Étape 1 - Garica et Planche

Importation des bibliothèques disponnibles.

In [ ]:
```
import json # Pour gérer le fichier donneesbus.json
from math import sin, cos, acos, pi
```

Ouverture du fichier donneesbus.json et création du dictionnaire.

In [ ]:
```
with open("Fichiers/donneesBus.json") as fic_donnees_bus:
    donneesBus = json.load(fic_donnees_bus)
```

Travail à réaliser.

In [ ]:
```
# Création d'une liste des noms des arrêts.
noms_arrets = list(donneesBus.keys())


def nom(ind: int) -> str:
    """
    Renvoie le nom de l'arret à l'indice ind
    :param ind: Indice de l'arrêt
    :type ind: int
    :return: nom de l'arrêt
    :rtype: str
    """
    return noms_arrets[ind]


def indice_som(nom_som: str) -> int:
    """
    Renvoie l'indice de l'arrêt à partir de son nom
    :param nom_som: Nom de l'arrêt
    :type nom_som: str
    :return: indice de l'arrêt
    :rtype: int
    """
    return noms_arrets.index(nom_som)


def lattitude(nom_som: str) -> float:
    """
    Renvoie la lattitude de l'arrêt à partir de son nom
    :param nom_som: Nom de l'arrêt
    :type nom_som: str
    :return: lattitude de l'arrêt
    :rtype: float
    """
    return donneesBus[nom_som][0]


def longitude(nom_som: str) -> float:
    """
    Renvoie la longitude de l'arrêt à partir de son nom
    :param nom_som: Nom de l'arrêt
    :type nom_som: str
    :return: longitude de l'arrêt
    :rtype: float
    """
    return donneesBus[nom_som][1]


def voisin(nom_som: str) -> list:
    """
    Renvoie la liste des arrêts voisins à partir de son nom
    :param nom_som: Nom de l'arrêt
    :type nom_som: str
    :return: liste des arrêts voisins
    :rtype: list
    """
    return donneesBus[nom_som][2]


# Création de la liste d'adjacence sous forme d'une liste.
mat_bus = [
    [1 if nom_som in voisin(nom_som1) else 0 for nom_som in noms_arrets] for nom_som1 in noms_arrets
]

# Création de la liste d'adjacence sous forme d'un dictionnaire.
dict_bus = {
    nom_arret: voisin(nom_arret) for nom_arret in noms_arrets
}


def distanceGPS(latA: float, latB: float, longA: float, longB: float) -> float:
    """
    Retourne la distance en mètres entre deux points GPS.
    :param latA: lattitude du premier point
    :param latB: lattitude du deuxième point
    :param longA: longitude du premier point
    :param longB: longitude du deuxième point
    :return:
    """
    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


def distance_arrets(arret1: str, arret2: str) -> float:
    """
    Renvoie la distance à vol d'oiseau entre deux arrêts.
    :param arret1: nom de l'arrêt 1
    :type arret1: str
    :param arret2: nom de l'arrêt 2
    :type arret2: str
    :return: distance entre les deux arrêts
    :rtype: float
    """
    return distanceGPS(
        lattitude(arret1),
        lattitude(arret2),
        longitude(arret1),
        longitude(arret2),
    )


def distance_arc(arret1: str, arret2: str) -> float:
    """
    Renvoie la distance à vol d'oiseau entre deux arrêts si ils sont .
    :param arret1: nom de l'arrêt 1
    :type arret1: str
    :param arret2: nom de l'arrêt 2
    :type arret2: str
    :return: distance entre les deux arrêts
    :rtype: float
    """
    return distanceGPS(
        lattitude(arret1),
        lattitude(arret2),
        longitude(arret1),
        longitude(arret2),
    ) if arret2 in voisin(arret1) else float("inf")


# Création de la matrice des poids sous forme d'une liste.
poids_bus = [
    [distance_arc(nom_som1, nom_som2) for nom_som2 in noms_arrets] for nom_som1 in noms_arrets
]
```

Nous n'avons pas rencontré de difficultés particulières.