Exercice 3 (8 points)

Cet exercice porte sur l'architecture matérielle (réseau), les arbres binaires de recherche et la programmation Python.

L'entreprise CaféNet possède plusieurs cafés répartis dans différentes villes. Le réseau de la chaîne de cafés est représenté en Figure 1.

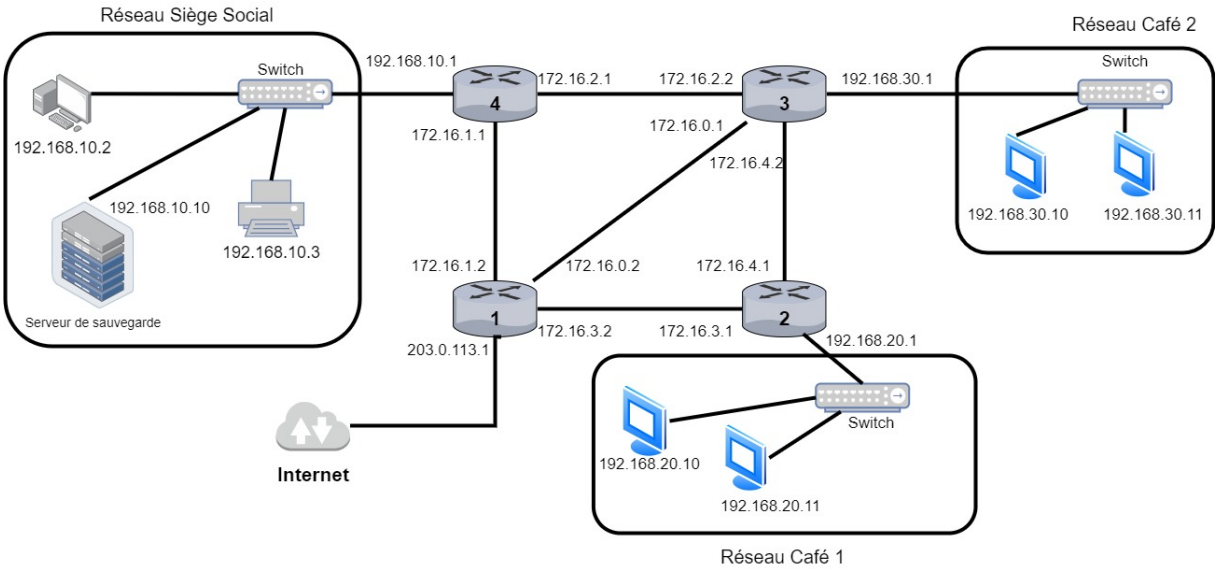


Figure 1. Schéma d'une partie du réseau

Sur le schéma sont représentés 4 routeurs, le réseau du siège social, le réseau du café 1, le réseau du café 2. Dans les réseaux du café 1 et du café 2, des bornes de commandes sont connectées à des switchs (ce sont des boitiers de connexion qui n'ont pas eux-mêmes d'adresse IP). Les 4 routeurs représentés sont composés d'au moins 3 interfaces réseau capable de relier des réseaux ensemble. Chaque interface possède donc une adresse IPV4 sur le réseau auquel elle est reliée.

Les masques des sous-réseaux sont tous 255.255.255.0. Avec ce masque, les trois premiers octets des adresses IP codent l'adresse réseau. Le dernier octet, c'est-à-dire les 8 derniers bits, code l'adresse des machines à l'intérieur de chaque sous-réseau.

Partie A

Le gérant veut faire installer une troisième borne de commande dans le café 1.

1. Indiquer les deux seules adresses IP valides pour cette nouvelle borne, parmi les quatre adresses IP proposées.

   (a) 192.168.20.2 (c) 192.168.20.261

   (b) 192.168.20.157 (d) 192.168.24.10

L’adresse de diffusion, appelée aussi adresse de broadcast, est la dernière adresse disponible à l’intérieur d’un réseau local.

2. Déterminer l’adresse de diffusion du réseau du café 1.

3. Déterminer combien de machines informatiques il est encore possible de connecter au réseau du café 1 après l’installation de la troisième borne de commande.

Le réseau local du café 1 n’a pas besoin de plus de 8 adresses IP différentes. Ce décompte d’adresses IP inclut les adresses IP réservées (à savoir l’adresse de diffusion et l’adresse du réseau). Il est rappelé que la longueur du masque de sous-réseau est actuellement de 24 bits (c’est-à-dire 3 octets).

4. Expliquer quelle est la longueur maximale du masque de sous-réseau que l’on pourrait choisir pour le réseau local du café 1.

Partie B

RIP (Routing Information Protocol) est un protocole de routage utilisé dans les réseaux IP. Il est conçu pour réduire le nombre de sauts entre deux réseaux. Un “saut” correspond au transfert des données d’un routeur à un autre. Le protocole RIP utilise le nombre de sauts comme critère principal pour évaluer le coût d’un chemin. Autrement dit, il considère que le chemin le plus optimal est celui qui traverse le moins de routeurs.

La table de routage du routeur 2 de la Figure 1 est représentée ci-dessous :

Routeur 2			
Réseau destination	Interface de sortie	Prochain routeur	Nombre de sauts
192.168.20.0	192.168.20.1	aucun	0
172.16.3.0	172.16.3.1	aucun	0
172.16.4.0	172.16.4.1	aucun	0
192.168.10.0	172.16.3.1	172.16.3.2	2
172.16.0.0	172.16.4.1	172.16.4.2	1
172.16.2.0	172.16.4.1	172.16.4.2	1
192.168.30.0	…	…	…
172.16.1.0	…	…	…

5. Recopier et compléter les deux dernières lignes de la table de routage du routeur 2.

La table de routage du routeur 2 contient un réseau de destination pour lequel deux routes différentes sont possibles. La ligne correspondante dans la table de routage aurait donc pu être remplie différemment tout en respectant le protocole RIP.

6. Identifier, dans la table de routage du routeur 2, le réseau de destination que l'on peut atteindre d'une autre façon et indiquer comment cette ligne de la table de routage pourrait être modifiée.

Une adresse IP qui n'est pas référencée dans la table de routage doit être routée par défaut vers Internet.

7. Recopier et compléter la ligne à ajouter à la table de routage du routeur 2.

Réseau destination	Interface de sortie	Prochain routeur
autre	…	…

Partie C

OSPF est également un protocole d'échanges de données entre les routeurs qui prend en compte le coût des routes. Le coût est lié au débit des liaisons entre les routeurs par la formule suivante :

$cout = \frac{10^9}{debit}$ avec le débit en $bit.s^{-1}$.

8. Recopier et compléter la dernière colonne du tableau ci-dessous :

Tableau des coûts		
Type de connexion	Débit en $bit.s^{-1}$	coût
Ethernet	$10\ Mbit.s^{-1} = 10^7\ bit.s^{-1}$	100
Fast Ethernet	$100\ Mbit.s^{-1} = 10^8\ bit.s^{-1}$	…
Fibre optique	$1\ Gbit.s^{-1} = 10^9\ bit.s^{-1}$	…

Le schéma ci-dessous met en évidence les types de connexion qui relient les routeurs.

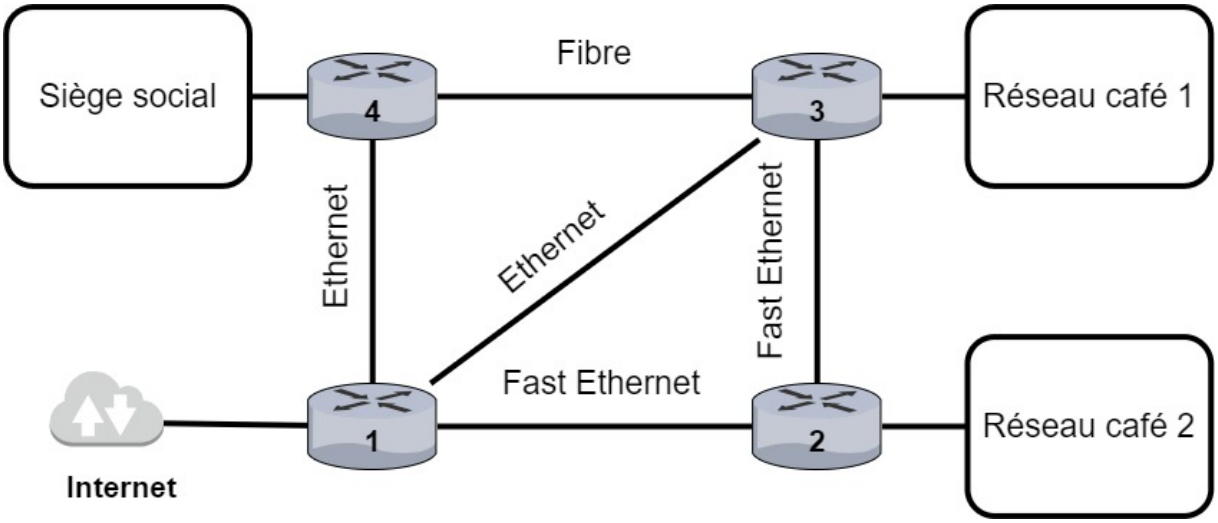


Figure 2. Schéma des types de connexion

9. Déterminer la route dont le coût est minimal pour aller du routeur 1 jusqu'au routeur 4 et calculer son coût au sens du protocole OSPF.

Partie D

Le but de cette partie est de classer les adresses IP des différents réseaux afin de faciliter leur recherche.

La fonction `ip_bin` prend en argument une chaîne de caractères décrivant une adresse IP en notation décimale, et renvoie une chaîne de caractères, de longueur 35 (32 bits et les 3 points), décrivant l'adresse IP en notation binaire.

Exemple :

```
>>> ip_bin('192.168.10.1')
'11000000.10101000.00001010.00000001'
```

10. Donner la chaîne de caractères renvoyée par `ip_bin('192.168.20.12')`.

La fonction `precede` prend en paramètres deux adresses IP en notation binaire, sous forme de chaînes de caractères identiques à celles renvoyées par la fonction `ip_bin`. La fonction `precede` renvoie un booléen qui vaut `True` si la première adresse IP en paramètre précède la seconde adresse IP.

Exemple :

```
>>> a = '11000000.10101000.00001010.00000001'
>>> b = '11000000.10101000.00001111.00000001'
>>> precede(a, b)
True
```

L'algorithme compare bit à bit les deux chaînes binaires, en lisant les chaînes de caractères dans le sens usuel (de gauche à droite). Dans l'exemple ci-dessus, tous les caractères sont identiques jusqu'au sixième caractère du troisième octet. Comme le bit de l'adresse a est inférieur à celui de l'adresse b, on en déduit que l'adresse IP a précède l'adresse IP b.

Si la première adresse IP ne précède pas la seconde, la fonction doit renvoyer `False`.

L'algorithme de comparaison est traduit dans le langage Python sous la forme suivante :

```
1 def precede(ip_1, ip_2):
2     for i in range(35):
3         if ip_1[i] < ip_2[i]:
4             return ...
5         elif ip_1[i] > ip_2[i]:
6             return ...
7     return ...
```

11. Expliquer dans quel cas la fonction `precede` exécutera la dernière instruction `return` de la ligne 7.

12. Recopier et compléter les lignes 4, 6 et 7 du code de la fonction `precede`.

Les tables de routage de chaque routeur sont implémentées sous la forme d'arbre binaire de recherche avec la classe `Abr`.

```
 1 class Abr:
 2     def __init__(self, adresse_ip,
 3                  interface, passerelle,
 4                  cout):
 5         self.adresse_ip = adresse_ip
 6         self.interface  = interface
 7         self.passerelle = passerelle
 8         self.cout       = cout
 9         if adresse_ip != '':
10             self.gauche = Abr('','','',0)
11             self.droite = Abr('','','',0)
12
13     def est_vide(self):
14         return ...
```

Dans cette représentation :

- `adresse_ip` désigne l'adresse IP de la destination ;
- `interface` désigne l'interface réseau ;
- `passerelle` désigne l'adresse IP du prochain routeur ;
- `cout` désigne le nombre de sauts pour atteindre la destination.

- par convention, l'arbre binaire vide est une instance de `Abr` pour laquelle `adresse_ip` est une chaîne de caractères vide ;
- un arbre binaire de recherche non vide possède nécessairement un sous-arbre gauche et un sous-arbre droit, éventuellement vides, qui sont tous les deux des arbres binaires de recherche. Ces sous-arbres sont désignés par `gauche` et `droite` dans la classe `Abr` ;
- si elle n'est pas vide, l'adresse IP du sous-arbre gauche précède l'adresse IP de l'instance parent ;
- si le sous-arbre droit n'est pas vide, alors l'adresse IP de l'instance parent précède l'adresse IP du sous-arbre droit.

13. Citer un attribut et citer une méthode de la classe `Abr`.

14. Recopier et compléter la ligne 14 du code de la classe `Abr`.

15. Justifier, en mobilisant des connaissances de cours, l'intérêt qu'il peut y avoir à représenter la table de routage par un arbre binaire de recherche.

La section de code qui définit `modifie` est incluse dans la classe `Abr`.

```
16      def modifie(self, adresse_ip,
17                  interface, passerelle,
18                  cout):
19          if self.est_vide():
20              self.adresse_ip = adresse_ip
21              self.interface  = interface
22              self.passerelle = passerelle
23              self.cout       = cout
24              self.gauche = Abr('','','',0)
25              self.droite = Abr('','','',0)
26          else:
27              self.adresse_ip = adresse_ip
28              self.interface  = interface
29              self.passerelle = passerelle
30              self.cout       = cout
```

Les lignes 20 à 23 sont exactement les mêmes que les lignes 27 à 30.

16. Réécrire le code de la fonction `modifie` en évitant cette répétition.

La classe `Abr` est complétée afin de permettre l'ajout de nouvelles lignes à la table de routage, tout en conservant les propriétés que doit posséder un arbre binaire de recherche.

```
32      def rechercher(self, adresse_ip):
33          if self.est_vide() or adresse_ip==self.adresse_ip:
34              return self
35          elif precede(...):
36              return self.gauche.rechercher(adresse_ip)
37          else:
38              return self.droite.rechercher(adresse_ip)
39
40      def inserer(self, adresse_ip,
41                  interface, passerelle,
42                  cout):
43          destination = self.rechercher(adresse_ip)
44          destination.modifie(adresse_ip,
45                              interface, passerelle,
46                              cout)
```

On rappelle que la fonction `precede` prend en arguments des adresses IP écrites sous forme binaire.

17. Recopier et compléter la ligne 35 du code de la fonction `rechercher`.