NSI - Corrigé du Bac Blanc 2024

EXERCICE 1 (7 POINTS)

1. **0,25 pt** Dans un logiciel libre, le code source est accessible à tous et on a la liberté de le copier, modifier, diffuser. Ce qui n'est pas le cas d'un logiciel propriétaire.

2. **0,25 pt** Un système d'exploitation est un ensemble de logiciels qui fait l'interface entre le matériel (mémoire, processeur, cartes, etc.) et l'utilisateur (les applications). Il permet aux autres logiciels d'utiliser les ressources proposées par le matériel sans avoir à se préoccuper en détail de ce matériel.

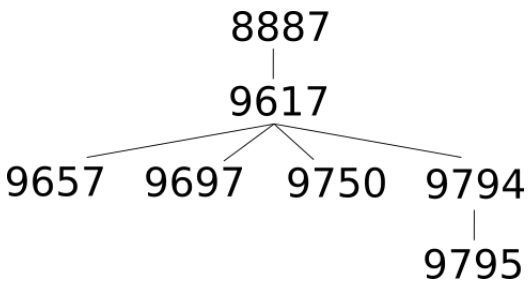
3. **0,25 pt** Le chemin absolu du fichier `rapport.odt` est :

   `/home/elsa/documents/boulot/rapport.odt`.

4. **0,25 pt** Depuis le répertoire `elsa`, le chemin relatif du fichier `photo_1.jpg` est :

   `../max/images/photos_vac/photo_1.jpg`.

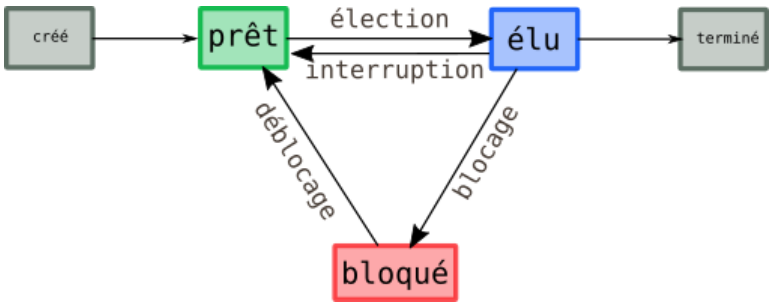
5. **0,5 pt** Une structure d'arbre est la plus adaptée :



6. **0,25 pt** C'est la commande `bash`.

7. **0,25 pt** `kill 8887`

8. **1 pt**



9. **0,25 pt** L'accès à une ressource déjà occupée par un autre processus contraint un processus à passer de l'état élu à l'état bloqué.

10. **0,25 pt** Avec deux processus A et B et deux ressources R1 et R2 :

    - A demande et obtient R1
    - B demande et obtient R2
    - A demande R1
    - B demande R2

11. **0,25 pt** Une structure de file est de type FIFO, c'est-à-dire qu'on traite les éléments dans leur ordre d'arrivée : premier arrivé, premier servi.

12. a) **0,25 pt** 11, 20, 32, 11, 20, 32, 11, 32, 11.

    b) **0,25 pt** Lorsque le quantum du tourniquet est de deux cycles CPU, l'ordre des PID par cycle est : 11, 11, 20, 20, 32, 32, 11, 11, 32.

13. **0,5 pt**

```
liste_attente = [Processus(11, 4), Processus(20, 2), Processus(32, 3)]
```

14. **1 pt**

```
    def execute_cycle(self):
        self.cycles_restants -= 1

    def change_etat(self, nouvel_etat):
        self.etat = nouvel_etat

    def est_termine(self):
        return self.cycles_restants == 0
```

15. **1,25 pts**

```
def tourniquet(liste_attente, quantum):
    ordre_execution = []
    while liste_attente != []:
        # On extrait le premier processus
        processus = liste_attente.pop(0)
        processus.change_etat("Elu")
        compteur_tourniquet = 0
        while compteur_tourniquet < quantum and not processus.est_termine():
            ordre_execution.append(processus.pid)
            processus.execute_cycle()
            compteur_tourniquet = compteur_tourniquet + 1
        if not processus.est_termine():
            processus.change_etat("Suspendu")
            liste_attente.append(processus)
        else:
            processus.change_etat("Termine")
    return ordre_execution
```

EXERCICE 2 (7 POINTS)

1. **0,25 pt**

```
a = [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
b = 'Fondation'
```

2. **0,75 pt**

```
def titre_livre(dico, id_livre):
    for i in range(len(dico['id'])):
        if dico['id'][i] == id_livre :
            return dico['titre'][i]
    return None
```

3. **0,5 pt**

```
def note_maxi(dico):
    note_max = 0
    for note in dico['note']:
        if note > note_max:
            note_max = note
    return note_max
```

4. **0,5 pt**

```
def livres_note(dico, n):
    liste_livres = []
    for i in range(len(dico['note'])):
        if dico['note'][i] == n:
            liste_livres.append(dico['titre'][i])
    return liste_livres
```

5. **0,5 pt**

```
def livre_note_maxi(dico):
    return livres_note(dico, note_maxi(dico))
```

Partie B

6. **0,5 pt** Les attributs sont `id`, `titre`, `auteur` , `annee_pub` et `note`.

   Les méthodes sont `get_id`, `get_titre`, `get_auteur` et `get_ann_pub`.

7. **0,25 pt**

```
    def get_note(self):
        return self.note
```

8. **0,75 pt**

```
b = Bibliotheque()
b.ajout_livre(Livre(8, 'Blade Runner', 'K.Dick', 1968, 8))
```

9. **0,5 pt**

```
class Bibliotheque:
    def __init__(self):
        self.liste_livre = []

    def ajout_livre(self, livre):
        self.liste_livre.append(livre)

    def titre_livre(self, id_livre):
        for livre in self.liste_livre :
            if self.id == id_livre :
                return self.titre
        return None
```

Partie C

10. **0,25 pt** L'attribut `auteur` ne peut pas être choisi comme clé primaire car il y a plusieurs valeurs identiques pour cet attribut (par ex. `'K.Dick'`).

11. **0,25 pt** La requête renvoie : `'Ubik'`, `'Blade Runner'`.

12. **0,5 pt**

```
SELECT titre
FROM livres
WHERE auteur = 'Asimov' AND annee_pub > 1950;
```

13. **0,25 pt** Écrire une requête SQL permettant de modifier la note du livre Ubik en la passant de 9 à 10.

```
UPDATE titre
SET note = 10
WHERE titre = 'Ubik';
```

14. **0,25 pt** Cela permet d'éviter la redondance de certaines données.

15. **0,25 pt** C'est une clé étrangère.

16. **0,5 pt**

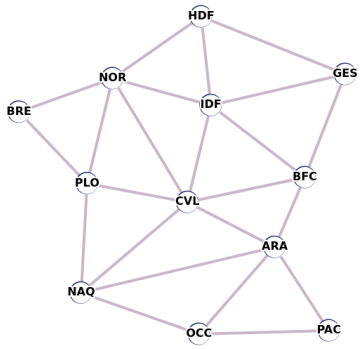
```
SELECT auteurs.nom, auteurs.prenom
FROM auteurs
JOIN livres ON auteurs.id = livres.id_auteur
WHERE livres.annee_pub > 1960;
```

17. **0,25 pt** La requête renvoie les titres des livres écrits par des auteurs de moins de 30 ans.

EXERCICE 3 (6 POINTS)

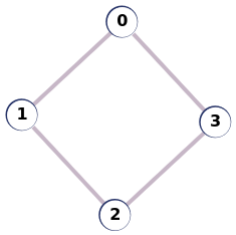
Partie A : Implémentation du graphe

1. 0,25 pt



2. a) 0,25 pt L'ordre du graphe "Régions de France" est 12.

b) 0,25 pt



c) 0,5 pt

```
def voisins(M, k):
    v = []
    for i in range(len(M[k]):
        if M[k][i] == 1:
            v.append(i)
    return v
```

ou

```
def voisins(M, k):
    return [i for i in range(len(M[k]) if M[k][i] == 1]
```

d) 0,25 pt Il y en a $1000 \times 1000 = 1000000$.

3. a) 0,25 pt

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

b) 0,25 pt

```
def voisins(G, k):
    return G[k]
```

Partie B : Coloration séquentielle ; un premier algorithme

4. **0,25 pt** Il s'agit d'un algorithme glouton, car pour chaque sommet on choisit la ≪meilleure≫ couleur, c'est-à-dire une couleur non attribuée à ses voisins.
5. **0,25 pt** Il pourrait y avoir une `IndexError` si la liste des couleurs n'est pas suffisante.
6. **0,5 pt** On obtiendrait : A → rouge, B → rouge, C → bleu, D → vert, E → vert, F → rouge.
7. **0,5 pt** On peut obtenir sur le graphe suivant une coloration avec seulement deux couleurs au lieu de 3 : A → rouge, B → bleu, C → bleu, D → rouge.

Partie C : L'algorithme de Welsh et Powell

8. **0,5 pt**

```
# Code 2
def tri_sommets(G) :
    """
    Renvoie la liste des sommets, triee par degre decroissant.
    """
    sommets = [sommet for sommet in G]
    for i in range(len(G)):
        i_sommet_max = i
        degre_sommet_max = len(G[sommets[i]])
        for j in range(i, len(sommets)):
            if len(G[sommets[j]]) > degre_sommet_max:
                i_sommet_max = j
                degre_sommet_max = len(G[sommets[j]])
        tmp = sommets[i]
        sommets[i] = sommets[i_sommet_max]
        sommets[i_sommet_max] = tmp
    return sommets
```

9. **0,5 pt** C'est un tri par sélection, sa complexité est en $O(n^2)$.
10. **0,5 pt** On peut citer le tri fusion ou le tri rapide, dont les complexités sont en $O(n \log(n))$.
11. **0,5 pt** On obtient la liste :

```
["CVL", "ARA", "IDF", "NOR", "PLO", "NAQ", "BFC", "OCC", "GES", "HDF",  "BRE", "PAC"]
```

12. **0,5 pt** Il suffit simplement de modifier la ligne :

```
    for s in G:
```

en

```
    for s in tri_sommets(G):
```

13. **0,5 pt** On obtient :

```
>>> coloration(R)
{'BRE': 'Rouge', 'PLO': 'Bleu', 'NAQ': 'Vert', 'OCC': 'Rouge',
 'PAC': 'Vert', 'ARA': 'Bleu', 'BFC': 'Vert', 'CVL': 'Rouge',
 'IDF': 'Bleu', 'GES': 'Rouge', 'HDF': 'Jaune', 'NOR': 'Vert'}
```