**Exercice 1.** ⋆ $G = \{A, B, C, D, E, F\}$ , $\mathcal{A} = \{\{A, C\}, \{A, F\}, \{F, C\}, \{E, F\}, \{E, C\}, \{B, C\}, \{B, D\}, \{D, E\}, \{B, E\}\}$.

⋆ Il s'agit d'un graphe non orienté, simple et pondéré

**Exercice 2.** ⋆ La matrice est symétrique car le graphe est non orienté

### Étude sur un exemple

**Exercice 3.** ⋆ Appliquer l'algorithme de Dijkstra avec origine $E$.

| $D$ | $B$ | $E$ | $C$ | $F$ | $A$ |
|---|---|---|---|---|---|
| $\infty$ | $\infty$ | 0 | $\infty$ | $\infty$ | $\infty$ |
| 1 | 1 | 0 | 3 | 5 | $\infty$ |
| 1 | 1 | 0 | 1 | 4 | 6 |

**Exercice 4.** ⋆ Appliquer l'algorithme de Dijkstra avec origine $C$.

| $D$ | $B$ | $E$ | $C$ | $F$ | $A$ |
|---|---|---|---|---|---|
| $\infty$ | $\infty$ | $\infty$ | 0 | $\infty$ | $\infty$ |
| $\infty$ | 2 | 3 | 0 | 1 | 3 |
| $\infty$ | 2 | 3 | 0 | 1 | 2 |
| 5 | 2 | 3 | 0 | 1 | 2 |
| 4 | 2 | 3 | 0 | 1 | 2 |

### Script Python

**Exercice 5.** On va modifier le tableau précédent avec pour départ $D$ pour permettre de retrouver la plus courte chaîne. Pour cela, si à une étape, le coefficient associé à un sommet a changé on signifie alors au dessus de la nouvelle valeur de quel sommet provient cette modification. Par exemple en ligne 3 du tableau, la valeur associé à $B$ est passée de $+\infty$ à 3 on rajoute en exposant $(D)$ et pour la ligne 4 la valeur associée à $C$ est passée de $+\infty$ à 4, on écrit $(E)$ en exposant.

| $D$ | $B$ | $E$ | $C$ | $F$ | $A$ |
|---|---|---|---|---|---|
| 0 | $+\infty$ | $+\infty$ | $+\infty$ | $+\infty$ | $+\infty$ |
| 0 | $3^{(D)}$ | $1^{(D)}$ | $+\infty$ | $+\infty$ | $+\infty$ |
| 0 | $2^{(E)}$ | 1 | $4^{(E)}$ | $6^{(E)}$ | $+\infty$ |
| 0 | 2 | 1 | 4 | $5^{(C)}$ | $7^{(C)}$ |
| 0 | 2 | 1 | 4 | 5 | $6^{(F)}$ |

⋆ La plus courte chaîne de $D$ vers $A$ est réalisé par $D - E - C - F - A$.

⋆ La plus courte chaîne de $D$ vers $E$ est réalisé par $D - E$.

⋆ La plus courte chaîne de $D$ vers $B$ est réalisé par $D - E - B$.

**Exercice 6.**

```
import numpy as np
Inf = np.inf
G=np.array([[0,3,1,Inf,Inf,Inf],[3,0,1,2,Inf,Inf],[1,1,0,3,5,Inf],
[Inf,2,3,0,1,3],[Inf,Inf,5,1,0,1],[Inf,Inf,Inf,3,1,0]])
def dijkstra(G,depart):
    # On récupère le nombre de sommets du graphe
    N = np.size(G,0)
    # Initialisation du tableau des plus courts chemins
    pcc = list()
    #le tableau des plus courtes chaînes est complété avec Inf
    for i in range(N):
        pcc.append([Inf, False,None])
    #le poids associé au dpéart est 0
    sommet_u = depart
```

```
    dist_u = 0
    pcc[depart][0] = 0
    # Le premier sommet sélectionné est le sommet depart
    pcc[depart][1] = True
    # On compte le nombre de sommets sélectionnés
    cpt = 0
    while cpt != N-1:
        # À chaque étape, la solution optimale doit être conservée
        minimum = Inf
        # Étape de relâchement
        for k in range(N):
            # Si le sommet k n'a pas encore été sélectionné
            if pcc[k][1] == False:
                dist_uv = G[sommet_u][k]
                # Distance totale du chemin s -> ... -> u -> v
                dist_totale = dist_u + dist_uv
                # Mise à jour du tableau des plus courts chemins
                if dist_totale < pcc[k][0]:
                    pcc[k][0] = dist_totale
                    pcc[k][2]=sommet_u

                # Mise à jour de la solution minimale à cette étape si ça l'améliore
                if pcc[k][0] < minimum:
                    minimum = pcc[k][0]
                    prochain_sommet_select = k
                # On a traité complétement un sommet

        cpt = cpt + 1
        # Le sommet à traiter est sélectionné

        sommet_u = prochain_sommet_select
        pcc[sommet_u][1] = True
        # Mise à joure de la distance du sommet selectionné
        dist_u = pcc[sommet_u][0]
    return(pcc)
```

**Exercice 7.**

```
⋆ def dijkstraPCC(G, depart, arrivee):
      pcc = dijkstra(G, depart)
  # Reconstitution du plus court chemin
      chaine = list()
  # On reconstitue le plus court chemin d'arrivee vers depart
      etape = arrivee
      chaine.append(etape)
      while etape != depart:
          etape = pcc[etape][2]
          chaine.append(etape)
  # On demande le miroir de la liste obtenue pour
  # que les sommets apparaissent dans l'ordre
      return(list(reversed(chaine)))
```