

**Méto : Récursivité, IA, graphes, correction d'erreurs, A\***

Vérifiez les conventions à suivre pour l'écriture du code (cf Moodle).

1. Un des points très délicat des fichiers clients à analyser c'est qu'il peut y avoir des erreurs dans les noms. Ou bien dans le fichier **Metro.txt** ou bien dans votre programme. Par exemple, il peut y avoir des noms de station identiques, mais en majuscule ou en minuscule, il peut manquer un accent (e à la place de é) etc.. Dans ce cas, vous aurez plusieurs stations avec le même nom, mais une orthographe différente – donc plusieurs stations différentes pour votre programme.

En aucun cas, vous ne modifierez le fichier client **Metro.txt** !

Pour éviter ce cas de figure au maximum, on crée généralement une méthode chargée de transformer le nom (éventuellement à problème) pour éliminer ces cas. Fabriquez une méthode qui transforme le nom d'une station en enlevant les accents, en mettant le nom en minuscule, et en enlevant les espaces du nom. Regardez les tirets dans le fichier Metro.txt et cherchez une manière de vérifier que la représentation est uniforme. Cette méthode est **statique** à la classe Réseau.

Vérifiez avec ou sans : trouvez -vous le même nombre de stations avec et sans ? Demandez à vos camarades de TD s'ils ont le même nombre.

2. Dans ce TD, vous allez réaliser des méthodes capables de trouver les chemins possibles entre deux stations de métro.

Créez une méthode de réseau **cheminDeVers( ? station1\_, ? station2\_)** qui essaye de parcourir le graphe pour aller de la **station1\_** à la **station2\_**. Vous pouvez choisir le type de paramètre en fonction de votre implémentation : cette méthode n'est pas définitive, elle sert à vous faire la main pour expérimenter le parcours du graphe avec vos outils d'accès (ceux développés dans le TD précédent).

**Si station1 == station2 alors bravo, vous avez trouvé un chemin sinon...**

**Supposons que la station1 soit directement connectée aux stations { sa, sb, sc, .... } alors,**

**chercher le chemin de la station1 à la station2 revient à chercher :**

**Le chemin passant par station1 et continuant par sa**

**Le chemin passant par station1 et continuant par sb**

**Le chemin passant par station1 et continuant par sc**

**Etc..**


Essayez et affichez les chemins. Affichez un message quand vous tombez sur une solution. Est-ce que votre programme se termine ?

3. Il faut éviter de passer plusieurs fois par la même station. Modifiez votre méthode **cheminVers** pour ajouter une collection des stations déjà visitées : vous ne passerez pas par les stations déjà rencontrées (et rajouterez les stations déjà visitées dans cette collection). Prenez un couple de stations et comparez les résultats avec vos camarades. Est-ce que vous trouvez toutes les solutions ?
4. On souhaite maintenant modifier **cheminVers** pour récupérer la liste de tous les chemins possibles d'une station vers l'autre. Modifiez le type de retours (et éventuellement rajoutez un ou plusieurs arguments) pour permettre de conserver les chemins trouvés.
5. On souhaite maintenant ne renvoyer que le chemin le plus court. Une des méthodes classiques consiste à calculer tous les chemins, puis à ne sélectionner que le plus court. Mais ce n'est pas une excellente solution : le calcul de tous les chemins possibles demande du temps et de l'espace de stockage. Vous allez utiliser une autre méthode : quand un premier chemin est trouvé, ce chemin est stocké comme meilleur chemin (pour l'instant). Au cours d'une recherche suivante, la longueur du chemin en cours de recherche est comparée à la longueur du meilleur chemin en cours. Si cette longueur est plus grande que la longueur du meilleur chemin trouvé, on abandonne cette recherche (sinon, on continue). Si une solution est trouvée avec un chemin plus court, alors cette solution devient la meilleure (pour l'instant).

Modifiez la méthode **cheminVers** pour en faire une méthode de **Reseau** qui donne le chemin le plus court entre deux stations. Exceptionnellement, vous pourrez utiliser une variable d'instance pour simplifier la programmation java (si nécessaire).

Vous avez implémenté un des algorithmes de base de l'IA (extrait de Wikipédia)

## Algorithme A\*

 Pour les articles homonymes, voir [A\\*](#).

L'algorithme de recherche **A\*** (qui se prononce **A étoile**, ou **A star** à l'anglaise) est un algorithme de [recherche de chemin](#) dans un [graphe](#) entre un [noeud](#) initial et un noeud final tous deux donnés. De par sa simplicité il est souvent présenté comme exemple typique d'algorithme de [planification](#), domaine de l'[intelligence artificielle](#). L'algorithme A\* a été créé pour que la première solution trouvée soit l'une des meilleures, c'est pourquoi il est célèbre dans des applications comme les jeux vidéo privilégiant la vitesse de calcul sur l'exactitude des résultats. Cet algorithme a été proposé pour la première fois par [Peter E. Hart](#) <sup>(en)</sup>, [Nils John Nilsson](#) <sup>(en)</sup> et [Bertram Raphael](#) <sup>(en)</sup> en 1968<sup>1</sup>. Il s'agit d'une extension de l'[algorithme de Dijkstra](#) de 1959.

Maintenant, essayez de réaliser cet algorithme sans utiliser la récursivité (c'est possible).