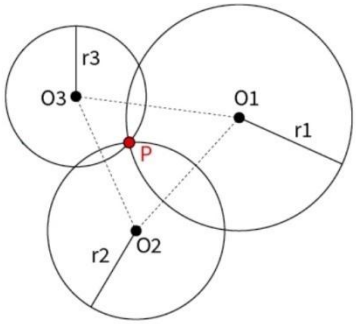
Nom : Prénom :	THEME 5 Localisation, cartographie et mobilité	LCM-TP3 1/3	
Date :	Principe de la géolocalisation et les itinéraires		

1- Le principe de la tringularisation



Dans la **triangulation**, on détermine sa position (P) par rapport à trois points de repère (au moins), de position connue. Pour cela, il faut déterminer précisément à quelle distance on se trouve de chacun de ces points de repère.

Si on se trouve à une distance d1 du repère O1, alors on est quelque part sur un cercle de rayon r1 autour de O1. Si on se trouve à une distance d2 du repère O2, alors on est sur un cercle de rayon r2 autour de O2.

On est alors à l'intersection des deux cercles. Pour choisir à quelle intersection, il faut ajouter un troisième cercle de rayon r3, distance à laquelle on se trouve du repère O3.

Application : Une situation pour appliquer la géolocalisation : Une personne cherche à se géolocaliser dans les alentours de notre établissement. Sa montre sonne 15h, mais elle entend l'horloge l'Horloge de la mairie avec retard de 1s, la sonnerie de l'établissement avec un retard de 1.6s et l'horloge de l'église avec un retard de 2.3 s. Cela lui permet de se géolocaliser.

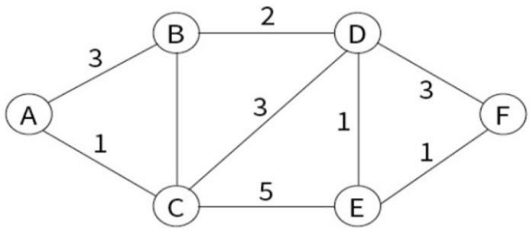
Travail demandé : Proposer une méthode pour aider un personne à se géolocaliser dans les alentours de notre établissement. se localiser sur la carte sachant que la vitesse de propagation du son dans l'air est approximativement de 340m/s.

Nom : Prénom :	THEME 5 Localisation, cartographie et mobilité	LCM-TP3 2/3	
Date :	Principe de la géolocalisation et les itinéraires		

2- Les itinéraires

Deux points sur une carte et plusieurs routes possibles

Les différents itinéraires possibles pourront se représenter sous la forme d'un graphe, dont les nœuds seront les différentes villes ou des intersections possibles. Les arrêtes entre les nœuds correspondent aux routes entre les villes. Elles sont pondérées par exemple par une distance ou une durée du trajet.



Itinéraires (6 sommets de A à E)

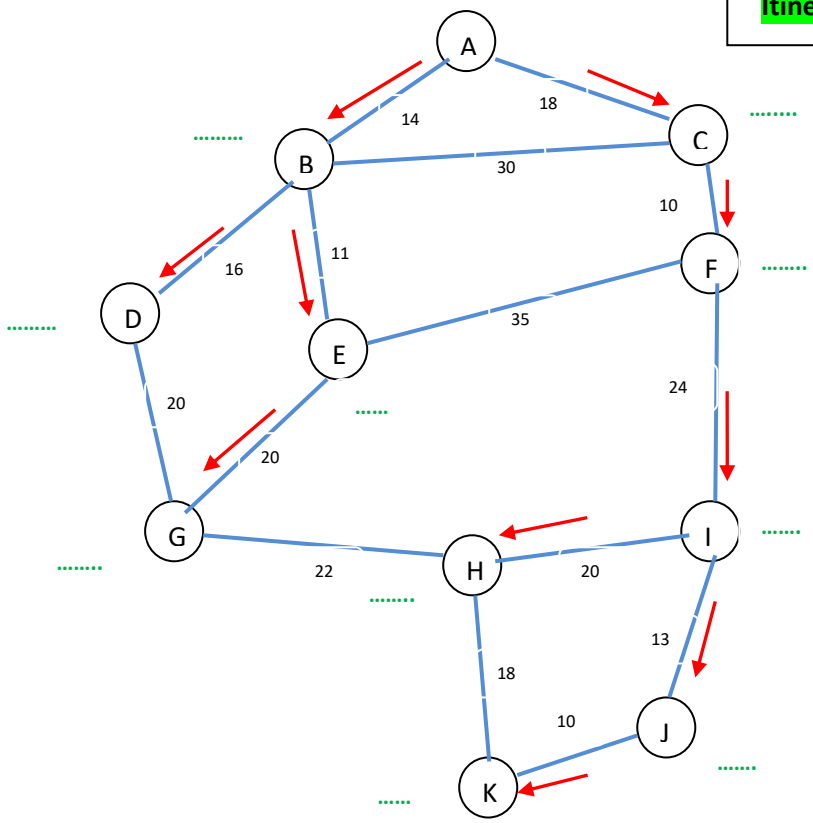
Algorithme de Djikstra

Chaque sommet du graphe correspond à une ville. L'un des sommets est défini comme point de départ (sommet-source). Chaque arête est pondérée par une valeur correspondant à la distance au nœud précédent. Les nœuds non adjacents sont pondérés provisoirement d'une valeur infinie pour rejetés par la suite.
A chaque itération de l'algorithme, on sélectionne le nœud pour lequel la valeur est la plus petite. L'avancée de l'algorithme est souvent présentée sous forme d'un tableau

Travail demandé : Trouver l'itinéraire le plus court pour aller de la ville située au point A au point K. Ecrire les distances calculées entre les sommets sur les pointillés.

Les flèches correspondent aux chemins autorisés à prendre et le sens de la circulation.

Itinéraires (11 sommets de A à K)



Quel est donc l'itinéraire le plus court pour rejoindre le point K en partant du point A ?

Réponse : ……

Nom : Prénom :	THEME 5 Localisation, cartographie et mobilité	LCM-TP3 3/3	
Date :	Principe de la géolocalisation et les itinéraires		

3- Application de l'algorithme de Djikstra en utilisant un script Python

Travail demandé :

3.1) Charger le script python « djikstra_élèves.py » correspondant au schéma de la page 2 pour 6 sommets de A à F

```
nb_sommets = 6 # Nombre de sommets

voisins_A = [0,3,1,1000,1000,1000]
voisins_B = [3,0,1,2,1000,1000]
voisins_C = [1,1,0,3,5,1000]
voisins_D = [1000,2,3,0,1,3]
voisins_E =  [1000,1000,5,1,0,1]
voisins_F = [1000,1000,1000,3,1,0]
m_voisins = [voisins_A,voisins_B,voisins_C,voisins_D,voisins_E,voisins_F]
""" ********** calcul du plus court chemin **********"""
etapes=[[1000,'non determine','non'] for i in range(nb_sommets)]
sommet_depart=0
dist_interm=0
while sommet_depart != nb_sommets-1:
    minimum=1000 #
    for n in range(1,nb_sommets):
        if etapes[n][2]=='non':
            dist=m_voisins[sommet_depart][n]
            dist_totale=dist_interm+dist
            if dist != 0 and dist_totale < etapes[n][0]:
                etapes[n][0]=dist_totale
                etapes[n][1]=sommet_depart
            if etapes[n][0]<minimum:
                minimum=etapes[n][0]
                psommet_depart=n
    sommet_depart=psommet_depart
    etapes[sommet_depart][2]='oui'
    dist_interm=etapes[sommet_depart][0]
    for i in range(1,nb_sommets):
        print(etapes[i])
    print("\n")
""" ********** reconstitution du plus court chemin, en partant du sommet d'arrivée **********"""
chemin=[]
sommet=nb_sommets-1
chemin.append(sommet)
while sommet != 0:
    sommet=etapes[sommet][1]
    chemin.append(sommet)
chemin.reverse() #inversion du chemin pour affichage dans le sens départ -> arrivée
print("le plus court chemin passe par les sommets : ",chemin,' et sa mesure est : ',etapes[nb_sommets-1][0])
```

3.2) **Modifier le script correspondant aux itinéraires sur 11 sommets de A à K** (modifier uniquement la partie réservée à la description du graphe)

```
""" ********** description du graphe **********"""
nb_sommets = 6

voisins_A = [0,3,1,1000,1000,1000]
voisins_B = [3,0,1,2,1000,1000]
voisins_C = [1,1,0,3,5,1000]
voisins_D = [1000,2,3,0,1,3]
voisins_E =  [1000,1000,5,1,0,1]
voisins_F = [1000,1000,1000,3,1,0]
m_voisins = [voisins_A,voisins_B,voisins_C,voisins_D,voisins_E,voisins_F]
```

3.3) **Ecrivez dans le cadre ci-dessous les résultats obtenus**

.....

.....

.....