HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Labo 3 : Réseau routier

Rapport

Étudiants	Gilliand Loris - Tutic Mateo - Wachter Luc
Cours	ASD2
Nom du professeur	Laura Elena Raileanu
Nom de l'assistant	Valentin Minder
Date	25 novembre 2018

Table des matières

1. Introduction

Le laboratoire trois d'ASD2 permet la mise en pratique des connaissances sur les graphes et les algorithmes étudiés en cours. En effet, il demande d'utiliser les arbres recouvrants de poids minimal et les arbres de plus courts chemins pour répondre à des questions pratiques sur des itinéraires de Suisse.

Ce document a pour but de présenter nos méthodes et résultats dans l'implémentation de l'algorithme de Dijkstra, des wrappers permettant de manipuler les réseaux routiers fournis et des autres méthodes nécessaires au fonctionnement du laboratoire.

2. Choix d'implémentations

2.1. Shortest path

Dans les deux algorithmes (`BellmanFordSP` et `DijkstraSP`), l'arête (ou arc) qui mène jusqu'au sommet source est représentée par une boucle de poids 0 sur le sommet source. En effet, on retrouve dans les deux algorithmes le code suivant : `this->edgeTo[v] = Edge(v,v,0);` (v étant le sommet source). Cette représentation permet dans `PathTo(int v)` de parcourir le tableau des `edgeTo` en partant de v et en remontant jusqu'au sommet source (le sommet source qui est donc représenté par la boucle).

2.2. Wrappers

Nous avons choisi d'implémenter deux classes : l'une pour voir le réseau routier comme un graphe non orienté (`RoadGraphWrapper`) et l'autre pour le voir comme un graphe orienté (`RoadDiGraphWrapper`).

2.2.1. Constructeurs et fonction de coût

Les deux classes sont très similaires. Dans les deux cas, le type de poids de leurs arêtes ou arcs est générique. Elles proposent toutes deux un constructeur prenant un `RoadNetwork` en paramètre et un autre qui prend en plus une fonction de coût.

Cette fonction de coût est stockée dans le wrapper, à l'aide de la bibliothèque `<functional>`. Son type est `std::function<edgeWeightType(const RoadNetwork::Road&)>` (où `edgeWeightType` est le type du poids des arêtes ou arcs). Si la fonction de coût n'est pas spécifiée, le poids des arêtes sera égal à la longueur de la route.

2.2.2. API

Les deux classes offrent le même API :

- `V()` - retourne le nombre de villes du réseau routier (sommets).
- `forEachEdge(Func f)` - applique la fonction *f* sur tous les arêtes ou arcs du graphe.
- `forEachAdjacentEdge(int v, Func f)` - applique la fonction *f* sur tous les arêtes ou arcs adjacents au sommet *v*.
- `forEachVertex(Func f)` - applique la fonction *f* sur tous les sommets du graphe.

3. Réponses aux questions

3.1. Chemin le plus court entre Genève et Emmen

Le résultat obtenu est le suivant.

```
1. Chemin le plus court entre Geneve et Emmen
Geneve -> Nyon -> Morges -> Lausanne -> Chietres -> Berne -> Emmen
Distance totale : 238km
```

3.2. Chemin le plus court entre Lausanne et Bâle

Le résultat obtenu est le suivant.

```
2. Chemin le plus court entre Lausanne et Bale
Lausanne -> Yverdon-Les-Bains -> Neuchatel -> Bienne -> Delemont -> Basel
Distance totale : 185km
```

3.3. Chemin le plus rapide entre Genève et Emmen, en passant par Yverdon

Le résultat obtenu est le suivant.

```
3. Chemin le plus rapide entre Geneve et Emmen en passant par Yverdon
Geneve -> Nyon -> Morges -> Lausanne -> Yverdon-Les-Bains -> Neuchatel -> Bienne -> Soleure -> Olten -> Emmen
Temps total : 156.429 minutes
```

3.4. Chemin le plus rapide entre Genève et Emmen, en passant par Vevey

Le résultat obtenu est le suivant.

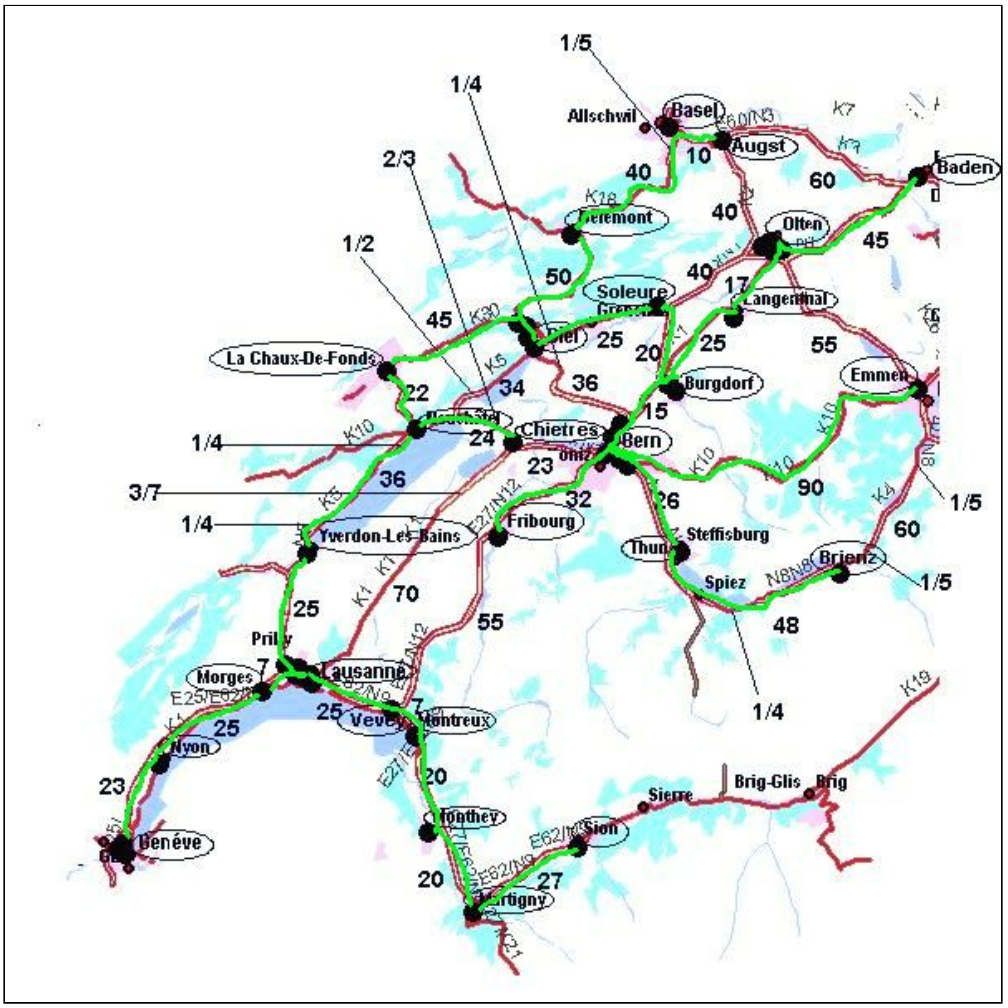
```
4. Chemin le plus rapide entre Geneve et Emmen en passant par Vevey
Geneve -> Nyon -> Morges -> Lausanne -> Vevey -> Fribourg -> Berne -> Burgdorf -> Soleure -> Olten -> Emmen
Temps total : 148.5 minutes
```

3.5. Routes à rénover et coût de la rénovation

Le résultat obtenu est le suivant. Les routes surlignées en vert sur la carte ci-dessous sont ceux qui font partie de l'arbre recouvrant de poids (prix de rénovation) minimum du graphe, trouvé à l'aide de l'algorithme de Prim strict.

```
5. Quelles routes doivent etre renovees ? Quel sera le cout de la renovation de ces routes ?
Augst - Basel : 150MF
Basel - Delemont : 344MF
Delemont - Bienne : 350MF
Bienne - Soleure : 175MF
Soleure - Burgdorf : 300MF
Burgdorf - Langenthal : 175MF
Langenthal - Olten : 119MF
Berne - Burgdorf : 225MF
La Chaux-De-Fonds - Bienne : 315MF
Neuchatel - La Chaux-De-Fonds : 330MF
Chietres - Neuchatel : 232MF
Berne - Thun : 390MF
Yverdon-Les-Bains - Neuchatel : 396MF
Lausanne - Yverdon-Les-Bains : 375MF
Morges - Lausanne : 105MF
Nyon - Morges : 375MF
Geneve - Nyon : 345MF
Lausanne - Vevey : 375MF
Vevey - Montreux : 105MF
Montreux - Monthey : 300MF
Monthey - Martigny : 300MF
Martigny - Sion : 405MF
Fribourg - Berne : 480MF
Thun - Brienz : 624MF
Emmen - Berne : 630MF
Olten - Baden : 675MF

Cout total : 8595MF
```



4. Conclusion

Pour réaliser ce laboratoire, nous avons dû regrouper toutes nos compétences, autant pour la réalisation des algorithmes sur les différents graphes que pour l'application en C++ de ces derniers.

Premièrement, la réalisation de l'algorithme de Dijkstra nous à pris un certain temps à implémenter du fait de certains mauvais choix. Nous avions d'abord choisi de réaliser cet algorithme avec une queue de priorité puis nous avons réalisé qu'il serait plus judicieux d'utiliser un set.

Concernant la partie wrapper, la mise en route fut plus laborieuse. Du fait du nombre de fichiers et de la généricité des classes, il était difficile de s'y retrouver ainsi que de voir clairement la méthode à appliquer. Après de longues heures de lecture des fichiers reçus, un début d'idée nous a permis d'avancer. En quelques lignes de code, nous avions un wrapper non orienté qui permettait d'afficher toutes les arêtes du graphes. Nous avons ensuite implémenté les fonctions nécessaires ainsi que le stockage d'une fonction transmise en paramètre. Dès lors, le laboratoire devenait plus clair et nous voyions la fin approcher.

Après avoir terminé la version non orientée du wrapper, l'implémentation de la version orientée fut rapide. Une fois les deux wrappers opérationnels, nous avons pu réaliser les exercices demandés. Afin de valider nos résultats, nous avons appliqué, à la main, les différents algorithmes sur le réseau routier reçu. Nous avons ensuite utilisé notre code pour trouver les réponses aux exercices de la donnée. Ces résultats sont les mêmes que ceux obtenus à la main.

Pour conclure, nous sommes fiers d'avoir pu terminer l'implémentation des wrappers à temps et d'avoir compris l'utilisation de ces derniers. Le déroulement de ce laboratoire n'ayant pas été calme, nous relevons aussi que la cohésion du groupe n'a jamais fléchi. Nous avons su encourager nos collègues et nous entraider tant que possible.