Corrigé – Bac 2026 – Numérique et sciences informatiques – Métropole, jour 1

Réseaux, protocoles de routage RIP et OSPF, sécurisation des communications, Jeu du Recto-Verso : binaire, graphes et récursivité, Plateforme de débat : arbres, programmation objet et bases de données

Corrigé détaillé rédigé par Ben, professeur particulier – benprofparticulier.fr

Exercice 1 – Réseaux, protocoles de routage RIP et OSPF, sécurisation des communications *(6 pts)*

Partie A : configuration IP du site3

1. On écrit chaque octet en binaire sur 8 bits (poids 128, 64, 32, 16, 8, 4, 2, 1) : $172 = 128 + 32 + 8 + 4 \rightarrow 10101100$; $16 \rightarrow 00010000$; $32 \rightarrow 00100000$; $15 = 8 + 4 + 2 + 1 \rightarrow 00001111$; $255 \rightarrow 11111111$; $240 = 128 + 64 + 32 + 16 \rightarrow 11110000$; $0 \rightarrow 00000000$.

```
Adresse IP : 10101100.00010000.00100000.00001111
Masque     : 11111111.11111111.11110000.00000000
```

Le masque compte 20 bits à 1 : il correspond à la notation CIDR **/20**.

2. Avec un masque /20, il reste $32 - 20 = 12$ bits pour la partie machine, soit $2^{12} = 4096$ adresses. Deux adresses sont réservées : l'adresse du réseau (bits machine tous à 0) et l'adresse de diffusion (bits machine tous à 1). Le réseau peut donc accueillir **4 094 hôtes**.

Adresse du réseau : ET logique entre l'adresse IP et le masque. Sur le 3e octet, 00100000 ET 11110000 = 00100000 = 32 ; le 4e octet devient 0. Le réseau est donc 172.16.32.0/20. Les 4 bits de poids faible du 3e octet sont libres : ce 3e octet varie de 32 (00100000) à 47 (00101111).

- Première adresse utilisable : **172.16.32.1** ;
- dernière adresse utilisable : **172.16.47.254** (172.16.47.255 est l'adresse de diffusion).

3. Le paramètre mal configuré est la **passerelle** : 172.16.48.254 n'appartient pas au réseau de l'hôte (172.16.32.0/20 s'arrête à 172.16.47.255). Cette adresse appartient au réseau du site4 (172.16.48.0/20), dont la passerelle est d'ailleurs 172.16.63.254 selon la convention. L'hôte joint sans problème les machines de son propre sous-réseau (aucune passerelle n'est nécessaire), mais tout paquet destiné à l'extérieur, donc à Internet, est envoyé vers une passerelle injoignable.

D'après la convention de l'administrateur, la passerelle est la dernière adresse disponible du sous-réseau, soit 172.16.47.254 ; le test de connectivité réussi vers cette adresse confirme qu'il s'agit bien de l'interface de R2 dans le site3. Correction :

```
gateway 172.16.47.254
```

4. Le réseau 192.168.0.8/30 laisse $32 - 30 = 2$ bits pour la partie machine, soit 4 adresses : 192.168.0.8 (adresse du réseau), 192.168.0.9, 192.168.0.10 et 192.168.0.11 (adresse de diffusion). Seules deux adresses sont utilisables, une par routeur : **R2 : 192.168.0.9** et **R4 : 192.168.0.10** (ou l'inverse), avec le masque 255.255.255.252.

Partie B

5. Chaque ligne du traceroute correspond à un routeur traversé : le paquet suit le chemin **R2 → R3 → R4 → R5**. Il emprunte trois liens (R2–R3, R3–R4, R4–R5) : son coût RIP est **3**.

6. RIP choisit toujours la route qui emprunte le moins de liens. Or le schéma montre une liaison directe R2–R4 (réseau 192.168.0.8/30) : la route R2 → R4 → R5 ne coûte que **2**. Le paquet passant par une route de coût 3, la route optimale n'est pas utilisée : c'est le dysfonctionnement.

Cause possible : la liaison R2–R4 est hors service (câble débranché ou coupé, interface d'un des deux routeurs désactivée ou mal configurée, par exemple avec des adresses qui ne sont pas dans le même réseau /30). Ne recevant plus d'informations de routage par ce lien, les routeurs ont recalculé leurs tables sans lui et passent par R3.

7. On cherche à **maximiser le débit** (plus de données transmises par seconde) et à **minimiser le coût** : OSPF retient le chemin dont la somme des coûts est minimale. Comme le coût est inversement proportionnel au débit, minimiser le coût revient à privilégier les liaisons rapides.

8. Coût de chaque type de liaison avec coût $= 10^8$ / débit (débit en bit/s) :

Liaison	**Débit**	**Coût**
Ethernet	10 Mbit/s $= 10^7$ bit/s	$10^8 / 10^7 = 10$
Fast Ethernet	100 Mbit/s $= 10^8$ bit/s	$10^8 / 10^8 = 1$
Fibre optique	4 Gbit/s $= 4 \times 10^9$ bit/s	$10^8 / (4 \times 10^9) = 0{,}025$

D'où : R1–R2 = 1 ; R1–R3 = 0,025 ; R1–R4 = 1 ; R2–R3 = 10 ; R2–R4 = 10 ; R3–R4 = 0,025 ; R4–R5 = 0,025.

Le site4 est relié à R2 et Internet n'est accessible que par R5, lui-même relié uniquement à R4. On compare les chemins de R2 à R5 :

- R2 → R4 → R5 : $10 + 0{,}025 = 10{,}025$;
- R2 → R3 → R4 → R5 : $10 + 0{,}025 + 0{,}025 = 10{,}05$;
- R2 → R1 → R4 → R5 : $1 + 1 + 0{,}025 = 2{,}025$;
- R2 → R1 → R3 → R4 → R5 : $1 + 0{,}025 + 0{,}025 + 0{,}025 = 1{,}075$.

Le paquet emprunte le chemin **site4 → R2 → R1 → R3 → R4 → R5 → Internet**, de coût **1,075** (la liaison R5–Internet est commune à tous les chemins).

> ⚠ **Piège.** Avec OSPF, le chemin le plus court en nombre de routeurs n'est pas forcément le meilleur : ici, le lien direct R2–R4 (Ethernet, coût 10) est le pire choix. Calcule toujours le coût de chaque liaison avant de comparer les chemins.

Partie C

9. Alice et Bob doivent préférer le **chiffrement asymétrique**.

Avec un chiffrement symétrique, la même clé secrète sert à chiffrer et à déchiffrer : Alice et Bob devraient se la transmettre. Comme le pirate est déjà présent sur le réseau au début de l'échange, il pourrait intercepter cette clé et déchiffrer ensuite tous les messages.

Avec un chiffrement asymétrique, chacun possède une paire de clés : une clé publique, qui peut circuler librement, et une clé privée, qui n'est jamais transmise. Alice chiffre avec la clé publique de Bob et seule la clé privée de Bob permet de déchiffrer. Même en interceptant la clé publique et les messages chiffrés, le pirate ne peut pas en lire le contenu. En pratique, on utilise le chiffrement asymétrique pour s'échanger de façon sûre une clé symétrique de session, plus rapide pour chiffrer la suite de la conversation.

10. Avec HTTP, les données circulent en clair : le pirate, qui peut capturer les paquets sur le réseau du lycée, lit (et peut modifier) les messages d'Alice et de Bob. HTTPS est HTTP sécurisé par le protocole TLS, qui apporte :

- la **confidentialité** : les échanges sont chiffrés (échange d'une clé de session par chiffrement asymétrique, puis chiffrement symétrique) ; le pirate n'intercepte que des données illisibles ;
- l'**authentification** du serveur : il présente un certificat signé par une autorité de certification, ce qui empêche le pirate de se faire passer pour lui (attaque de l'homme du milieu) ;
- l'**intégrité** : toute modification des données en transit est détectée.

Exercice 2 – Jeu du Recto-Verso : binaire, graphes et récursivité *(6 pts)*

Partie A : représentation binaire des configurations

1. Le plateau compte 9 jetons et chacun a 2 états possibles (Recto ou Verso), indépendamment des autres. Le nombre de configurations est donc $2 \times 2 \times \ldots \times 2$ (9 facteurs) $= 2^9 = 512$. Autrement dit, une configuration correspond à un mot de 9 bits.

2. D'après la figure 1, la configuration de départ est (R = Recto noir, V = Verso blanc) : ligne 1 : V V R ; ligne 2 : V V R ; ligne 3 : R R V.

- après « ligne 1 » : R R V / V V R / R R V ;
- après « ligne 2 » : R R V / R R V / R R V ;
- après « colonne 2 » : R V V / R V V / R V V.

Seuls les trois jetons de la colonne 1 sont côté Recto : **cette séquence ne mène pas à la victoire**.

> ⚠ **Piège.** Après « ligne 1 » et « ligne 2 », seule la colonne 3 est en Verso : c'est « colonne 3 » qui gagnerait, pas « colonne 2 ». Applique les coups un par un sur la figure, sans conclure trop vite.

3. La configuration de la figure 1 se lit $[0, 0, 1, 0, 0, 1, 1, 1, 0]$. Les bits ont, de gauche à droite, les poids $2^8 = 256$, $2^7 = 128$, 64, 32, 16, 8, 4, 2 et 1 :

$0 \times 256 + 0 \times 128 + 1 \times 64 + 0 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 64 + 8 + 4 + 2 = \mathbf{78}$.

(Vérification de la méthode sur l'exemple du sujet : $[0, 0, 1, 0, 0, 0, 0, 1, 1]$ donne $64 + 2 + 1 = 67$.)

4.

```
def representant(bits):
    n = 0
    for b in bits:
        n = 2 * n + b
    return n
```

À chaque bit lu, on décale le nombre déjà obtenu d'un rang vers la gauche (multiplication par 2) et on ajoute le nouveau bit. Sur $[0, 0, 1, 0, 0, 0, 0, 1, 1]$, n prend successivement les valeurs 0, 0, 1, 2, 4, 8, 16, 33 et 67.

> 💡 **Astuce.** Cette méthode (schéma de Horner) évite de calculer des puissances de 2 et fonctionne quelle que soit la longueur de la liste. Une boucle avec n = n + bits[i] * 2 ** (len(bits) - 1 - i) est aussi juste, mais plus facile à rater.

5.

```
def xor_etendu(l_x, l_y):
    resultat = []
    for i in range(len(l_x)):
        resultat.append(xor(l_x[i], l_y[i]))
    return resultat
```

Avec l_x = [1, 1] et l_y = [0, 1], on obtient [xor(1, 0), xor(1, 1)] = [1, 0], comme dans l'énoncé.

6.

```
def configurations_suivantes(config):
    config_bin = binaire(config)
    voisins = []
    for nom in coups_possibles:
        coup = coups_possibles[nom]   # ligne facultative
        nouvelle = xor_etendu(config_bin, coup)
        voisins.append(representant(nouvelle))
    return voisins
```

On parcourt les 8 coups du dictionnaire ; chacun est appliqué à la configuration par un ou exclusif, puis le résultat est reconverti en entier. Pour config = 67, le coup « ligne 1 » donne [1, 1, 0, 0, 0, 0, 0, 1, 1], soit $256 + 128 + 2 + 1 = 387$: c'est bien le premier voisin de 67 donné dans le sujet.

Partie B : graphe des configurations

7. S'il existe une arête $x \to y$, il existe un coup c tel que $y = x \oplus c$. Jouons ce même coup depuis y : $y \oplus c = (x \oplus c) \oplus c = x \oplus (c \oplus c)$. Or pour tout bit b, $b \oplus b = 0$, donc $c \oplus c$ est la liste de neuf 0, et $x \oplus 0 = x$. Ainsi $y \oplus c = x$: le coup c, joué depuis y, mène à x, donc **l'arête $y \to x$ existe**. Concrètement, retourner deux fois les mêmes jetons les remet dans leur état de départ : le graphe est non orienté.

8. Le graphe a $2^9 = 512$ sommets.

- **Matrice d'adjacence** : un tableau de 512×512 cases, soit $2^9 \times 2^9 = 2^{18} = 262144$ entiers.
- **Dictionnaire** : chaque sommet a exactement 8 voisins (un par coup ; chaque coup retourne au moins un jeton et deux coups différents donnent deux configurations différentes). On stocke donc 512 clés et 512 listes de 8 entiers : $2^9 + 8 \times 2^9 = 9 \times 2^9 = 4608$ entiers (dont $2^{12} = 4096$ pour les voisins).

Le dictionnaire demande environ 57 fois moins de mémoire : le graphe est très peu dense (8 voisins sur 512 sommets possibles), la matrice serait presque entièrement remplie de 0. Le choix du dictionnaire est donc justifié.

9.

```
def parcours_profondeur(graphe, configuration, vus):
    vus.append(configuration)
    for s in graphe[configuration]:
        if s not in vus:
            parcours_profondeur(graphe, s, vus)
```

On marque le sommet courant comme visité, puis on relance récursivement le parcours sur chacun de ses voisins non encore visités.

10. La victoire correspond à neuf jetons Recto, soit la liste [1, 1, 1, 1, 1, 1, 1, 1, 1], c'est-à-dire l'entier $2^9 - 1 = 511$. Le parcours depuis 4 visite toutes les configurations accessibles depuis 4 ; la configuration 4 est perdante si 511 n'en fait pas partie.

```
vus = []
parcours_profondeur(graphe, 4, vus)
if 511 not in vus:
    print("La configuration 4 est perdante")
else:
    print("La configuration 4 est gagnante")
```

(On peut aussi écrire assert 511 not in vus.) L'exécution confirme que 511 n'est pas atteint : la configuration 4 est bien perdante.

11. Il faut utiliser un **parcours en largeur**. Il visite les sommets par distance croissante au sommet de départ : d'abord ceux à 1 coup, puis ceux à 2 coups, etc. La première fois qu'il atteint la configuration 511, c'est donc par un chemin comportant le minimum de coups ; en mémorisant pour chaque sommet celui par lequel on l'a découvert, on reconstruit ce chemin. Le parcours en profondeur trouve un chemin vers la victoire s'il en existe un, mais pas forcément le plus court : il s'enfonce le plus loin possible dans une branche avant d'explorer les autres.

Exercice 3 – Plateforme de débat : arbres, programmation objet et bases de données *(8 pts)*

Partie A : structure arborescente des arguments

1. Le nœud « point de départ du débat » est la **racine** de l'arbre.

2. Un arbre binaire est un arbre dont chaque nœud a au plus deux fils. Ce n'est pas le cas ici : dans la figure 1, le sujet « La Science est une activité humaine objective. » a quatre fils (deux arguments pour et deux arguments contre). Un arbre de débat **n'est donc pas un arbre binaire** : une affirmation peut recevoir autant d'arguments que les utilisateurs le souhaitent.

3.

Attribut	Type
contenu	str (chaîne de caractères)
sorte	str (””, ”sujet”, ”pour” ou ”contre”)
arguments	list (liste d'objets Affirmation)
nb_likes	int (entier)
nb_dislikes	int (entier)

4.

```
        assert argument.sorte == ""
        self.arguments.append(argument)
        argument.sorte = "pour"
```

L'assertion vérifie que l'argument n'a encore été utilisé nulle part (sorte vide), on l'ajoute à la liste des arguments de self, puis on le marque comme argument pour.

5.

```
def nb_contre(self):
    n = 0
    for arg in self.arguments:
        if arg.sorte == "contre":
            n = n + 1
    return n
```

6. La méthode mystere renvoie **le plus grand nombre d'arguments contre directement rattachés à une même affirmation, parmi toutes les affirmations du sous-arbre de racine self** (self compris). En effet, m vaut d'abord le nombre d'arguments contre de self, puis on le remplace par le résultat récursif de chaque argument s'il est plus grand : on calcule un maximum sur tout le sous-arbre. Sur la figure 1, appelée sur

le sujet, elle renvoie 2 (le sujet est attaqué par deux arguments contre, aucune autre affirmation n'en a plus d'un).

7.

```
def evaluation(self):
    total = self.nb_likes - self.nb_dislikes
    for arg in self.arguments:
        if arg.sorte == "pour":
            total = total + arg.evaluation()
        else:
            total = total - arg.evaluation()
    return total
```

On part des likes moins les dislikes de l'affirmation, puis on ajoute l'évaluation (récursive) de chaque argument pour et on retranche celle de chaque argument contre. Les arguments rattachés à une affirmation sont tous « pour » ou « contre », d'où le else. La récursivité s'arrête sur les feuilles : leur liste arguments est vide, elles renvoient simplement likes − dislikes. Les lignes avec arguments_pour ou arguments_contre sont fausses (ces attributs n'existent pas), def evaluation() oublie self, et return 1 + total fausserait le résultat.

Partie B : base de données des utilisateurs et de leurs contributions

8. Les rôles d'un SGBD sont **a), c) et d)** : sécuriser les accès (authentification, droits des utilisateurs), assurer la persistance des données même en cas de panne (journalisation des transactions, sauvegardes) et gérer les accès simultanés de plusieurs utilisateurs (transactions, cohérence). L'alimentation électrique (b) relève de l'infrastructure matérielle et les connexions HTTPS (e) relèvent du serveur web et des protocoles réseau, pas du SGBD.

9. L'attribut **email** aurait pu servir de clé primaire : une adresse électronique identifie de façon unique un utilisateur (deux comptes n'ont pas la même adresse). En revanche, nom et prénom ne conviennent pas : deux utilisateurs peuvent être homonymes.

10.

```
SELECT COUNT(*)
FROM affirmation
WHERE nb_likes >= 50;
```

11. L'auteur d'une affirmation est désigné par son pseudo (clé étrangère auteur) : il faut une jointure avec la table utilisateur pour connaître son nom et son prénom.

```
SELECT affirmation.contenu, affirmation.nb_likes
FROM affirmation
JOIN utilisateur ON affirmation.auteur = utilisateur.pseudo
WHERE affirmation.sorte = 'sujet'
  AND utilisateur.prenom = 'Pierre'
  AND utilisateur.nom = 'Durand';
```

12. La requête associe chaque affirmation (aff) à chacune de ses réponses directes (rep), puis ne garde que les réponses qui sont des arguments contre et qui ont au moins deux fois plus de likes que l'affirmation qu'elles attaquent. Elle permet d'obtenir, **pour chaque affirmation contredite par un argument contre au moins deux fois plus liké qu'elle, le contenu de cette affirmation et le contenu de cet argument contre** (les affirmations « désavouées » par la communauté). Les attributs affirmation_repondue et id_affirmation de la requête correspondent à aff_repondue et id_aff dans le schéma.

13.

```
INSERT INTO reaction
VALUES (108, 'i<3descartes', 'like');

UPDATE affirmation
SET nb_likes = nb_likes + 1
WHERE id_aff = 108;
```

Les valeurs suivent l'ordre des attributs de la table reaction : aff_reagie, utilisateur, sorte.

14. L'attribut utilisateur de la table reaction est une **clé étrangère** qui référence la clé primaire pseudo de la table utilisateur. Par la contrainte d'intégrité référentielle, une ligne de reaction ne peut pas désigner un utilisateur qui n'existe pas : tant que des réactions font référence à 'i<3rgpd', le SGBD refuse de supprimer cet utilisateur. Il faut donc d'abord supprimer ses réactions (et, pour la même raison, retirer son pseudo de l'attribut auteur de ses affirmations).

15.

```
UPDATE affirmation
SET auteur = NULL
WHERE auteur = 'i<3rgpd';

DELETE FROM reaction
WHERE utilisateur = 'i<3rgpd';

DELETE FROM utilisateur
WHERE pseudo = 'i<3rgpd';
```

> ⚠ **Piège.** L'ordre des trois requêtes compte : l'utilisateur se supprime en dernier, une fois que plus aucune clé étrangère (auteur, utilisateur) ne pointe vers lui. Et on ne touche pas à nb_likes ni à nb_dislikes, comme l'exige la procédure.

Conseils pour le jour J

Ce sujet rapporte beaucoup de points sur des questions courtes et mécaniques : conversions binaires, calcul de plage d'adresses, requêtes SQL, attributs d'une classe. Traite-les toutes avec soin et sans calculatrice : pose les puissances de 2 (256, 128, 64…) et vérifie chaque conversion sur l'exemple fourni par le sujet (la configuration 67, le premier voisin 387). Un exemple donné dans l'énoncé est un outil de vérification gratuit, sers-t'en.

En réseau, lis le schéma avant de répondre : la question 6 repose entièrement sur la liaison directe R2–R4 que le traceroute n'emprunte pas, et la question 8 demande de calculer le coût de chaque liaison pour découvrir que le détour par R1 et R3 est le meilleur chemin. Écris toujours le chemin complet et le coût total.

En Python, respecte exactement les noms imposés (representant, xor_etendu, nb_contre, evaluation) et réutilise les fonctions déjà écrites : configurations_suivantes s'appuie sur binaire, xor_etendu et representant, mystere et evaluation sont récursives sur la liste arguments. En SQL, pense à la jointure dès qu'une information se trouve dans une autre table, et à l'ordre des suppressions imposé par les clés étrangères.

Ce document est la propriété intellectuelle de Ben, professeur particulier (benprofparticulier.fr), protégée par le Code de la propriété intellectuelle (art. L. 111-1 et L. 122-4). Il est mis à disposition pour un usage personnel de révision. Toute reproduction, diffusion, mise en ligne ou revente, totale ou partielle, sans autorisation écrite est interdite et constitue une contrefaçon (art. L. 335-2).