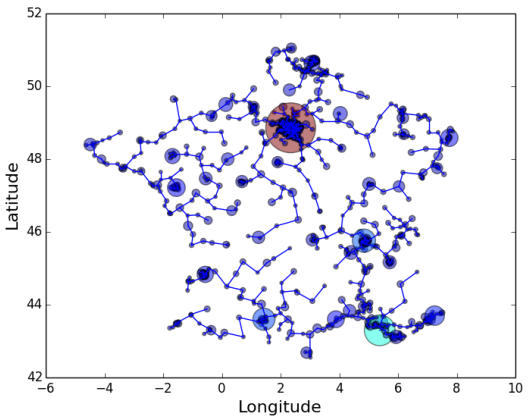

Figure 1: Exemple de réseau le plus court. La taille de chaque bulle correspond à la population. Cette image a été générée avec le script `visualisation.py`.

Étape 1 : Pré-requis et démarrage du projet

Au niveau logiciel, vous aurez besoin du compilateur `icc` *et d'une version de* `python` *avec la librairie* `mathplotlib` *[* `https://matplotlib.org/` *]. Si vous devez installer* `mathplotlib` *:*

- *Sous Linux/Ubuntu :* `sudo apt install python-matplotlib`
- *Sous macOS (si vous utilisez macport) :* `sudo port install py-matplotlib`
- *Pour les utilisations Windows, voir le site Internet de la librairie.*

Pour démarrer ce projet, on fournit :

1. Une base de données (`citiesList.csv`)[2] avec les villes françaises de métropole et leurs caractéristiques.
2. Une fonction `C` (`citiesReader.c` et `citiesReader.c`) pour lire les latitudes/longitudes (en degrés) des villes françaises (de N habitants ou plus) à partir la base de donnée, et pour écrire ces coordonnées dans un autre fichier pour la visualisation.
3. Un script Python (`visualisation.py`) pour générer des images avec la position des villes et le graphe obtenu.
4. Un fichier `C` de départ. (`main.c`)

Pour comprendre le fonctionnement des fichiers fournis, testez :

```
>> icc -std=c99 main.c citiesReader.c
>> ./a.out
Minimal population? 250000
...
>> python visualisation.py
```

[1] Plus rigoureusement : un des réseaux les plus courts.

[2] Ce fichier est une version adaptée/modifiée d'un fichier téléchargé à l'adresse: `http://sql.sh/736-base-donnees-villes-francaises`

Étape 2 : Implémentation de l'algorithme de Prim

Implémentez le calcul du réseau le plus court reliant les villes de N habitants ou plus *(en utilisant l'algorithme de Prim sur toutes les villes de N habitants ou plus)*. La distance entre deux villes (a et b) est calculée avec la formule:

$$R \cdot \mathrm{acos}\Big(\sin(\mathrm{lat}_a)\ \sin(\mathrm{lat}_b) + \cos(\mathrm{lon}_a - \mathrm{lon}_b)\ \cos(\mathrm{lat}_a)\ \cos(\mathrm{lat}_b)\Big)$$

avec $R \approx 6378\mathrm{km}$. Ajoutez le calcul de la taille du réseau *(il est obtenu additionnant le poids des arêtes du graphe final)*, et validez l'algorithme avec un petit nombre de villes.

Pour la validation, vous pouvez tester avec quelques villes uniquement, et estimer si la distance calculée est raisonnable. Sur quelques villes, vous pouvez également vérifier "à l'oeil" si votre arbre est bien un arbre de poids minimal. Si il y a une erreur, vérifiez chaque résultat de chaque étape de votre implémentation.

Étape 3 : Variante à poids minimal par département

On considère le réseau alternatif obtenu en combinant (1) le réseau national reliant la plus grande ville de chaque département aux plus grandes villes des autres départements *(en utilisant l'algorithme de Prim sur ces grandes villes uniquement)* et (2) les réseaux départementaux reliant les villes de N habitants ou plus de chaque département *(en utilisant l'algorithme de Prim département par département)*. Implémentez cette variante et validez-là.

Pour la validation de cette variante, vous pouvez procéder comme pour la validation de la première variante. Attention, il y a des incohérences dans la base de donnée, pas d'inquiétudes. :-)

Étape 4 : Optimisation et parallélisation

Optimisez au maximum votre *(ou vos)* implémentation*(s) (pour un grand nombre de villes)*. En particulier, analysez la possibilité *(ou non)* de vectoriser et/ou de paralléliser les boucles et les tâches.

Étudiez la pertinance des optimisations principales en estimant le gain de rapidité avec chacune de ces optimisations *(pour un grand nombre de villes)*. Ensuite, étudiez les performances de votre implémentation en fonction du nombre de villes.

Pour les études de performance, ne considérez que les étapes de calcul. Les lectures et écritures de fichiers sont ignorées.

Consignes finales

L'évaluation de ce projet se fera sur base d'un rapport écrit et des codes finaux *(nettoyé !)*, à envoyer par e-mail à `axel.modave@ensta-paris.fr` ***au plus tard le jeudi 5 mai 2022***. En outre, une soutenance est prévue le ***mardi 10 mai 2022***.

Dans votre rapport de ***maximum 6 pages***, montrez et discutez des résultats de validation, expliquez les stratégies d'implémentation *(structures de donnés, vectorisation, parallélisation, ...)* et étudiez la performance de ces stratégies.

Vos codes finaux doivent :

1. contenir la commande de compilation, en commentaire, au dessus du fichier principal (`main.c`);
2. demander la variante *(si les deux variantes ont été réalisées)* et le population minimale N des villes à considérer;
3. écrire le fichier `resuGraph.dat` avec la liste des arêtes, et `resuCities.dat` avec la population et les coordonnées des villes;
4. afficher le temps d'exécution *(les parties lecture/écriture de fichiers sont ignorées)*.