```python
# -*- coding: utf-8 -*-

"""
Auteurs :
    Chabanat Matis
    Le Menn Arthur
    TD1 | TP1
"""

import json
from math import sin, cos, acos, pi
from graphics import *

#Import du fichier .json
donneesbus = open("donneesbus.json")

#Transformation du fichier json en dictionnaire
donneesbus = json.load(donneesbus)

#Création de la liste de toutes les clé du dictionnaire
noms_arrets = list(donneesbus.keys())

#Renvoie le nom de l'arrêt dans noms_arrets à l'indice donné par l'user
def nom(ind):
    return noms_arrets[ind]

#Renvoie l'indice dans la liste noms_arrets correspondant au nom donné par l'user
def indic_som(nom_som):
    for i in range(len(noms_arrets)):
        if noms_arrets[i] == nom_som:
            return i

#Renvoie la latitude de nom_som
def latitude(nom_som):
    return donneesbus[nom_som][0]

#Renvoie la longitude de nom_som
def longitude(nom_som):
    return donneesbus[nom_som][1]

#renvoie la liste des voisins de nom_som
def voisin(nom_som):
    return donneesbus[nom_som][2]

#Renvoie la matrice d'adjacence du dictionnaire
def mat_bus(dico):
    mat_bus = []
    nomKey = list(dico.keys())
    for i in range(len(nomKey)):
        ligne = []
        for y in range(len(nomKey)):
            if nomKey[y] in voisin(nomKey[i]):
                ligne.append(1)
            else:
                ligne.append(0)
        mat_bus.append(ligne)
    return mat_bus

#Renvoie le graphe du dictionnaire en paramètre sous forme de dictionnaire
def dic_bus(dico):
    dic_bus = {}
    nomKey = list(dico.keys())
    for i in range(len(nomKey)):
        dic_bus[nomKey[i]] = voisin(nomKey[i])
    return dic_bus

#Renvoie la distance en mètre à vol d'oiseau de deux coordonnées gps
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

#Renvoie la distance à vol d'oiseau entre deux sommets voisins ou pas
def distarrets(arret1, arret2):
    lat1 = latitude(arret1)
    lat2 = latitude(arret2)
    long1 = longitude(arret1)
    long2 = longitude(arret2)
    return distanceGPS(lat1, lat2, long1, long2)

#Renvoie le poids de l'arc entre deux sommets
def distArc(arret1, arret2):
    if arret1 in voisin(arret2):
        return distarrets(arret1, arret2)
    else:
        return float ('inf')

#Renvoie le graphe pondéré du dictionnaire en paramètre sous forme de matrice
def poids_bus(dico):
    matPoids = []
    nomKey = list(dico.keys())
    for i in range(len(nomKey)):
        ligne = []
        for y in range(len(nomKey)):
            ligne.append(distArc(nomKey[i], nomKey[y]))
        matPoids.append(ligne)
    return matPoids




def dijkstra(depart, arrive):
    #Déclaration des tableaux
    G = poids_bus(donneesbus)
    sommetATraite = []
    dist = []
    pred = []
    tab = []
    sommetTraite = indic_som(depart)

    #Initialisation tab des tableaux
    for i in range(len(G)):
        pred.append(float("inf"))
        dist.append(float("inf"))
        tab.append(float("inf"))
        sommetATraite.append(i)

    #On traite le sommet de départ
    sommetATraite.remove(sommetTraite)
    pred[sommetTraite] = sommetTraite #Comme c'est l'arrêt de départ son
    prédecesseur est lui même
    dist[sommetTraite] = 0

    #Début de la boucle
    while sommetATraite != []:
        #Réinitialisation des variable
        for i in range(len(G)):
            if i in sommetATraite:
                tab[i] = G[sommetTraite][i]
            else:
                tab[i] = float('inf')

        #Relachement
        for i in range(len(tab)):
            if tab[i] != float('inf'):
                if dist[i] > (dist[sommetTraite]+tab[i]):
```

```
144                     pred[i] = sommetTraite
145                     dist[i] = dist[sommetTraite]+tab[i]
146
147         #Reinitialisation
148         for i in range(len(G)):
149             if i in sommetATraite:
150                 tab[i] = dist[i]
151             else:
152                 tab[i] = float('inf')
153
154         #On établit le prochain arrêt à traiter
155         sommetTraite = minTableau(tab)[1]
156         sommetATraite.remove(sommetTraite)
157
158         #On verifie qu'on ne soit pas arrivé au sommet
159         if sommetTraite == indic_som(arrive):
160             break
161
162     #On remonte les chemin
163     result = []
164     sommet = indic_som(arrive)
165     while sommet != indic_som(depart):
166         result.append(nom(sommet))
167         sommet = pred[sommet]
168     result.append(depart)
169     return inverseTab(result)
170
171
172
173
174 def belmann(arret_dep,arret_arriv):
175     #init des dist et pred
176     dicoDistPred = {som: [float('inf'), None] for som in noms_arrets}
177     #On init le poids de l'arret de départ a 0
178     dicoDistPred[arret_dep][0]=0
179
180     #fonction pour relacher un arc
181     def relachement(depart,arrivee):
182         if dicoDistPred[depart][0] + distArc(depart, arrivee) <
        dicoDistPred[arrivee][0]:
183             dicoDistPred[arrivee][0] = dicoDistPred[depart][0] + distArc(depart,
            arrivee)
184             dicoDistPred[arrivee][1] = depart
185             #s'il y a un amelioration, on retourne true pour dire qu'un relachement
            a été fait
186             #et que l'on doit etudier les autres arrets
187             return True
188         else:
189             return False
190
191
192     #début boucle : on considère qu'aucun changement a été réalisé
193     change=False
194     #etude de tous les arrets
195     for i in range(0, len(noms_arrets) - 1):
196         for arret1 in noms_arrets:
197             #2e boucle pour etudier les voisins de l'arret1
198             for arret2 in voisin(arret1):
199                 #on relache le sommet
200                 if change and not relachement(arret1, arret2):
201                     break
202                 change = relachement(arret1, arret2)
203
204     #Après avoir tout relaché, on retrouve le chemin à réaliser pour faire
    arret1->arret2
205     lastArret = dicoDistPred[arret_arriv][1]
206     cheminArrets = [lastArret]
207     while lastArret != arret_dep:
208         lastArret=dicoDistPred[lastArret][1]
209         cheminArrets = [lastArret]+cheminArrets
210
211     #On place a la fin du chemin le nom de l'arret d'arrivée
```

```
212         cheminArrets.append(arret_arriv)
213 
214         #la fonction renvoie le chemin et la distance a parcourir (round permet
            d'arrondir la distance)
215         return cheminArrets,round(dicoDistPred[arret_arriv][0])
216 
217 
218 
219 
220     def floydWarshall(depart, arrivee):
221         #Initialisation
222         indiceDepart = indic_som(depart)
223         indiceArrivee = indic_som(arrivee)
224         matrice = poids_bus(donneesbus)
225         n = len(matrice)
226         #On remplit les tableaux distance et pred
227         distances, pred = [[0 for _ in range(n)] for _ in range(n)], [[0 for _ in
            range(n)] for _ in range(n)]
228         chemin = []
229 
230         #On place les poids et les predécesseurs déjà existant
231         for i in range(n):
232             for j in range(n):
233                 distances[i][j] = matrice[i][j]
234                 pred[i][j] = j
235         #boucles de Floyd Warshall
236         for k in range(n):
237             for i in range(n):
238                 for j in range(n):
239                     if distances[i][j] > distances[i][k] + distances[k][j]:
240                         distances[i][j] = distances[i][k] + distances[k][j]
241                         pred[i][j] = pred[i][k]
242 
243         #Recherche du chemin depart --> arrivée
244         chemin.append(depart)
245         position = indiceDepart
246         while position != indiceArrivee:
247             position = pred[position][indiceArrivee]
248             chemin.append(nom(position))
249         return chemin
250 
```