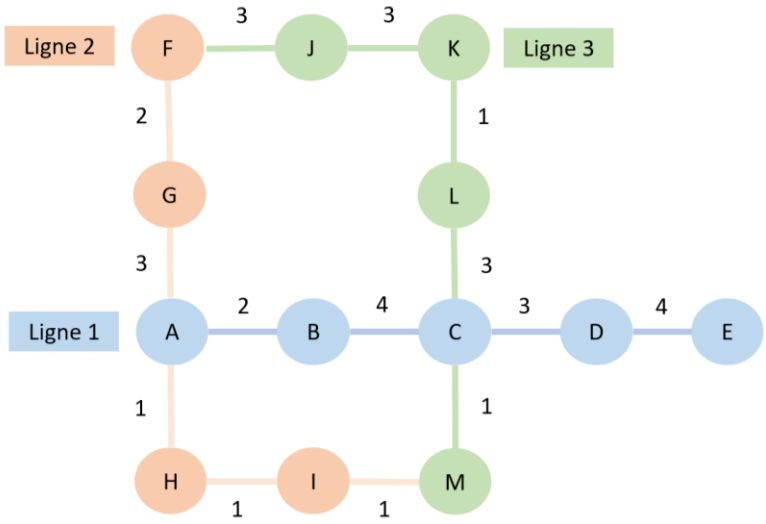
Informatique

11
Bases des graphes

TD 11-2
Dijkstra

Exercice 1: Dijkstra

Voici un plan de métro :



Le réseau est composé de trois lignes. Les stations sont nommées par des lettres afin de faciliter l'étude. Les temps de parcours en minutes sont indiqués sur les branches de chaque ligne entre chaque station. Nous allons programmer l'algorithme de Dijkstra afin de déterminer le plus court chemin sur ce réseau d'une gare de départ à une gare d'arrivée, par exemple dans notre cas, de F à C.

On définit le réseau ainsi :

```
Ligne_1 = ['A','B','C','D','E']
Ligne_2 = ['F','G','A','H','I','M']
Ligne_3 = ['F','J','K','L','C','M']
Lignes = [Ligne_1,Ligne_2,Ligne_3]

Segments_1 = [[2,2],[4,4],[3,3],[4,4]]
Segments_2 = [[2,2],[3,3],[1,1],[1,1],[1,1]]
Segments_3 = [[3,3],[3,3],[1,1],[3,3],[1,1]]
Segments = [Segments_1,Segments_2,Segments_3]
```

On notera que les segments définis ci-dessus permettent de donner le temps de parcours dans les deux sens. Ainsi, bloquer le segment de B à C et de B à A revient à écrire :

```
Segments_1 = [[2,0],[0,4],[3,3],[4,4]]
```

A partir de cela, on propose les fonctions permettant de créer automatiquement

- La liste **Stations** des différentes stations du réseau
- Le nombre de stations **Nb**
- La matrice **Graphe** (array de numpy) des distances du réseau

Téléchargez le fichier en lien ici afin d'avoir le graphe à disposition. Vous le compléterez à la suite dans la partie réservée à l'algorithme de Dijkstra.

Description de l'algorithme de Dijkstra

L'algorithme se déroule ainsi :

- Initialisation : Création des données initiales
  - Création de **Depart**, chaine de caractères contenant la station de départ
  - Création de **Arrivee**, chaine de caractères contenant la station d'arrivée
  - Initialisation de la liste **Distances** avec **Nb** valeurs infinies (np.inf) (elle contient les distances minimales « Depart → Stations[i] »
  - Initialisation à 0 de la distance de la station **Depart** dans la liste **Distances**. On obtiendra l'indice de **Depart** dans **Stations** avec l'instruction : Stations.index(Depart)
  - Initialisation de la liste **Reste** contenant toutes les stations de **Stations** (copie)
  - Initialisation de la liste **Provenances** contenant **Nb** 0 par exemple (elle contient les « Provenances » d'une station. P=Provenance[i] sera la station de provenance de la station S=Stations[i] qui a permis d'atteindre S avec le plus court chemin depuis le départ)
  - Initialisation de **S**, station courant traitée par l'algorithme, et de son indice **iS**
- Itérations : Tant que **S** n'est pas l'arrivée, qu'il reste des stations dans **Reste** et que Distance[iS] est différente de l'infini (cette condition permet de gagner du temps si **Arrivee** n'est pas accessible depuis **Depart**)
  - **S** ← Station parmi **Reste** ayant la distance minimum dans **Distances** (la première si exæquo)
  - Mise à jour de **Reste** (retirer S)
  - **Voisins** ← Liste des stations de **Reste** voisines de **S** (utilisation de Graphe)
  - Traitement des voisins : Pour chaque voisin V, si la distance Depart→S→V < Depart→V actuellement stockée dans **Distances** :
    - Mise à jour de **Distances** avec cette nouvelle distance Depart→S→V
    - Mise à jour de **Provenances** afin d'indiquer que S est le prédécesseur de V

  *Nous obtenons la distance la plus courte du départ à chaque station dans **Distances** et la liste des prédécesseurs de chaque station dans **Provenances***
- Traitement final :
  - Détermination de la distance à parcourir pour atteindre **Arrivee** depuis **Depart** dans **Distances**
  - Si la solution existe (distance non infinie), remontée du chemin le plus court entre **Depart** et **Arrivee** en exploitant **Provenances** et inversion du résultat

Programmation

Question 1: Créer l'algorithme permettant de déterminer le plus court chemin de F à C en exploitant la description ci-dessus

Vous pourrez vérifier votre algorithme en vous aidant de l'exemple d'exécution suivant : LIEN PDF (à télécharger et ouvrir en mode présentation 😉) ou LIEN VIDEO.

Remarque : On pourrait améliorer l'algorithme en traitant judicieusement le choix de la prochaine station lorsqu'il y a des ex æquo, mais ce n'est pas l'objet de ce TD