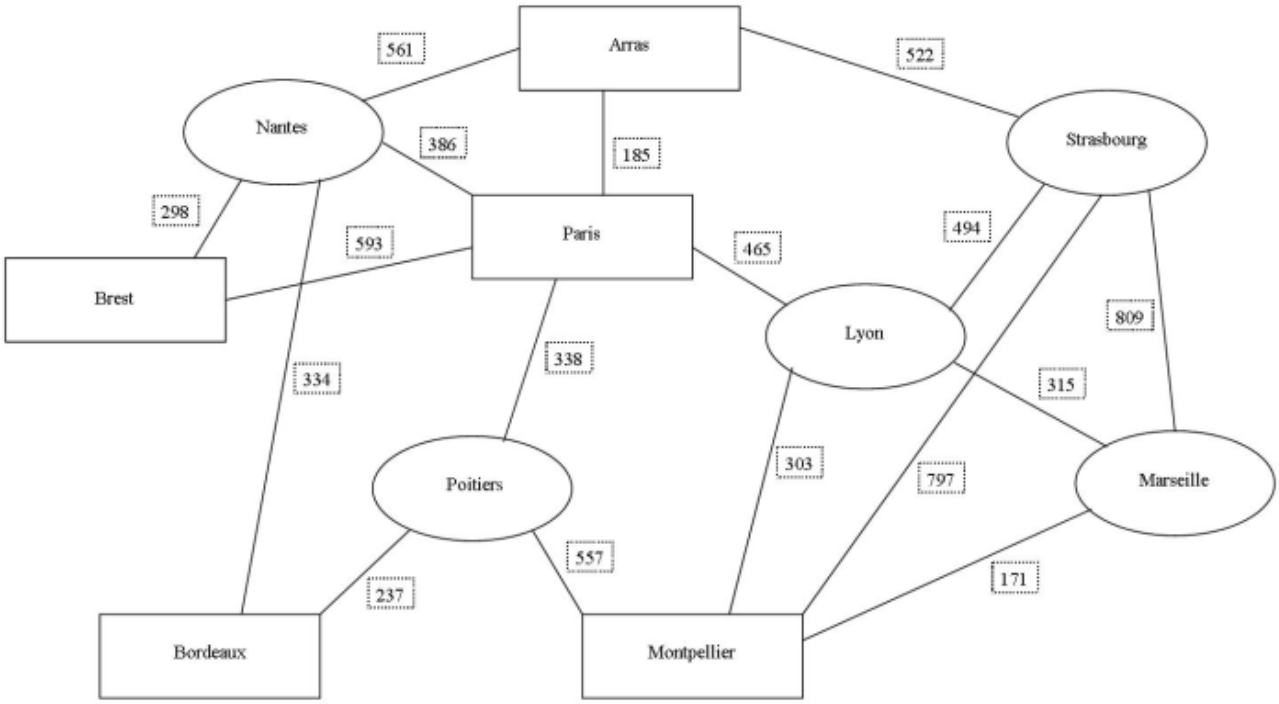
TD3 : Parcours de graphe (Algorithme de Dijkstra)

Principe de l'algorithme de Dijkstra

Le principe de l'algorithme de Dijkstra est de trouver le chemin ayant le poids le plus faible entre 2 noeuds, sachant que le poids d'un chemin est la somme des poids des arêtes qui le composent.



Pour appliquer l'algorithme, il va nous falloir deux tableaux.

- Un tableau à 3 lignes, que l'on va appeler "tableau des poids" contenant :
    - dans la première ligne, la liste des nœuds (désignés soit par leur nom, soit plutôt par un numéro) ;
    - dans la seconde ligne, un poids affecté à chaque nœud ;
    - dans la troisième ligne, une variable *vrai* ou *faux*, qui servira à savoir si l'on est déjà passé par ce nœud.
- Un autre tableau à une dimension, que l'on nommera "tableau des prédécesseurs" contenant ce que nous allons appeler les prédécesseurs de chaque nœud, c'est-à-dire le "noeud-père" qui précède le nœud dans le chemin que l'on prend pour y aller.
  Exemple : si pour aller de Nantes à Strasbourg je suis ce trajet *(cf. : graphe)* : Strasbourg => Arras => Nantes, le prédécesseur de Nantes sera Arras, le prédécesseur de Arras sera Strasbourg, et Strasbourg lui, n'aura pas de prédécesseur.

Étapes de l'algorithme de Dijkstra

0) Tout d'abord on initialise les tableaux

Concernant le tableau des poids, on va mettre tous les poids à -1, montrant par là qu'aucun poids n'a encore été affecté à un nœud, et on va mettre toutes les variables oui ou non à non, car on n'est passé par aucun nœud. Ensuite, on met le poids du point de départ à 0.

Nom du nœud	Poids	Déjà parcouru ?
Arras	-1	Non
Bordeaux	-1	Non
Brest	0	Non
Lyon	-1	Non
Marseille	-1	Non
Montpellier	-1	Non
Nantes	-1	Non
Paris	-1	Non
Poitiers	-1	Non
Strasbourg	-1	Non

Puis on met à 0 le tableau des antécédents.

Nœud	Antécédent du nœud
Arras	Aucun
Bordeaux	Aucun
Brest	Aucun
Lyon	Aucun
Marseille	Aucun
Montpellier	Aucun

Nœud	Antécédent du nœud
Nantes	Aucun
Paris	Aucun
Poitiers	Aucun
Strasbourg	Aucun

1) On recherche le nœud non parcouru ayant le poids le plus faible et on indique donc qu'on l'a parcouru

La première fois, il s'agit forcément du nœud de départ. On met donc la variable oui ou non de Brest à oui.

2) On va rechercher ce que l'on appelle "les fils" du nœud où l'on se trouve

Si l'on découvre des fils au nœud, on va effectuer une condition sur chaque fils que l'on a trouvé :

```
SI ( le noeud-fils n'a pas encore été parcouru )

ET QUE ( Poids(Noeud-père) + Poids(Liaison Noeud-père/Noeud-fils) <
Poids(Noeud-fils) ) OU Poids(Noeud-fils) = -1

{

    Poids(Noeud-fils) = Poids(Noeud-père) + Poids(Liaison Noeud-père/Noeud-
fils)

    Antecedent(Noeud-fils) = Noeud-Père

}
```

3) On boucle en retournant à l'étape 1) tant que...

Tant que le nœud ayant le poids le plus faible n'est pas le nœud d'arrivée.
En effet, quand le nœud ayant le poids le plus faible sera le nœud d'arrivée, cela voudra dire que l'on a trouvé le chemin le plus court pour y aller en partant du nœud de départ.

Travail demandé :

Écrire un programme JAVA permettant d'implémenter l'algorithme de Dijkstra. Le programme sera constitué des classes suivantes :

- La classe **Node** qui désigne un nœud du graphe.

- La classe **Edge** qui désigne une liaison entre deux nœuds.

- La classe **Graphe** qui contiendra une collection de **Node** et une autre de **Edge**.

- La classe **Dijkstra** qui contiendra l'implémentation de l'algorithme et dont le constructeur aura pour paramètre un objet de type **Graphe**.

- La classe **TestDijkstra** qui contiendra la fonction main, et qui permet de tester l'algorithme en lui passant le graphe de la première page (on affichera par exemple le plus court chemin de Bordeaux à Strasbourg).