Sujet 0 - 2024 - sujet 1 (corrigé)

Exercice 1 (Architecture matérielle, réseaux, protocoles de routage)

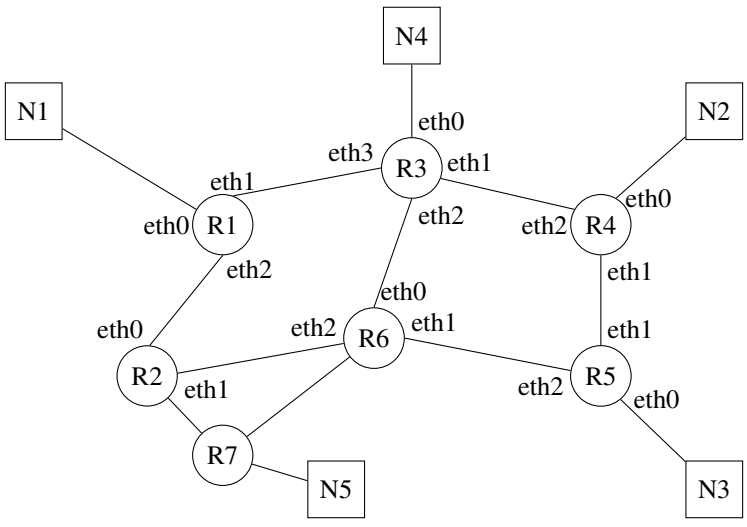
1. M1 a pour adresse réseau `192.168.1.0` alors que M3 a pour adresse réseau `192.168.2.0`. Ces deux ordinateurs ne sont donc pas sur le même réseau local. Ils ne peuvent donc pas communiquer directement, d'où le résultat « Hôte inaccessible ».
2. L'acronyme RAM signifie Random Access Memory. Il s'agit de la mémoire vive d'un ordinateur. Cette mémoire permet de stocker les données et les programmes au cours de leur exécution. Il s'agit d'une mémoire volatile.
3. Linux, plus précisément GNU/Linux, est un système d'exploitation libre.
4. Un routeur permet de relier au moins deux réseaux locaux entre eux. Il faut autant d'interfaces réseaux qu'il y a de réseaux locaux à relier. Il faut donc au minimum deux interfaces réseau.
5. Une adresse IP possible pour eth0 (mais il y en a beaucoup d'autres) : `192.168.1.254`
6. On a le chemin suivant : N1 - R1 - R3 - R4 - N2
7. On a la table suivante :

Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth2	4
N3	eth2	3

8. On a directement les coûts suivants :
   - ⋆ Fibre : $0{,}1$;
   - ⋆ Fast-Ethernet : 1 ;
   - ⋆ Ethernet : 10.
9. Pour la liaison R1 - R2 - R6 - R5 - N3, d'après la table de routage de R1, on a un coût de $0{,}3$. On a donc $0{,}1 + x + 0{,}1 = 0{,}3$, et on en déduit donc que $x = 0{,}1$. Ainsi, le coût de la liaison R2 - R6 est de $0{,}1$ et nous avons une liaison de type Fibre.
10. On a la table de routage suivante :

Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth1	0,2
N2	eth2	1,3
N3	eth1	1,2
N3	eth2	0,3
N4	eth1	0,1
N4	eth2	1,2

11. On a le graphe suivant :



Exercice 2 (Listes, dictionnaires et programmation Python)

1. La fonction suivante convient :

```
def corrige(cop, corr):
    c = []
    for i in range(len(corr)):
        c.append(cop[i] == corr[i])
    return c
```

2. La fonction suivante convient :

```
def note(cop, corr):
    note = 0
    for i in range(len(corr)):
        if cop[i] == corr[i]:
            note = note + 1
    return note
```

3. La fonction suivante convient :

```
def note_paquet(p, corr):
    d = {}
    for k,v in p.items():
        d[k] = note(v, corr)
    return d
```

4. Les clés d'un dictionnaire ne doivent pas être des valeurs qui peuvent être modifiées (valeurs mutables). Les listes Python étant mutables, il n'est pas possible d'utiliser une liste Python comme clé d'un dictionnaire.
5. Il est possible d'attribuer un identifiant (`id`) unique à chaque candidat et d'utiliser cet identifiant comme clé.
6. La fonction renvoie le tuple suivant :

```
((('Tom', 'Matt'), 6), (('Lambert', 'Ginne'), 4), (('Kurt', 'Jett'), 4),
{('Carl', 'Roth'): 2, ('Ayet', 'Finzerb'): 3})
```

7. La fonction `enigme` renvoie un tuple contenant :
   - les 3 meilleurs candidats sous forme de trois tuples (classés dans un ordre décroissant de note) ;
   - un dictionnaire contenant les autres candidats.
8. Si par exemple, on a seulement deux candidats, la fonction renvoie un tuple contenant :
   - les deux candidats sous forme de deux tuples (classés dans un ordre décroissant de note) ;
   - `None` ;
   - un dictionnaire vide.
9. La fonction suivante convient :

```
def classement(d):
    t = []
    while len(d) != 0:
        l = enigme(d)
        a = l[0]
        b = l[1]
        c = l[2]
        d = l[3]
        if a != None:
            t.append(a)
        if b != None:
            t.append(b)
        if c != None:
            t.append(c)
    return t
```

10. On a le code suivant :

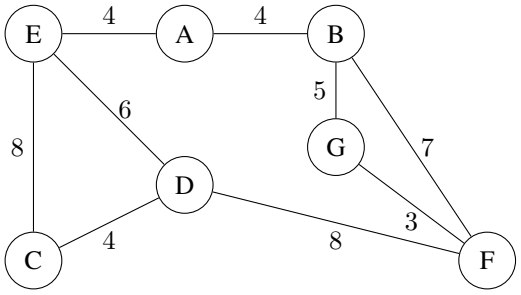
```
1  def renote_express2(copcorr) :
2      gauche = 0
3      droite = len(copcorr)
4      while droite - gauche > 1 :
5          milieu = (gauche + droite)//2
6          if copcorr[milieu] :
7              gauche = milieu
8          else :
9              droite = milieu
10     if copcorr[gauche] :
11         return droite
12     else :
13         return gauche
```

11. Le coût en temps de `renote_express` est en $O(n)$ (linéaire), alors que le coût en temps de `renote_express2` est en $O(\log_2(n))$ (logarithmique).
12. La fonction prendra deux paramètres `cop` et `corr` et il faudra modifier les lignes 3, 6 et 10 de la façon suivante :

```
1  def renote_express3(cop, corr) :
2      gauche = 0
3      droite = len(cop)
4      while droite - gauche > 1 :
5          milieu = (gauche + droite)//2
6          if cop[milieu] == corr[milieu] :
7              gauche = milieu
8          else :
9              droite = milieu
10     if cop[gauche] :
11         return droite
12     else :
13         return gauche
```

Exercice 3 (Graphes et bases de données)

1. On a le graphe G1 suivant :



2. On obtient le chemin A - E - D avec une distance de 10 km.
3. On a la matrice d'adjacence suivante :

$$\begin{pmatrix} 0 & 4 & 0 & 0 & 4 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 4 & 8 & 0 & 0 \\ 0 & 0 & 4 & 0 & 6 & 8 & 0 \\ 4 & 0 & 8 & 6 & 0 & 0 & 0 \\ 0 & 7 & 0 & 8 & 0 & 0 & 3 \\ 0 & 5 & 0 & 0 & 0 & 3 & 0 \end{pmatrix}$$

4. L'implémentation suivant convient :

```
d = {'A':['B', 'C', 'H'], 'B':['A', 'I'], 'C':['A', 'D', 'E'], 'D':['C', 'E'],
     'E': ['C', 'D', 'G'], 'F':['G', 'I'], 'G':['E', 'F', 'H'], 'H':['A', 'G', 'I'],
     'I':['B', 'F', 'H']}
```

5. Le parcours en largeur suivant est possible : A, B, C, H, D, E, G, I, F.
6. A la ligne 8, la fonction `cherche_itineraires` s'appelle elle-même, donc elle peut donc être qualifiée de fonction récursive.
7. La fonction `cherche_itineraires` permet de recenser tous les itinéraires (tous les chemins) pour aller de la ville `start` à la ville `end`.
8. La fonction suivante convient :

```
def itineraires_court(G,dep,arr):
    cherche_itineraires(G, dep, arr)
    tab_court = []
    mini = float('inf')
    for v in tab_itineraires:
        if len(v) <= mini :
            mini = len(v)
    for v in tab_itineraires:
        if len(v) == mini:
            tab_court.append(v)
    return tab_court
```

9. Le problème vient de la liste Python `tab_itineraires`. Cette liste est vide au début de l'exécution du programme `p1` (ligne 1). Tous les itinéraires possibles pour aller de la ville `start` à la ville `end` sont ajoutés à cette liste grâce à l'appel à la fonction `cherche_itineraires` (ligne 14).
   A la suite de l'appel de la fonction `itineraires_court(G2, 'A', 'E')` depuis la console, la liste `tab_itineraires` contient les itinéraires suivants :

   `[['A', 'B', 'I', 'F', 'G', 'E'], ['A', 'B', 'I', 'H', 'G', 'E'], ['A', 'C', 'D', 'E'], ['A', 'C', 'E'], ['A', 'H', 'G', 'E'], ['A', 'H', 'I', 'F', 'G', 'E']]`

   Quand on appelle la fonction `itineraires_court(G2, 'A', 'F')` toujours depuis la console, la liste `tab_itineraires` n'a pas été « vidée » (puisque le programme `p1` n'a pas été exécuté de nouveau), elle contient donc toujours, entre autres, l'itinéraire `['A', 'C', 'E']`. La fonction `itineraires_court` permet de choisir l'itinéraire le plus court parmi ceux qui se trouvent dans la liste `tab_itineraires`. Sachant que les itinéraires pour aller de A à F ont au minimum une taille de 4, c'est donc l'itinéraire `['A', 'C', 'E']` qui est renvoyé par la fonction, d'où le problème constaté.
   L'exécution du programme `p1` entre les deux appels fait disparaitre le problème puisque la liste est vidée lors de la seconde exécution.

10. L'utilisation d'un SGBD apporte beaucoup d'avantage par rapport à l'utilisation d'un simple fichier au format texte :
    - ⋆ les SGBD permettent de gérer les autorisations d'accès ;
    - ⋆ les accès concurrents sont gérés par les SGBD ;
    - ⋆ les SGBD proposent aussi de gérer la redondance des données (pour palier aux problèmes de pannes ou de surcharges).
11. On a le schéma relationnel suivant :

    `ville(id : INT, nom : TEXT, num_dep : INT, nombre_hab : INT, superficie : FLOAT)`
12. `id_ville` est une clé étrangère, elle permet d'effectuer une jointure entre la table `sport` et la table `ville`.
13. Cette requête renvoie `Chamonix`
14. La requête suivante convient :

```sql
SELECT nom FROM sport WHERE type = 'piscine';
```

15. La requête suivante convient :

```sql
UPDATE sport SET note = 7 WHERE nom = 'Ballon perdus';
```

16. La requête suivante convient :

```sql
INSERT INTO ville VALUES (8, 'Toulouse', 31, 471941, 118);
```

17. La requête suivante convient :

```sql
SELECT sport.nom
FROM sport
JOIN ville ON ville.id = id_ville
WHERE ville.nom = 'Annecy' AND sport.type = "mur d'escalade";
```