Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

Exercice 1 - Base de données - Théâtre

1. À cause de la contrainte de référence (une des trois contraintes d'intégrité d'une base de données avec la contrainte de relation et la contrainte de domaine), il est impossible d'ajouter un enregistrement dans la relation **Role** faisant référence à des enregistrements inexistants dans les relations **Piece** et **Acteur**.

2.
```
INSERT INTO Role VALUES (46721, 389761, 'Tartuffe');
```
ou
```
INSERT INTO Role(idPiece, idActeur, nomRole)
   VALUES (46721, 389761, 'Tartuffe');
```

3. Cette requête met à jour l'attribut **Langue** de la table **Piece** avec la valeur "Anglais" partout où cet attribut a les valeurs "Américain" ou "Britannique".

4a.
```
SELECT nom, prénom FROM Acteur WHERE anneeNaiss > 1990;
```

4b.
```
SELECT MAX(anneeNaiss) FROM Acteur;
```
ou
```
SELECT anneeNaiss FROM Acteur ORDER BY anneeNaiss DESC LIMIT 1;
```

4c.
```
SELECT R.nomRole FROM Role AS R JOIN Acteur AS A ON R.idActeur = A.idActeur
   WHERE A.prenom = 'Vincent' AND A.nom = 'Macaigne';
```
ou
```
SELECT R.nomRole FROM Role AS R, Acteur AS A
   WHERE A.prenom = 'Vincent' AND A.nom = 'Macaigne' AND R.idActeur = A.idActeur;
```

4d.
```
SELECT P.titre FROM Piece AS P
   JOIN Role AS R ON P.idPiece = R.idPiece
   JOIN Acteur AS A ON R.idActeur = A.idActeur
   WHERE P.langue = 'Russe' AND A.prenom = 'Jeanne' AND A.nom = 'Balibar';
```
ou
```
SELECT P.titre FROM Piece AS P, Acteur AS A, Role AS R
   WHERE P.langue = 'Russe' AND A.prenom = 'Jeanne' AND A.nom = 'Balibar'
   AND R.idActeur = A.idActeur AND R.idPiece = P.idPiece ;
```

Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

Exercice 2 - Pile - Fonction mystère

1. a.

```
pile1 = Pile()
pile1.empiler(7)
pile1.empiler(5)
pile1.empiler(2)
```

1. b. L'affichage est : 7, 5, 5, 2

2. a. Cas n°1 : 3, 2
Cas n°2 : 3, 2, 5, 7
Cas n°3 : 3
Cas n°4 : pile vide

2. b. La fonction mystère permet d'obtenir la pile inversée correspondant aux éléments supérieurs de la pile jusqu'à l'élément recherché inclus.

3.

```
def etendre(pile1, pile2):
    """Ajoute les éléments de pile2 à pile1 en inversant leur ordre."""
    while not pile2.est_vide():
        pile1.empiler(pile2.depiler())
```

4.

```
def supprime_toutes_occurences(pile, element):
    """Supprime tous les éléments element de la pile."""
    p = Pile()
    while not pile.est_vide(): #Elimine les éléments
        e = pile.depiler()
        if e != element:
            p.empiler(e)
    while not p.est_vide(): #Remet les éléments dans l'ordre
        pile.empiler(p.depiler())
```

Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

Exercice 3 - Processus, OS et Routage

Partie A : Processus

1. /sbin/init (Son PID est 1.)

2. Les processus actifs sont ceux dont l'attribut **STAT** a pour valeur "R", il s'agit des processus :

- PID 5440
- PID 5450

3. Le PPID (parent process identifier) du processus 5450 (commande ps) est 1912 qui correspond à un processus de l'application Bash.
Depuis cette application (PID 1912), les autres commandes qui ont été exécutées sont :

- Bash (PID 2014)
- Bash (PID 2013)
- python programme1.py (PID 5437)

4. La commande **python programme1.py** a été exécutée avant la commande **python programme2.py** car sont PID est plus petit.

5. Non, il est impossible de prédire quel processus finira avant l'autre, tout dépend de leur taille et de l'ordonnancement qui est fait. Les quantums de temps processeurs sont distribués aux différents processus selon des algorithmes d'ordonnancement. Le processus 5437 **`python programme1.py`** est dormant, sûrement en attente d'une ressource (processeur par exemple) tandis que le processus 5440 **`python programme2.py`** est en cours d'exécution.

Correction

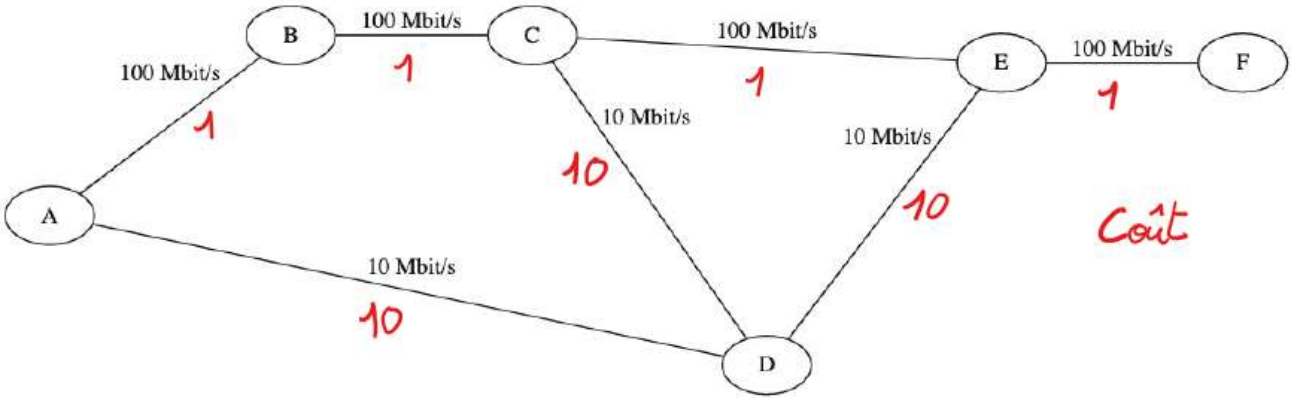
NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

Partie B : Routage

1. Dans cette table à compléter (routage RIP), la machine de destination est toujours F.

Machine (Source)	Prochain saut (Passerelle)	Distance (Sauts RIP)
A	D	3
B	C	3
C	E	2
D	E	2
E	F	1

2. Calcul du coût de chaque liaison



Dans cette table à compléter (routage OSPF), la machine de destination est toujours F.

Machine (Source)	Prochain saut (Passerelle)	Distance (Coût OSPF)
A	B	4
B	C	3
C	E	2
D	E	11
E	F	1

3. Le protocole OSPF est le plus performant car le chemin qu'il sélectionne possède un débit de 100 Mbits/s tandis que le protocole RIP sélectionne une route limitée à 10 Mbits/s

Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

Exercice 4 - Labyrinthe

Partie A : Représentation d'un labyrinthe

1. `lab2[1][0] = 2`

2.
```
def est_valide(i, j, n, m):
    """ Renvoie True si les coordonnées (i, j) correspondent à des coordonnées
        valides du labyrinthe de taille (n, m), et False sinon."""
    return -1 < i < n and -1 < j < m
```

3.
```
def depart(lab):
    n = len(lab)
    m = len(lab[0])
    for y in range(n):
        for x in range(m):
            if lab[y][x] == 2:
                return (y, x)
    return None
```

4.
```
def nb_cases_vides(lab):
    """ Renvoie le nombre de cases vides du
    labyrinthe (avec l'arrivée et le départ)."""
    total = 0
    for y in range(len(lab)):
        for x in range(len(lab[0])):
            if lab[y][x] in (0, 2, 3):
                total += 1
    return total
```

Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

Partie B : Recherche d'une solution dans un labyrinthe

1. L'appel `voisines(1, 2, [[1, 1, 4], [0, 0, 0], [1, 1, 0]])`
renvoie `[(1, 1), (2, 2)]`

2. a.
```
# entrée: (1, 0), sortie (1, 5)
chemin = [(1, 0)]
chemin.append((1, 1))
chemin.append((2, 1))
chemin.pop() # Supprime la dernière case de la liste (impasse)
chemin.append((1, 2))
chemin.append((1, 3))
chemin.append((2, 3))
chemin.append((3, 3))
chemin.append((3, 4))
chemin.pop()
chemin.pop()
chemin.pop()
chemin.append((1, 4))
chemin.append((1, 5))
```

2. b.
```
def solution(lab):
    chemin = [depart(lab)] # ajout de la case départ
    case = chemin[0]
    i = case[0]
    j = case[1]
    while lab[i][j] != 3: # Tant qu'on est pas sur l'arrivée
        lab[i][j] = 4
        cases_voisines = voisines(i, j, lab)
        if not cases_voisines :   # S'il n'y a pas de voisines
            chemin.pop()          # enlève la dernière case
        else:                     # Sinon ajout de la première case voisine
            chemin.append(cases_voisines[0])
            i, j = cases_voisines[0]
    return chemin
```

Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

Exercice 5 - Itérative vs Récursive

1. Le couple (1, 3) est une inversion car 8 > 7.
2. Le couple (2, 3) n'est pas une inversion car 3 < 7.

Partie A : Méthode itérative

1. a.

Cas n°1 : `fonction1([1, 5, 3, 7], 0)` renvoie `0`
Cas n°2 : `fonction1([1, 5, 3, 7], 1)` renvoie `1`
Cas n°3 : `fonction1([1, 5, 2, 6, 4], 1)` renvoie `2`

1. b. Cette fonction permet de déterminer combien de valeur, située après la $i^{ième}$ valeur sont inférieures à cette $i^{ième}$ valeur.

2.
```
def nombre_inversions(tab):
    """ Renvoie le nombre d'inversions. """
    cpt = 0
    for i in range(len(tab)):
        cpt += fonction1(tab, i)
    return cpt
```

3. Pour une liste de taille n, on voit qu'il faut parcourir presque toute la liste pour chaque élément de la liste, ce qui conduit à une complexité quadratique $O(n^2)$.

Partie B : Méthode itérative

1. Le tri fusion a une complexité linéarithmique O(n.log(n)).

2.

Solution 1	Solution 2

`def moitie_gauche(tab):`
`  milieu = len(tab) // 2`
`  if len(tab)%2: milieu += 1`
`  return tab[:milieu]`
	`from math import ceil`
	`def moitie_gauche(tab):`
	`  milieu = ceil(len(tab)/2)`
	`  return tab[:milieu]`

3.
```
def nb_inversions_rec(tab, n = 0):
    """ Renvoie le nombre d'inversions. """
    if len(tab) <= 1: # Cas trivial
        return 0
    else:  # Cas récursif
        gauche = moitie_gauche(tab)
        droite = moitie_droite(tab)
        ng = nb_inversions_rec(gauche, n) # Nombre d'inversion dans la moitie gauche
        nd = nb_inversions_rec(droite, n) # Nombre d'inversion dans la moitie droite
        n = nb_inv_tab(sorted(gauche), sorted(droite)) # Nb d'inversion pr les 2 tableaux triés
        return ng + nd + n
```