Correction 26-NSIJ1JA1

Exercice 1

Partie A

1. ```
   INSERT INTO choregraphie
   VALUES ('Tout autour', 2015, 9)
   ```

2. ```
   SELECT COUNT(*)
   FROM choregraphie
   WHERE annee = 1983;
   ```

3. La requête affiche les noms chorégraphies avec le nom du chorégraphe qui l'a créée.

4. ```
   SELECT nom FROM personne
   JOIN participe ON personne.id = participe.idpersonne
   JOIN spectacle ON participe.idspectacle = spectacle.id
   WHERE titre = 'Tout autour';
   ```

5. Comme id est une clé étrangère dans la table spectacle, il faut d'abord supprimer tous les enregistrements associés dans la table participe avant de pouvoir supprimer un spectacle.

Partie B

6. ```
   n = 0
   for clef in dmateriel:
       n += dmateriel[clef]['nb']
   ```

7. ```
   def tab_clefs_tab_volumes(dmatos):
       tab_clefs = []
       tab_volumes = []
       for clef, dico in dmatos.items():
           tab_clefs.append(clef)
           tab_volumes.append(dico['volume'])
       return tab_clefs, tab_volumes
   ```

8. C'est un algorithme glouton.

9. Il y a, par exemple, les tris insertion ou sélection, de complexité $O(n^2)$ (quadratique), ou encore le tri rapide (quicksort) ou le tri fusion de complexité $O(n\log(n))$ (pseudo linéaire).

10. ```
    def remplissage(dmatos, vmax):
        vol = 0
        clefs_triees = clefs_triees_selon_volume(dmatos)
        i = len(clefs_triees) - 1
        elements = []
        while i >= 0 and vol < vmax:
            clef = clefs_triees[i]
            if dmatos[clef]['nb'] == 0:
                i -= 1
            elif vol + dmatos[clef]['volume'] <= vmax:
                vol += dmatos[clef]['volume']
                elements.append(clef)
                dmatos[clef]['nb'] -= 1
            else:
                i -= 1
        return elements
    ```


Antoine Dequay

11.
```python
def nb_voitures(dmatos, vmax):
    elements = remplissage(dmatos, vmax)
    if elements == []:
        return 0
    n = 1
    while len(elements) != 0:
        elements = remplissage(dmatos, vmax)
        n += 1
    return n
```

Exercice 2

1. Dans ce cas, la valeur de *somme* est $0,7 \times 1 + -0,2 \times 3 + 0,1 = 0,2 > 0$. Ainsi, la valeur de *sortie* sera 1.

2. Une méthode est `get_biais` et un attribut est `biais`.

3.
```python
detecteur = DetecteurSpam(2)
detecteur.set_poids([0.7, -0.2])
detecteur.set_biais(0.1)
```

4. La ligne 16 est un `assert`, c'est à dire ici une instruction qui vérifie que la condition `len(nouveau_poids) == len(self.poids)` est vraie. Elle permet de s'assurer que la liste de poids passée en argument a la même longueur que la liste de poids déjà présente dans l'objet, de sorte à rester cohérent avec le nombre d'entrées du perceptron.

5. L'argument self n'est pas obligatoire, on pouvait en faire une méthode statique !
```python
def fonction_activation(self, valeur):
    if valeur >= 0:
        return 1
    return 0
```

6.
```python
def prediction(self, entrees):
    somme = 0
    for i in range(len(entrees)):
        somme += entrees[i] * self.poids[i]
    somme += self.biais
    sortie = self.fonction_activation(somme)
    return sortie
```

7. Ici, n est la taille de `entrees`. Comme on fait un parcours de la liste `entrees` une fois, et que les autres instructions (dont l'appel à `fonction_activation`) sont en temps constant, la complexité temporelle de la méthode `prediction` est $O(n)$.

8. Les valeurs demandées sont :
   - `echantillon[1] : [0.2, 0.1]`
   - `echantillon[2][1] : 0.8`

9.
```python
    def entrainement(self, echantillon, cibles, taux=0.1):
        nb_erreurs = 0
        for i in range(len(echantillon)):
            entrees = echantillon[i]
            cible = cibles[i]
            # calcul de la prédiction
            predict = self.prediction(entrees)
            if predict != cible:
                # prédiction incorrecte
                nb_erreurs += 1
                erreur = cible - predict
                for j in range(len(self.poids)):
                    self.poids[j] += taux * erreur * entrees[j]
                self.biais += taux * erreur
        return nb_erreurs
```

10.
```python
detecteur = DetecteurSpam(2)

emails = [[0, 0], [0, 1], [1, 0], [1, 1]]
spams = [0, 0, 0, 1]

nb_epoques_max = 1000
nb_erreurs = detecteur.entrainement(emails, spams, taux=0.1)
nb_epoques = 1

while nb_erreurs > 0 and nb_epoques < nb_epoques_max:
    nb_erreurs = detecteur.entrainement(emails, spams, taux=0.1)
    nb_epoques += 1

if nb_erreurs == 0:
    print(nb_epoques - 1)
else:
    print(-1)
```

Exercice 3

Partie A

1. Comme $24 = 3 \times 8$, la part fixe des adresses IP des machines branchées dans le réseau correspond aux 3 premiers octets, c'est à dire à 192.168.20.

2. Pour le suffixe 30, il y a donc 2 bits alloués à la partie variable de l'adresse IP, ce qui correspond à $2^2 = 4$ adresses IP possibles. En enlevant les deux deux adresses réservées, il reste donc **2 adresses IP disponibles** pour les machines branchées dans le réseau.

3. On a :

Table de routage de R2			
Destination	Interface	Passerelle	Distance
192.168.20.0/24	172.24.0.2	172.24.0.1	1
172.24.0.0/30	172.24.0.2	–	0
172.24.1.0/30	172.24.1.2	–	0
192.168.40.0/24	172.24.1.2	172.24.1.1	1

Partie B

4. `(255, 255, 255, 252)`

5. En binaire, 255 s'écrit 11111111. Comme pour x dans $\{0, 1\}$, on a par définition `1 ET x = x`, et que `&` correspond à l'opération ET bit à bit, on a bien `11111111 & X = X` pour tout X écrit sur 8 bits. Comme un entier sur 8 bits peut prendre toutes les valeurs de 0 à 255, on a bien `255 & A = A` pour tout entier A compris entre 0 et 255.

6. Comme 252 = 255 - 3, 252 sécrit en binaire 11111100. De plus, 189 = 255 - 64 - 2 s'écrit 10111101 en binaire et 191 = 255 - 64 s'écrit 10111111. En effectuant l'opération ET bit à bit, on trouve `11111100 & 10111101 = 10111100` et `11111100 & 10111111 = 10111100`. Ainsi, sur les adresses IP complètes, les deux premiers octets sont identiques et recopiés tels quels pour la partie fixe des adresses IP. Le troisième octet est lui aussi identique d'après les calculs ci-dessus. Le quatrième octet devient 0 (car pour x dans $\{0, 1\}$, on a par définition `0 ET x = 0`). Ainsi, les deux adresses IP correspondent bien à la même adresse IP de réseau, à savoir `172.20.188.0`.

7.
```
def meme_reseau(adresse1, adresse2, masque):
    for i in range(4):
        octet1 = adresse1[i] & masque[i]
        octet2 = adresse2[i] & masque[i]
        if octet1 != octet2:
            return False
    return True
```

Partie C

8.
```
def diffuser(self):
    while not self.file.est_vide():
        paquet = self.file.defiler()
        orig, donnees = paquet
        for m in self.machines:
            if m != orig:
                self.machines[m].enfiler(paquet)
```

9. On aura :

```
{(172, 64, 0, 0): (1, None, 0),
(172, 60, 0, 0): (0, (172, 58, 0, 2), 1),
(192, 168, 47, 0): (0, (172, 58, 0, 2), 2),
(172, 62, 0, 0): (1, (172, 64, 0, 2), 1),
(192, 168, 72, 0): (1, (172, 64, 0, 2), 1)}
```

10.
```
def m_a_j(self, passerelle, i_interf, vecteurs):
    a_ete_modif = False
    for adresse, d in vecteurs:
        if adresse not in self.table or d+1 < self.table[adresse][2]:
            self.table[adresse] = (i_interf,
            passerelle, d+1)
            a_ete_modif = True
    if a_ete_modif:
        self.envoyer_vecteurs(i_interf)
```

11.
```
def vecteurs(self, i):
    res = []
    for adresse in self.table:
        i_interf, passerelle, dist = self.table[adresse]
        if i_interf != i:
            res.append((adresse, dist))
    return res
```

12.
```
    def envoyer_vecteurs(self, i_interf):
        for i in range(len(self.interfaces)):
            if i != i_interf:
                adresse = self.interfaces[i].adresse
                v = self.vecteurs(i)
                paquet = (adresse, v)
                self.interfaces[i].sortie.enfiler(paquet)
```

13.
```
for routeur in routeurs:
    routeur.envoyer_vecteurs(-1)

continuer = True
while continuer:
    continuer = False
    for reseau in reseaux:
        reseau.diffuser()
    for routeur in routeurs:
        routeur.traitement()
    for reseau in reseaux:
        if not reseau.file.est_vide():
            continuer = True
    print("tour")

for routeur in routeurs:
    print(f"{routeur.nom}: {routeur.table}")
```


Antoine Dequay