Nouvelle Calédonie – 2025 – sujet1 - Correction

Exercice 1 (6 points)

PARTIE A — Graphes & RIP

1) Parcours en largeur depuis A

Un **parcours en largeur (BFS)** explore :

- d'abord les voisins directs,
- puis les voisins des voisins,
- etc.

D'après la table initiale de A, ses voisins directs sont :
B, C et E

Donc le parcours doit commencer par : `A → B C E`

Ensuite viennent les sommets à distance 2.

Parmi les propositions :

- ❌ ABCDEF
- ❌ ABCEDF
- ✅ ABCDFE

👉 **Réponse correcte : ABCDFE**

Justification :

Après A :

- niveau 1 : B, C, E
- niveau 2 : D, F

Le BFS impose que tous les sommets du niveau 1 soient visités avant ceux du niveau 2.

---

2) Table de routage de F à l'initialisation

À l'initialisation (règle a), un routeur ne connaît que ses voisins directs.

D'après le graphe (figure 1) :

F est relié à D uniquement.

Donc :

routeur	nombre de sauts	prochain routeur
D	1	–

👉*Commentaire : On met « – » ou* `None` *car il n'y a pas d'intermédiaire.*

---

3) Table de A après 1 itération

A connaît initialement :

B	1
C	1
E	1

Après réception des tables :

- B peut connaître D
- C peut connaître D
- E peut connaître D

Donc A apprend qu'il peut atteindre D en 2 sauts.

Deux réponses possibles car deux chemins possibles :

routeur	sauts	prochain
B	1	–
C	1	–
E	1	–
D	2	B

ou

| D | 2 | C |

👉*Commentaire : RIP ne garantit pas l'unicité du choix si les distances sont égales.*

---

4) Stabilisation

Le graphe a un diamètre de 3.

Donc convergence en **2 itérations**.

👉 Réponse : à partir de la **2e itération** les tables ne varient plus.

---

5) Si E et F sont reliés

Nouveau chemin A → E → F = 2 sauts

Au lieu de 3 via D.

Donc la table finale de A devient :

routeur	sauts	prochain
B	1	–
C	1	–
E	1	–
D	2	B ou C
F	2	E

PARTIE B — Programmation orientée objet

6) Méthode relie

```
def relie(self, autre):

    if autre not in self.voisins:

        self.voisins.append(autre)
        autre.voisins.append(self)

        self.nb_sauts[autre] = 1
        autre.nb_sauts[self] = 1

        self.prochain[autre] = None
        autre.prochain[self] = None
```

👉*Commentaire :*

- *On évite les doublons*
- *Connexion bidirectionnelle*
- *Distance 1*
- *Pas d'intermédiaire*

---

7) Méthode relie_liste

```
def relie_liste(self, lst):
    for r in lst:
        self.relie(r)
```

---

8) Connexions du graphe

```
A.relie_liste([B, C, E])
B.relie(D)
C.relie(D)
D.relie(F)
```

---

9) Méthode met_a_jour_table

Principe : appliquer règle c.

```
def met_a_jour_table(self, voisin):

    modif = False

    for r in voisin.nb_sauts:

        distance = voisin.nb_sauts[r] + 1

        if r not in self.nb_sauts:
            self.nb_sauts[r] = distance
            self.prochain[r] = voisin
            modif = True

        elif distance < self.nb_sauts[r]:
            self.nb_sauts[r] = distance
            self.prochain[r] = voisin
            modif = True

    return modif
```

---

9) Méthode itere_rip

```
def itere_rip(self):

    modif = False

    for v in self.voisins:
        if self.met_a_jour_table(v):
            modif = True

    return modif
```

---

12) Itération globale

```
def iteration_globale(l_routeurs):

    modif = False

    for r in l_routeurs:
        if r.itere_rip():
            modif = True

    return modif
```

---

13) Programme principal (boucle de convergence)

```
modification = True

while modification:
    modification = False
    for r in liste_routeurs:
        if r.itere_rip():
            modification = True
```
