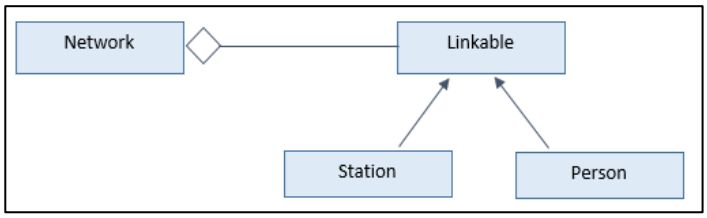
Rapport par Samy REZIG et Yves TRAN

1. Structure du code

Le projet se compose des classes :

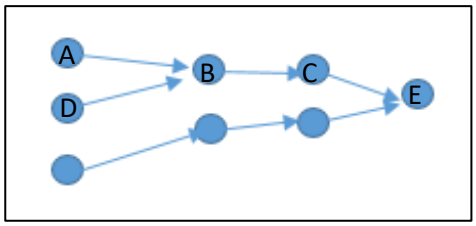
- *Network<T extends Linkable>* : représentant un réseau générique de *Linkable* ;
- *Linkable* : représentant des objets pouvant se lier ;
- *Station* : représentant une station de métro ;
- *Person* : représentant un individu dans le contexte d'un réseau social ;
- *ReversedTree* : structure de données représentant un arbre où l'accès se fait par ses feuilles ;
- *Node* : liste chainée contenant des *Linkable* représentant un chemin dans le réseau.

Notre but est de créer une structure représentant un réseau de données quelconques. Ceci justifie l'utilisation de la généricité pour la classe Network. Les classes *Station* et *Person* deviennent respectivement des exemples de réseaux (de transport et sociaux). Pour regrouper les classes pouvant former un réseau, nous introduisons l'interface *Linkable*, regroupant *Station* et *Person,* qui précise la définition de notre réseau.



Un réseau contient un *Set* de *Linkable* où chacun de ces objets sont relier entre eux. En effet, ces objets renferment la liste de leurs suivants. Les classes filles de *Linkable* gèrent ces listes et le nombre de connexions à leur manière (ref Q1).

La classe *Node* permet de constituer un chemin du réseau. Elle englobe un *Linkable* pour le lier à un autre objet *Node* qui englobe un autre *Linkable* qui se suivent dans le réseau. Afin de former un ensemble de chemin, nous définissons la classe *ReversedTree* qui a le pouvoir de stocker plusieurs chemins vers un même *Node* en minimisant le coût de la mémoire. En effet, les chemins peuvent s'entrecroiser et évite la redondance des *Node*. Sur la figure, un chemin vers E se lit A – B – C – E ou D – B – C – E.



2. Justification pour la méthode de la question 4

Pour vérifier si l'on observe les "six degrés de séparation", on a décidé modifier la méthode de recherche de chemin entre deux sommets afin de récupérer le plus court chemin avec la classe *ReversedTree*. On parcourt ensuite l'ensemble des chemins un par un puis on récupère la taille des chemins et on vérifie ensuite la taille du chemin et si on tombe sur un chemin de longueur (soit nombre de séparations) supérieur à 6, on retourne faux sinon on retourne vrai si on peut relier tous les sommets avec un chemin de longueur 6 max.

3. Implémentation de la question Bonus

Nous voyons les lignes de métro comme des composantes connexes. L'algorithme se divise en trois étapes. Premièrement, depuis un sommet de départ, nous définissons le plus court itinéraire vers

la destination. Si ce chemin contient toutes les composantes à parcourir, alors on la choisit, sinon on se déplace vers le premier sommet sur une composante non visitée (*) en sauvegardant l'itinéraire parcouru et en mettant la liste des composantes à parcourir à jour. Puis on recommence, depuis ce nouveau sommet, à chercher le chemin le plus court, regarder ces composantes et ainsi de suite.

Une fois arrivée sur le sommet final, nous obtenons un chemin avec des passages redondants. Nous allons raccourcir ce chemin en parcourant tous ses segments. Pour chaque segment (A, B), nous le comparons avec le plus court chemin de A vers B et si ce dernier passe par autant de composantes, le segment est remplacé par ce chemin, sinon le segment est laissé tel quel.

Finalement, nous avons un chemin correct qui dépend de la recherche en (*). Ce parcours visite les suivants d'un nœud courant dans un ordre aléatoire pour éviter des boucles infinies ce qui peut donner des résultats différents. Pour s'approcher de la solution optimale, nous exécutons cet algorithme un certain nombre de fois pour choisir le plus court chemin, favorisant un résultat optimal. Par exemple, de Danube vers Bercy, on a : [Danube – Botzaris – Place des Fêtes -…- Quai de la gare – Bercy], soit 40 stations. La longueur varie peu après plusieurs exécutions et reste entre 39 et 40 stations.

4. Difficulté du projet

La partie sur laquelle nous avons passé pas mal de temps était l'implémentation de la méthode de recherche de chemin. En effet, nous avons modifié la méthode à plusieurs reprises lors du développement. Au début, on avait une méthode de recherche qui nous renvoyé un chemin qui n'était pas le plus court, nous avons donc dû la modifier pour la question 4 afin qu'elle nous renvoie le chemin le plus court d'un sommet à un autre et qu'on puisse ainsi vérifier sa longueur.

Vu que nous avions décidé de modéliser un réseau de données de manière générale et non de le faire cas par cas, il a été question de rendre les méthodes les plus génériques possibles, c'est-à-dire que si on fait un autre type de réseau les méthodes soit toujours utilisables.

Dans l'ensemble, le projet était plutôt challengeant au niveau algorithmique. La question la plus difficile était tout de même la question bonus que ce soit l'algorithme ou même la vérification du chemin que ça renvoie. Il fallait que la solution reste optimale et que l'on garde un temps d'exécution court.