Éléments de correction sujet 04 (2023)

Exercice 1

Partie A

1.
   a. adresse IP routeur F : 192.168.5.254
   b. Il sera possible de connecter 256 - 2 = 254 machines au maximum (en comptant le routeur)
2.
   a. masque de sous-réseau : 255.255.240.0 soit en binaire 11111111.11111111.11110000.00000000.
   b. On effectue un ET logique entre le masque de sous réseau et l'adresse IP 192.168.2.2 (en binaire 11000000.10101000.00000010.00000010), ce qui donne une adresse réseau 11000000.10101000.00000000.00000000 soit en décimal **192.168.0.0**
   c. Cette interconnexion permet au réseau de mieux résister en cas de panne d'un routeur en proposant plusieurs chemins possibles. Par exemple, pour aller du routeur A au routeur E, il existe 2 chemins possibles : A-F-E ou A-B-E. Si le routeur F tombe en panne, il y a toujours la possibilité de passer par A-B-E

Partie B

1.
   a. entre A et E : A - B - E
   entre F et B : F - H - G - B ou F - D - A - B ou F- H - E - B ou F- D - G - B
   b.

Table de routage du routeur E		
Destination	Routeur suivant	Distance
A	B	2
B	B	1
C	H	2
D	G	2
F	H	2
G	G	1
H	H	1

Table de routage du routeur G		
Destination	Routeur suivant	Distance
A	B	2
B	B	1
C	D	2
D	D	1
E	E	1
F	H	2
H	H	1

2.

a.

Table de routage du routeur F		
Destination	Routeur suivant	Coût total
A	D	1,1
B	H	10,11
C	D	1,1
D	D	0,1
E	H	10,1
G	D	1,1
H	H	0,1

b.

le chemin serait E - H - F - D pour un coût de 10 + 0,1 + 0,1 = 10,2

<u>Exercice 2</u>

1.

a.

On obtient les données suivantes :

6	1.70	100

b.

```
SELECT nom, age
FROM animal
WHERE nom_espece = “bonobo”
ORDER BY age
```

2.

a.

clé primaire d'espece : "nom_espece" (chaque entrée est unique et peut donc jouer le rôle de clé primaire)
clé étrangère d'espece : "num_enclos" ("num_enclos" correspond à "num_enclos" de la relation enclos)

b.

```
animal (id_animal  : INT, nom : TEXT, age : INT, taille :
FLOAT, poids : INT, #nom_espece : TEXT)
enclos (num_enclos : INT, ecosysteme : TEXT, surface : INT,
struct : TEXT, date_entretien : DATE)
espece (nom_espece : TEXT,  classe : TEXT, alimentation :
TEXT, #num_enclos : INT)
```

3.

a.

```
UPDATE espece
SET classe = "mammifères"
WHERE nom_espece = "ornithorynque"
```

b.

```
INSERT INTO animal
VALUES
(179, "Serge", 0, 0.8, 30, "lama")
```

4.

a.

```
SELECT nom, animal.nom_espece
FROM animal
JOIN espece ON animal.nom_espece = espece.nom_espece
JOIN enclos ON espece.num_enclos = enclos.num_enclos
WHERE enclos.struct = 'vivarium' and alimentation =
'carnivore'
```

b.

```
SELECT COUNT(*)
FROM animal
JOIN espece ON animal.nom_espece = espece.nom_espece
WHERE classe = 'oiseaux'
```

Exercice 3

1.

résultat de l'exécution : Bonjour Alan
x et y sont de type booléen. x est False et y est True

```
def occurrences_lettre(une_chaine, une_lettre):
    compteur = 0
    for l in une_chaine:
        if l == une_lettre:
            compteur = compteur + 1
    return compteur
```

Exercice 3

1.
```
indice = randint(0, 3)
enfiler(f, couleurs[indice])
```
2.
```
def vider(f):
   while not est_vide(f):
      defiler(f)
```
3.
```
while not est_vide(sequence):
  c = defiler(sequence)
  affichage(c)
  time.sleep(0.5)
  enfiler(stock, c)
while not est_vide(stock):
  enfiler(sequence, defiler(stock))
```
4.
   a.
```
def tour_de_jeu(sequence):
    affich_seq(sequence) #ZONE A
    stock = creer_file_vide()
    while not est_vide(sequence):
        c_joueur = saisie_joueur()
        c_seq = defiler(sequence) #ZONE B
        if c_joueur == c_seq:
            enfiler(stock, c_seq) #ZONE C
        else :
            vider(sequence) #ZONE D
    while not est_vide(stock): #ZONE E
        enfiler(sequence, defiler(stock)) #ZONE F
```
   b.
```
def tour_de_jeu_modif(sequence):
    affich_seq(sequence)
    t_ini = taille(sequence)
    stock = creer_file_vide()
    while not est_vide(sequence):
        c_joueur = saisie_joueur()
        c_seq = defiler(sequence)
        if c_joueur == c_seq:
            enfiler(stock, c_seq)
        else:
            vider(sequence)
    while not est_vide(stock):
        enfiler(sequence, defiler(stock))
    if t_ini != taille(sequence) :
        vider(sequence)
    tour_de_jeu_modif(sequence)
```

Exercice 5 : *Cet exercice porte sur la programmation en général et la récursivité en particulier.*
On considère un tableau de nombres de n lignes et p colonnes.
Les lignes sont numérotées de 0 à $n-1$ et les colonnes sont numérotées de 0 à $p-1$. La case en haut à gauche est repérée par (0,0) et la case en bas à droite par $(n-1,p-1)$.
On appelle chemin une succession de cases allant de la case (0,0) à la case $(n-1,p-1)$, en n'autorisant que des déplacements case par case : soit vers la droite, soit vers le bas.
On appelle somme d'un chemin la somme des entiers situés sur ce chemin.
Par exemple, pour le tableau T suivant :

4	**1**	**1**	3
2	0	**2**	1
3	1	**5**	**1**

- Un chemin est (0,0), (0,1), (0,2), (1,2), (2,2), (2,3) (en gras sur le tableau).
- La somme du chemin précédent est 14.
- (0,0), (0,2), (2,2), (2,3) n'est pas un chemin.

L'objectif de cet exercice est de déterminer la somme maximale pour tous les chemins possibles allant de la case (0,0) à la case $(n-1,p-1)$.

1) On considère tous les chemins allant de la case (0,0) à la case (2,3) du tableau T donné en exemple.
   a) Un tel chemin comprend nécessairement 3 déplacements vers la droite. Combien de déplacements vers le bas comprend-il ?
   Solution : Il faut 2 déplacements vers le bas.
   b) La longueur d'un chemin est égal au nombre de cases de ce chemin. Justifier que tous les chemins allant de (0,0) à (2,3) ont une longueur égale à 6.
   Solution : Il faut 2 déplacements vers le bas et 3 vers la droite, ce qui fait 5 cases à visiter. En rajoutant le départ, ça fait un chemin de longueur 6.
2) En listant tous les chemins possibles allant de (0,0) à (2,3) du tableau T, déterminer un chemin qui permet d'obtenir la somme maximale et la valeur de cette somme.
   Solution : Voici les différentes valeurs possibles :

$$4+1+1+3+1+1=11$$
$$4+1+1+2+1+1=10$$
$$4+1+1+2+5+1=14$$
$$4+1+0+2+1+1=9$$
$$4+1+0+2+5+1=13$$
$$4+1+0+1+5+1=12$$
$$4+2+0+2+1+1=10$$
$$4+2+0+2+5+1=14$$
$$4+2+0+1+5+1=13$$
$$4+2+3+1+5+1=16$$

   Le chemin de somme maximale est (0,0), (1,0), (2,0), (2,1), (2,2), (2,3) et vaut 16.
3) On veut créer le tableau T' où chaque élément $T'[i][j]$ est la somme maximale pour tous les chemins possibles allant de (0,0) à (i,j).
   a) Compléter le tableau T' se trouvant annexe associé au tableau T ci-dessous.

T =

4	1	1	3
2	0	2	1
3	1	5	1

T' =

4	5	6	?
6	?	8	10
9	10	?	16

b) Justifier que si j est différent de 0, alors : $T'[0][j] = T[0][j] + T'[0][j-1]$
Solution : Pour aller en $(0,j)$, avec $j > 0$, il faut forcément venir de $(0,j-1)$.
On a donc $T'[0][j] = T[0][j] + T'[0][j-1]$.

4) Justifier que si i et j sont différents de 0, alors :

$$T'[i][j] = T[i][j] + \max(T'[i-1][j], T'[i][j-1])$$

Solution : Pour aller en (i,j), avec $i > 0$ et $j > 0$, il faut soit venir de $(i-1,j)$, soit de $(i,j-1)$.
Les deux valeurs à comparer sont $T[i][j] + T[i-1][j]$ et $T[i][j] + T[i][j-1]$.
Comme il faut prendre le plus grand des deux, on otbient l'égalité demandée.

$$T'[i][j] = T[i][j] + \max(T'[i-1][j], T'[i][j-1])$$

5) On veut créer la fonction récursive `somme_max` ayant pour paramètres un tableau T, un entier i et un entier j. Cette fonction renvoie la somme maximale pour tous les chemins possibles allant de la case (0,0) à la case (i,j).

a) Quel est le cas de base, à savoir le cas qui est traité directement sans faire appel à la fonction `somme_max`? Que renvoie-t-on dans ce cas?
Solution : Dans le cas `somme_max(T, 0, 0)`, on renvoie `T[0][0]`.

b) À l'aide de la question précédente, écrire en Python la fonction récursive `somme_max`.

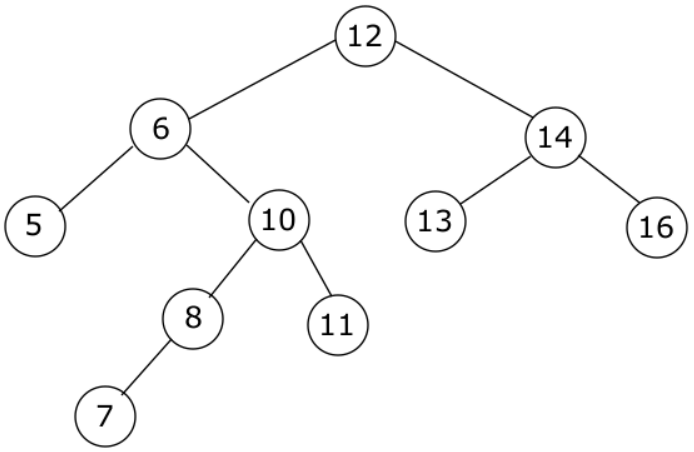
Solution :

```
def somme_max(T, i, j):
    if i == 0 and j == 0:
        return T[0][0]
    elif i == 0 and j > 0:
        return T[i][j] + T[i][j-1]
    elif i > 0 and j == 0:
        return T[i][j] + T[i-1][j]
    else:
        return T[i][j] + max(T[i][j-1], T[i-1][j])
```

c) Quel appel de fonction doit-on faire pour résoudre le problème initial?
Solution : Il faut appeler `somme_max(T, 2, 3)`.

Exercice 3

1. Il s'agit d'une structure FIFO, c'est-à-dire une file
2.
   a. il s'agit de la taille d'un arbre
   b. il s'agit de la racine de l'arbre
   c. il s'agit de la feuille d'un arbre
3.
   a. attributs de la classe Noeud : tache, indice, gauche et droite
   b. La méthode *insere* est dite récursive, car elle s'appelle elle-même. Dans cette méthode récursive, on trouve bien le traitement du cas de base, ce qui permet d'affirmer que cette méthode se termine.
   c. il s'agit du signe > (strictement supérieur)
   d.



4.

```
def est_present(self, indice_recherche) :
      """renvoie True si l'indice de priorité indice_recherche
      (int) passé en paramètre est déjà l'indice d'un nœud
      de l'arbre, False sinon"""
      if self.est_vide():
          return False
      if self.racine.indice == indice_recherche:
          return True
      if self.racine.indice > indice_recherche :
          return self.racine.gauche.est_present(indice_recherche)
      else :
          return self.racine.droite.est_present(indice_recherche)
```

5.
   a.
   parcours infixe : 6 - 8 - 10 - 12 - 13 - 14
   b.
   Le parcours infixe permet d'obtenir les valeurs des nœuds d'un arbre binaire de recherche dans un ordre croissant. Le parcours infixe va donc permettre d'obtenir les tâches à accomplir dans l'ordre des priorités

6.

```
def tache_prioritaire(self):
        """renvoie la tache du noeud situé le plus
        à gauche de l'ABR supposé non vide"""
        if self.racine.gauche.est_vide(): #pas de nœud plus à
gauche
            return self.racine.tache
        else:
            return self.racine.gauche.tache_prioritaire()
```

7.