TP6. Conception d'un réseau ferroviaire en C

Dans ce TP, on s'intéresse aux arbres couvrants de poids minimal dans le contexte d'un graphe géométrique (les sommets sont des points, toutes les arêtes sont définies, et leur poids est égal à la distance entre leurs extrémités).

I. Présentation

Le problème que nous allons résoudre ici est celui de concevoir le plus petit réseau possible reliant un ensemble de villes. Il se trouve que si l'on pose ce problème avec un formalisme de graphe avec arêtes pondérées, le réseau voulu correspond à l'arbre couvrant de poids minimal.
Le dossier de code qui vous est fourni est organisé comme suit :

- Le répertoire common : contient les fichiers pour la structure du graphe, le parser (lecture des fichiers de données) et le générateur de .geojson, utilisé pour la visualisation. Vous n'avez pas à les modifier et vous n'avez pas besoin de les lire.
- Le répertoire eleves : contient les fichiers que vous avez à compléter. Il y a :
  - `union_find.c` : structure d'Union-find, utilisée dans l'algorithme de Kruskal ;
  - `tri_aretes.c` : trie en place un tableau d'arêtes ;
  - `arbre_couvrant.c` : là où il faudra implémenter l'algorithme de Kruskal
- Le fichier `tp6.c` : point d'entrée du programme. Il contient la fonction `main`. Vous n'avez ni à le modifier ni à le lire.
- Les fichiers `test_tri_aretes.c` et `test_union-find.c` : pour tester vos fonctions de bases.
- Fichiers `*.h` : Fichiers d'entêtes pour tous les codes. L'entête `graphe_ville.h` contient les définitions des structures pour les nœuds et les arêtes (ainsi que des fonctions pour simplifier leurs utilisations). Il est nécessaire de les comprendre pour implémenter l'algorithme de Kruskal. Il est également utile de lire `union_find.h` pour comprendre quelle implémentation de la structure est attendue.

Pour compiler votre code, on utilisera les instructions suivantes :
`gcc -Wall -I. tp11.c eleves/*.c common/*.c -lm`
Pour compiler le corrigé, on utilisera :
`gcc -Wall -I. tp11.c corrigé/*.c common/*.c -lm`
Similairement, pour compiler les programmes de test fournis pour Union-find et pour le tri des arêtes :
`gcc -Wall -I. test_union_find.c eleves/*.c common/*.c -lm`
et
`gcc -Wall -I. test_tri_aretes.c eleves/*.c common/*.c -lm`
Pour décortiquer la commande : `-I.` précise que les fichiers `.h` sont dans le dossier courant, puis on spécifie les sources (l'étoile veut dire n'importe quelle suite de caractères), et `-lm` note qu'il faut charger la librairie mathématique.
Pour tester ces algorithmes, un jeu de données vous est fourni : `villes-france-100k.csv`

II. Implémentation de l'algorithme de Kruskal

1. Lire l'entête `union_find.h` et comprendre quelle représentation est proposée et quelles sont les primitives à implémenter.
Implémenter la structure d'Union-Find dans le fichier `union_find.c`. On pourra commencer par une implémentation naïve, puis revenir sur cette question pour y ajouter la fusion par rang et la compression de chemins.

On rappelle que pour l'algorithme de Kruskal commence par trier toutes les arêtes par ordre de poids croissant. Comme on considère un graphe géométrique, il contient toutes les arêtes (non-orientées) possibles.

2. Implémenter un algorithme de tri en place (c'est-à-dire sans allouer de mémoire supplémentaire mais en procédant par simples échanges) dans le fichier `tri_aretes.c`. On pourra proposer un tri quadratique simple puis revenir sur cette question une fois les autres traitées pour implémenter un tri rapide.

3. Si n est le nombre de sommets du graphe G géométrique considéré, combien G possède-t-il d'arêtes ? Quel est le nombre d'arêtes d'un arbre couvrant de G ?

4. Implémenter l'algorithme de Kruskal dans `arbre_couvrant.c`.
Indications : Commencer par générer toutes les arêtes dans un tableau de la bonne taille, puis trier ce tableau. On peut ensuite parcourir toutes les arêtes et ajouter les arêtes admissibles dans un tableau d'arêtes de la bonne taille que l'on renverra à la fin. N'oubliez pas d'arrêter le parcours des arêtes si on sait déjà que l'on a terminé.
Pour la première étape, on aura besoin d'une indexation explicite de l'ensemble des arêtes c'est-à-dire des couples $\{(i, j) : i < j\}$.

III. Visualisation

5. Aller sur `http://umap.openstreetmap.fr/fr/map/new` et importer le fichier de sortie `.geojson` produit par votre pro- gramme. Recommandation : dans les paramètres aller dans Propriétés de forme par défaut → Forme de l'icône et choisir Cercle, pour avoir des points plus petits et un rendu plus lisible.
Comparer le graphe obtenu en France avec une carte du réseau de TGV.