Exercice 1 : Cœur de réseau
Partie A

1. **Attention :** La métrique est le nombre de routeur traversés (et non le nombre de saut).

Table de routage R_1		
Destination	Prochain saut	Distance
R_2	R_2	0
R_3	R_2	1
R_4	R_4	0
R_5	R_5	0
R_6	R_5	1

2. La requête d'un utilisateur du LAN1 vers INTERNET utilise selon le protocole RIP le chemin :
LAN1 - R1 - R5 - R6 – INTERNET

3.

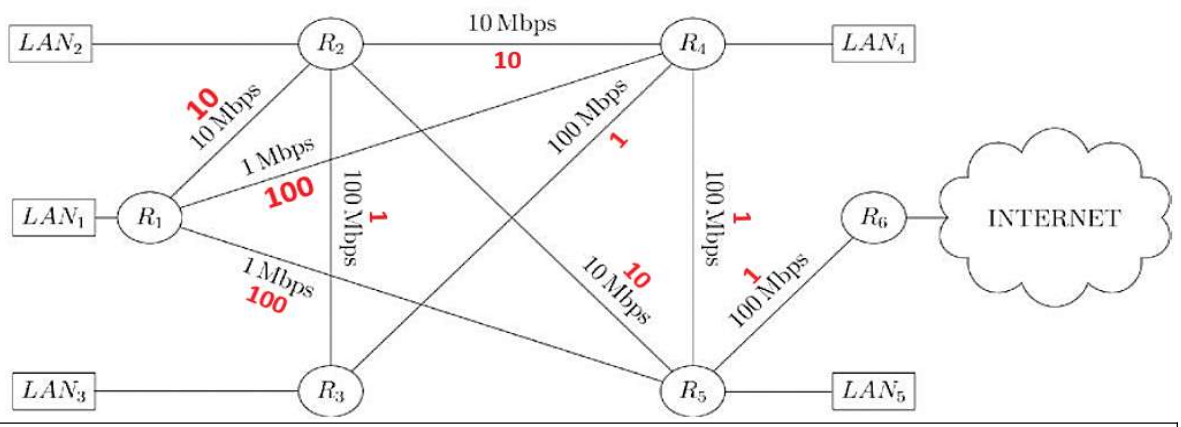

Table de routage R_1		
Destination	Prochain saut	Distance
R_2	R_2	10
R_3	R_2	11
R_4	R_2	12
R_5	R_2	13
R_6	R_2	14

4. La requête d'un utilisateur du LAN1 vers INTERNET utilise selon le protocole OSPF le chemin :
LAN1 – R1 - R2 - R3 – R4 - R5 – R6 – INTERNET ayant un coût/distance de 14.

5. Après une panne du routeur 2, la requête d'un utilisateur du LAN1 vers INTERNET utilise selon le protocole OSPF le chemin :
LAN1 – R1 - R5 – R6 – INTERNET ayant un coût/distance de 101.

Partie B

6. Détails des calculs : $1_{(10)} = 00000001_{(2)}$ et $100(10) = 64 + 32 + 4 = 01100100_{(2)}$

Calcul d'une adresse de réseau				
machine (déc. pointée)	192	168	1	100
machine (binaire)	11000000	10101000	**00000001**	**01100100**
masque (binaire)	11111111	11000000	00000000	00000000
réseau (binaire)	11000000	**10000000**	**00000000**	**00000000**
réseau (déc. pointée)	192	**128**	**0**	**0**

7. Adresse de Braodcast.
Détails des calculs : $10111111_{(2)} = 128+32+16+8+4+2+1$ ou $255-64 = 191_{(10)}$

Calcul d'une adresse de réseau				
réseau (binaire)	11000000	10000000	00000000	00000000
masque (binaire)	11111111	11000000	00000000	00000000
complément du masque (binaire)	**00000000**	**00111111**	**11111111**	**11111111**
broadcast (binaire)	**11000000**	**10111111**	**11111111**	**11111111**
broadcast (déc. pointée)	**192**	**191**	**255**	**255**

8.
Adresse du réseau LAN_1 : 172.16.0.0/16
Adresse de broadcast : 172.16.255.255/16
Nombre total d'adresses machines disponible : 2^{16} – 2 (reseau et broadcast) = 65534

9.

```
13      for i in range(4):
14          # Opération booléenne :
15          tmp.append(ip[i] & crible[i])
16      return ".".join([str(element) for element in tmp])
```

ou

```
15          tmp.append(str(ip[i] & crible[i]))
16      return ".".join(tmp)
```

Attention : Il faut convertir les entiers de la liste **tmp** en chaînes de caractères pour éviter l'erreur : **TypeError: sequence item 0: expected str instance, int found** de la méthode **join**.

10.

```
17    for index in range(4):
18        somme = liste_courante[3 - index] + retenue
19        valeur, retenue = somme%256, somme//256
20        liste_suivante = [str(valeur)] + liste_suivante
21    return '.'.join(liste_suivante)
```

Explication : On ajoute 1 au dernier octet (4e octet à droite). Si cet octet est égal à 256, on ramène la valeur à 0 (somme%256) et on retient 1 (somme//256) pour l'ajouter à l'octet suivant (3e octet).