BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2026

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 2

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 13 pages numérotées de 1/13 à 13/13.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

EXERCICE 1 (6 points)

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

Les mots clés du langage SQL suivants pourront être utilisés dans les requêtes : `SELECT FROM, WHERE, JOIN ON, INSERT INTO VALUES, UPDATE SET, COUNT, AND, OR, DISTINCT.`

Pour rappel : le mot clé `DISTINCT` est utilisé pour supprimer les doublons des résultats d'une requête, en ne retournant que des lignes uniques.

On s'intéresse dans cet exercice à la mise à disposition de données « temps réel » de qualité de l'air.

Le Bureau de la Qualité de l'Air (BQA) au sein du Ministère de la Transition Ecologique et Solidaire (MTES) a mandaté le Laboratoire Central de Surveillance de la Qualité de l'Air (LCSQA) pour organiser la mise à disposition du public au niveau national des données « temps réel » des mesures des concentrations de polluants atmosphériques.

Les données d'observation sont issues de la surveillance réglementaire de la qualité de l'air. Elles décrivent **les concentrations moyennes horaires des polluants** réglementés surveillés par des appareils de mesure automatiques installés sur des stations fixes.

Les stations de mesure de la qualité de l'air sont gérées par les Associations Agréées de Surveillance de la Qualité de l'Air (**AASQA**). Les AASQA sont donc producteurs et responsables de ces données. On compte 18 AASQA, correspondant aux régions métropolitaines et aux départements et régions d'outre-mer.

Les données disponibles permettent de documenter les concentrations de polluants atmosphériques (ozone O3, monoxyde d'azote NO et dioxyde d'azote NO2, dioxyde de soufre SO2, particules de diamètre inférieur à 10 µm PM10, particules de diamètre inférieur à 2,5 µm PM2.5, monoxyde de carbone CO, etc.).

Les **stations** dont sont issues les observations ne disposent pas toutes de mesures pour l'ensemble de ces polluants, certaines ne mesurent qu'un seul polluant de la liste, d'autres en mesurent plusieurs.

Pour cet exercice, le modèle relationnel suivant a été retenu :

AASQA	
Id	CHAR(7)
Nom	VARCHAR(50)
FuseauHoraire	VARCHAR(50)
Siteweb	VARCHAR(50)

1 — n

Station	
Id	CHAR(7)
Nom	VARCHAR(50)
Lattitude	DECIMAL(10,6)
Longitude	DECIMAL(10,6)
Altitude	INT
TypeImplantation	VARCHAR(50)
AASQAId	CHAR(7)

1 — n

Mesure	
StationId	CHAR(7)
DateDebut	DATETIME
DateFin	DATETIME
Polluant	VARCHAR(50)
TypeInfluence	VARCHAR(50)
Valeur	DECIMAL(10,1)

Les clés primaires sont soulignées.

Les éventuelles clés étrangères ne sont volontairement pas identifiées.

Une AASQA gère zéro, une ou plusieurs stations et une station est gérée par une et une seule AASQA. De la même façon, une station réalise zéro, une ou plusieurs mesures et une mesure est réalisée par une et une seule station.

Quelques occurrences de chaque table ou relation sont données ci-dessous.

Relation AASQA

Id	Nom	FuseauHoraire	Siteweb
FR072A	AIR BREIZH	UTC	http://www.airbreizh.asso.fr
FR004A	AIRPARIF	UTC	http://www.airparif.asso.fr
FR061A	LIG'AIR	UTC	https://www.ligair.fr
FR065A	QUALITAIR CORSE	UTC	http://www.qualitaircorse.org
FR064A	GWAD'AIR	UTC-4	http://www.gwadair.fr
FR077A	HAWA MAYOTTE	UTC+3	http://www.hawa-mayotte.fr
FR069A	ATMO NORMANDIE	UTC	http://www.atmonormandie.fr/
FR068A	ATMO OCCITANIE	UTC	http://atmo-occitanie.org/
FR071A	ATMO AUVERGNE-RHONE-ALPES	UTC	http://www.atmo-auvergnerhonealpes.fr/
FR076A	ATMO BOURGOGNE-FRANCHE-COMTE	UTC	https://atmo-bfc.org/
FR074A	ATMO HAUTS DE FRANCE	UTC	http://www.atmo-hdf.fr

Relation `Station` :

Id	Nom	Latitude	Longitude	Altitude	TypeImplantation	AASQAId
FR37040	ABYMES RN1	16.254086	-61.53713	13	Périurbaine	FR064A
FR05083	Gonfreville l Orcher	49.50273	0.232486	80	Urbaine	FR069A
FR33203	Annecy Rocade	45.9097	6.11825	452	Urbaine	FR071A
FR07056	Pays du Mezenc	44.98377	4.226006	1191	Rurale régionale	FR071A
FR04143	Paris Centre	48.859	2.351	37	Urbaine	FR004A

Relation `Mesure`. La colonne **Valeur** est donnée en µg/m³)

StationId	DateDebut	DateFin	Polluant	TypeInfluence	Valeur
FR05083	15/04/2022 00:00	15/04/2022 01:00	SO2	Industrielle	3.3
FR05083	15/04/2022 01:00	15/04/2022 02:00	SO2	Industrielle	3.3
FR33203	15/04/2022 07:00	15/04/2022 08:00	PM10	Trafic	66.5
FR33203	15/04/2022 08:00	15/04/2022 09:00	PM10	Trafic	45
FR07056	15/04/2022 13:00	15/04/2022 14:00	O3	Fond	101.2
FR07056	15/04/2022 14:00	15/04/2022 15:00	O3	Fond	102.7

Partie A : Modèle relationnel

1. Expliquer le rôle de la clé primaire dans une relation.
2. Donner les quatre attributs formant la clé primaire pour la relation `Mesure`.
3. Expliquer le rôle d'une clé étrangère.
4. Donner un exemple de clé étrangère dans le modèle relationnel ci-dessus.

Partie B : Alimentation de la base de données

Chaque jour des mesures sont réalisées. Il faut ajouter ces mesures à la base de données.

5. 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 10h à 11h pour la station "Annecy Rocade".

6. 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 00h à 01h pour la station d'identifiant "FR41017", l'exécution échoue et l'exception suivante est levée :

   IntegrityError: FOREIGN KEY constraint failed

   Expliquer cette erreur.

Partie C : Exploitation de la base de données

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

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

9. 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";
```

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).

10. 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".

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

12. 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.

EXERCICE 2 (6 points)

Cet exercice porte sur la programmation Python, les structures de données, les graphes et la recherche de motif dans un texte.

Cet exercice comporte 2 parties qui peuvent être traitées indépendamment.

Le jeu du taquin consiste à déplacer des tuiles adjacentes à la case vide jusqu'à arriver à mettre tous les nombres dans l'ordre. Voici un exemple de positions pour un jeu du taquin de dimension 3 x 3 soit 9 cases.

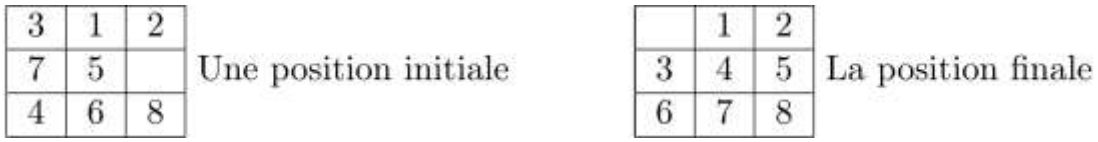

Figure 1. Exemple de jeu du taquin sur 9 cases

Partie A

Dans un premier temps, on s'intéresse au jeu du taquin sur 4 cases. Voici une figure qui représente les 12 positions que l'on peut atteindre à partir de la position finale en déplaçant les tuiles. Deux positions sont reliées si on peut passer de l'une à l'autre en un seul coup, c'est-à-dire en déplaçant une tuile.

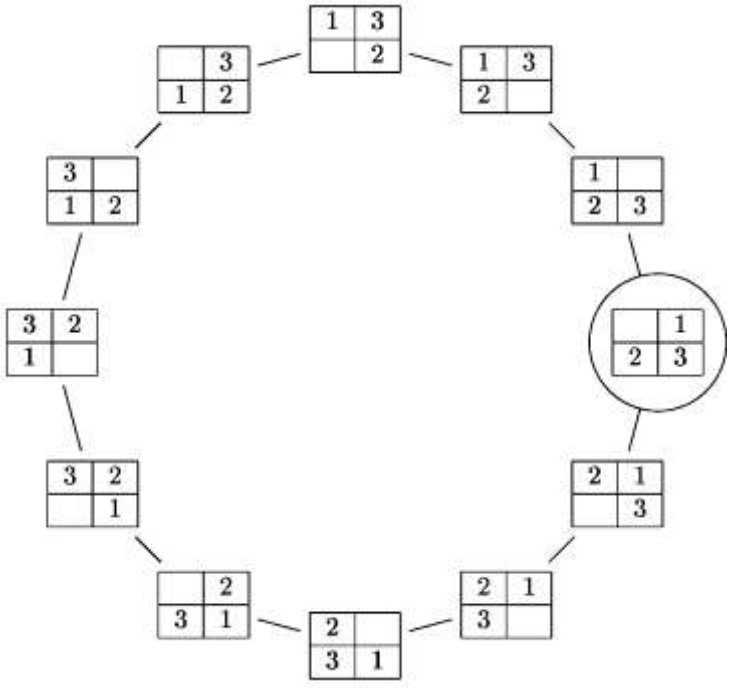

Figure 2. Positions accessibles depuis la position finale du jeu du taquin sur 4 cases

1. Représenter une position des 3 tuiles qui n'est pas accessible depuis la position finale.

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

On s'intéresse maintenant au jeu du taquin sur 9 cases.

On représente, en Python, une position de taquin par une liste qui contient les nombres présents de gauche à droite puis de haut en bas. La case vide est représentée par le nombre 0. Par exemple, les positions de jeu du taquin de dimension 3 x 3 de la Figure 1 sont représentées respectivement par :

`[3, 1, 2, 7, 5, 0, 4, 6, 8]` et `[0, 1, 2, 3, 4, 5, 6, 7, 8]`.

3. 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).

4. 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.

```
1 def est_finale(pos):
2     for i in range(len(pos)):
3         if ...:
4             return ...
5     return ...
```

5. 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`.

```
1 def case_vide(pos):
2     res = ...
2     for i in range(9):
3         ...
4     return res
```

Exemple :

```
>> case_vide([3, 1, 2, 7, 5, 0, 4, 6, 8])
5
>> case_vide([3, 0, 2, 7, 5, 0, 4, 6, 8])
-1
```

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).

On suppose que l'on a à disposition une fonction `voisins` qui prend en paramètre une position représentée par une liste `pos` et qui renvoie la liste des positions que l'on peut

atteindre en déplaçant une tuile. 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.

```
1  def est_possible(pos):
2      if est_finale(pos):
3          return True
4      # Liste pour retenir les positions déjà rencontrées
5      visites = []
6      for v in voisins(pos):
7          if v not in visites:
8              visites.append(v)
9              if est_possible(v):
10                 return True
```

Lorsqu'on teste la fonction, l'exécution ne semble pas se terminer.

7. Expliquer le problème et proposer une solution (sans la programmer).

Partie B

On cherche maintenant à savoir si un mot est présent dans une liste de mots acceptés. Cette liste de mots acceptés est stockée sous la forme d'un texte contenant chacun des mots séparés par des espaces. Pour savoir si un mot est valide, il suffit donc de le chercher dans le texte, entre deux espaces.

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

On propose l'implémentation suivante pour rechercher un motif dans un texte.

```
1  def recherche(motif, texte):
2      """Renvoie le plus petit indice i tel que motif apparait dans
3      texte à l'indice i ou -1 si motif n'apparait pas"""
4      n = len(texte)
5      m = len(motif)
6      i = 0
7      while i + m - 1 < n:
8          j = m - 1
9          while texte[i+j] == motif[j]:
10             if j == 0:
11                 return i
12             j = j - 1
13         i = i + m
14     return -1
```

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

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

EXERCICE 3 (8 points)

Cet exercice porte sur le routage et sur l'utilisation d'une structure arborescente.

Partie A

Rappels :

Une adresse IPv4 est composée de 4 octets, soit 32 bits. Elle est notée a.b.c.d, où a, b, c et d sont les valeurs décimales des 4 octets.

La notation a.b.c.d/n, appelée notation CIDR (**C**lassless **I**nter **D**omain **R**outing), signifie que les n premiers bits à gauche de l'adresse IP représentent la partie « réseau », les bits à droite qui suivent représentent la partie « machine ».

On donne la table de routage suivante :

```
$ ip route show scope global table 100
default via 10.0.0.5 dev out2
172.16.0.0/25
    nexthop via 10.0.0.7 dev out3 weight 1
    nexthop via 10.0.0.9 dev out4 weight 1
172.16.0.10 nexthop via 10.0.0.3 dev out1
172.16.0.20 nexthop via 10.0.0.3 dev out1
172.16.0.30 nexthop via 10.0.0.3 dev out1
172.16.0.40 nexthop via 10.0.0.3 dev out1
```

Un calculateur de masque IPv4 donne les informations suivantes après avoir saisi la commande `172.16.0.0/25` :

Adresse IPv4
CIDR : 25
Masque de réseau : 255.255.255.128
Masque inverse : 0.0.0.127

En mode réseau :
Adresse de réseau : 172.16.0.0
Première adresse : 172.16.0.1
Dernière adresse : 172.16.0.126
Adresse de broadcast : 172.16.0.127
Nombre d'adresses IP disponibles : 126

En mode filtre :
Première adresse : 172.16.0.0
Dernière adresse : 172.16.0.127
Nombre d'adresses IP disponibles : 128

Figure 1. Masque IPv4

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

2. Recopier et compléter le tableau suivant, en indiquant le prochain saut à effectuer pour atteindre l'adresse `ip` considérée, conformément à la table de routage.

IP destination	Prochain saut
172.16.0.10	
172.16.0.100	
172.16.0.200	

On donne l'architecture du réseau de routeurs correspondant à la table précédente du routeur R0.

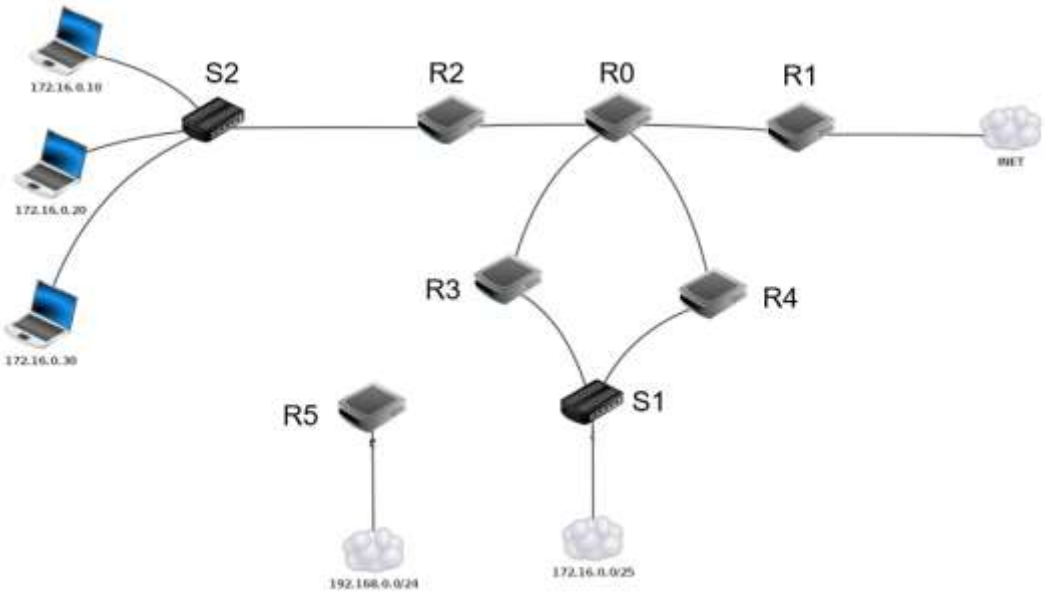

Figure 2. Architecture Réseau

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

Routeur	IP de l'interface
R1	
R2	
R3	
R4	

On connecte le réseau `192.168.0.0/24` aux réseaux existants via le routeur R5 de la façon suivante :

- un premier lien est établi entre R5 (adresse `172.16.0.3`) et le switch S1 ;
- un second lien est établi entre R5 (adresse `10.0.0.1`) et R2.

4. Recopier et compléter l'architecture réseau (Figure 2.) en ajoutant ces nouveaux liens.

Le protocole RIP (Routing Information Protocol) est un protocole de routage qui minimise le nombre de routeurs par lesquels les paquets transitent.

5. 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.

Le protocole OSPF (Open Shortest Path First) est un protocole de routage qui minimise le coût du transit des paquets.

Le coût d'une liaison est donné par la formule :

$\text{coût} = \frac{10^9}{BP}$ où BP est la bande passante de la connexion en bit par seconde.

Toutes les liaisons sont réalisées par des câbles Ethernet sauf les liaisons R0-R2, R0-R4, R4-S1 et S1-R5 qui sont réalisées par des câbles fibre optique.

Un câble Ethernet a une bande passante de 1 Gbit/s. Un câble fibre optique a une bande passante de 10 Gbit/s.

6. Calculer le coût des liaisons pour les 2 valeurs des bandes passantes utilisées.

7. 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.

Partie B

Dans cette partie, on se propose de mettre en œuvre une structure arborescente pour la manipulation d'une table de routage.

IP destination	Prochain saut
168.16.0.100	10.0.0.3
168.16.0.200	10.0.0.4
203.18.17.10	10.0.0.5
0.0.0.0	10.0.0.6
168.16.0.10	10.0.0.7
203.18.15.30	10.0.0.8
203.18.15.20	10.0.0.9
203.18.15.10	10.0.0.10

Chaque nœud de l'arbre contient un dictionnaire qui associe à une lettre un nœud enfant. Depuis la racine de l'arbre, le chemin parcouru jusqu'à un nœud est une suite d'arêtes pondérées chacune par une lettre. Ce chemin forme un mot.

Si ce mot est une adresse IP de destination dans la table de routage, le nœud contient l'adresse IP du prochain saut. De plus, si le dictionnaire des successeurs n'est pas vide, ce mot est le préfixe d'au moins une adresse IP de prochain saut.

Après insertion des quatre premières lignes, on obtient :

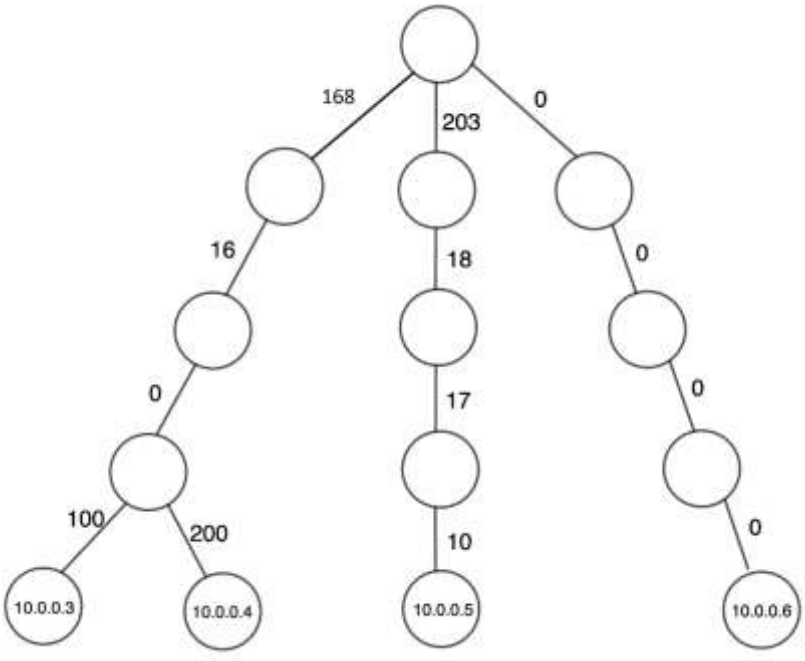

Figure 1. Arbre créé après quatre insertions

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

La classe `Noeud` est définie comme suit :

```
1 class Noeud():
2     def __init__(self):
3         self.saut = ''
4         self.dict_adjacence = {}
5     def rechercher_enfant(self, octet):
6         if not (octet in self.dict_adjacence):
7             return ...
8         return ...
```

Les instances de cette classe contiennent deux attributs :

- `saut` qui est une chaîne de caractères non nulle lorsque le chemin suivi pour atteindre ce nœud correspond à une adresse IP de destination ;
- `dict_adjacence` qui est un dictionnaire contenant des couples octet, enfant.

La classe contient une méthode `rechercher_enfant` qui prend en paramètre un octet et qui renvoie le nœud correspondant à cet octet depuis le nœud courant. Si l'octet n'est pas enregistré dans le dictionnaire du nœud, la fonction renvoie `None`.

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

La classe `Arbre` est définie comme suit :

```
1 class Arbre() :
2     def __init__(self):
3         self.racine = Noeud()
4     def rechercher(self, ip_dest):
5         # Corps de la fonction rechercher
6         ...
11    def mystere(self, ip_dest, saut):
12        noeud = self.racine
13        octets = ip_dest.split('.')
14        i = 0
15        while i < len(octets):
16            noeud_suivant = noeud.rechercher_enfant(octets[i])
17            if noeud_suivant is not None :
18                noeud_suivant = Noeud()
19                noeud.dict_adjacence[octets[i]] = noeud_suivant
20            noeud = noeud_suivant
21            i = i+1
22        noeud.saut = saut
```

Les instances de cette classe contiennent un attribut `racine` qui est un nœud.

La classe contient deux méthodes :

- `rechercher` qui renvoie le nœud atteint après le parcours du chemin correspondant à `ip_dest`. Si `ip_dest` n'est pas dans l'arbre, la fonction renvoie `None` ;
- `mystere` qui prend pour paramètre un `ip_dest` et un `saut`.

Pour rappel, la fonction `split` en Python divise une chaîne de caractères en une liste de sous-chaînes en fonction d'un séparateur spécifié.

Par exemple;

```
texte = 'Bonjour-le-monde'
separateur = '-'
resultat = texte.split(separateur)
```

donne comme résultat `['Bonjour', 'le', 'monde']`

10. Compléter le corps (à partir de la ligne 6) de la fonction `rechercher`.
11. Expliquer, en justifiant, le rôle de la fonction `mystere`.

On souhaite maintenant inverser la table de routage c'est-à-dire associer à un saut possible une adresse IP de destination.

12. On possède un arbre contenant les informations d'une table de routage. Proposer un algorithme permettant d'inverser cette table.