NOUVELLE CALÉDONIE – 2025 – sujet1

Exercice 1 (6 points)

Cet exercice porte sur les graphes, les protocoles réseaux et la programmation orientée objet.

PARTIE A

Le graphe suivant modélise un ensemble de routeurs ; les sommets sont les routeurs, les arêtes les liaisons entre ceux-ci.

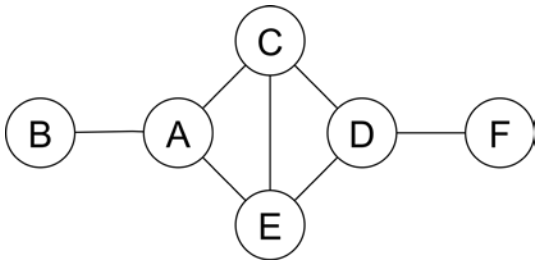

Figure 1. Schéma des routeurs et des liaisons On désire

parcourir ce graphe *en largeur* depuis le sommet A.

1. Dire lequel de ces parcours est un parcours en largeur en justifiant :
    - ABCDEF ;
    - ABCEDF ;
    - ABCDFE.

Voici un résumé sommaire du fonctionnement du protocole RIP permettant à chaque routeur d'un réseau de taille modérée d'établir sa table de routage :

Règle a (*règle d'initialisation*). Chaque routeur initialise sa table en y ajoutant ses voisins directs. Ils sont accessibles en un saut, sans passer par aucun routeur intermédiaire.

Règle b (*règle de transmission/réception*). À intervalles de temps réguliers chaque routeur envoie sa table de routage à ses voisins.

Règle c (*règle de mise à jour*). Lorsqu'un routeur reçoit les informations d'un routeur voisin, trois cas peuvent survenir :

- une route vers un nouveau routeur lui est présentée : il l'ajoute à sa table de routage ;
- une route vers un routeur déjà connu lui est présentée, plus longue en nombre de sauts que celle inscrite dans sa table : elle est ignorée ;
- une route vers un routeur déjà connu lui est présentée, mais strictement plus courte en nombre de sauts que la précédente : l'ancienne est remplacée par celle-ci.

La réception par chaque routeur des tables de tous ses voisins et la mise à jour de sa table de routage en conséquence constitue une *itération* du protocole. Au bout d'un petit nombre de ces itérations, plus aucune table de routage ne varie, on dit que le processus est *stabilisé*.

Pour tout cet exercice, on n'envisagera pas les cas problématiques dans lesquels une liaison est coupée ou un routeur tombe en panne.

On considère des routeurs A, B, C, D, E et F connectés comme indiqué sur le graphe de la figure 1.

Voici la table de routage de A à l'initialisation du protocole RIP :

Table de routage de A		
routeur	nombre de sauts	prochain routeur
B	1	–
C	1	–
E	1	–

2. Donner la table de routage de F à l'initialisation du protocole RIP.
3. Donner la table de routage de A après une première itération de RIP (deux réponses sont possibles).
4. Donner le numéro de l'itération de RIP à partir duquel les tables des routeurs du réseau ne varient plus.

On suppose dans la question suivante que les routeurs E et F sont reliés.

5. Donner la nouvelle table de routage de A après stabilisation de RIP (deux réponses sont possibles).

PARTIE B

Pour simuler la situation précédente et les tables de routage, on modélise le fonctionnement d'un routeur par la classe `Routeur`. Chaque instance `r` de la classe `Routeur` possède quatre attributs.

- `nom` : une chaine de caractères qui identifie le routeur.
- `voisins` : une liste d'objets de type `Routeur`. Il s'agit de routeurs qui sont directement connectés au routeur `r`.
- `nb_sauts` : un dictionnaire qui associe à chaque routeur accessible depuis `r` le nombre de sauts nécessaires pour l'atteindre depuis `r`.
- `prochain` : un dictionnaire qui associe à chaque routeur `r_accessible`, accessible depuis `r`, le premier routeur sur un chemin qui mène à `r_accessible`, en `n` sauts, où `n` est la valeur associée à `r_accessible` dans `nb_sauts`. S'il y a un unique saut de `r` à `r_accessible`, alors la valeur associée au routeur `r_accessible` est `None`.

Initialement, tous les routeurs sont déconnectés : l'attribut `voisins` est initialisé avec la liste vide, et les attributs `nb_sauts` et `prochain` avec le dictionnaire vide.

```
class Routeur:
    def __init__(self, nom):
        self.nom = nom
        self.voisins = []
        self.nb_sauts = {}
        self.prochain = {}
```

Dans le programme principal, on crée les routeurs de la manière suivante :

```
A = Routeur('A')
B = Routeur('B')
C = Routeur('C')
D = Routeur('D')
E = Routeur('E')
F = Routeur('F')
```

Ainsi que la liste des routeurs

```
liste_routeurs = [A, B, C, D, E, F]
```

Afin de pouvoir relier les routeurs entre eux, on souhaite écrire une méthode `relie`, de la classe `Routeur`, dont on donne le code incomplet ci-dessous. Cette méthode prend en argument le routeur `self` ainsi qu'un routeur `autre` et met à jour si nécessaire les attributs `voisins`, `nb_sauts` et `prochain` des routeurs `self` et `autre` afin d'indiquer la présence d'une connexion entre ces deux routeurs. Dans le cas où les routeurs sont déjà connectés, cette méthode ne fait rien.

```
1      def relie(self, autre):
2          if autre not in self.voisins:
3              self.voisins.append(...)
4              self.nb_sauts[autre] = ...
5              self.prochain[autre] = ...
6              if self not in autre.voisins:
7                  autre.relie(...)
8
```

6. Recopier et compléter le code de la méthode `relie`.

7. Écrire la méthode `relie_liste` de la classe `Routeur` qui prend en paramètre une liste de routeurs `lst` et qui relie le routeur `self` à chacun des routeurs de la liste `lst`.

Par exemple, pour relier le routeur `A` aux routeurs `B`, `C` et `E`, on exécute l'instruction :

```
A.relie_liste([B, C, E])
# On n'appelle pas B.relie(A) car la liaison est déjà faite
```

8. Écrire les instructions manquantes pour relier les routeurs de manière à obtenir le graphe de la figure 1.

D'après la règle *c* (règle de mise à jour) du protocole RIP, lorsqu'un routeur reçoit les informations d'un routeur voisin, il doit mettre à jour sa table de routage. On donne cidessous le code incomplet de la méthode `met_a_jour_table` qui implémente la règle *c* du protocole RIP.

```
def met_a_jour_table(self, autre):
    for r in autre.nb_sauts:
        if r != self:
            if (r not in self.nb_sauts or
                self.nb_sauts[r] > ...):
                self.nb_sauts[r] = ...
                self.prochain[r] = ...
```

9. Recopier et compléter le code de la méthode `met_a_jour_table` ci-dessus.
10. Écrire la méthode `itere_rip` qui prend en paramètre le routeur `self` et met à jour sa table de routage lorsqu'il reçoit la table de routage de chacun des routeurs présents dans la liste de ses voisins.
11. Écrire une fonction qui prend en paramètre une liste de routeurs `l_routeurs` et qui réalise une itération du protocole RIP pour tous les routeurs de `l_routeurs`.

Au bout de quelques itérations, le protocole RIP converge : plus aucune table de routage du réseau n'est modifiée. On aimerait pouvoir itérer le protocole dans le programme principal jusqu'à ce que ce soit le cas, à l'aide d'une boucle `while`.

On suppose que la méthode `met_a_jour_table` de la classe `Routeur` a été modifiée de telle sorte qu'elle renvoie `True` dans le cas où le routeur `self` a procédé à une mise à jour de sa table de routage, et `False` sinon.

12. Écrire une version modifiée du code de la méthode `itere_rip` de la classe `Routeur` de telle sorte que celle-ci renvoie `True` dans le cas où le routeur `self` a procédé à une modification de sa table de routage au cours de l'exécution de la méthode `itere_rip`, et `False` sinon.

On donne ci-dessous le code du programme principal. On suppose que les instructions permettant de relier les routeurs ont été écrites à la suite et que la situation est celle décrite dans le graphe de la figure 1.

```
A = Routeur('A')
B = Routeur('B')
C = Routeur('C')
D = Routeur('D')
E = Routeur('E') F = Routeur('F')
liste_routeurs = [A, B, C, D, E, F]
# instructions permettant de relier les routeurs
```

13. Compléter le code du programme principal afin que celui-ci mette à jour les tables de routage des routeurs présents dans la liste `liste_routeurs` jusqu'à ce qu'il ne soit plus nécessaire de faire des mises à jour des tables de routage.

On ne demande pas de réécrire les instructions permettant de connecter les routeurs entre eux.