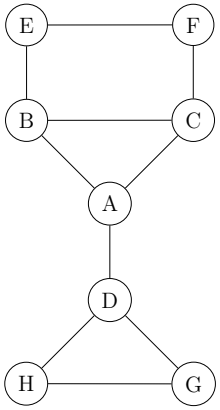
MPSI - Informatique Tronc Commun
Cours 26 - Algorithme de Dijkstra

1 - Mise en garde Ce document contient surtout les codes python liés au cours, mais l'essentiel du cour se déroulera au tableau et vous devez donc **prendre des notes**.

2 - Exemple de graphe représenté par dictionnaire de listes d'adjacence

```
non_pondere = {
    "A" : ["B", "C", "D"],
    "B" : ["A", "C", "E"],
    "C" : ["A", "B", "F"],
    "D" : ["A", "G", "H"],
    "E" : ["B", "F"],
    "F" : ["C", "E"],
    "G" : ["D", "H"],
    "H" : ["D", "G"],
}
```



3 - Rappel : implantation d'une file en python

```
from collections import deque
def file_vide():
    return deque()

def enfiler(f, x):
    f.appendleft(x)

def defiler(f):
    return f.pop()

def est_vide(f):
    return f == deque()
```

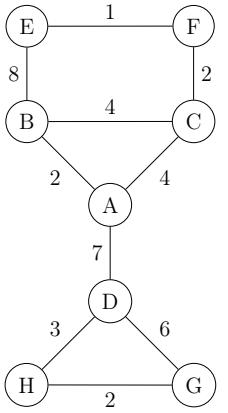

4 - Calcul de toutes les distances à un sommet donné

```
NON_VU = 0
VISITE_PREVUE = 1
DEJA_VISITE = 2

def distances(graphe, source):
    res = {s : None for s in graphe.keys() }
    statut = {s : NON_VU for s in graphe.keys() }
    a_visiter = file_vide()
    def prevoir_visite(sommet, distance):
        if statut[sommet] == NON_VU:
            enfiler(a_visiter, (sommet, distance))
            statut[sommet] = VISITE_PREVUE
    # Fin de la fonction auxiliaire prevoir_viste
    prevoir_visite(source, 0)
    while not est_vide(a_visiter):
        (u, d) = defiler(a_visiter)
        res[u] = d
        statut[u] = DEJA_VISITE
        for voisin in graphe[u]:
            prevoir_visite(voisin, d+1)
    return res
```

5 - Premier exemple de graphe pondéré

```
g1 = {
    "A" : [("B", 2), ("C", 4), ("D", 7)],
    "B" : [("A", 2), ("C", 4), ("E", 8)],
    "C" : [("A", 4), ("B", 4), ("F", 2)],
    "D" : [("A", 7), ("G", 6), ("H", 3)],
    "E" : [("B", 8), ("F", 1)],
    "F" : [("C", 2), ("E", 1)],
    "G" : [("D", 6), ("H", 2)],
    "H" : [("D", 3), ("G", 2)],
}
```



6 - Code de l'algorithme de Dijkstra

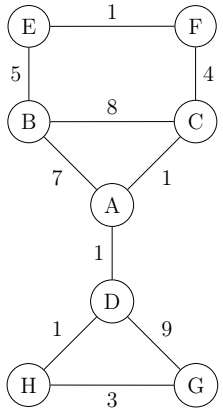
```
def dijkstra(graphe, source):
    res = {s : None for s in graphe.keys() }
    statut = {s : NON_VU for s in graphe.keys() }
    a_visiter = file_prio()
    enfiler_ou_mettre_a_jour(a_visiter, source, 0)
    while not est_vide(a_visiter):
        (u, d) = defiler(a_visiter)
        res[u] = d
        statut[u] = DEJA_VISITE
        for (voisin, delta) in graphe[u]:
            if not statut[voisin] == DEJA_VISITE:
                enfiler_ou_mettre_a_jour(a_visiter, voisin, d+delta)
    return res
```

7 - Les files de priorité Pour l'algorithme de Dijkstra on a donc besoin de files de priorité ayant au moins les opérations suivantes :

- Une fonction `file_prio()` qui ne prend pas d'arguments et qui renvoie une nouvelle file de priorité, vide.
- Une fonction `est_vide(f)` qui prend en entrée une file de priorité et renvoie en sortie un booléen qui indique si l'entrée est vide.
- Une fonction `enfiler_ou_mettre_a_jour(f, x, nouvelle_prio)` qui prend en entrée une file de priorité `f`, un élément `x` et une priorité `nouvelle_prio`. Cette fonction ne renvoie rien (dit autrement, elle renvoie `None`). Elle modifie en place la file de priorité, de sorte que
  - Si `x` n'est pas dans la file de priorité, il est rajouté, avec priorité `nouvelle_prio`.
  - Si `x` est dans la file, avec priorité `ancienne_prio`, alors il reste dans la file, avec comme priorité le minimum entre l'ancienne et la nouvelle priorité.
- Une fonction `defiler(f)` qui retire de la file, et renvoie, l'élément de `f` de plus faible priorité, ainsi que sa priorité. Le résultat renvoyé par cette fonction est donc une paire `(élément, priorité)`.

8 - Deuxième exemple de graphe pondéré

```
g2 = {
    "A" : [("B", 7), ("C", 1), ("D", 1)],
    "B" : [("A", 7), ("C", 8), ("E", 5)],
    "C" : [("A", 1), ("B", 8), ("F", 4)],
    "D" : [("A", 1), ("G", 9), ("H", 1)],
    "E" : [("B", 5), ("F", 1)],
    "F" : [("C", 4), ("E", 1)],
    "G" : [("D", 9), ("H", 3)],
    "H" : [("D", 1), ("G", 3)],
}
```



9 - Exercice Déroulez l'algorithme de Dijkstra sur le graphe `g2` depuis le sommet `'A'`. Vous garderez sur votre feuille une trace des différents états par lesquels passe la file de priorité.

10 - Exercice

1. Implantez les files de priorité, en utilisant des listes de paires `(élément, priorité)`.
2. Quelle est la complexité de vos différentes opérations sur les files de priorité ?
3. Quelle est la complexité de l'algorithme de Dijkstra en utilisant votre implantation de files de priorité ?