Plus courts chemins à source unique

Chaque jour partout dans le monde des millions de personnes utilisent des logiciels de calculs d'itinéraires pour se rendre d'un point A à un point B.

Ces logiciels sont basés sur des algorithmes de plus courts chemins dans des graphes.

1 Un exemple de graphe orienté pondéré

Un graphe orienté pondéré est un graphe orienté dont les arcs ont des poids (pas forcément positifs).

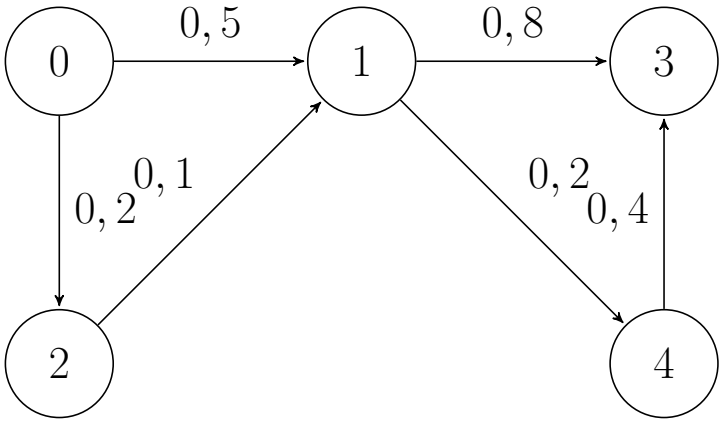
La **longueur d'un chemin** est la somme des poids des arcs composant ce chemin.

La seule contrainte demandée est qu'**il n'y a pas de cycle négatif dans le graphe orienté**.

Un cycle négatif est un cycle tel que la longueur du cycle est strictement négatif.

L'existence d'un cycle négatif ne permet pas de définir dans tous les cas de chemin le plus court d'une source vers un sommet.

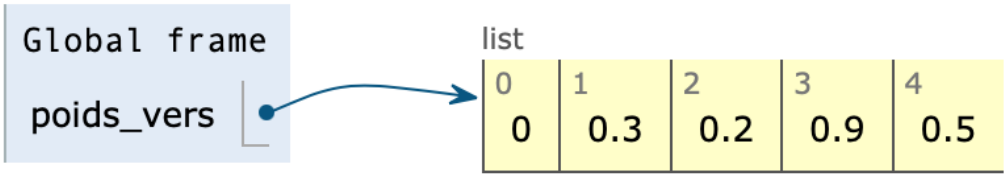
(Voir exercice)



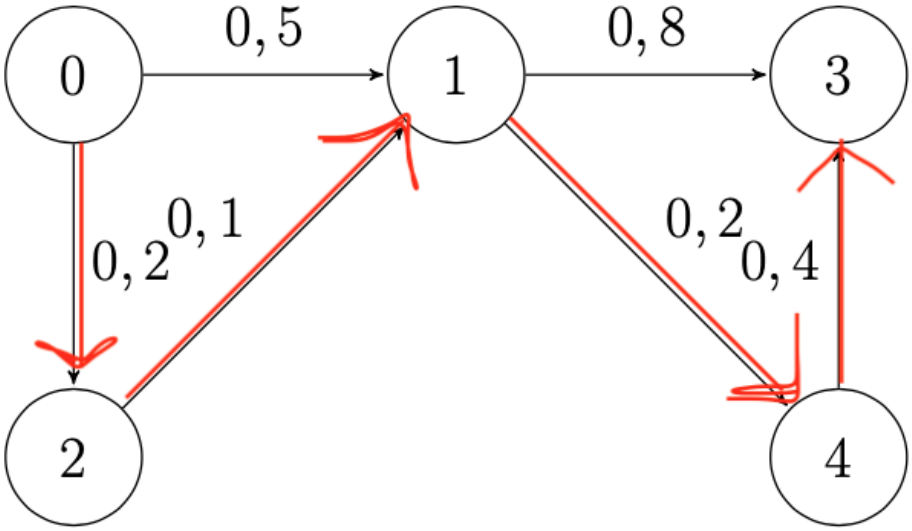
En partant de la source 0 les chemins les plus courts vers les autres sommets sont :

1. Pour aller vers 1 : le chemin 0 -> 2 -> 1 de poids 0,3
2. Pour aller vers 2 : le chemin 0 -> 2 de poids 0,2
3. Pour aller vers 3 : le chemin 0 -> 2 -> 1 -> 4 -> 3 de poids 0,9
4. Pour aller vers 4 : le chemin 0 -> 2 -> 1 -> 4 de poids 0,5

A la fin de la recherche des chemins les plus courts en partant de la source 0 on a un tableau `poids_vers` qui donne les poids de tous les chemins les plus courts de la source 0 vers un sommet i du graphe en exemple.



L'ensemble des chemins les plus courts de la source vers les autres sommets forme un **arbre**



Pour pouvoir travailler avec des arcs pondérés on va introduire une classe `Arc_P`

```
class Arc_P:
      def __init__(self,d,f,p):
        self.deb = d
        self.fin = f
        self.poids = p

    def origine(self):
        return self.deb

    def fin(self):
        return self.fin

    def poids(self):
        return self.poids
```

On implémente en Python une classe `Graphe_OP` pour graphe orienté pondéré

```
class Graphe_OP:
      def __init__(self,n):
```

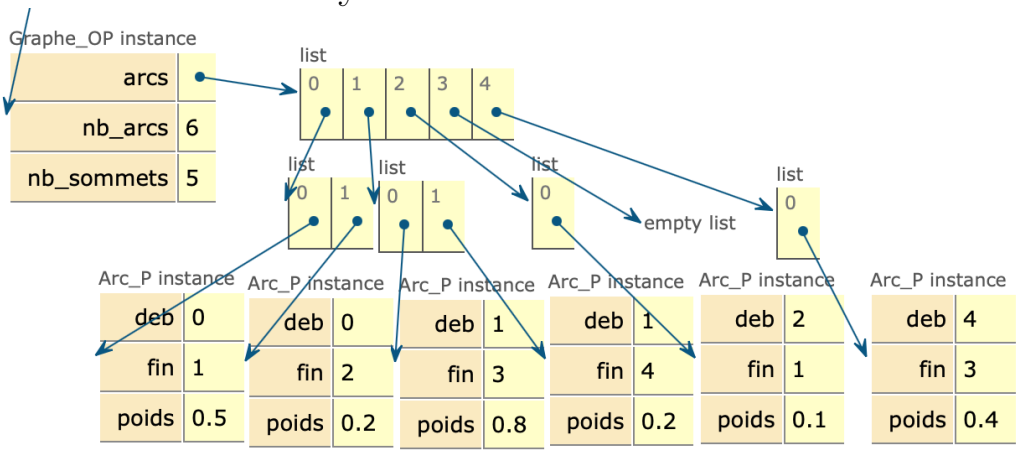
```
        self.nb_sommets = n
        self.nb_arcs = 0
        self.arcs =  \
        [[]for i in range(nb_sommets)]

    def ajoute_arc(self,i:int,arc_p:Arc_P):
        self.arcs[i] .append( arc_p)
        self.nb_arcs += 1

    def voisins(self,i):
        return self.arcs[i]

    def __str__(self):
        ch = str(self.nb_sommets)+\
        " sommets "+str(self.nb_arcs)+\
        " arcs \n"
        for i in range(self.nb_sommets):
            ch = ch + str(i) + " --> "
            for arc in self.arcs[i]:
                ch = ch + str(arc.fin)+\
                 "("+str(arc.poids)+")"+" "
            ch = ch +"\n"
        return ch
```

Voici l'instanciation du graphe en exemple ci-dessus

```
g = Graphe_OP(5)
g.ajoute_arc(0,Arc_P(0,1,0.5))
g.ajoute_arc(0,Arc_P(0,2,0.2))
g.ajoute_arc(1,Arc_P(1,3,0.8))
g.ajoute_arc(1,Arc_P(1,4,0.2))
g.ajoute_arc(2,Arc_P(2,1,0.1))
```

```
g.ajoute_arc(4,Arc_P(4,3,0.4))
```

On observe dans Python Tutor



On obtient dans la console lorsqu'on fait `print(g)`

```
5 sommets 6 arcs
0 --> 1(0.5) 2(0.2)
1 --> 3(0.8) 4(0.2)
2 --> 1(0.1)
3 -->
4 --> 3(0.4)
```

2 Algorithme de plus court chemin en partant d'une source

Définition 1. *Etant donné un graphe pondéré G et une source s de ce graphe*

Le ***poids du plus court chemin*** *entre la source s et un sommet i de ce graphe est défini par :*

S'il existe au moins un chemin p de s vers i alors

$\delta(s,i) = min(l(p) : s \xrightarrow{p} i)$ où $l(p)$ est la longueur d'un chemin p,

Sinon $\delta(s,i) = \infty$

Un plus court chemin *de la source s vers un sommet*

i est un chemin quelconque p de longueur l(p) tel que δ(s,i) = l(p)

Théorème 1. *(Conditions d'optimalité)*

G un graphe orienté pondéré sans cycle négatif et s le sommet source.

Le tableau `poids_vers` *donne le tableau des poids des chemins les plus courts en partant de s si et seulement si :*

1. `poids_vers[s] = 0`
2. *Pour tout sommets v et w ,*
   `poids_vers[v]` $\leqslant$ `poids_vers[w]` + `poids(w->v)`

   (Inégalité triangulaire)

Définition 2. *Le relâchement d'un arc u -> v consiste à tester si l'on peut améliorer le plus court chemin de la source s à v en passant par u.*

Algorithme 1 : Relâchement d'un arc u-> v

```
relâchement (u->v)
début
    Données : Un arc pondéré u -> v, un tableau
              poids_vers
    Résultat : Rien
1   si poids_vers[v] > poids_vers[u] +poids(u->v) alors
2       poids_vers[v] ← poids_vers[u] +poids(u->v)
3       //fil d'ariane pour construire un chemin le plus
         court
4       predecesseur[v] ← u
5   fin
fin
```

D'où un algorithme très général pour rechercher les meilleurs chemins dans un graphe pondéré sans cycle négatif, en partant d'une source.

Algorithme 2 : Meilleurs chemins en partant de s

```
meilleurs_chemins (G,s)
début
    Données : Un graphe orienté pondéré, une source s
    Résultat : Rien
1   poids_vers[s] ← 0
2   Pour tout autre sommet poids_vers[w] ← inf
3   tant que conditions optimalité non vérifiées faire
4       Relâcher tous les arcs du graphe
5   fin
fin
```

Cet algorithme est très général au sens où il ne précise pas de méthode pour relâcher les arcs du graphe pondéré.

3 Algorithme de Dijkstra

On suppose les poids positifs.

L'algorithme de Dijkstra organise le relâchement des arcs ainsi :

1. On met dans une file de priorité minimale la source et comme clé poids_vers[source] = 0
2. Tant que la file n'est pas vide on prend la racine r de la file de priorité, on relâche tous les arcs d'origine r et on insère dans la file les extrémités de ces arcs avec leurs clés

On exécute l'algorithme de Dijkstra dans un tableau sur le graphe pondéré ci-dessus en prenant comme source le sommet 0.

étape	0	1	2	3	4	choix
1	0	∞	∞	∞	∞	0
2	$\underline{0}$	0.5	0.2	∞	∞	2
3	$\underline{0}$	0.3	$\underline{0.2}$	∞	∞	1
4	$\underline{0}$	$\underline{0.3}$	$\underline{0.2}$	1.1	0.5	4
5	$\underline{0}$	$\underline{0.3}$	$\underline{0.2}$	0.9	$\underline{0.5}$	

Algorithme 3 : Algorithme de Dijkstra

```
dijkstra (G,s)
début
    Données : Un graphe orienté pondéré, une source s
    Résultat : Rien
1   poids_vers[s] ← 0
2   poids_vers[w] ← inf
3   file ← file_priorité_min()
4   insérer la source et sa clé 0 dans la file
5   tant que file non vide faire
6       r ← racine(file)
7       pour chaque voisin v de racine faire
8           //en relâchant on met à jour la file de priorité
9           //s'il y a eu relachement
10          //si v est dans la file on diminue sa clé
11          //sinon on met v dans la file
12          relâcher(r,v)
13      fin
14  fin
fin
```

4 Algorithme de Dijkstra dans un DAG

Si le graphe est un DAG l'algorithme de Dijkstra est plus efficace en relâchant les arcs selon leur ordre dans le tri topolo-

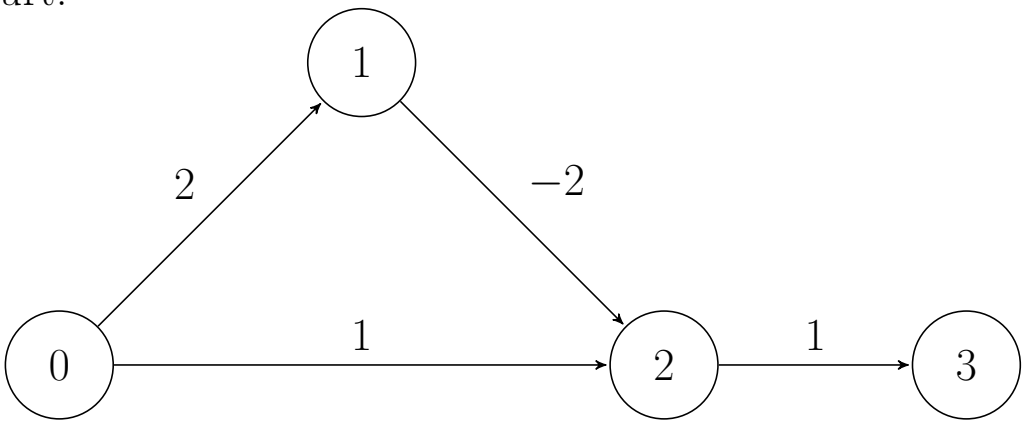
gique.

5 Algorithme de Bellman-Ford

Les poids peuvent être négatifs. Mais il n'y a pas de cycle négatif.

On observe sur le graphe suivant que l'algorithme de Dijkstra ne donne pas le chemin le plus court pour aller de 0 à 3.

L'algorithme de Dijkstra donne le chemin 0-2-3 avec un poids de 2 alors que le chemin 0-1-2-3 avec un poids de 1 est plus court.



Lemme 1. *(Propriété de relâchement de chemin)(admis)*

Si on a un chemin ***le plus court*** *de la source s vers un sommet i,*

$s = u_0 \to u_1 \to u_2 ..... \to u_k = i$

Si on relâche les arcs dans l'ordre $u_0 \to u_1$, $u_1 \to u_2$, ..., $u_{k-1} \to u_k$ *alors à la fin on obtient poids_vers[i]* $= \delta(s,i)$

L'Algorithme de Bellman-Ford est basée sur l'idée suivante :

S'il existe un chemin le plus court entre une source s et un sommet i, ce chemin ne contient pas de cycle (chemin simple) et par conséquent il a au plus n - 1 arcs où n est le nombre de sommets de ce graphe.

Supposons le pire des cas où il a n-1 arcs :

s $\rightarrow u_1 \rightarrow u_2 ..... \rightarrow u_{n-1}$ = i

Si on répète n-1 fois relâcher tous les arcs du graphe (dans n'importe quel ordre) alors :

A la première itération on aura relâché forcément s $\rightarrow u_1$

A la deuxième $u_1 \rightarrow u_2$, donc pour l'instant on respecte l'ordre des relâchements s $\rightarrow u_1$ puis $u_1 \rightarrow u_2$ pour pouvoir utiliser la propriété de relâchement de chemin

Et ainsi de suite jusqu'à la n-1 itération où on aura relâché $u_{n-2} \rightarrow u_{n-1}$

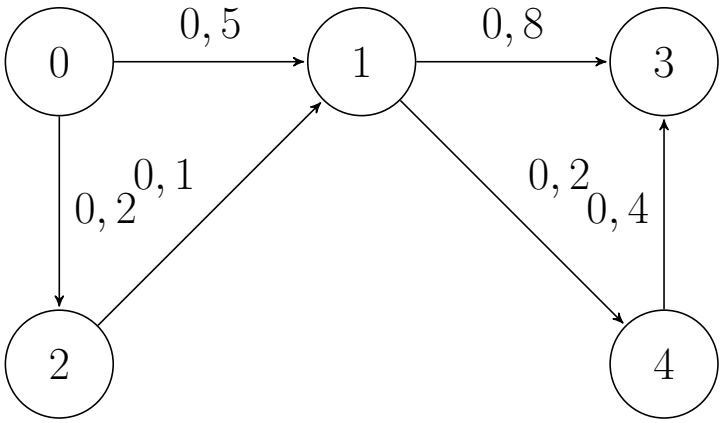
On aura fait cela pour tous les chemins de s vers i donc parmi un de ces chemins il y aura un chemin le plus court donc on est sûr d'après la propriété de relâchement de chemin d'avoir à la fin des n-1 itérations à la fois la distance la plus courte de s à i et aussi le prédécesseur de i permettant d'avoir un chemin le plus court de s à i

Algorithme 4 : Algorithme de Bellman-Ford

```
Bellman-Ford (G,s)
début
    Données : Un graphe orienté pondéré de n sommets,
              une source s
    Résultat : Rien
1   poids_vers[s] ← 0
2   Pour tout autre sommet poids_vers[w] ← inf
3   pour i ← 1 jusqu'à n-1 faire
4       Relâcher tous les arcs du graphe dans n'importe
        quel ordre(mais toujours le même)
5   fin
fin
```

Appliquons l'algorithme de Bellman-Ford sur le graphe suivant en relâchant les arcs dans l'ordre arbitraire suivant :

(4,3) - (1,3) - (1,4) - (2,1) - (0,2) - (0-1)



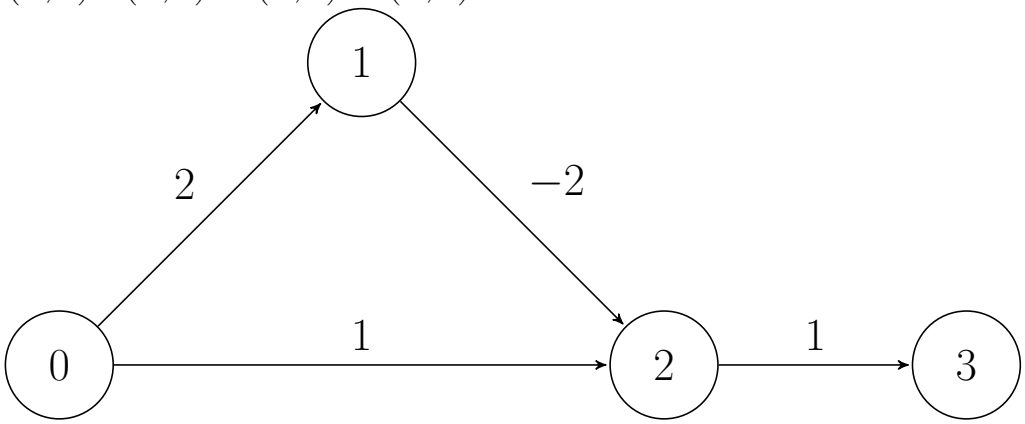
Il y a cinq sommets on va donc répéter 4 fois de l'étape 1 à l'étape 4

étape	0	1	2	3	4
0	0	∞	∞	∞	∞
1	0	0.5	0.2	∞	∞
2	0	0.3	0.2	1.3	0.7
3	0	0.3	0.2	1.1	0.5
4	0	0.3	0.2	0.9	0.5

On sait que dans le graphe suivant le chemin le plus court du sommet 0 au sommet 3 est constitué des arcs (0,1)- (1,2) - (2,3)

Appliquons l'algorithme de Bellman-Ford en relâchant les arcs dans l'ordre :

(2,3)- (1,2) - (0,1) - (0,2)



On obtient le tableau suivant :

étape	0	1	2	3
0	0	∞	∞	∞
1	0	2	1	∞
2	0	2	0	2
3	0	2	0	1

Exercices

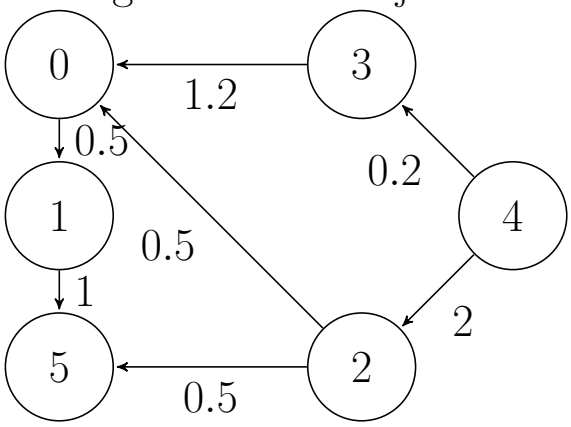
Ex 1

Supposons que dans un graphe orienté pondéré il y ait un cycle négatif .

Etant donné une source s , et un sommet w tel qu'il existe un chemin de s vers w et tel que w appartient au cycle négatif, montrer qu'il n'existe pas de plus court chemin de s vers w.

Ex 2

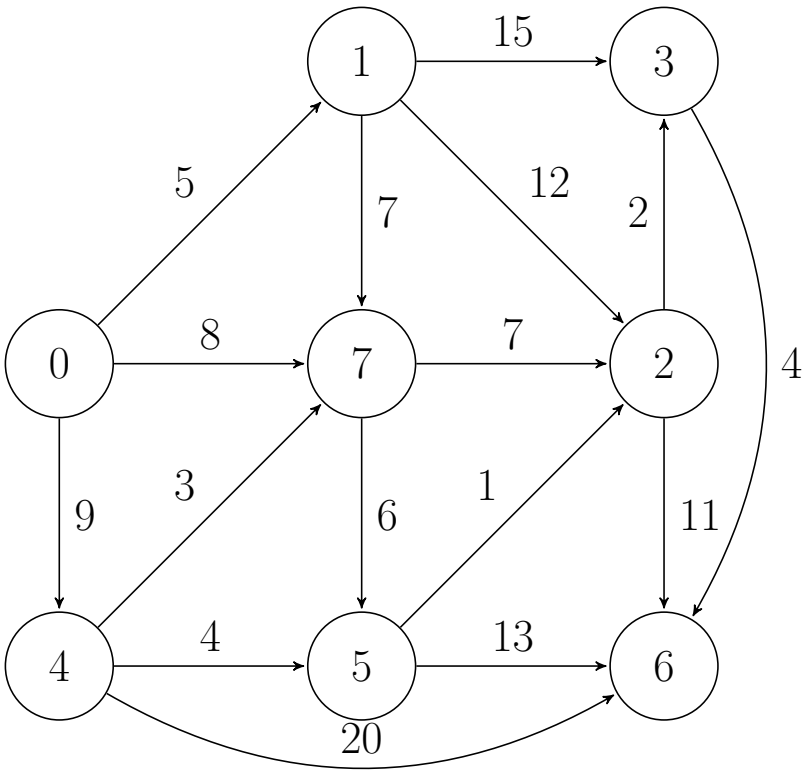
Faire le tri topologique du graphe orienté suivant puis appliquer l'algorithme de Dijkstra en prenant pour source s = 2



Ex 3

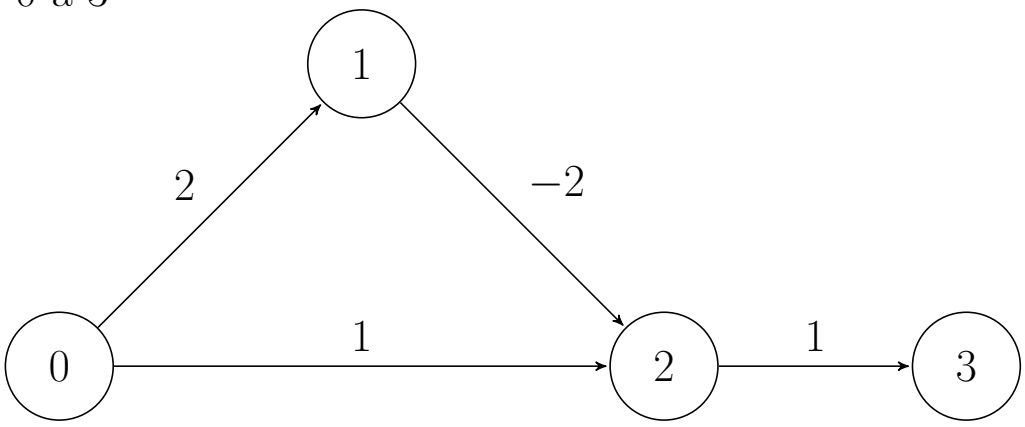
Appliquer l'algorithme de Dijkstra sur le graphe suivant en partant de la source :

1. s = 0
2. s = 4



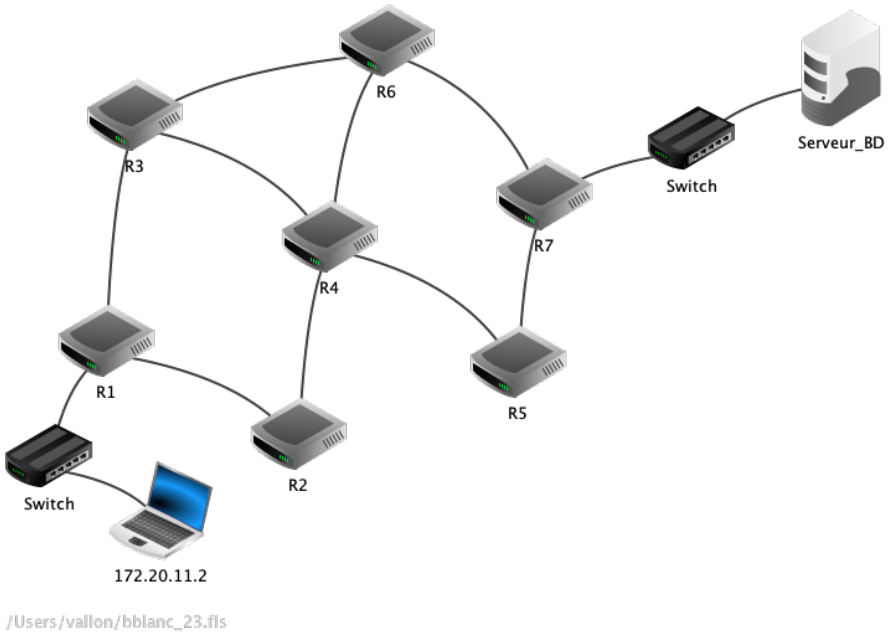
Ex 4

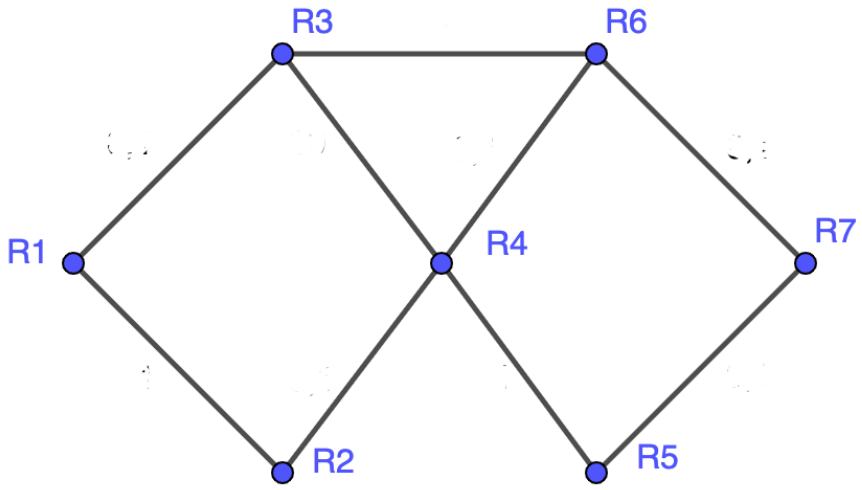
Appliquer l'algorithme de Dijkstra sur le graphe suivant et observer qu'il ne donne pas le chemin le plus court pour aller de 0 à 3

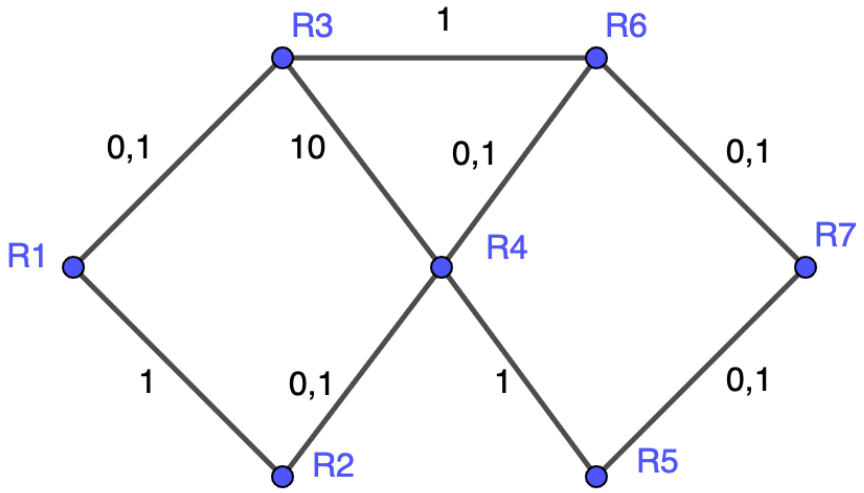


Ex 5

On a modélisé le réseau des routeurs sur un campus par un graphe non orienté pondéré des coûts des connexions entre les routeurs.







Appliquer l'algorithme de Dijkstra en partant de la source R1, et en déduire le chemin le plus court entre les routeurs R1 et R7.