Proposition de correction

Exercice 1

Q1

Ping a envoyé une requête ICMP vers M3 mais il n'y a pas eu l'acquittement. M3 n'est pas joignable.

Q2

Random Access Memory

Q3

Nom du noyau GNU/Linux, donné à partir de son créateur Linus (Torvalds) et du système Unix.

Q4

Une interface d'entrée et une interface de sortie afin de relayer les paquets vers des réseaux différents.

Q5

192.168.1.254/24

Q6

N1 → R1 → R3 → R4 → N2

Q7

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

Q8

- Fibre : $10^8 / 10^9 = 0,1$
- Fast-Ethernet : $10^8 / 10^8 = 1$
- Ethernet : $10^8 / 10^7 = 10$

Q9

0,1 + R2 – R6 +0,1 = 0,3

R2 – R6 = 0,1 (Fibre)

Q10

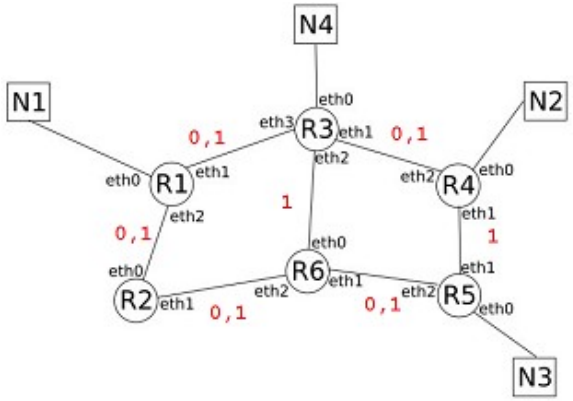
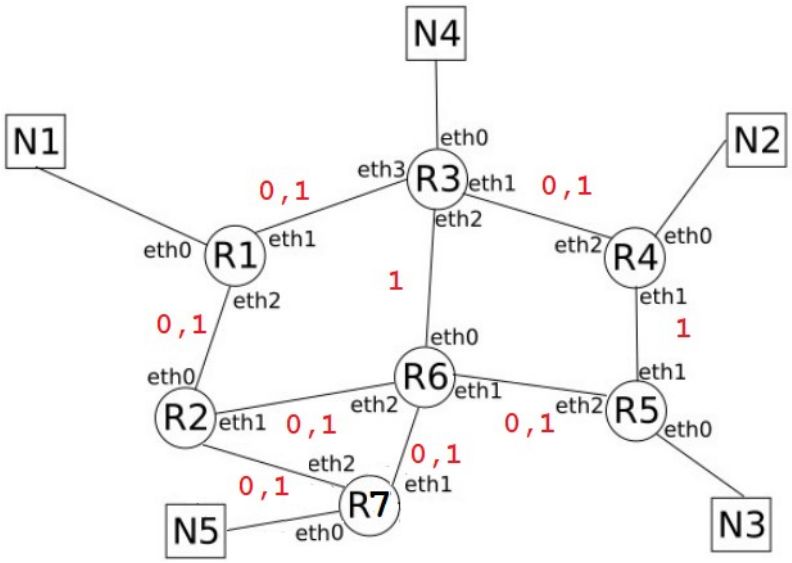


Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth1	0,2
~~N2~~	~~eth2~~	~~1,3~~
~~N3~~	~~eth1~~	~~1,4~~
N3	eth2	0,3
N4	eth1	0,1
~~N4~~	~~eth2~~	~~1,2~~

La table de routage ne conserve que la meilleure route.

Q11



Exercice 2

Q1

```
def corrige(cop : list, corr : list) -> list:
    """
    @param cop -- liste d'entiers [1;5] de la copie
    @param corr -- liste d'entiers [1;5] de la correction
    @return la liste des booléens associée à la copie cop selon la correction corr.
    """
    corrTM = []
    for i in range(len(cop)):
        if cop[i] in range(1,6):
            if i <= len(corr):
                if corr[i] in range(1,6):
```

```
            corrTM += [cop[i] == corr[i]]
    return corrTM
```

Q2

```
def note(corr : list, cop : list) -> int:
    """
    @param corr -- liste d'entiers [1;5] de la correction
    @param cop -- liste d'entiers [1;5] de la copie
    @return la note attribuée à la copie cop
    """
    la_note = 0
    for correction in corrige(cop, corr):
        la_note += int(correction)
    return la_note
```

Q3

```
def notes_paquet(p : dict, corr : list) -> dict:
    """
    @param p -- paquet de copies
    @param corr -- une correction corr
    @return dictionnaire dont les clés sont les noms des candidats du paquet p et les valeurs sont leurs notes selon la correction corr.
    """
    return {key: note(value, corr) for key, value in p.items()}
```

Q4

Une liste ne peut être hachée car elle est mutable.

Q5

Rajouter un champ date de naissance sur un entier de 8 digits dans le tuple (diminue le risque de collision).

Attention, la personne peut alors être identifiée (données personnelles).

Q7

Ce programme retourne un tuple contenant trois tuples et un dictionnaire. Ces trois tuples représentent les trois plus grandes valeurs du dictionnaire, tandis que le dictionnaire contient les valeurs restantes dans l'ordre où elles apparaissent.

Q6

Dans notre cas :

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

Q8

None pour les valeurs du tuple pour le(s) nom(s) absent(s) et dictionnaire vide.

Q9

```
def classement(notes : dict) ->list:
    """
    @param notes -- dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes
    @return la liste des couples ((prénom, nom), note) des candidats classés par notes décroissantes
    """
    candidats = []
    suite     = None
    while suite != {}:
        nom1, nom2, nom3, suite = enigme(notes)
        for nom in nom1, nom2, nom3:
            if nom is not None:
                candidats += [nom]
        notes = suite
    return candidats
```

Q10

```
def renote_express2(copcorr) :
    gauche = 0
    droite = len(copcorr)
    while droite - gauche > 1 :
        milieu = (gauche + droite)//2
        if copcorr[milieu] :
            gauche = milieu + 1
        else :
            droite = milieu - 1

    if copcorr[gauche] :
        return gauche + 1
    else :
        return gauche
```

Q11

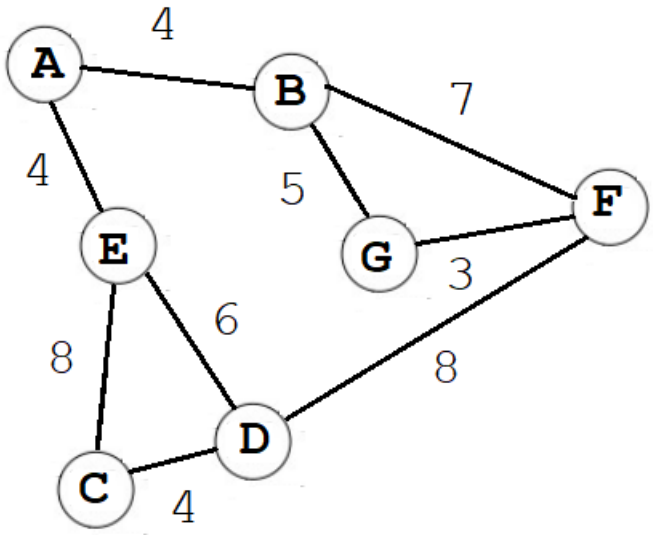
- renote_express : O(n)
- renote_express2 : $O(\ln_2(n))$

Q12

Remplacer copcorr[…] par cop[…] == corr[…]

Exercice 3

Q1



Q2

A → E → D

Q3

1	1	0	0	1	0	0
1	1	0	0	0	1	1
0	0	1	1	1	0	0
0	0	1	1	1	0	0
1	0	1	1	1	0	0
0	1	0	0	0	1	1
0	1	0	0	0	1	1

Q4

```
G2 = {
   "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", "I", "G"],
   "I":["B", "F", "H"]
   }
```

Q5

A, B, C, H, I, D, E, G, F

Q6

la fonction s'appelle elle-même (ligne 8)

Q7

À déterminer tous les chemins possibles pour se rendre d'un point à un autre.

Q8

```
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
```

Q9

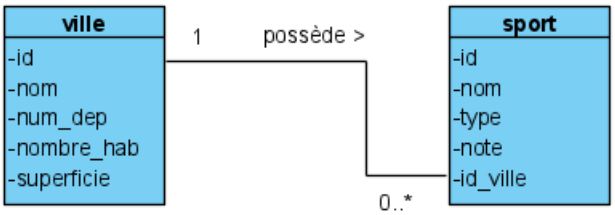
La variable globale tab_itineraires doit être réinitialisée entre chaque appel.

```
def wrapper_itineraires_court(G, dep, arr):
    global tab_itineraires
    tab_itineraires=[]
    return itineraires_court(G, dep, arr)
```

Q10

Évite de dupliquer les données sur les villes pour chaque infrastructure (gain de place + facilité de mise à jour)

Q11



Q12

Permet de mettre en relation la table ville et la table sport.

Q13

Chamonix

Q14

```
SELECT nom
FROM sport
WHERE type = "piscine"
ORDER BY nom
```

Q15

```
UPDATE sport
SET note = 7
WHERE id = 3
```

Q16

```
INSERT INTO ville
VALUES(8, "Toulouse", 31, 471941, 118)
```

Q17

```
SELECT sport.nom
FROM sport, ville
WHERE ville.nom = "Annecy"
AND sport.type = "mur d’escalade"
AND sport.id_ville = ville.id
ORDER BY sport.nom
```