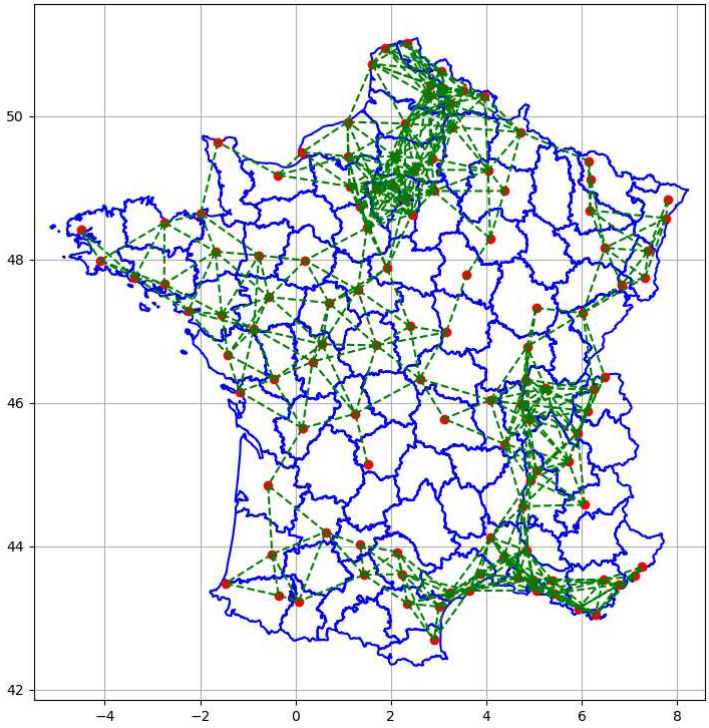
TP : Floyd-Warshall : Le plus court chemin

Mise en situation

On se propose dans ce TP d'utiliser l'algorithme de Floyd-Warshall afin de déterminer le plus court chemin entre deux villes de France. Ce chemin se fait par des lignes droites entre deux grandes villes de France dont la population est supérieure à 30 000 habitants et qui sont distantes de 120 km au plus l'une de l'autre.

Un premier TP a permis de déterminer et tracer le graphe correspondant aux villes de 30 000 habitants au moins et distantes de 120 km au plus l'une de l'autre. Voir carte ci-contre.

Tout cela peut-être obtenu avec le code python fourni dans le fichier « Itineraire grandes villes – Floyd-Warshall.py ». Ce code python permet d'obtenir le graphe qui lui-même permet de faire le tracé ci-contre.



Objectif

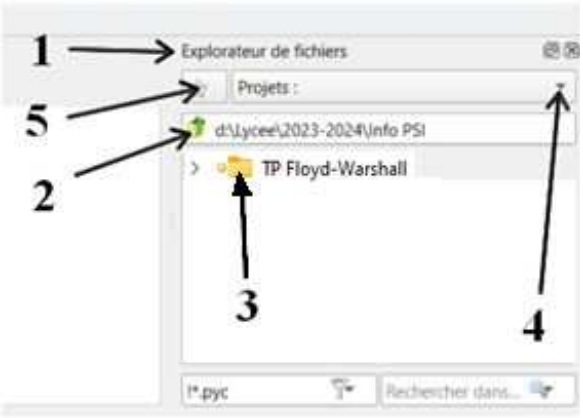
L'objectif est de créer les fonctions qui en indiquant, les noms de deux villes, la population minimale des villes françaises (par exemple 30 000 habitants) qui peuvent accueillir les étapes d'un parcours entre les deux villes et la distance maximale (par exemple 120 km) entre les villes étapes ; permettront d'obtenir l'itinéraire le plus court et de le tracer sur cette carte.

Récupérations des données

Copier et enregistrer dans votre dossier personnel le dossier « Itinéraire grandes villes - Floyd-Warshall » qui contient les deux fichiers : « dots.db » et « Itineraire grandes villes - Floyd-Warshall.py». Puis ouvrir avec Pyzo le fichier « Itineraire grandes villes - Floyd-Warshall.py».

Procédure pour rendre courant dans le shell un dossier :

- ☞ (1) Visualiser la fenêtre de « l'explorateur de fichier »
- ☞ (2) Sélectionner le dossier contenant le dossier en question.
- ☞ (3) Clic droit sur le dossier en question puis sélectionner « Ajouter ce répertoire à la liste des projets »
- ☞ (4) Sélectionner le dossier en question dans la liste des projets
- ☞ (5) Clic gauche puis sélectionner « Aller dans ce dossier»
- ☞ (6) Exécuter le code (Touche F5)



➢ Tester le code en tapant la commande **`Trace_graphe(30,120)`** puis **`plt.show()`**, cela doit permettre l'affichage de la carte ci-dessus.

Si l'affichage ne se fait pas correctement (superposition des cartes des départements) mettre les lignes 46 et 47 en commentaire.

Codage de l'algorithme de Floyd-Warshall

➢ Le graphe obtenu par la fonction « Constr_graphe(30,120) » renvoie un couple de dictionnaires. Taper les commandes **`Graphe,Dvilles=Constr_graphe(30,120)`** , puis **`Dvilles['BREST']`** , et enfin **`Graphe['BREST']`** . Ce que renvoient ces commandes doit vous permettre (avec vos connaissances en géographie) de comprendre la structure des deux dictionnaires renvoyés par la fonction « Constr_graphe(30,120) »

Le premier dictionnaire renvoyé par la commande « Constr_graphe(30,120) » est un graphe d'adjacence. Appelons le « Graphe_adj ». Ce graphe est décrit par un dictionnaire dont les clés sont toutes les villes d'au moins 30 000 habitants. Les valeurs sont des dictionnaires dont les clés sont uniquement les villes voisines de la ville clé (maximum 120 km). Les deux dictionnaires ont la même dimension. Notons « n » cette dimension. Le second dictionnaire est une collection de toutes les villes de plus de 30 000 habitants avec les coordonnées GPS et leur population.

Nous allons construire une matrice d'adjacence de ce même graphe (matrice n×n). Appelons la « Mat_adj ». Cette matrice sera décrite par un dictionnaire de dictionnaire dont les clés (clésA) sont toutes les villes d'au moins 30 000 habitants. Les valeurs seront des dictionnaires dont les clés (clésB) sont aussi toutes les villes d'au moins 30 000 habitants et les valeur un couple (distance,suivante).

☞ La variable du couple « suivante » sera au départ la même chaine de caractère que la clé (clésB).

☞ La variable du couple « distance » sera un float. Ce réel est :

⇒ La distance entre les villes cléA et cléB, si les villes « cléA », « cléB » sont distantes de mois de « autonomie » km, autrement dit si cléB est une des clés du dictionnaire « Graphe_adj[cleA] »

⇒ Une distance nulle si cléA = clé B

⇒ Une distance infinie (« inf » renvoyé la commande « np.inf » de Numpy), si les villes « cléA », « cléB » sont distantes de plus de « autonomie » km, C'est-à-dire si cléB n'est pas une des clés du dictionnaire « Graphe_adj[cleA] ».

➢ **Question 1** Ecrire une fonction **`Constr_mat(Graphe_adj)`** » qui prend en argument un graphe d'adjacence et qui renvoie la matrice d'adjacence (**`Mat_adj`**), sous la forme d'un dictionnaire de dictionnaires, correspondant au graphe d'adjacence renvoyé par la commande **`Constr_graphe(Pop_min,autonomie)`**

Après avoir écrit ce code et l'avoir exécuté, taper successivement dans le shell les commandes **`Graphe,Dvilles=Constr_graphe(30,120)`** , puis **`Mat_adj=Constr_mat(Graphe)`** , et enfin **`Mat_adj['BREST']['LORIENT']`** cela doit vous retourner : **`(111.33592697619756, 'LORIENT')`** car la ville de Lorient est à 111 km de Brest ce qui est inférieur à 120 km.

En tapant ensuite **`Mat_adj['BREST']['VANNES']`** cela doit retourner : **`(inf, 'VANNES')`** car Vannes est à plus de 120 km de Brest. Et donc à une distance infinie si on ne peut faire que des étapes de 120 km maximum.

➢ **Question 2** La fonction précédente renvoie une matrice donnant la distance la plus courte entre deux villes et la direction à prendre pour rejoindre ces deux villes sans étapes intermédiaire. Nous allons maintenant écrire une fonction qui permet de d'ajouter une ville permettant une étape intermédiaire.

Ecrire une fonction **`Ajout_ville(M,Ajout)`** qui prend en argument une matrice d'adjacence (dictionnaire de dictionnaires) et une ville (chaîne de caractère) et qui retourne une nouvelle matrice d'adjacence pour laquelle on a ajouté la possibilité de faire étape à la ville **`Ajout`**.

Après avoir écrit ce code et l'avoir exécuté, taper successivement dans le shell les commandes **`Mat_adj= Ajout_ville(Mat_adj,'LORIENT')`** puis **`Mat_adj['BREST']['VANNES']`** cela doit retourner **`(159.0655081587547, 'LORIENT')`** car Vannes est à 159 km de Brest et que l'on peut aller de Brest à Vannes en faisant étape à Lorient et qu'enfin pour cela il faut aller vers Lorient.

➢ **Question 3** L'algorithme de Floyd-Warshall consiste à ajouter une à une la possibilité de passer par une étape intermédiaire et de retenir le chemin le plus court (La distance et la direction à prendre)

Ecrire une fonction **`Floyd_Warshall(Graphe)`** qui prend en argument un graphe d'adjacence (Celui obtenue par la commande **`Graphe,Dvilles=Constr_graphe(30,120)`**) et qui renvoie une matrice d'adjacence (**`MFW`**) à laquelle on a appliqué l'algorithme de Floyd-Warshall en ajoutant une a une toutes les villes correspondant aux clés de la 1ière matrice d'adjacence renvoyée par la fonction **`Constr_mat(Graphe_adj)`**.

Cette fonction utilisera les fonctions **`Constr_mat(Graphe_adj)`** et **`Ajout_ville(M,Ajout)`**.

Après avoir écrit ce code et l'avoir exécuté, taper successivement dans le shell les commandes **`Graphe,Dvilles=Constr_graphe(30,120)`** , puis **`MFW=Floyd_Warshall(Graphe)`** , et enfin **`MFW ['BREST']['NANTES']`** cela doit vous retourner : **`(261.35043233309915, 'LORIENT')`** car la ville de Nantes est à 261 km de Brest par étapes de moins de 120 km et qu'enfin pour cela il faut aller vers Lorient.

En tapant ensuite dans le shell la commande : **`MFW['BREST']['CAEN']`** cela doit vous retourner : **`(448.2001559976825, 'QUIMPER')`** car la ville de Caen est à 448 km de Brest par étapes de moins de 120 km et qu'enfin pour cela il faut aller vers Quimper.

Bien sur la direction à prendre et la distance obtenue sont celles du plus court itinéraire.

Tracé de l'itinairaire le plus court

➢ **Question 4** Ecrire une fonction **`Itineraire(Pop_min,autonomie,Depart,Arrivee)`** qui prend en argument deux nombres et deux chaines de caractères (deux noms de villes de plus de **`Pop_min`** habitant) et qui renvoie un triplet :

- La distance du plus court chemin entre les villes Départ et Arrivée (en passant par des villes de plus de **`Pop_min`** habitants avec des étapes de moins de **`autonomie`** km)
- L'itinéraire emprunté pour rejoindre les 2 villes : La liste des noms des villes (du Départ à l'Arrivée) étapes de ce plus court chemin.
- Le dictionnaire des villes de plus de **`Pop_min`** habitants tel que renvoyé par la fonction **`Constr_graphe(Pop_min,autonomie`**)

Cette fonction reprendra les étapes suivantes :

- Construction du graphe d'adjacence (enregistrement dans la variable **`Graphe`**) et du dictionnaire des villes de plus de **`Pop_min`** habitants (enregistrement dans la variable **`Dvilles`**)
- Construction de la matrice d'adjacence après application de l'algorithme de Floyd-Warshall à partir du graphe d'adjacence **`Graphe`** (Enregistrement dans la variable **`MFW`**).
- Enregistrement de la distance de l'itinéraire le plus court de la ville **`Départ`** à la ville **`Arrivée`** dans une variable **`distance`**.
- Construction de la liste des villes étapes de la ville **`Départ`** à la ville **`Arrivée`** (Enregistrement dans la variable **`Etapes`** des chaines de caractères que sont les noms des villes étapes)
- Retour du triplet **`(distance,Etapes,Dvilles)`**

➢ Tester votre fonction **`Itineraire(Pop_min,autonomie,Depart,Arrivee)`** en tapant les commandes **`Dist,Iti,Villes=Itineraire(30,120,'BREST','CAEN')`** puis **`print(Dist,Iti)`** cela doit renvoyer : **`448.2001559976825 ['BREST', 'QUIMPER', 'SAINT-BRIEUC', 'SAINT-MALO', 'CHERBOURG-OCTEVILLE', 'CAEN']`**

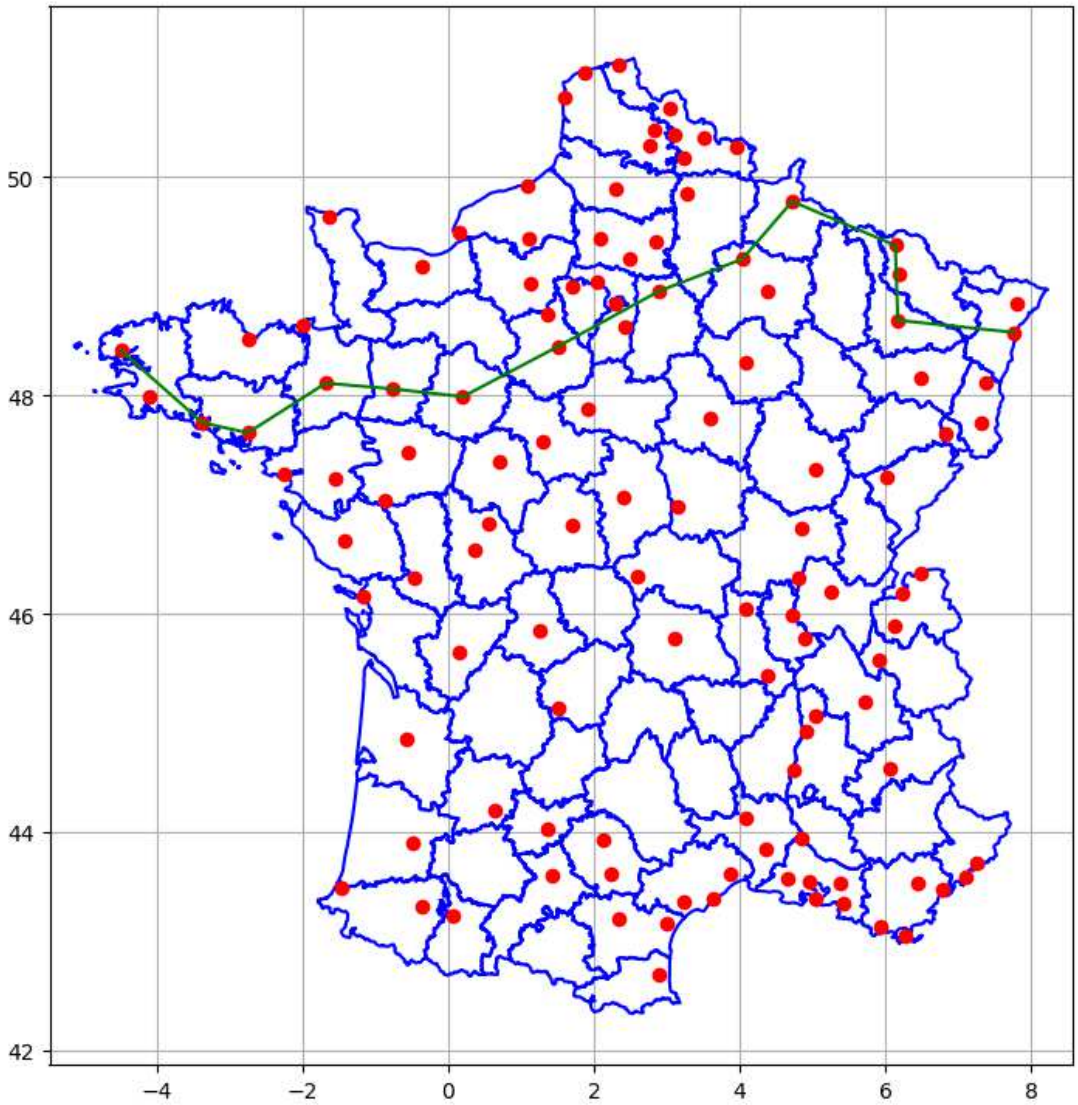
Et **`Dist,Iti,Villes=Itineraire(30,120,'BREST','NICE')`** et enfin **`print(Dist,Iti)`** cela doit renvoyer : **`1167.8240813100008 ['BREST', 'LORIENT', 'VANNES', 'NANTES', 'CHOLET', 'CHATELLERAULT', 'CHATEAUROUX', 'MONTLUCON', 'CLERMONT-FERRAND', 'SAINT-ETIENNE','MONTELIMAR','AVIGNON','AIX-EN-PROVENCE','DRAGUIGNAN','NICE']`**

➢ **Question 5** Ecrire une fonction **`Trace_itineraire(Pop_min,autonomie,Depart,Arrivee)`** qui prend en argument deux nombres et deux chaines de caractères (deux noms de villes de plus de **`Pop_min`** habitants) et qui trace sur la carte de France l'itinéraire renvoyé

Cette fonction reprendra les étapes suivantes :

- ☞ Application de la commande **`Itineraire(Pop_min,autonomie,Depart,Arrivee)`** et enregistrement des valeurs retournées dans les variable **`Distance`**, **`Itineraire`**, **`Dvilles`**
- ☞ Tracé de la carte de France avec la commande appelant la fonction **`carteFrance()`**
- ☞ Tracé de l'itinéraire sur la carte de France en utilisant les variables **`Itineraire`** et **`Dvilles`**.
- ☞ Affichage dans le Shell de la distance et de l'itinéraire (avec la commande **`print()`**)

➢ Tester votre fonction **`Trace_itineraire(Pop_min,autonomie,Depart,Arrivee)`** en tapant les commandes **`Trace_itineraire(30,120,'BREST','STRASBOURG')`** puis **`plt.show()`** . Cela doit vous affiche la carte ci-dessous :



➢ **Question 6** Tester les commandes >>> **`Trace_itineraire(30,120,'BREST','STRASBOURG')`** puis **`Trace_itineraire(30,90,'BREST','STRASBOURG')`**, et ensuite **`Trace_itineraire(30,80,'BREST','STRASBOURG')`** , et enfin **`plt.show()`**.

Pourquoi un des trois tracés est-il abhérant ? Comment peut-on modifier le code pour éviter d'afficher un tel tracé aberrant ? Modifier votre code pour cela.