Exercice 3 (8 points)

Cet exercice porte sur la programmation Python, les graphes et les réseaux.

Partie A

On considère un réseau d'antennes radios, représenté dans la figure 1, où les disques représentent la zone d'émission de chaque antenne. Pour éviter toute interférence, deux antennes "proches" géographiquement doivent émettre à des fréquences différentes.

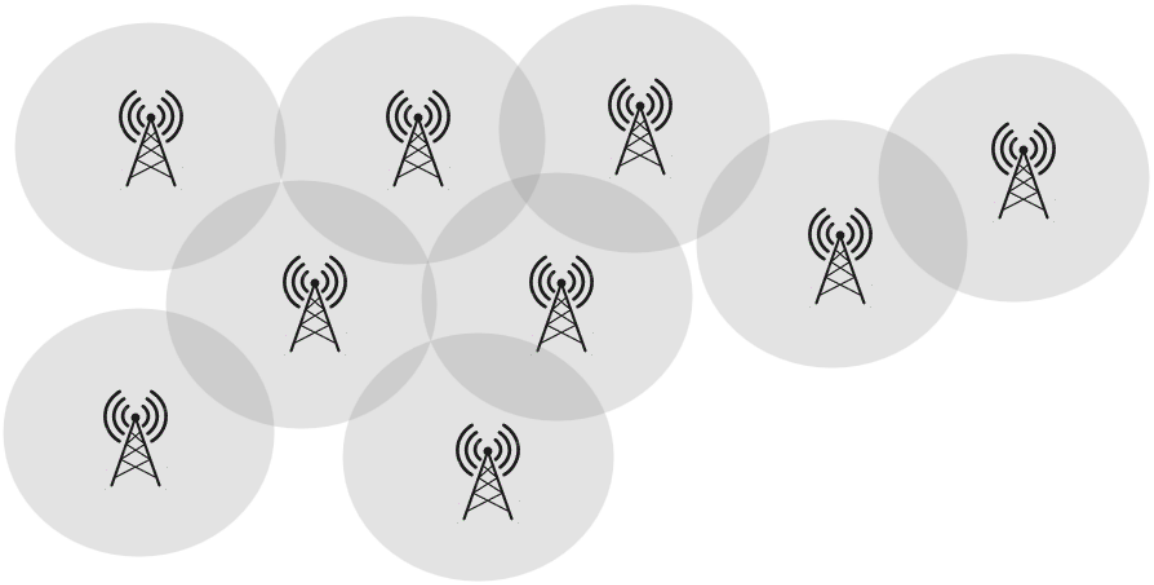

Figure 1. Réseau d'antennes

On modélise ainsi le réseau d'antennes par un graphe non orienté, appelé graphe d'interférences, dont les sommets sont les antennes numérotées de 1 à n, n étant un entier naturel supérieur ou égal à 1, et les sommets sont reliés par une arête si leurs zones d'émission s'intersectent.

Soit `G` le graphe associé au réseau d'antennes précédent :

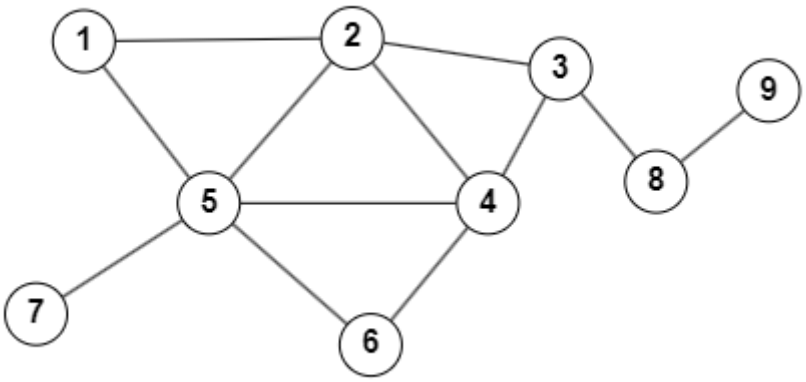

Figure 2. Modélisation du réseau sous la forme d'un graphe G

Les fréquences à allouer sont associées à des couleurs comme rouge, vert, jaune, bleu, etc.
Pour éviter les interférences, la coloration doit être une coloration propre : deux sommets adjacents ne peuvent recevoir la même couleur.

Dans cet exercice on représente le graphe `G` par un dictionnaire de listes d'adjacence dont les clefs sont les sommets de type `int` et les valeurs sont des listes de voisins du sommet clef, chaque liste contenant des éléments de type `int`.

1. Donner la valeur associée à la clé `1` dans ce dictionnaire.

2. Écrire une fonction `voisins`, qui prend en paramètres un dictionnaire et un entier, telle que `voisins(graphe, k)` renvoie une liste contenant les voisins du sommet `k` dans le graphe qui est modélisé par le dictionnaire de listes d'adjacence `graphe`.

   Exemple :

   ```
   >>> voisins(G, 2)
   [1, 3, 4, 5]
   ```

L'algorithme de Welsh et Powell consiste à colorer séquentiellement le graphe en visitant les sommets par ordre de degrés décroissants. Le degré d'un sommet d'un graphe non orienté est le nombre d'arêtes dont le sommet est une extrémité. L'idée est que les sommets ayant beaucoup de voisins sont plus difficiles à colorer : il faut les colorier en premier.

3. Écrire la fonction `degre_du_sommet` qui prend en paramètres un graphe modélisé par le dictionnaire de listes d'adjacence `graphe` et un sommet `sommet` et qui renvoie le degré du sommet `sommet`.

Exemple :

```
>>> degre_du_sommet(G, 2)
4
```

4. Écrire la fonction `degre_sommets` qui prend en paramètre un graphe modélisé par le dictionnaire de listes d'adjacence `graphe` et qui renvoie la liste des tuples `(sommet,degre)` de chaque sommet du graphe.

   Exemple :

```
>>> degre_sommets(G)
[(1, 2), (2, 4), (3, 3), (4, 4), (5, 5), (6, 2), (7, 1),
(8, 2), (9, 1)]
```

On définit la fonction `tri_liste` ci-après :

```
 1 def tri_liste(l_deg):
 2   """l_deg : liste de tuples (sommets,degré).
 3   Trie la liste l_deg par degrés décroissants"""
 4   for i in range(len(l_deg)+1):
 5     som_max = i
 6     deg_max = l_deg[i][1]
 7
 8     for j in range(i+1, len(l_deg)):
 9       if deg_max < l_deg[j][1]:
10         som_max = j
11         deg_max = l_deg[j][1]
12     temp = l_deg[i]
13     l_deg[i] = l_deg[som_max]
14     l_deg[som_max] = temp
15   return l_deg
```

À l'exécution, `tri_liste([(1, 2), (2, 2), (3, 3)])` renvoie l'erreur suivante :

```
IndexError: list index out of range
```

5. Commenter puis corriger cette erreur.

6. Choisir parmi les tris proposés celui qui correspond à la fonction `tri_liste` : tri par insertion, tri par sélection, tri fusion, tri bulle.

7. Écrire une fonction `tri_sommets` qui prend en paramètre un graphe `graphe` et qui ne renvoie que la liste des sommets du graphe `graphe` triés par degré décroissant. On pourra utiliser les fonctions définies dans les questions précédentes.

Exemple :

```
>>> tri_sommets(G)
[5, 2, 4, 3, 1, 6, 8, 7,9]
```

On suppose que le graphe est planaire, c'est-à-dire qu'il existe une représentation de ce graphe dans un plan pour laquelle les arêtes ne se croisent pas, et on définit la fonction coloration ci-après.

```
 1 def coloration(g):
 2     """Renvoie une coloration du graphe g"""
 3     # Algorithme de Welsh-Powell, limité à 4 couleurs
 4
 5     couleur = ['Rouge', 'Bleu', 'Vert', 'Jaune']
 6     coloration_sommets = {}
 7     for s_i in g:
 8         coloration_sommets[s_i] = None
 9     for s_i in tri_sommets(g):
10         couleurs_voisins_s_i = [coloration_sommets[s_j] for
s_j in voisins(g, s_i)]
11         k = 0
12         while couleur[k] in couleurs_voisins_s_i :
13             k = k + 1
14         coloration_sommets[s_i] = couleur[k]
15     return coloration_sommets
```

8. Donner le type et le contenu de la variable `coloration_sommets` de la fonction `coloration_graphe` ci-dessus pour le graphe `G`, après execution de la boucle des lignes 7 et 8.

9. Recopier et compléter le retour de la fonction `coloration` appliquée au graphe `G` donné plus haut.

   ```
   {1: 'Vert', 2: ..., ...}
   ```

Partie B

On s'intéresse maintenant à un réseau informatique.

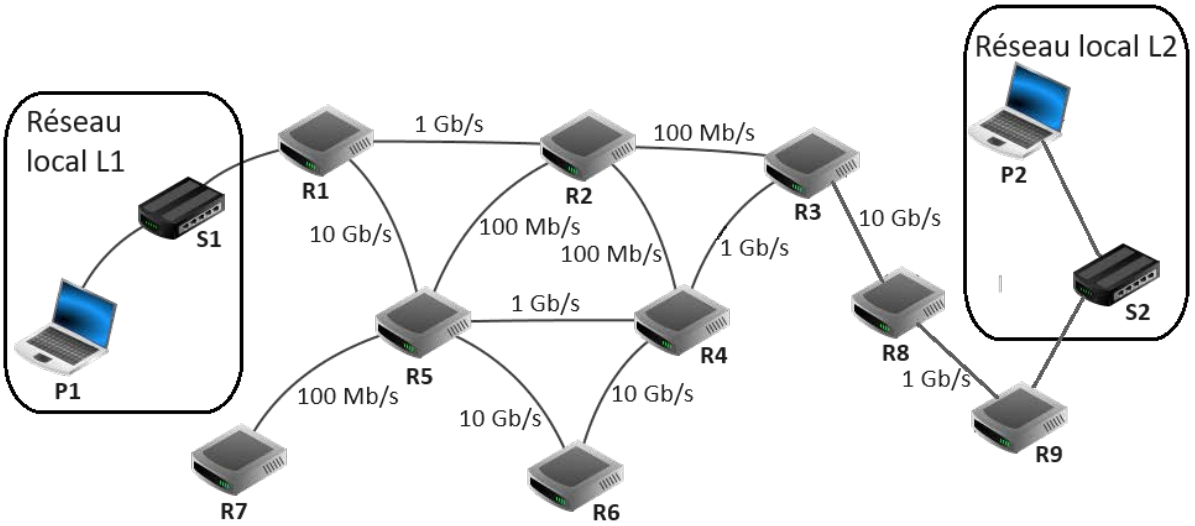

Figure 3. Réseau entreprise

Dans cette partie, les adresses IP sont composées de 4 octets, soit 32 bits. Elles sont notées X1.X2.X3.X4, où X1, X2, X3 et X4 sont les représentations décimales des 4 octets. La notation X1.X2.X3.X4/n signifie que les n premiers bits de l'adresse IP représentent la partie « réseau », les bits suivants représentent la partie « hôte ».

On fournit les données suivantes concernant le réseau de cette entreprise.

Réseau local L1 :
- Adresse IP de l'ordinateur P1 : 190.12.10.25/24
- S1 : switch

Réseau local L2 :
- Adresse réseau : 12.128.0.0
- Masque de sous réseau : 255.255.0.0
- S2 : switch
- P2 : ordinateur

Extrait de l'arborescence du système de fichiers de l'ordinateur P2 :

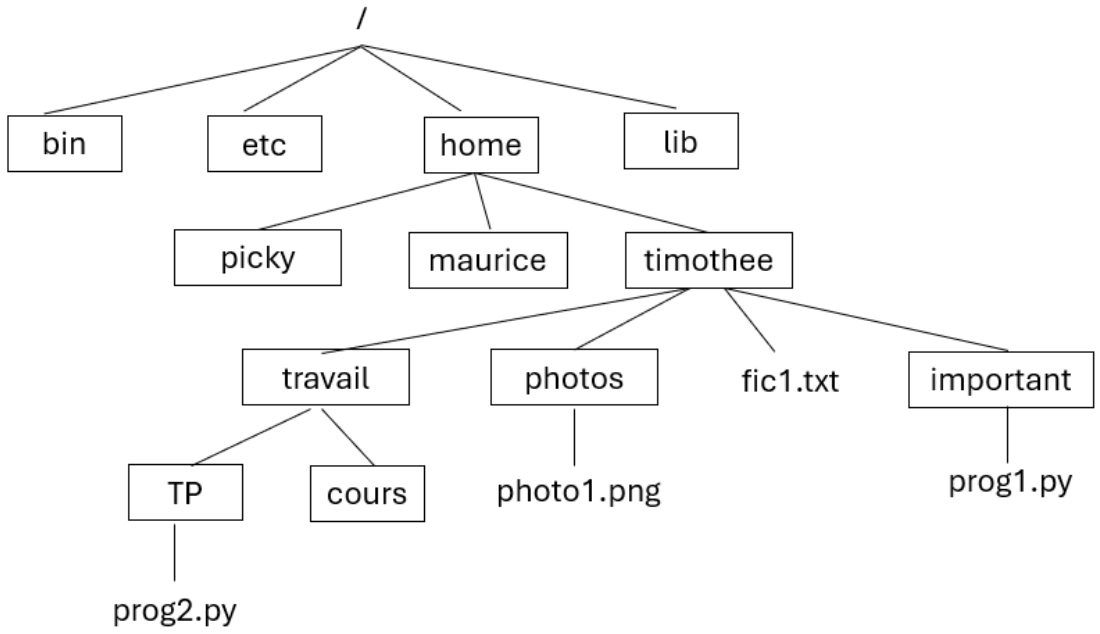

Figure 4. Arborescence

Extrait du manuel de la commande cp :

```
[root@localhost ~]# man cp
CP(1)                          User Commands                          CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
```

Figure 5. Manuel de la commande cp

10. Donner une commande en ligne qui permet de copier le fichier `prog1.py` dans le répertoire `TP` lorsqu'on se trouve dans le répertoire nommé `important`.

11. Donner la commande qui permet de vérifier si l'ordinateur P1 est accessible lorsque l'on travaille sur l'ordinateur P2.

12. Donner une adresse possible pour l'ordinateur P2 du réseau local L2.

Dans le cadre du protocole RIP, le chemin emprunté par les informations est celui qui aura la distance la plus petite en nombre de sauts.
Dans le cadre du protocole OSPF, le chemin emprunté par les informations est celui qui aura le coût total minimal.

Extraits des tables de routage :

Routeur	Destination	Passerelle
R1	R9	R2
R2	R9	R3
R3	R9	R8
R4	R9	R3
R5	R9	R4
R6	R9	R4
R7	R9	R5
R8	R9	R9
R9	R9	LOCALHOST

13. Donner le chemin emprunté par un paquet de données allant de l'ordinateur P1 à l'ordinateur P2, en utilisant l'extrait de la table de routage.

14. Donner le nom du protocole de routage qui semble être utilisé.

Dans les questions suivantes, on utilise le protocole de routage OSPF.

Pour calculer le coût C d'une liaison, on utilise la formule : $C = \frac{10^8}{BP}$ où BP est la bande passante en bits par seconde.

15. Calculer les coûts pour des liaisons de 100 Mbits/s, 1 Gbits/s et 10 Gbits/s.

16. Déterminer la route qui sera empruntée par le paquet de données envoyé de l'ordinateur P1 à l'ordinateur P2, en respectant le protocole OSPF.