SAE S2-02

EXPLORATION ALGORITHMIQUE D'UN PROBLÈME

L'objectif de cette SAÉ est de mettre en pratique les **algorithmes de recherche de plus court chemin** *(Dijkstra, Bellmann Ford, Floyd Warshall…) et de* **visualiser leur exécution pas à pas** *en utilisant une bibliothèque graphique.*

LES ÉTAPES

Semaine du 28 mars : présentation de la SAÉ.
Constitution des groupes pour l'étape 1, au choix des étudiants : travail individuel, par groupe de 2 ou de 3 étudiants.
Les groupes sont susceptibles d'être modifiés dans les étapes suivantes.

- **Étape 1** : Récupération, préparation des données – Date limite : 15 avril
  4 séances prévues dans l'emploi du temps sur la période (en comptant la séance de présentation)
  4 points

- **Étape 2** : Recherche de chemins de poids minimum – Date limite : 21 mai
  5 séances prévues dans l'emploi du temps sur la période
  8 points

- **Étape 3** : Visualisation par une bibliothèque graphique – Date limite : 11 juin
  5 séances prévues dans l'emploi du temps sur la période
  8 points

Remarque : il n'est pas interdit de travailler sur la SAÉ en dehors des séances prévues !

ÉTAPE 1 : PRISE EN MAIN DES DONNÉES – IMPORTATION – PRÉPARATION

ORIGINE DES DONNÉES

https://www.data.gouv.fr/fr/datasets/offre-transport-du-reseau-chronoplus-gtfs/

LE FORMAT GTFS

https://gtfs.org/

General Transit Feed Specification (GTFS, traduction littérale : spécification générale pour les flux relatifs aux transports en commun) est un format informatique standardisé pour communiquer des horaires de transports en commun et les informations géographiques associées (topographie d'un réseau : emplacement des arrêts, tracé des lignes). (Wikipédia)

LES DONNÉES

Les fichiers txt récupérés sont au format CSV (séparateur « , »)

- agency.txt : regroupe les informations sur le service de transport
- calendar.txt et calendar_dates.txt : informations sur les jours de circulations
- routes.txt : nom des lignes, type
- stops.txt : information sur les arrêts
- trips.txt : info sur les trajets
- stops_times.txt : pour chaque ligne, heure de passage aux points d'arrêt.

Pour cette SAE nous ne nous intéressons aux distances et non à la durée des trajets. Nous avons donc juste besoin d'informations géographiques sur les arrêts : latitudes, longitudes et voisins.

FICHIERS TRAITÉS

Vous n'aurez pas à manipuler les tables précédentes, mais vous pouvez les charger et les consulter par curiosité. Les fichiers que nous traiterons sont deux fichiers textes qui ont été créés à partir de ces tables, leurs contenus peuvent, par exemple, être visualisés dans le Bloc-notes. Vous choisirez d'utiliser celui de votre choix

- donneesbus.json : 465 arrêts
  {"NOVE": [43.538772, -1.452244, ["CANT", "PALI"]], "AVRI": [43.50246, -1.4624575, ["PLGA", "ETIE"]],…}

- donneesbus.csv : 465 arrêts
  arret;lattitude;longitude;listesucc
  NOVE;43,538772;-1,452244;['CANT', 'PALI']
  AVRI;43,50246;-1,4624575;['PLGA', 'ETIE']
  7PUI;43,4721865;-1,526852;['HOUN', 'AEROPO']…

> *Remarque sur le format **JSON** : même si ce format vous est inconnu, il peut être intéressant que vous vous y intéressiez. De même qu'il est assez naturel d'importer un CSV dans une feuille d'un tableur ou dans un DataFrame, la structure d'un **format JSON** peut assez facilement être rapprochée de celle d'un **dictionnaire Python**. Ce qui peut nous intéresser ici.*

TRAVAIL À RÉALISER POUR L'ÉTAPE 1 (AVANT LE 15 AVRIL)

A. Importer les données dans un dictionnaire **donneesbus** : la clé est le nom de l'arrêt et la valeur de la clé contiendra une liste avec la latitude, la longitude et la liste des arrêts voisins.

B. Créer une liste **noms_arrets** contenant les noms des arrêts

C. Créer les fonctions suivantes qui pourrons (ou non !) se révéler utiles pour les futures manipulations.
   - **nom(ind)** : permet d'associer le nom du sommet à son indice **ind** dans **noms_arrets**
   - **indice_som(nom_som)** : permet d'associer l'indice de l'arrêt à son nom **nom_som**
   - **latitude(nom_som)** : renvoie la latitude d'un arrêt nommé **nom_som**
   - **longitude(nom_som)** : renvoie la longitude d'un arrêt nommé **nom_som**
   - **voisin(nom_som) :** renvoie la liste des voisins d'un arrêt nommé **nom_som**

D. Le réseau de bus peut, bien sûr, être modélisé par un graphe.
   Représenter ce graphe sous Python :
   - La liste d'adjacence par un dictionnaire ***dic_bus***
   - La matrice d'adjacence par une liste de liste ***mat_bus***

E. Distance.
   Créer deux fonctions :
   - ***distarrets(arret1,arret2)*** renvoyant la distance à vol d'oiseau entre les arrêts *arret1* et *arret2.*
   - ***distarc(arret1,arret2)*** renvoyant la distance à vol d'oiseau entre les les arrêts *arret1* et *arret2* quand l'arc ***(arret1,arret2)*** existe, et sinon la valeur qui vous paraît la plus cohérente.

UNE FONCTION UTILE : DISTANCEGPS

```
from math import sin, cos, acos, pi
import numpy as np
##########################################################
# calcul de la distance entre deux points A et B dont   #
# on connait la lattitude et la longitude               #
##########################################################
def distanceGPS(latA,latB,longA,longB):
    # Conversions des latitudes en radians
    ltA=latA/180*pi
    ltB=latB/180*pi
    loA=longA/180*pi
    loB=longB/180*pi
    # Rayon de la terre en mètres (sphère IAG-GRS80)
    RT = 6378137
    # angle en radians entre les 2 points
    S = acos(round(sin(ltA)*sin(ltB) + cos(ltA)*cos(ltB)*cos(abs(loB-loA)),14))
    # distance entre les 2 points, comptée sur un arc de grand cercle
    return S*RT
```

F. Le réseau de bus peut aussi être modélisé par un graphe ***pondéré***, qui pourra être utilisé par les algorithmes de plus courts chemins.
   Représenter ce graphe sous Python par une matrice des poids (liste de listes), ***poids_bus*** prenant en compte à la fois l'existence des arcs et leur poids.
   Sachant que notre objectif est de chercher les chemins de poids minimum, on affectera aux arcs qui n'existent pas dans le graphe une valeur adaptée à cette future recherche…

ÉVALUATION : COMPTE-RENDU ET ENTRETIEN

Dès que vous avez terminé de créer les fonctions **distarrets, distarc, nom(ind)**, **indice_som(nomsom)** et les objets **dic_bus, mat_bus** et **poids_bus.**

- → Vous déposez un compte-rendu sur ELearn qui contiendra :
  - les codes bien commentés de ces fonctions,
  - une description des principales difficultés rencontrées.
- → Vous prenez Rdv. Des créneaux de 30 min seront proposés les semaines du 11 avril et du 2 mai
  L'entretien portera sur :
  - Le code Python (des questions pourront être posées, collectivement ou individuellement aux membres du groupe)
  - Comment envisagez-vous la suite ?
  - Les compétences mises en œuvre.

Si le 15 avril vous n'avez pas terminé la programmation des différentes fonctions et objets, vous déposez quand même votre travail sur Elearn avec :
- les codes commentés que vous avez produits à cette date,
- les principales difficultés rencontrées.

Vous prenez Rdv pour la semaine du 2 mai

Le compte-rendu sera noté sur 4 points, cette note pouvant être corrigée, à la hausse ou à la baisse, individuellement ou collectivement, selon la **qualité des réponses fournies lors de l'entretien.**

ÉTAPE 2 : RECHERCHE DE CHEMINS DE POIDS MINIMUM

Première partie de l'étape 2 (sur 6 points)

Créer des fonctions « plus court chemin » entre deux arrêts en utilisant les différents algorithmes vus en Graphes (Dijkstra, Bellmann Ford, Floyd Warshall).

- Dijkstra(arret_dep,arret_arriv)
- Belmann(arret_dep,arret_arriv)
- FloydWarshall(arret_dep,arret_arriv)

Ces fonctions prennent en paramètres les deux arrêts et renvoient **le plus court chemin**, sous forme de la liste des arrêts parcourus ainsi, que **la distance minimum**.
Vous pouvez traiter les algorithmes dans l'ordre de votre choix (Dijkstra n'est pas le plus simple à programmer)

Deuxième partie de l'étape 2 (sur 2 points)

- Créer une fonction « plus court chemin entre deux arrêts » en utilisant une heuristique dans un algorithme A*. Les étudiants devront chercher par eux-mêmes des informations sur cet algorithme.
- Comparer l'efficacité des différents algorithmes

La validation de chaque algorithme de la **première partie** se fera pendant les séances (les étudiants montreront et commenteront leur code en direct).

La **deuxième partie** (il est possible que certains groupes n'y arrivent pas) donnera lieu à un compte rendu déposé sur Elearn avant le 21 mai.