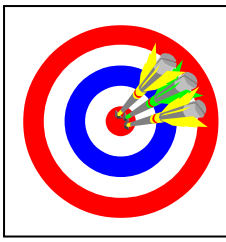

Algorithmes avancés

Réseau CFF – théorie des graphes

ITI – orientation logiciel & systèmes complexes — Travaux pratiques
hepia, HES-SO//Genève

Objectifs

- Modéliser un réseau ferroviaire à l'aide d'un graphe non-orienté valué.
- Appliquer des algorithmes de la théorie des graphes, notamment vérification de la connexité et calcul du plus court chemin entre deux sommets.

Enoncé

Ecrire un paquetage qui permette la modélisation d'un réseau ferroviaire sous forme de graphe pondéré (on considérera l'exemple de la Suisse). Il faudra en particulier trouver le plus court chemin entre deux villes. Chaque sommet du graphe représentera une ville. Deux villes avec une connexion directe en train, sont reliées par une arête dont le poids est le temps de parcours.

Cahier des charges

1. Créer un paquetage qui permet la gestion d'un graphe pondéré générique avec les fonctionnalités suivantes :
   - Charger le graphe à partir d'un fichier XML ;
   - Sauvegarder le graphe dans un fichier XML ;
   - Transformer la matrice des poids du graphe en une liste de poids, et inversément ;
   - Pouvoir afficher le graphe sous forme de matrice des poids ou sous forme de liste des poids ;
   - Ajouter ou détruire soit un sommet, soit une arête entre deux sommets du graphe ;
   - Vérifier la connexité du graphe ;
   - Implémenter les algorithmes de Dijkstra et Floyd ;
   - Pouvoir afficher les matrices des poids des parcours et de précédences pour Floyd ou les tableaux correspondants pour Dijkstra ;
   - Donner le chemin de poids minimal entre deux sommets et le poids du parcours associé à ce chemin avec Floyd et Dijkstra ;
2. En utilisant ce paquetage, créer une application en mode console qui implémente le réseau ferroviaire suisse. Utiliser le fichier XML **villes.xml** pour créer le graphe.

Données

Dans le cours **Algorithmes avancés** sur **https://cyberlearn.hes-so.ch/course/** vous trouverez deux fichiers:

- **`villes.xml`** : une suite de villes avec leurs coordonnées 'suisses' et la liste des temps de parcours avec les autres villes en liaisons directes ;
- **`Main_skel.java`** : la définition du menu utilisateur et des formats de sortie des données pour l'application en mode console ; à noter que les impressions du menu sont envoyées sur le canal d'erreur (via `System.err.print()`) pour les différencier des sorties de l'application, lesquelles sont envoyées sur la sortie standard (via `System.out.print()`) ; on utilise `Integer.MAX_VALUE` pour représenter une valeur entière infinie pour laquelle on imprimera la chaîne de caractères "`inf`", et `-1` pour désigner l'absence de prédécesseur dans les algorithmes de Floyd et Dijkstra.

Consignes et rendu

Vous devez placer l'ensemble des fichiers et paquetages de votre application en mode console dans un **dossier à votre nom** (seulement votre nom, pas le prénom)**, tout en minuscules sans espaces ni accents**. Ce dossier contiendra en particulier la classe principale **obligatoirement** appelée **`Main.java`**, basée sur le fichier Main_skel.java, et le fichier **`villes.xml`**. Vous zipperez ce dossier dans une **archive .zip** du même nom (**p.ex. `feka.zip` pour l'étudiant Ermal Feka**) ; ceci peut être fait en ligne de commande avec `zip -r feka.zip feka/`).
Cette archive devra être déposée sous **https://cyberlearn.hes-so.ch/course/** dans :

Algorithmes avancés → Graphes → Rendu du TP CFF

Une fois l'archive dézippée, la compilation de `Main.java` doit pouvoir se faire dans le dossier résultant simplement avec la commande **`javac Main.java`** et le lancement de l'application avec **`java Main`**

Attention ! Vous ne devez rien modifier ni aux données en entrée, ni au menu, ni au format de sortie des données. Le non respect des consignes sera sanctionné.

La date de rendu sera indiquée dans cyberlearn.
Votre application doit être structurée avec des classes et des paquetages.
Les méthodes doivent être testées et les codes sources commentés.