Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

CORRIGÉ NON OFFICIEL

Proposition de corrigé rédigée par FormaV avec l'aide d'outils d'intelligence artificielle. Elle peut contenir des erreurs et ne remplace ni le corrigé ni le barème officiels de l'examen.

Les sujets officiels des sessions passées et nos sujets d'entraînement sont classés sur https://bac-maths-nsi.fr/annales-sujets-et-corriges

Document mis en ligne par FormaV®. Reproduction ou diffusion sans autorisation interdite.

BACCALAURÉAT GÉNÉRAL • CORRIGÉ DÉTAILLÉ

Numérique et sciences informatiques

Session 2026 • Polynésie • Jour 2

Code sujet **26-NSIJ2PO1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Bases de données relationnelles et langage SQL
12 réponses • sujet p. 2–4

Exercice 2 — Programmation Python, structures de données, graphes et recherche de motif dans un texte
10 réponses • sujet p. 5–7

Exercice 3 — Routage et utilisation d’une structure arborescente
12 réponses • sujet p. 8–13

Les numéros, parties et pages renvoient au sujet original. Chaque réponse rappelle la consigne, explique la démarche et donne le résultat.

FormaV - document d'entraînement non officiel

Exercice 1

6 points

Bases de données relationnelles et langage SQL

Partie A : Modèle relationnel

Question 1.

Sujet p. 2, 3

> Expliquer le rôle de la clé primaire dans une relation.

Une clé primaire identifie de manière unique chaque ligne d'une relation : deux lignes ne peuvent pas avoir la même valeur de clé primaire. Elle ne peut pas non plus être absente. Elle peut être constituée d'un seul attribut ou d'une combinaison de plusieurs attributs.

RÉPONSE À RETENIR

La clé primaire identifie sans ambiguïté chaque enregistrement de la relation.

Question 2.

Sujet p. 2, 3

> Donner les quatre attributs formant la clé primaire pour la relation Mesure.

La clé primaire de `Mesure` est composée de quatre attributs : il faut considérer leurs valeurs ensemble pour identifier une mesure.

RÉPONSE À RETENIR

Les quatre attributs sont `StationId`, `DateDebut`, `DateFin` et `Polluant`.

Question 3.

Sujet p. 2, 3

> Expliquer le rôle d'une clé étrangère.

Une clé étrangère est un attribut, ou un groupe d'attributs, d'une relation qui fait référence à une clé d'une autre relation. Elle relie les deux relations et permet de maintenir la cohérence des références : la valeur référencée doit exister dans la relation cible.

RÉPONSE À RETENIR

Une clé étrangère référence une clé d'une autre relation et garantit la cohérence des liens entre les données.

Question 4.

Sujet p. 2, 3

> Donner un exemple de clé étrangère dans le modèle relationnel ci-dessus.

La relation `Station` contient l'attribut `AASQAId`, qui indique l'AASQA gestionnaire de la station. Il référence l'identifiant de cette AASQA.

RÉPONSE À RETENIR

Un exemple est `Station.AASQAId`, qui référence `AASQA.Id`. On peut aussi citer `Mesure.StationId`, qui référence `Station.Id`.

FormaV - document d'entraînement non officiel

Partie B : Alimentation de la base de données

Question 5.

Sujet p. 2, 3, 4

> Écrire une requête SQL permettant de rajouter à la table Mesure la mesure de valeur 34, du polluant PM10 (type d'influence « Trafic »), le 20/06/2022 de 10 h à 11 h pour la station « Annecy Rocade ».

La relation `Mesure` stocke l'identifiant de la station, et non son nom. Dans les occurrences fournies, « Annecy Rocade » a pour identifiant `FR33203`. On insère cet identifiant ainsi que les dates-heures, le polluant, le type d'influence et la valeur.

RÉPONSE À RETENIR

```
INSERT INTO Mesure (StationId, DateDebut, DateFin, Polluant, TypeInfluence, Valeur)
VALUES ('FR33203', '2022-06-20 10:00:00', '2022-06-20 11:00:00',
        'PM10', 'Trafic', 34);
```

Question 6.

Sujet p. 2, 3, 4

> L'attribut StationId dans la table Mesure est une clé étrangère faisant référence à l'identifiant d'une station. Lors de l'ajout à la table Mesure de la mesure de valeur 13.5, du polluant O3 (type d'influence « Fond »), le 20/06/2022 de 00 h à 01 h pour la station d'identifiant « FR41017 », l'exécution échoue et l'exception « IntegrityError: FOREIGN KEY constraint failed » est levée. Expliquer cette erreur.

`StationId` est une clé étrangère : sa valeur doit correspondre à l'identifiant d'une ligne de la relation `Station`. L'erreur indique que la valeur `FR41017` utilisée lors de l'insertion ne satisfait pas cette contrainte : elle ne correspond à aucun identifiant de station valide dans la table référencée.

RÉPONSE À RETENIR

L'insertion échoue parce que `FR41017` ne correspond à aucun identifiant de station existant ; la valeur de la clé étrangère `StationId` n'est donc pas valide.

Partie C : Exploitation de la base de données

Question 7.

Sujet p. 2, 3, 4

> Écrire une requête SQL permettant d'afficher le nom et le site web des AASQA.

On sélectionne les attributs `Nom` et `Siteweb` dans la relation `AASQA`. Le tableau ci-dessous donne le résultat sur les occurrences de cette relation fournies dans le sujet.

RÉPONSE À RETENIR

```
SELECT Nom, Siteweb
FROM AASQA;
```

Nom	Siteweb
AIR BREIZH	http://www.airbreizh.asso.fr
AIRPARIF	http://www.airparif.asso.fr
LIG'AIR	https://www.ligair.fr
QUALITAIR CORSE	http://www.qualitaircorse.org
GWAD'AIR	http://www.gwadair.fr
HAWA MAYOTTE	http://www.hawa-mayotte.fr
ATMO NORMANDIE	http://www.atmonormandie.fr/
ATMO OCCITANIE	http://atmo-occitanie.org/
ATMO AUVERGNE-RHONE-ALPES	http://www.atmo-auvergnerhonealpes.fr/
ATMO BOURGOGNE-FRANCHE-COMTE	https://atmo-bfc.org/
ATMO HAUTS DE FRANCE	http://www.atmo-hdf.fr

Question 8.

Sujet p. 2, 4

> Écrire une requête SQL permettant d'afficher le nombre total de stations.

`COUNT(*)` compte les lignes de la relation `Station`. Les occurrences de la table `Station` fournies dans le sujet sont au nombre de cinq ; ce décompte ne permet pas de connaître celui de la base complète.

RÉPONSE À RETENIR

```
SELECT COUNT(*)
FROM Station;
```

Résultat sur les cinq occurrences de `Station` fournies dans le sujet : `5`. Le total de la base complète n'est pas déductible du seul extrait.

Question 9.

Sujet p. 2, 4

> Expliquer à quoi sert la requête suivante : SELECT Station.Nom, Station.TypeImplantation FROM AASQA JOIN Station ON AASQA.Id = Station.AASQAId WHERE AASQA.Nom = "ATMO AUVERGNE-RHONE-ALPES";

La jointure associe chaque station à l'AASQA dont l'identifiant correspond à `Station.AASQAId`. La condition conserve ensuite uniquement les stations gérées par l'AASQA nommée « ATMO AUVERGNE-RHONE-ALPES ». La requête affiche pour chacune son nom et son type d'implantation.

RÉPONSE À RETENIR

Sur les occurrences fournies dans le sujet, la requête renvoie :

Nom	TypeImplantation
Annecy Rocade	Urbaine
Pays du Mezenc	Rurale régionale

Question 10.

Sujet p. 2, 3, 4

Pour les questions suivantes, on considère l'AASQA « Air Pays de la Loire » et la station « Mazagran », qui sont connues dans la base de données. Écrire une requête SQL permettant d'afficher la date de début, la date de fin et la valeur du polluant PM10 pour la station « Mazagran ».

Il faut relier `Station` à `Mesure` par l'identifiant de station, puis filtrer sur le nom « Mazagran » et le polluant `PM10`. Les occurrences de mesures fournies ne comprennent aucune mesure de Mazagran : elles ne permettent donc pas de donner les valeurs réelles de cette station.

RÉPONSE À RETENIR

```
SELECT Mesure.DateDebut, Mesure.DateFin, Mesure.Valeur
FROM Station JOIN Mesure ON Station.Id = Mesure.StationId
WHERE Station.Nom = 'Mazagran'
  AND Mesure.Polluant = 'PM10';
```

La requête ne renvoie aucune ligne parmi les seules occurrences présentées dans le sujet. Sur la base complète, elle renverra les mesures de PM10 effectivement enregistrées pour Mazagran.

Question 11.

Sujet p. 2, 3, 4

Écrire une requête SQL permettant d'afficher la liste sans doublon des polluants de la station « Mazagran ».

On relie la station à ses mesures, puis on sélectionne leur attribut `Polluant`. `DISTINCT` supprime les doublons. L'extrait ne présente aucune mesure pour Mazagran ; ses polluants ne peuvent donc pas être déterminés à partir de ces seules occurrences.

RÉPONSE À RETENIR

```
SELECT DISTINCT Mesure.Polluant
FROM Station JOIN Mesure ON Station.Id = Mesure.StationId
WHERE Station.Nom = 'Mazagran';
```

La requête ne renvoie aucune ligne correspondante parmi les occurrences fournies. Sur la base complète, elle renverra les polluants distincts mesurés à Mazagran.

Question 12.

Sujet p. 2, 3, 4

Écrire une requête SQL permettant d'afficher le nom des stations gérées par « Air Pays de la Loire » et, pour ces

> stations, la valeur, la date de début et la date de fin des mesures du polluant NO2.

La première jointure associe l'AASQA à ses stations, puis la seconde associe les stations à leurs mesures. Les filtres retiennent l'AASQA « Air Pays de la Loire » et les mesures du polluant `NO2`. Les occurrences fournies ne contiennent pas cette AASQA ni ses stations et mesures : elles ne permettent donc pas d'établir les résultats réels.

RÉPONSE À RETENIR

```
SELECT Station.Nom, Mesure.Valeur, Mesure.DateDebut, Mesure.DateFin
FROM AASQA
JOIN Station ON AASQA.Id = Station.AASQAId
JOIN Mesure ON Station.Id = Mesure.StationId
WHERE AASQA.Nom = 'Air Pays de la Loire'
  AND Mesure.Polluant = 'NO2';
```

La requête ne renvoie aucune ligne correspondante parmi les seules occurrences présentées dans le sujet. Sur la base complète, elle renverra le nom des stations concernées ainsi que la valeur et les dates de leurs mesures de NO2.

Exercice 2

6 points

Programmation Python, structures de données, graphes et recherche de motif dans un texte

Partie A

Question 1.

Sujet p. 5

> Dans le jeu du taquin sur 4 cases, représenter une position des 3 tuiles qui n'est pas accessible depuis la position finale.

La position finale du taquin sur 4 cases représentée dans la figure est :

0	1
2	3

On peut proposer la position suivante :

0	2
1	3

Pour justifier qu'elle est inaccessible, considérons la parité de la permutation des quatre symboles, y compris 0, et la couleur de la case qui contient 0 sur un damier. À chaque déplacement, 0 échange sa place avec une tuile : la parité de la permutation change, et 0 passe sur une case de l'autre couleur. La relation entre ces deux parités reste donc la même. Dans la position proposée, 0 est à la même place que dans la position finale, mais les tuiles 1 et 2 ont été échangées : la parité de permutation a changé sans changement de couleur de 0. Cette position ne peut donc pas être atteinte.

RÉPONSE À RETENIR

Une position inaccessible est :

0	2
1	3

Question 2.

Sujet p. 5, 6

> Nommer, en justifiant, la structure de données la plus adaptée pour modéliser, dans ce jeu, les passages d'une position à une autre.

Chaque position possible du taquin est modélisée par un sommet. Deux sommets sont reliés par une arête si un seul déplacement permet de passer d'une position à l'autre. Les déplacements étant réversibles, le graphe est non orienté.

RÉPONSE À RETENIR

La structure adaptée est un **graphe non orienté** : les sommets représentent les positions et les arêtes les déplacements possibles entre elles.

FormaV - document d'entraînement non officiel

Question 3.

Sujet p. 6

> Étant donnée une liste pos représentant une position du taquin, donner une instruction basée sur l'instruction assert qui permet d'interrompre le programme si la taille de la liste pos n'est pas correcte, c'est-à-dire différente de 9.

`len(pos)` donne la taille de la liste. L'instruction `assert` interrompt le programme si sa condition est fausse ; on affirme donc que cette taille vaut 9.

RÉPONSE À RETENIR

```
assert len(pos) == 9
```

Question 4.

Sujet p. 5, 6

> Recopier et compléter la fonction est_finale, qui prend en paramètre une position représentée par une liste pos et qui renvoie un booléen valant True si la position correspond à la position finale et False sinon.

La position finale donnée dans le sujet est `[0, 1, 2, 3, 4, 5, 6, 7, 8]`. À chaque indice `i`, on vérifie que la valeur est `i`. Dès qu'une valeur diffère, la position n'est pas finale ; si aucune ne diffère, elle l'est.

RÉPONSE À RETENIR

```
def est_finale(pos):
    for i in range(len(pos)):
        if pos[i] != i:
            return False
    return True
```

Question 5.

Sujet p. 6

> Compléter, en utilisant autant de lignes que nécessaire, la fonction case_vide, qui prend en paramètre une liste d'entiers pos et qui renvoie l'indice de 0. Si la liste pos ne contient pas 0, ou contient plusieurs 0, case_vide(pos) renvoie -1.

On initialise `res` à `-1`, qui signifie qu'aucun zéro n'a encore été trouvé. À la découverte d'un zéro, on mémorise son indice. Si un autre zéro est rencontré, la fonction renvoie `-1`. À la fin, `res` vaut l'indice du zéro unique, ou reste égal à `-1` s'il n'y en avait aucun.

RÉPONSE À RETENIR

```
def case_vide(pos):
    res = -1
    for i in range(9):
        if pos[i] == 0:
            if res != -1:
                return -1
            res = i
    return res
```

Question 6.

Sujet p. 6

> Si la case vide se trouve en indice i dans une grille de 9 cases, donner les 4 indices qui sont susceptibles de désigner les tuiles qui peuvent être déplacées sur la case vide, sans chercher à éliminer ceux qui sont invalides au niveau des bords.

Dans une grille de 3 colonnes, les cases immédiatement au-dessus et au-dessous ont pour indices respectifs `i - 3` et `i + 3`. Les cases immédiatement à gauche et à droite ont pour indices `i - 1` et `i + 1`. Comme demandé, on ne retire pas les indices qui ne sont pas valides à cause des bords.

> **RÉPONSE À RETENIR**
>
> Les quatre indices sont **`i - 3`, `i + 3`, `i - 1` et `i + 1`**.

Question 7.

Sujet p. 6, 7

> On écrit la fonction est_possible pour parcourir toutes les positions du taquin atteignables à partir d'une position donnée afin de déterminer si la position finale est accessible depuis cette position. Lorsqu'on teste la fonction, l'exécution ne semble pas se terminer. Expliquer le problème et proposer une solution, sans la programmer.

La liste `visites` est créée à nouveau à chaque appel récursif : elle ne conserve donc pas les positions rencontrées par les autres appels. De plus, le test ne porte que sur les voisins immédiats de l'appel courant. Comme les déplacements sont réversibles, la fonction peut revenir à une position déjà explorée et recommencer ce cycle indéfiniment. Le code ne prévoit pas non plus de retour explicite `False` si aucune branche ne mène à la position finale.

Il faut partager une structure de positions visitées pendant tout le parcours, et marquer chaque position avant d'explorer ses voisins. On peut faire un parcours en profondeur ou en largeur, de préférence itératif avec une pile ou une file et un ensemble de positions visitées. Le nombre de positions possibles étant fini, chaque position n'est alors explorée qu'une fois et le parcours termine.

> **RÉPONSE À RETENIR**
>
> Le problème vient de la portée locale de `visites` : les cycles ne sont pas empêchés entre les appels récursifs. Il faut partager un ensemble de positions visitées sur tout le parcours et marquer une position avant d'explorer ses voisines ; un parcours itératif avec pile ou file évite également une profondeur récursive excessive.

Partie B

Question 8.

Sujet p. 7

> Nommer un algorithme de recherche de motif dans un texte plus efficace que la recherche naïve.

Un algorithme plus efficace que la recherche naïve est l'algorithme de Boyer-Moore : il compare le motif en commençant par sa droite et peut décaler le motif de plusieurs caractères après une discordance.

> **RÉPONSE À RETENIR**
>
> **Algorithme de Boyer-Moore.**

Question 9.

Sujet p. 7

> Donner le résultat de l'appel recherche('na', 'banana').

Pour `i = 0`, la fonction compare d'abord les derniers caractères : `texte[1]` et `motif[1]` valent tous deux `'a'`, mais `texte[0]` vaut `'b'` et ne correspond pas à `motif[0]`. L'indice avance de la longueur du motif, soit à `i = 2`. À cet indice, `texte[3] = 'a'` correspond à `motif[1]`, puis `texte[2] = 'n'` correspond à `motif[0]`. La fonction renvoie donc 2.

> **RÉPONSE À RETENIR**
>
> `recherche('na', 'banana')` renvoie **2**.

Question 10.

Sujet p. 7

> Donner, en justifiant, un exemple de motif et un exemple de texte pour lesquels la fonction codée ne renvoie pas le résultat attendu.

Choisissons le motif `'aa'` et le texte `'baa'`. À l'indice 0, la fonction compare les caractères de droite à gauche : le dernier `'a'` correspond, mais `'b'` ne correspond pas à `'a'`. Elle avance alors de 2 indices, jusqu'à `i = 2`. La condition de boucle est désormais fausse, donc elle renvoie `-1` sans examiner l'indice 1, où le motif apparaît pourtant.

> **RÉPONSE À RETENIR**
>
> Avec le motif **`'aa'`** et le texte **`'baa'`**, la fonction renvoie `-1`, alors que le plus petit indice où le motif apparaît est **1**.

Exercice 3

8 points

Routage et utilisation d'une structure arborescente

Partie A

Question 1.

Sujet p. 8, 9

> Donner le nombre maximum de machines pouvant être connectées au réseau ayant pour adresse 172.16.0.0/25.

Une adresse IPv4 comporte 32 bits. Le préfixe `/25` en réserve 25 à la partie réseau ; il reste donc `32 − 25 = 7` bits pour la partie machine. Il y a $2^7 = 128$ adresses au total. Comme l'indique la figure 1, l'adresse de réseau `172.16.0.0` et l'adresse de diffusion `172.16.0.127` ne sont pas attribuables à des machines.

RÉPONSE À RETENIR

Le réseau peut accueillir au maximum **126 machines** : `128 − 2 = 126`.

Question 2.

Sujet p. 8, 9

> Recopier et compléter le tableau en indiquant le prochain saut à effectuer pour atteindre les adresses IP considérées, conformément à la table de routage : 172.16.0.10, 172.16.0.100 et 172.16.0.200.

On utilise la route la plus spécifique qui correspond à l'adresse destination. La route exacte vers `172.16.0.10` indique `10.0.0.3`. L'adresse `172.16.0.100` appartient au réseau `172.16.0.0/25` (adresses de `172.16.0.0` à `172.16.0.127`) ; deux prochains sauts de poids égal sont prévus. Enfin, `172.16.0.200` n'appartient pas à ce réseau et utilise la route par défaut.

RÉPONSE À RETENIR

IP destination	Prochain saut
172.16.0.10	10.0.0.3
172.16.0.100	10.0.0.7 ou 10.0.0.9
172.16.0.200	10.0.0.5

Pour `172.16.0.100`, la table propose deux prochains sauts de même poids ; l'un ou l'autre peut être utilisé.

Question 3.

Sujet p. 8, 9

> Recopier et compléter le tableau avec les adresses des interfaces des routeurs R1 à R4 à joindre depuis R0, compte tenu des informations de sa table.

Les adresses des prochains sauts données par la table sont celles des interfaces des routeurs voisins à joindre depuis R0. La route par défaut passe par `10.0.0.5` (R1), les routes exactes passent par `10.0.0.3` (R2), et le réseau `172.16.0.0/25` propose les prochains sauts `10.0.0.7` (R3) et `10.0.0.9` (R4).

RÉPONSE À RETENIR

Routeur	IP de l'interface
R1	10.0.0.5
R2	10.0.0.3
R3	10.0.0.7
R4	10.0.0.9

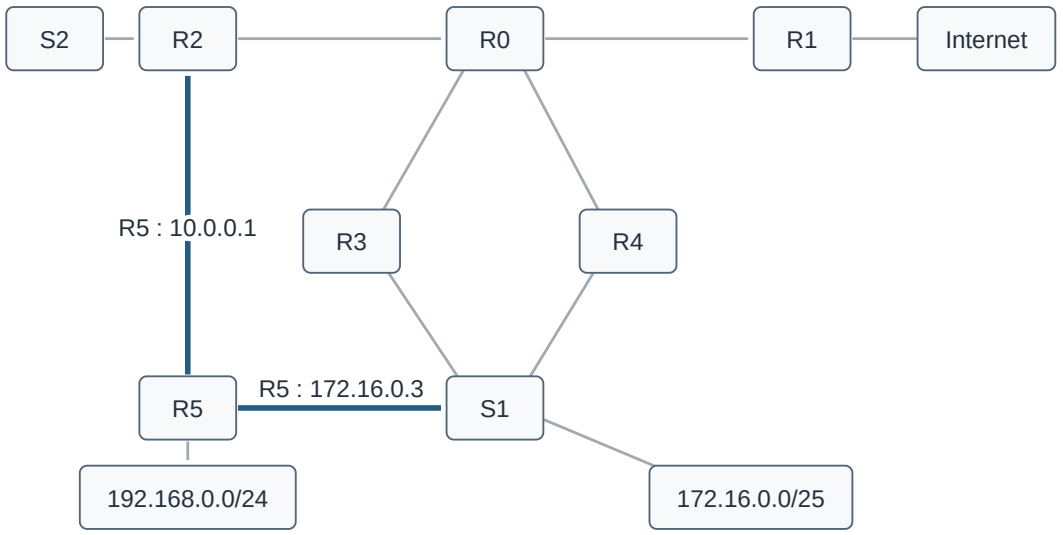
Question 4.

Sujet p. 9, 10

> On connecte le réseau 192.168.0.0/24 aux réseaux existants via le routeur R5 : un premier lien est établi entre R5 (adresse 172.16.0.3) et le switch S1, et un second lien entre R5 (adresse 10.0.0.1) et R2. Recopier et compléter l'architecture réseau (Figure 2) en ajoutant ces nouveaux liens.

Il faut ajouter les deux liens indiqués dans l'énoncé : R5–S1, avec l'adresse `172.16.0.3` sur l'interface de R5, et R5–R2, avec l'adresse `10.0.0.1` sur l'interface de R5. Le lien déjà représenté entre R5 et le réseau `192.168.0.0/24` est conservé.

Architecture complétée par les deux liens de R5



Les liens mis en évidence sont les deux liaisons à ajouter. Les adresses indiquées sont celles des interfaces de R5, comme dans l'énoncé.

RÉPONSE À RETENIR

Les deux liaisons à ajouter sont **R5–S1** (interface R5 : `172.16.0.3`) et **R5–R2** (interface R5 : `10.0.0.1`).

Question 5.

Sujet p. 9, 10

> Sachant que le protocole RIP est utilisé, déterminer, en justifiant, le coût du chemin allant de la machine dont l'adresse IP est 172.16.0.10 jusqu'au réseau 192.168.0.0/24.

Sur la figure 2, la machine `172.16.0.10` est reliée à S2, lui-même relié à R2. Le réseau `192.168.0.0/24` est relié à R5. Avec le lien direct ajouté entre R2 et R5, le trajet passe par les deux routeurs R2 et R5. Le détour par R0, R3 ou R4 ferait traverser davantage de routeurs. RIP choisit le trajet qui minimise le nombre de routeurs traversés.

> **RÉPONSE À RETENIR**
>
> Le coût RIP est de **2 sauts de routeur** : R2 puis R5.

Question 6.

Sujet p. 10

> Calculer le coût des liaisons pour les 2 valeurs des bandes passantes utilisées. Le coût d'une liaison est donné par la formule coût = 10^9/BP, où BP est la bande passante de la connexion en bit par seconde. Un câble Ethernet a une bande passante de 1 Gbit/s et un câble fibre optique a une bande passante de 10 Gbit/s.

On applique la formule $\text{coût} = 10^9 / \text{BP}$, en exprimant la bande passante en bits par seconde. Une liaison Ethernet a une bande passante de $1\ \text{Gbit/s} = 10^9\ \text{bit/s}$; une liaison fibre optique a une bande passante de $10\ \text{Gbit/s} = 10 \times 10^9\ \text{bit/s}$.

> **RÉPONSE À RETENIR**
>
> Pour Ethernet : $10^9 / 10^9 = 1$. Pour la fibre optique : $10^9 / (10 \times 10^9) = 0{,}1$. Les coûts sont donc **1** pour Ethernet et **0,1** pour la fibre optique.

Question 7.

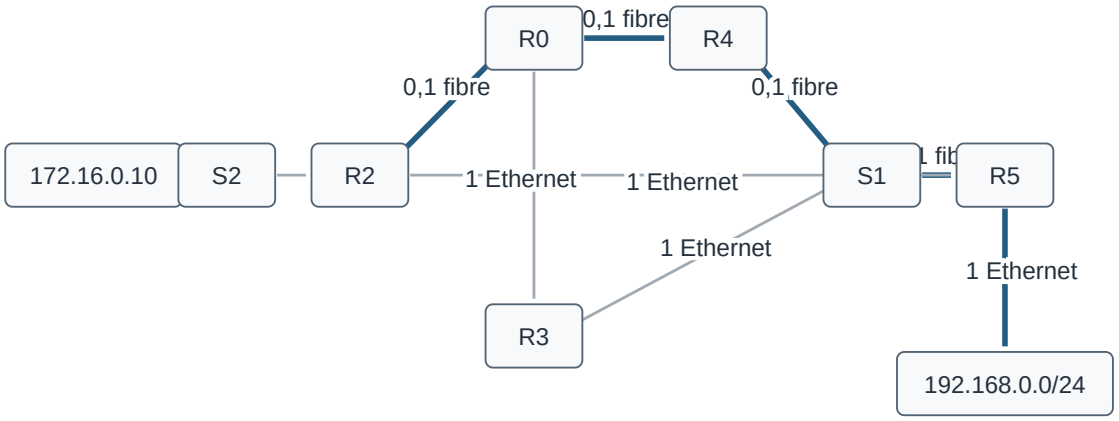
Sujet p. 9, 10

> Sachant que le protocole OSPF est utilisé, déterminer, en justifiant, le coût du chemin allant de la machine dont l'adresse IP est 172.16.0.10 jusqu'au réseau 192.168.0.0/24.

La machine `172.16.0.10` est reliée à S2 puis à R2. Pour atteindre R5, le chemin direct R2–R5 est en Ethernet et coûte 1. L'autre chemin pertinent est R2–R0–R4–S1–R5 : les quatre liaisons R2–R0, R0–R4, R4–S1 et S1–R5 sont en fibre, donc son coût est $4 \times 0{,}1 = 0{,}4$. OSPF choisit ce dernier chemin, de coût inférieur. Pour atteindre le réseau `192.168.0.0/24`, on ajoute la liaison Ethernet entre R5 et ce réseau, de coût 1 ; le coût total est donc $0{,}4 + 1 = 1{,}4$.

FormaV - document d'entraînement non officiel

Chemin OSPF minimal vers 192.168.0.0/24



Les coûts indiqués sont ceux obtenus à la question 6. Le chemin en évidence coûte 0,4 jusqu'à R5, puis 1 pour la liaison vers le réseau destination.

> **RÉPONSE À RETENIR**
>
> Le chemin retenu est **R2 → R0 → R4 → S1 → R5**, puis la liaison Ethernet vers `192.168.0.0/24`. Son coût total est **1,4**.

Partie B

Question 8.

Sujet p. 11

> Recopier et compléter l'arbre correspondant à la table de routage proposée.

Chaque adresse comporte quatre octets : on suit donc quatre arêtes depuis la racine. Le nœud atteint après le quatrième octet contient le prochain saut associé à l'adresse. Les chemins communs sont partagés. On ajoute aux quatre entrées déjà représentées les quatre autres entrées de la table.

RÉPONSE À RETENIR

```
Racine
├── 168
│   └── 16
│       └── 0
│           ├── 100  → saut 10.0.0.3
│           ├── 200  → saut 10.0.0.4
│           └── 10   → saut 10.0.0.7
├── 203
│   └── 18
│       ├── 17
│       │   └── 10  → saut 10.0.0.5
│       └── 15
│           ├── 30  → saut 10.0.0.8
│           ├── 20  → saut 10.0.0.9
│           └── 10  → saut 10.0.0.10
└── 0
    └── 0
        └── 0
            └── 0  → saut 10.0.0.6
```

Par exemple, les octets `168 → 16 → 0 → 10` conduisent au nœud qui contient le prochain saut `10.0.0.7`.

Question 9.

Sujet p. 12

> Compléter les lignes 7 et 8 de la fonction rechercher_enfant donnée ci-dessus.

Si l'octet n'est pas une clé du dictionnaire des enfants, la méthode doit renvoyer `None`. Sinon, elle renvoie le nœud enfant associé à cet octet.

RÉPONSE À RETENIR

```
if not (octet in self.dict_adjacence):
    return None
return self.dict_adjacence[octet]
```

Question 10.

Sujet p. 12, 13

> Compléter le corps, à partir de la ligne 6, de la fonction rechercher.

On parcourt les octets de l'adresse dans l'ordre. Si un enfant n'existe pas, l'adresse n'est pas dans l'arbre et on renvoie `None`. Après les quatre octets, on vérifie que le nœud contient bien un prochain saut : un nœud dont le chemin n'est qu'un préfixe n'en contient pas et ne correspond donc pas à une destination enregistrée. La méthode renvoie alors le nœud de destination.

RÉPONSE À RETENIR

```
noeud = self.racine
for octet in ip_dest.split('.'):
    noeud = noeud.rechercher_enfant(octet)
    if noeud is None:
        return None
if noeud.saut == '':
    return None
return noeud
```

Question 11.

Sujet p. 12, 13

> Expliquer, en justifiant, le rôle de la fonction mystere.

Le rôle visé de `mystere` est d'insérer dans l'arbre l'adresse destination `ip_dest` et de placer `saut` dans le nœud atteint après le parcours des quatre octets. Pour cela, la fonction devrait réutiliser un enfant s'il existe et créer un nœud si l'enfant est absent. La dernière instruction, `noeud.saut = saut`, associe ensuite le prochain saut à cette destination.

Mais le code imprimé ne réalise pas correctement cette insertion : à la ligne 17, il teste `noeud_suivant is not None`, puis crée un nœud dans ce cas. Le test est inversé : il faut créer le nœud lorsque `noeud_suivant is None`. Tel qu'il est donné, un enfant déjà existant est remplacé et un enfant absent peut conduire à poursuivre avec `None`.

RÉPONSE À RETENIR

Rôle visé : insérer ou mettre à jour l'association entre une adresse IP de destination et son prochain saut dans l'arbre. **Le code tel qu'imprimé comporte un test inversé** à la ligne 17 et ne remplit pas correctement ce rôle.

Question 12.

Sujet p. 11, 12, 13

> On possède un arbre contenant les informations d'une table de routage. Proposer un algorithme permettant d'inverser cette table, c'est-à-dire d'associer à un saut possible une adresse IP de destination.

On parcourt tous les nœuds de l'arbre en conservant le chemin d'octets qui mène à chacun. Lorsqu'un nœud contient un saut non vide, on ajoute l'adresse formée par ce chemin à la liste associée à ce saut. Une pile permet de réaliser ce parcours sans récursion. Il faut prévoir une liste par saut, car plusieurs destinations peuvent avoir le même prochain saut.

RÉPONSE À RETENIR

```
def inverser(racine):
    resultat = {}
    pile = [(racine, '')]

    while pile:
        noeud, adresse = pile.pop()

        if noeud.saut != '':
            if noeud.saut not in resultat:
                resultat[noeud.saut] = []
            resultat[noeud.saut].append(adresse)

        for octet, enfant in noeud.dict_adjacence.items():
            if adresse == '':
                nouvelle_adresse = octet
            else:
                nouvelle_adresse = adresse + '.' + octet
            pile.append((enfant, nouvelle_adresse))

    return resultat
```

Par exemple, avec l'arbre de la question 8, le résultat associe notamment le saut `10.0.0.3` à l'adresse destination `168.16.0.100`, et le saut `10.0.0.6` à `0.0.0.0`.

Sujet : https://www.sujetdebac.fr/annales-pdf/2026/spe-numerique-informatique-2026-polynesie-2-sujet-officiel.pdf
© FormaV EI • Proposition de correction, distincte d'un corrigé officiel.

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.