ASD2 - labo 4 - Rapport

- HES-SO // HEIG-VD // ASD2 - Algorithmes et structures de données // **Prof. Olivier CUISENAIRE**
- MM. **Léonard BERNEY et Valentin MINDER** // 04.01.2015

Introduction

La première partie de ce laboratoire consiste à produire une implémentation efficace de l'algorithme de Dijkstra en C++, afin de trouver les plus courts chemins d'une sommet d'un graphe à tous les autres. Cet algorithme sera utilisé pour résoudre des problèmes concrets de navigation guidée, comme le calcul d'un plus court ou d'un plus rapide trajet entre deux villes, étant donnée un réseau routier réel. Ces performances sont également comparées à l'algorithme de Bellmann-Ford pour une série de réseaux de test, le plus grand comportant 1'000'000 sommets et 15'172'126 arêtes.

Une deuxième partie consiste à déterminer un réseau couvrant minimal afin de planifier des rénovations du réseau au moindre coût tout en connectant tout le réseau. L'algorithme de Kruskal calculant le *Minimum Spanning Tree* est fourni, il n'y a qu'a faire la connexion entre le réseau routier et un graphe à symbole via un *Wrapper* qui encapsulera le réseau afin de l'utiliser.

Problématiques

Le réseau routier fourni est défini dans sa propre structure. Un des défis était de définir plusieurs *Wrapper* qui encapsuleront le réseau afin de l'utiliser comme un Graphe standard. En particulier, nous avons dû définir 3 wrapper différents en fonction des usages. En effet, les sommets représentent toujours les villes et les arêtes les routes qui les relient. En revanche, les poids sur les arêtes peuvent être de différentes natures: longueur du tronçon, temps nécessaire pour le parcours du tronçon ou le coût de rénovation du tronçon, les deux derniers dépendant de la longueur et de la nature du tronçon (fraction de route et autoroute). Ces trois natures différentes des poids des arêtes permettent de déterminer, respectivement, les plus courts chemins, plus rapides chemins et réseau à rénover minimal.

Concernant l'algorithme de *Dijkstra*, nous avions besoin d'une structure qui permettent de trier les sommets atteignables les plus proches, tels qu'une queue de priorité. Nous avons décidé de ne pas l'implémenter en utlisant une structure *IndexMinPQ*, et nous avons donc été confrontés à un sérieux problème de performances dû à la manière peu efficace dont les sommets étaient traités à chaque itération de l'algorithme. Plutôt que d'essayer d'implémenter tel quel le pseudo code vu en cours, nous avons finalement décidés d'utiliser comme base la version eager de l'algorithme de Prim. Cette dernière approche donne des résultats nettement meilleurs, qui sont dans la fourchette attendue (moins de 10 secondes).

Notre version utilise un set de <dist,v> ou dist est la distance avec laquelle on peut atteindre le sommet v. Ainsi, le premier élément du set contient toujours le sommet le plus proche actuellement atteignable. Lors d'une mise à jour, afin de changer la priorité d'un sommet, il est

nécessaire de le supprimer avec l'ancienne valeur puis de le réajouter dans le set avec la nouvelle. De plus, comme les poids sont tous positifs, chaque sommet qui a été retiré de la queue est traité définitivement et ne doit donc plus y être rajouté. Nous utilisons un simple tableau de booléen pour assurer cela.

Réponses aux questions posées.

1. Quel est le chemin le plus court entre Genève et Emmen ? Quelles sont les villes traversées ?

- Le plus court chemin mesure `238 km` et traverse les villes de Nyon, Morges, Lausanne, Chiètres et Berne.

1. Me□mes questions entre Lausanne et Ba□le ?

- Le plus court chemin mesure `185 km` et traverse les villes de Yverdon-les-Bains, Neuchatel, Bienne et Delémont.

1. Quel est le chemin le plus rapide entre Genève et Emmen en passant par Yverdon ? Quelles sont les villes traversées ?

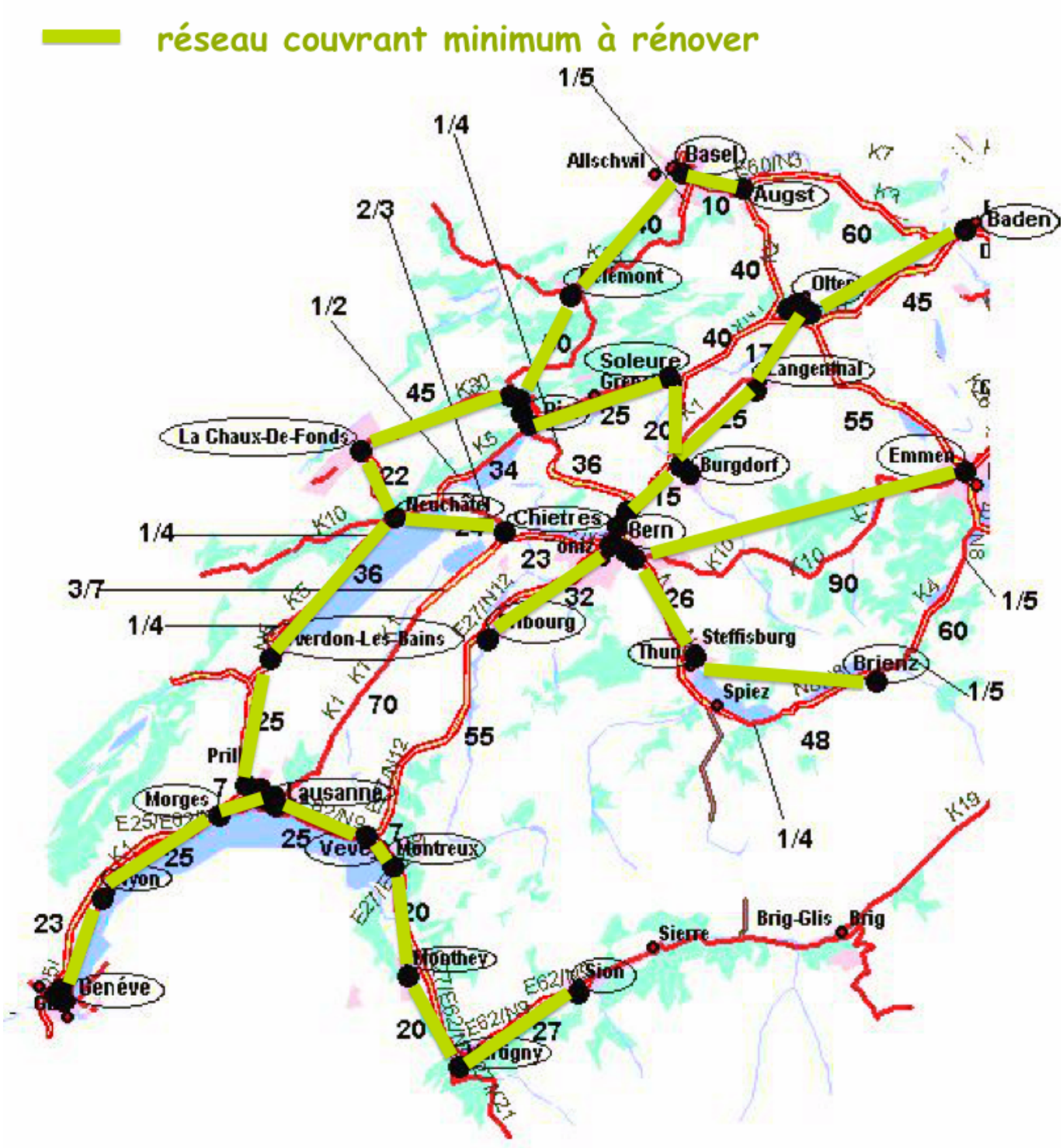
- Le chemin chemin le plus rapide prend `72 minutes` (1 heure 12 min) et traverse les villes de Nyon, Morges, Lausanne, Yverdon-les-Bains, Neuchatel, Bienne et Berne.

1. Me□mes questions mais en passant par Vevey ?

- Le chemin chemin le plus rapide prend `68.5 minutes` (1 heure 8 min 30 secondes) et traverse les villes de Nyon, Morges, Lausanne, Vevey, Fribourg et Berne.

1. Quelles routes doivent e□tre rénovées ? Quel sera le cou□t de la rénovation de ces routes ?

- Le réseau comptant `n = 27` villes, les `n - 1 = 26` tronçons suivants doivent être rénovés, dans l'ordre de cout minimal.
- Morges - Lausanne (105 MCHF), Vevey - Montreux (105 MCHF), Langenthal - Olten (119 MCHF), Augst - Basel (150 MCHF), Burgdorf - Langenthal (175 MCHF), Bienne - Soleure (175 MCHF), Berne - Burgdorf (225 MCHF), Chietres - Neuchatel (232 MCHF), Montreux - Monthey (300 MCHF), Monthey - Martigny (300 MCHF), Soleure - Burgdorf (300 MCHF), La Chaux-De-Fonds - Bienne (315 MCHF), Neuchatel - La Chaux-De-Fonds (330 MCHF), Basel - Delemont (344 MCHF), Geneve - Nyon (345 MCHF), Delemont - Bienne (350 MCHF), Lausanne - Vevey (375 MCHF), Nyon - Morges (375 MCHF), Lausanne - Yverdon-Les-Bains (375 MCHF), Berne - Thun (390 MCHF), Yverdon-Les-Bains - Neuchatel (396 MCHF), Martigny - Sion (405 MCHF), Fribourg - Berne (480 MCHF), Thun - Brienz (624 MCHF), Emmen - Berne (630 MCHF) ainsi que Olten - Baden (675 MCHF).
- Le cout total est de `8585 MCHF`, soit 8.585 milliards de francs suisses. La carte du réseau minimal est annexée.



Temps d'éxecution comparés, en millisecondes. Le nombre de sommets et d'arêtes est indiqué pour une notion d'échelle.

Fichier	V()	E()	Bellman-Ford	Dijkstra
tinyEWD.txt	8	15	0.044	0.038
mediumEWD.txt	250	2546	0.265	0.636
1000EWD.txt	1000	16866	1.362	3.564
10000EWD.txt	10000	123462	10.412	30.339
largeEWD.txt	1000000	15172126	5705.53_	4377.9__

Conclusion

Lors de tests sur le plus grand réseau, nous avons rapidement remarqué que la complexité d'un algorithme était absolument essentielle. En effet, un simple oubli de test de réajout dans la queue de priorité ou une queue de priorité gérée comme une liste dont il faut à chaque fois rechercher le plus petit élément, étaient des erreurs qui donnaient des résultats similaires sur de petits jeux de données mais catastrophiques sur de plus grands, le temps d'exécution pouvant dépasser les 15 minutes. Après avoir codé une version fonctionnelle, nous avons donc du améliorer la gestion interne des données afin d'optimiser les opérations sensibles et de rester dans des temps acceptables (moins de 15 secondes).

Ce laboratoire nous a permis de relier nos connaissances théoriques sur les algorithmes à des applications très concrètes telles qu'un réseau routier et la recherche de chemin, tantôt plus court ou plus rapide. Nous nous sommes également rendus compte de l'importance capitale de travailler avec les structures de données les plus appropriées au problème qu'on tente de résoudre afin de minimiser la complexité spatiale et temporelle d'un algoritme lorsque celui-ci doit être implémenté.