Algorithmes et structures de données 2
Laboratoire n°3
Arbres couvrants de poids minimaux et de plus courts chemins

28.10.2020

Introduction

Ce laboratoire a pour but de travailler les algorithmes d'arbres couvrants minimaux et de plus court chemin, il est composé de quatres parties distinctes :

1. Dans la première partie, vous implémenterez l'algorithme de Boruvka, permettant de trouver l'arbre couvrant de poids minimal d'un graphe.
2. Dans la deuxième partie, vous implémenterez l'algorithme de Dijkstra, vu en cours.
3. Dans la troisième partie, vous appliquerez les algorithmes d'arbres couvrants minimaux et de plus court chemin à des graphes contenant des données réelles. Nous nous intéresserons ici aux grandes lignes du réseau ferroviaire de la Suisse.
4. Dans la quatrième et dernière partie, vous détecterez la présence de circuit absorbant grâce à l'algorithme de Bellman-Ford, que vous appliquerez à un graphe représentant des taux de change.

Environnement de travail

- Pour expérimenter sur votre IDE, le plus simple est de télécharger les archives zip depuis Cyberlearn.
- Dans ce cas, faites attention à ce que les fichiers textes et d'images soient visibles par votre programme exécutable (dans CLion, ajouter le chemin du dossier de votre projet dans *Run -> Edit Configurations -> Working Directory*).
- Les fichiers textes utilisés pour construire les graphes doivent être passés en arguments de votre programme.

Profs: Laura Elena Raileanu, Marc Dikötter
Assistants: Antoine Rochat, Guillaume Hochet

Travail demandé

Exercice 1 – Algorithme de Boruvka

- Le but de cet exercice est d'implémenter l'algorithme de Boruvka avec Union-Find, qui est un autre algorithme de recherche d'arbre couvrant de poids minimal dans un graphe pondéré.

- **Travail à faire :**

1. Implémenter la méthode permettant de tester votre future implémentation de l'algorithme dans le fichier `main.cpp`.
2. Implémenter l'algorithme de Boruvka avec Union-Find dans la classe `MinimumSpanningTree.h`, dont le pseudo-code vous est fourni.

Exercice 2 – Algorithme de Dijkstra

- Le but de cet exercice est d'implémenter l'algorithme de Dijkstra vu en cours, qui est un autre algorithme de recherche de chemin le plus court dans un graphe orienté pondéré.

- **Travail à faire :**

1. Implémenter la méthode permettant de tester votre future implémentation de l'algorithme dans le fichier `main.cpp`.
2. Vous devez implémenter la classe `DijkstraSP`, sous-classe de `ShortestPath`, permettant d'appliquer l'algorithme de Dijkstra à un graphe. Vous pouvez vous inspirer du pseudo code vu en cours, de l'implémentation de Prim ou alors implémenter vous-même une structure IndexMinPQ.

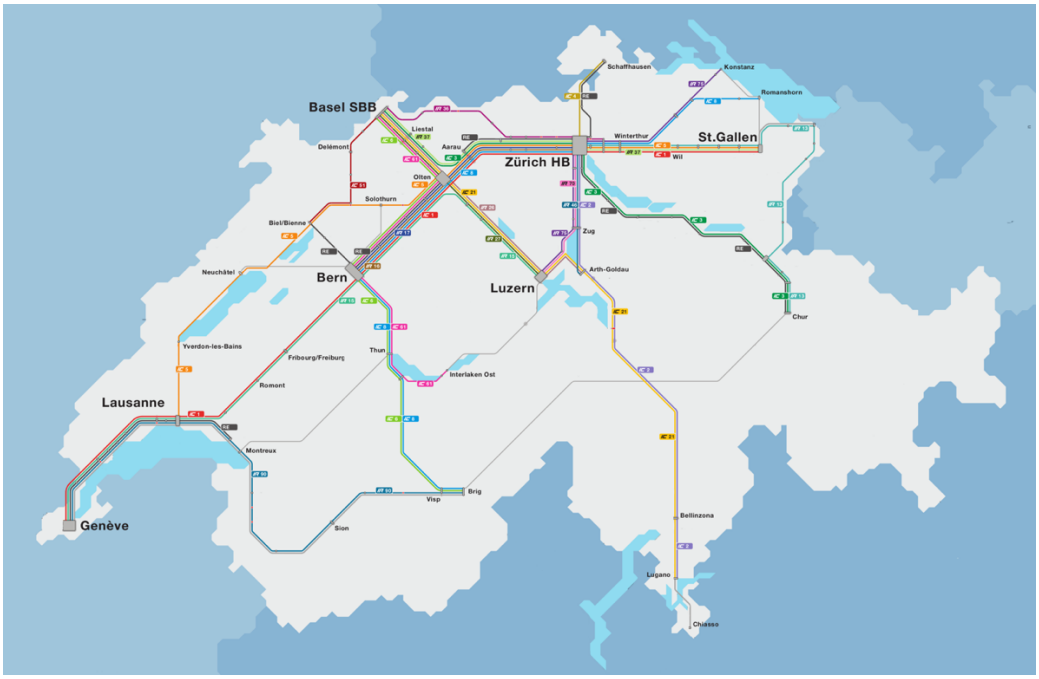
Exercice 3 – Réseau ferroviaire

- L'objectif de cet exercice d'appliquer des algorithmes de plus court chemin et d'arbre couvrant de poids minimal, vus en cours, à un graphe représentant des données réelles, ici un réseau ferroviaire.

- Ce réseau ferroviaire représente les grandes lignes du réseau ferroviaire suisse.

- Il peut être modélisé sous la forme d'un graphe :
  - Les sommets du graphe associé correspondent aux villes
  - Les arêtes correspondent aux lignes de chemin de fer les reliant.

- À chaque ligne sont associés trois poids : la distance en kilomètres séparant les deux villes, la durée moyenne du trajet en minutes, ainsi que le nombre de voies sur la ligne.



- **Objectifs :**
  - Votre implémentation va vous permettre de répondre aux différentes questions suivantes :

Plus court chemin :

- Quel est le chemin le plus court entre Genève et Coire ? Quelles sont les villes traversées ?
- Mêmes questions mais en supposant que la gare de Sion est en travaux et donc qu'aucun train ne peut transiter par cette gare.
- Quel est le chemin le plus rapide entre Genève et Coire en passant par Brigue ? Quelles sont les villes traversées ?
- Mêmes questions mais entre Lausanne et Zurich, en passant par Bâle.

Arbre recouvrant de poids minimum :

- Nous voulons faire des travaux d'entretien sur le réseau ferroviaire, mais dans un souci d'économie nous souhaitons réduire le coût des travaux au maximum. Toutes les villes devront être accessibles par une ligne rénovée. Chaque ligne possède un nombre de voies allant de 1 à 4. Le coût pour rénover 1 km de ligne de chemin de fer varie selon le nombre de voies :
  - 15M CHF par km pour les lignes ayant 4 voies
  - 10M CHF par km pour les lignes ayant 3 voies
  - 6M CHF par km pour les lignes ayant 2 voies
  - 3M CHF par km pour les lignes ayant 1 voie

  Quelles lignes doivent être rénovées ? Quel sera le coût total de la rénovation de ces lignes ?

- **Travail à faire :**
  - `ShortestPath.h`
    Vous devez implémenter la méthode `PathTo` permettant de lister les arcs constituant un chemin le plus court du sommet source au sommet puit.

  - `TrainGraphWrapper.h`
    Afin d'appliquer les algorithmes de plus court chemin ou d'arbre couvrant minimum à notre réseau ferroviaire, vous devez compléter les deux wrappers suivants :
    `TrainGraphWrapperDirected` : permet de simuler un graphe orienté (pour les algorithmes de plus court chemin)
    `TrainGraphWrapper` : permet de simuler un graphe non-orienté (pour les algorithmes de recherche d'arbre couvrant de poids minimum).

    Ces *Wrappers* encapsulent un *TrainNetwork* et incluent une fonction de coût, qui permet de calculer le coût d'une ligne (donc d'une arête) en fonction du problème à résoudre.

    Ces Wrappers doivent implémenter au minimum les méthodes suivantes, afin d'être appelés par les différents algorithmes d'arbre couvrant de poids minimal et de plus court chemin :
    - `V()`
    - `forEachVertex(Func f)`
    - `forEachEdge(Func f)`
    - `forEachAdjacentEdge(int v, Func f)`

  - `main.cpp`
    Implémentation des méthodes `PlusCourtChemin`, `PlusCourtCheminAvecTravaux`, `PlusRapideChemin` et `ReseauLeMoinsCher`. Il vous faudra utiliser les algorithmes fournis dans les classes `MinimumSpanningTree` et `ShortestPath` ainsi que vos wrappers. Voici un exemple d'utilisation des wrappers :

```
TrainNetwork tn("reseau.txt");
// ShortestPath
TrainGraphWrapperDirected tdgw(tn, costFunction);
BellmanFordSP<TrainGraphWrapperDirected> sp(tdgw, v);

// MinimumSpanningTree
TrainGraphWrapper tgw(tn, costFunction);
auto mst = MinimumSpanningTree<TrainGraphWrapper>::EagerPrim (tgw);
```

    Nous vous conseillons de tester vos méthodes des wrappers à l'aide des algorithmes de Boruvka et Dijkstra que vous avez implémentés dans la partie 1 et 2 de ce laboratoire.

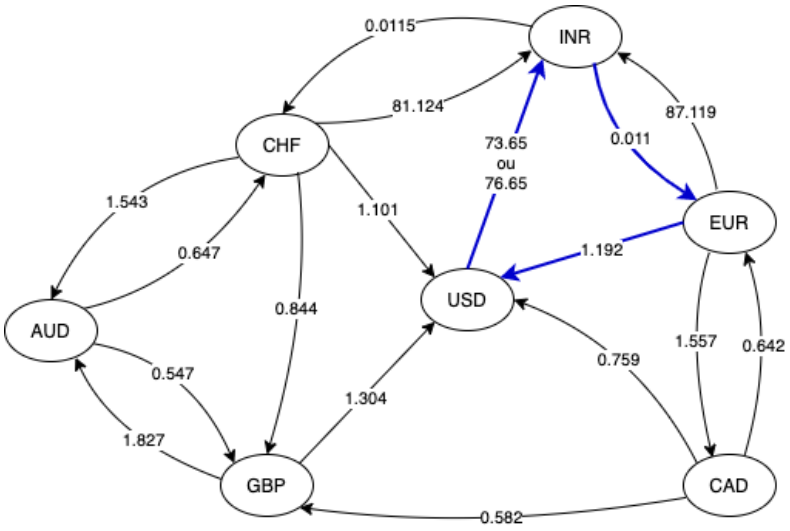
Exercice 4 – Réseau boursier

- L'objectif de cet exercice d'implémenter la détection de circuit absorbant à l'aide de l'algorithme de Bellman-Ford, et ensuite d'appliquer cette détection à un réseau boursier.

- Notre réseau boursier contient les taux de change entre différentes monnaies.

- Il peut être modélisé sous la forme d'un graphe :
  - Les sommets du graphe associé correspondent aux monnaies

- Les arcs correspondent aux taux de change d'une monnaie à une autre.



1 EUR (x1.192) -> 1.192 USD (x76.65) -> 91.3668 INR (x0.011) -> 1.00503 EUR

- L'objectif de cet exercice est d'analyser notre graphe afin de trouver s'il contient un chemin permettant de gagner de l'argent. Pour cela, nous allons utiliser l'algorithme de Bellman-Ford qui permet de détecter les circuits absorbants.

- **Travail à faire :**
  - Implémenter l'algorithme de Bellman-Ford avec détection de circuit absorbant dans la classe `ShortestPath.h`. Cet algorithme doit lancer une exception si un circuit absorbant est détecté.
  - Compléter le fichier `main.cpp` afin de lister un chemin absorbant (s'il y en a un) ainsi que le gain qu'il permet de réaliser.

Modalités pratiques

- À rendre sur Cyberlearn au plus tard le dimanche **01.12.2020** à **23h59.**
- Pas de rapport à rendre pour ce laboratoire, mais soignez vos commentaires et entêtes.
- Travail à faire par groupes de trois étudiant-e-s. Durée : 10 périodes encadrées. Sources sur Cyberlearn.
- Respectez les consignes indiquées sur Cyberlean.

Durée

- 10 périodes
- A rendre le dimanche **01.12.2020** à **23h59** au plus tard