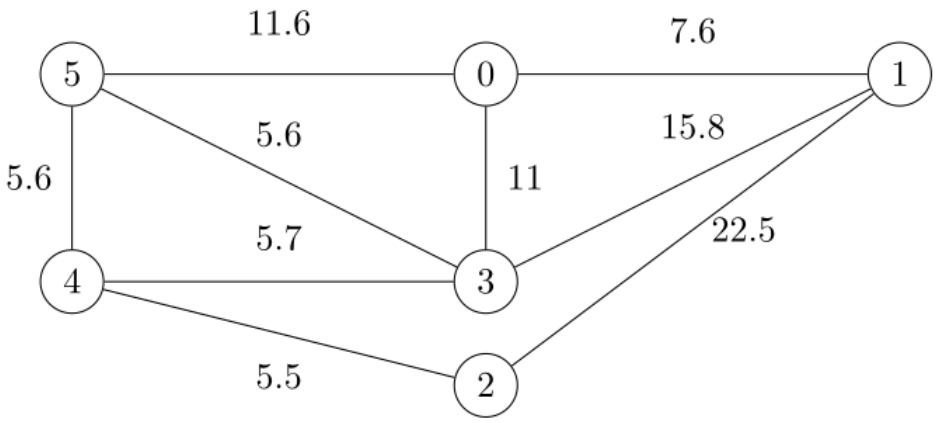
P17 - Algorithme de Dijkstra

Correction



In [1]:
```
import numpy as np
```

> **1. a) Définir la liste d'adjacence G du graphe.**

In [2]:
```
G = [[[1, 7.6], [3, 11], [5, 11.6]],
     [[0, 7.6], [2, 22.5], [3, 15.8]],
     [[1, 22.5], [4, 5.5]],
     [[0, 11], [1, 15.8], [4, 5.7], [5, 5.6]],
     [[2, 5.5], [3, 5.7], [5, 5.6]],
     [[0, 11.6], [3, 5.6], [4, 5.6]],
    ]
```

> **b) Comment obtenir les voisins (numéro et poids) du sommet 4 ?**

In [3]:
```
G[4]
```

Out[3]:
```
[[2, 5.5], [3, 5.7], [5, 5.6]]
```

> **c) Le sommet 4 a pour voisin $v = [5, 5.6]$. Comment obtenir le poids de l'arête correspondante ?**

In [4]:
```
v = [5, 5.6]
v[1]
```

Out[4]:
```
5.6
```

> **2. Écrire une fonction sommet_min(distances, a_explorer) qui renvoie le numéro du sommet de distance courante minimale parmi les sommets à explorer.**

In [5]:
```
def sommet_min(distances, a_explorer):
    mini = np.inf
    for i in range(0, len(distances)):
        if distances[i] < mini and i in a_explorer:
            mini = distances[i]
            imini = i
    return imini
```

In [6]:
```
# test
sommet_min([4,2,1,7,3], [0,1])
```

Out[6]:
```
1
```

> **3. Écrire une fonction supprime(s,L) qui supprime la valeur s dans la liste L.**

Indication : pour supprimer l'élément numéro i dans une liste L, on utilise del L[i]

In [7]:
```
def supprimer(s, L):
    i = 0
    while L[i] != s:
        i = i+1
    del L[i]
```

> **4. Compléter le programme suivant qui applique l'algorithme de Dijkstra.**

In [8]:
```
def Dijkstra(G, sd, sf):
    # valeurs initiales
    n = len(G)
    distances = [np.inf for i in range(n)]
    distances[sd] = 0
    a_explorer = [ i for i in range(n) ]
    # boucle
    while sf in a_explorer :
        print('a_explorer=',a_explorer)
        print('distances=',distances)
        s = sommet_min(distances, a_explorer)
        print('s=',s)
        supprimer(s, a_explorer)
        for v in G[s] : # v = voisin de s, v = [numero, poids]
            if v[0] in a_explorer:
                d = distances[s] + v[1]
                if d < distances[v[0]]:
                    distances[v[0]] = d
    # résultat final
    return distances[sf]
```

In [9]:
```
Dijkstra(G, 0, 2)
```

```
a_explorer= [0, 1, 2, 3, 4, 5]
distances= [0, inf, inf, inf, inf, inf]
s= 0
a_explorer= [1, 2, 3, 4, 5]
distances= [0, 7.6, inf, 11, inf, 11.6]
s= 1
a_explorer= [2, 3, 4, 5]
distances= [0, 7.6, 30.1, 11, inf, 11.6]
s= 3
a_explorer= [2, 4, 5]
distances= [0, 7.6, 30.1, 11, 16.7, 11.6]
s= 5
a_explorer= [2, 4]
distances= [0, 7.6, 30.1, 11, 16.7, 11.6]
s= 4
a_explorer= [2]
distances= [0, 7.6, 22.2, 11, 16.7, 11.6]
s= 2
```

Out[9]:
```
22.2
```

> **5. Comment modifier simplement la fonction pour qu'elle renvoie toutes les distances de sd aux autres sommets ?**

In [10]:
```
def Dijkstra_distances(G, sd):
    # valeurs initiales
    n = len(G)
    distances = [np.inf for i in range(n)]
    distances[sd] = 0
    a_explorer = [ i for i in range(n) ]
    # boucle
    while len(a_explorer) > 0 :
        s = sommet_min(distances, a_explorer)
        supprimer(s, a_explorer)
        for v in G[s] : # v = voisin de s, v = [numero, poids]
            if v[0] in a_explorer:
                d = distances[s] + v[1]
                if d < distances[v[0]]:
                    distances[v[0]] = d
    # résultat final
    return distances
```

In [11]:
```
Dijkstra_distances(G, 0)
```

Out[11]:
```
[0, 7.6, 22.2, 11, 16.7, 11.6]
```

> **6. a) On souhaite maintenant obtenir un chemin le plus court. Expliquer ce que fait la fonction suivante.**

In [12]:
```
def chemin(sommets_peres, sd, sf):
    s = sf
    c = [sf]
    while s != sd:
        s = sommets_peres[s]
        c = [s] + c
    return c
```

> **b) En déduire une fonction Dijkstra_chemin(G,sd,sf) qui renvoie un plus court chemin de sd à sf.**

In [13]:
```
def Dijkstra_chemin(G, sd, sf):
    # valeurs initiales
    n = len(G)
    distances = [np.inf for i in range(n)]
    distances[sd] = 0
    a_explorer = [ i for i in range(n) ]
    sommets_peres = [None for i in range(n)]
    # boucle
    while sf in a_explorer :
        s = sommet_min(distances, a_explorer)
        supprimer(s, a_explorer)
        for v in G[s] : # v = voisin de s, v = [numero, poids]
            if v[0] in a_explorer:
                d = distances[s] + v[1]
                if d < distances[v[0]]:
                    distances[v[0]] = d
                    sommets_peres[v[0]] = s
    # résultat final
    return chemin(sommets_peres, sd, sf)
```

In [14]:
```
Dijkstra_chemin(G, 0, 2)
```

Out[14]:
```
[0, 3, 4, 2]
```

In [ ]: