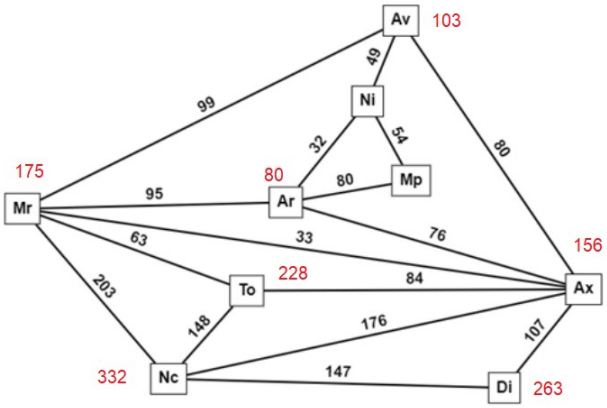
Proposition de correction

Exercice 1

Partie A



Q1

- Mp → Ar → Mr → Nc
- 332km

Q2

- Mp → Ar → Mr → Nc
- Mp → Ar → Ax → Nc

Partie B

Q3

```
G = {
    'Av': ['Mr', 'Ni', 'Ax'],
    'Ni': ['Av', 'Ar', 'Mp'],
    'Mp': ['Ni', 'Ar'],
    'Ar': ['Mr', 'Ni', 'Mp', 'Ax'],
    'Mr': ['Av', 'Ar', 'Ax', 'To', 'Nc'],
    'Ax': ['Av', 'Ar', 'Mr', 'To', 'Nc', 'Di'],
    'To': ['Mr', 'Ax', 'Nc'],
    'Nc': ['Mr', 'To', 'Ax', 'Di'],
    'Di': ['Nc', 'Ax']
    }
```

Q4

- LIFO : Last In First Out
- FIFO : First In First Out

Q5

FIFO

Q6

['Av', 'Mr', 'Ni', 'Ax', 'Ar', 'To', 'Nc', 'Mp', 'Di']

Q7

proposition A : parcours en largeur

Q8

```
def distance(graphe : dict, sommet : str) -> dict:
    """
    @param  graphe -- dictionnaire représentant un graphe sous la forme de listes d'adjacence
    @param  sommet -- un sommet du graphe
    @return un dictionnaire dont les clés sont les sommets du graphe
            et la valeur associée, la distance entre ce sommet clé et le sommet d'origine sommet
    """
    f = creerFile()
    enfiler(f, sommet)
    distances = {sommet: 0}
    visite    = [sommet]

    while not estVide(f):
        s = defiler(f)
        for v in graphe[s]:
            if v not in visite:
                visite.append(v)
                distances[v] = distances[s] + 1
                enfiler(f, v)

    return distances
```

Q9

{'Av': 0, 'Mr': 1, 'Ni': 1, 'Ax': 1, 'Ar': 2, 'To': 2, 'Nc': 2, 'Mp': 2, 'Di': 2}

Q10

```
def parcours2(G : dict, sommet : str) -> list:
    """
    @param  G -- un dictionnaire représentant un graphe sous la forme de listes d'adjacence
    @param  s -- un sommet du graphe
    @return parcours en profondeur depuis s
    """
    p = creerPile()
    empiler(p, sommet)
    visite = []                  # visite doit être initialisé à vide
    while not estVide(p):
        s = depiler(p)
        if not (s in visite):    # sinon on ne rentre pas dans ce bloc => fin
            visite.append(s)
            for v in G[s]:
                empiler(p, v)
    return visite
```

Q11

['Av', 'Ax', 'Di', 'Nc', 'To', 'Mr', 'Ar', 'Mp', 'Ni']

Exercice 2

Partie A

Q1

Le nœud initial est appelé **racine**

Un nœud qui n'a pas de fils est appelé **feuille**

Un arbre binaire est un arbre dans lequel chaque nœud a **au maximum** deux fils.

Un arbre binaire de recherche est un arbre binaire dans lequel tout nœud est associé à une clé qui est :

- supérieure à chaque clé de tous les nœuds de son **sous-arbre gauche**
- inférieure à chaque clé de tous les nœuds de son **sous-arbre droit**

Q2

1, 0, 2, 3, 4, 5, 6

Q3

0, 1, 2, 6, 5, 4, 3

Q4

0, 1, 2, 3, 4, 5, 6

Q5

```
arbre_no1 = ABR()
arbre_no2 = ABR()
arbre_no3 = ABR()
for cle_a_inserer in [1, 0, 2, 3, 4, 5, 6]:
    arbre_no1.inserer(cle_a_inserer)
for cle_a_inserer in [3, 2, 1, 0, 4, 5, 6]:
    arbre_no2.inserer(cle_a_inserer)
for cle_a_inserer in [3, 1, 5, 0, 2, 4, 6]:
    arbre_no3.inserer(cle_a_inserer)
```

Q6

- 5
- 3
- 2

Q7

```
def est_present(self, cle_a_rechercher):
    if self.est_vide() :
        return False
    elif cle_a_rechercher == self.cle() :
        return True
    elif cle_a_rechercher < self.cle() :
        return self.sag().est_present(cle_a_rechercher)
    else :
```

```
        return self.sad().est_present(cle_a_rechercher)
```

Q8

arbre_no3.est_presente(7)

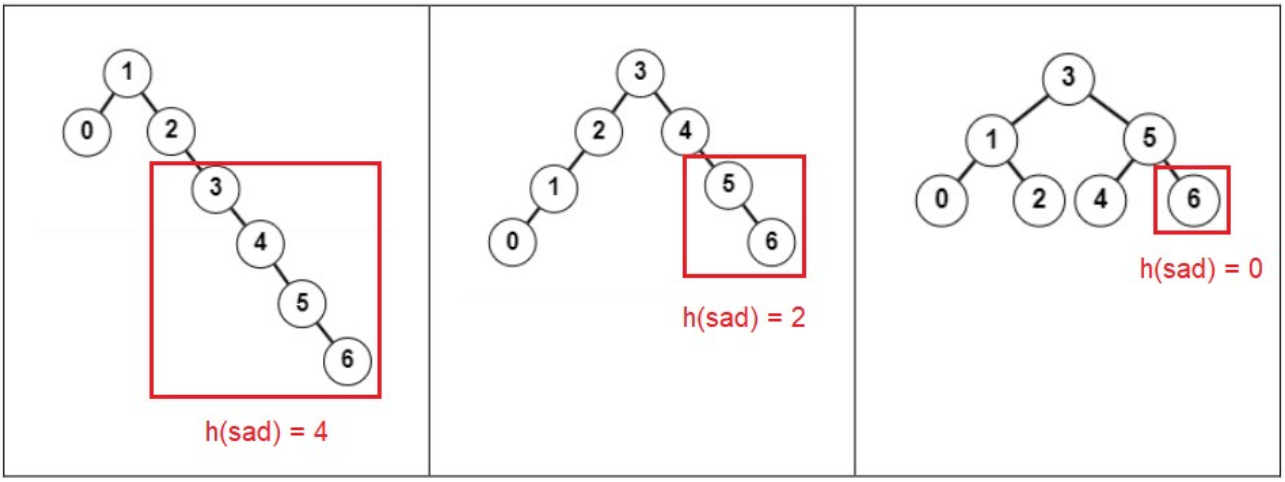
l'ABR est équilibré → recherche en $O(\ln_2(n))$

Partie B

Q9

la différence de hauteur entre les sous arbres gauche et droit doit être < 2

Q10



Arbre_2 et Arbre_3 partiellement équilibrés

Q11

Arbre_3 équilibré

Q12

```
def est_equilibre(self):
    if self.est_vide():
        return True
    else:
        equilibre = self.sad().hauteur() - self.sag().hauteur()
        return abs(equilibre) < 2 and self.sad().est_equilibre() and self.sag().est_equilibre()
```

Exercice 3

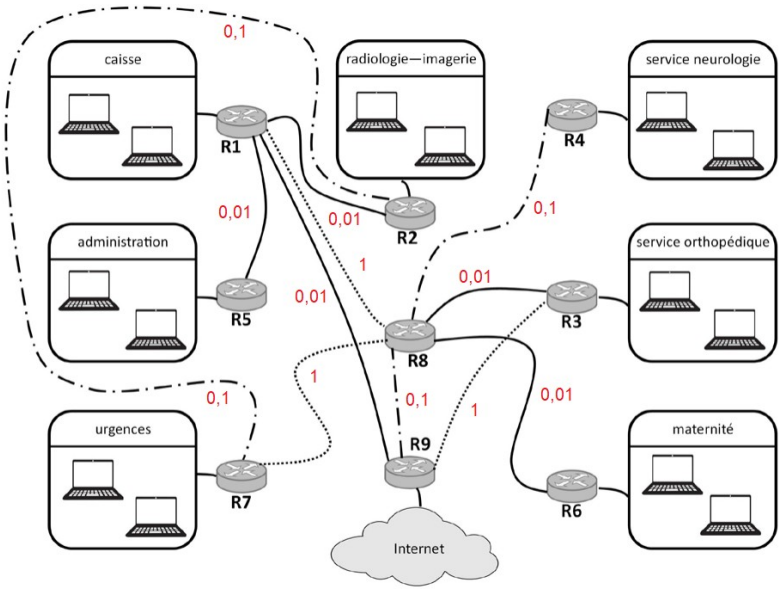
Partie A

Q1

neurologie → R4 → R8 → R1 → R2 → radiologie

Q2

Nœud R2	
Destination	**Coût**
R1	1
R3	3
R4	3
R5	2
R6	3
R7	1
R8	2
R9	2



Q3

$c = 10^8 / (10.10^9) = 0,01$

Q4

neurologie → R4 → R8 → R9 → R1 → R2 → radiologie

Partie B

Q5

Tardus Kylian

Montpart Vincent

Q6

```
SELECT num_SS
FROM patient
WHERE service = 'orthopédique'
AND date LIKE '%/%/2023'
ORDER BY num_SS
```

Q7

```
SELECT examen.type, examen.date
FROM examen
JOIN patient
```

```
ON examen.num_SS = patient.num_SS
WHERE patient.nom = ‘Baujean’
AND patient.prenom = ‘Emma’
ORDER BY examen.date
```

Q8

```
SELECT patient.nom, patient.prenom
FROM patient
JOIN consultation
ON patient.num_SS = consultation.num_SS
JOIN medecin
ON consultation.id_medecin = medecin.id_medecin
WHERE medecin.nom = ’ARNOS’
AND medecin.prenom = ‘Pierre’
ORDER BY patient.nom, patient.prenom
```

Partie C

Q9

```
def mdp_fort(mdp):
    if len(mdp) < 12 :
        return False
    majuscules = 0
    chiffres   = 0
    symboles   = 0
    for caractere in mdp :
        if caractere.isupper():
            majuscules += 1
        if caractere.isdigit():
            chiffres+=1
        if caractere in liste_symboles:
            symboles+=1
    if majuscules < 2 or chiffres < 2 or symboles < 2:
        return False
    return True
```

Q10

```
def creation_mdp(n, nbr_m, nbr_c, nbr_s):
    mdp = ''
    caracteres =' abcdefghijklmnopqrstuvwxyz' + \
        'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' + \
        '#@!?%<>=€$+-*/&'
    majuscules = 0
    chiffres   = 0
```

```
    symboles   = 0
    while len(mdp) < n or majuscules < 2 or chiffres < 2 or symboles < 2:
        # la variable 'c' contient un caractère
        # choisi aléatoirement dans la variable 'caracteres'
        c = choice(caracteres)
        if c.isupper():
            majuscules += 1
        if c.isdigit():
            chiffres+=1
        if c in liste_symboles:
            symboles+=1
        mdp = mdp + c
    return mdp
```

Q11

```
def recherche_mot(mdp):
    mot    = transforme(mdp)
    trouve = []
    i = 0
    while i < len(mot):
        if mot[i].isdigit(): # si le caractère est un chiffre
            i = i+1
        elif mot[i] in liste_symboles:
            i = i+1
        else:
            # si le caractère est une lettre, on prend les
            # lettres qui la suivent jusqu'au moment où
            # on trouve un chiffre ou un symbole
            chaine = ''
            while not(mot[i].isdigit() or mot[i] in liste_symboles):
                chaine = chaine + mot[i]
                i = i+1
            trouve.append(chaine)
    return trouve
```

Q12

```
def mdp_extra_fort(mdp : str) -> bool:
    """
    @param mdp -- une chaîne de caractères mdp
    @return True si mdp est un mot de passe extra fort et False sinon
    """
    for mot in recherche_mot(mdp):
        if len(mot) > 3:
            if mot in dicoFR:
                return False
    return True
```