BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2026

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU MARDI 16 JUIN 2026

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 16 pages numérotées de 1/16 à 16/16.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

Exercice 1 (6 points)

Cet exercice porte sur les réseaux, les protocoles de routage et la sécurisation des communications.

Le réseau informatique d'un lycée est réparti sur plusieurs sites : campus, internat, services administratifs, etc.

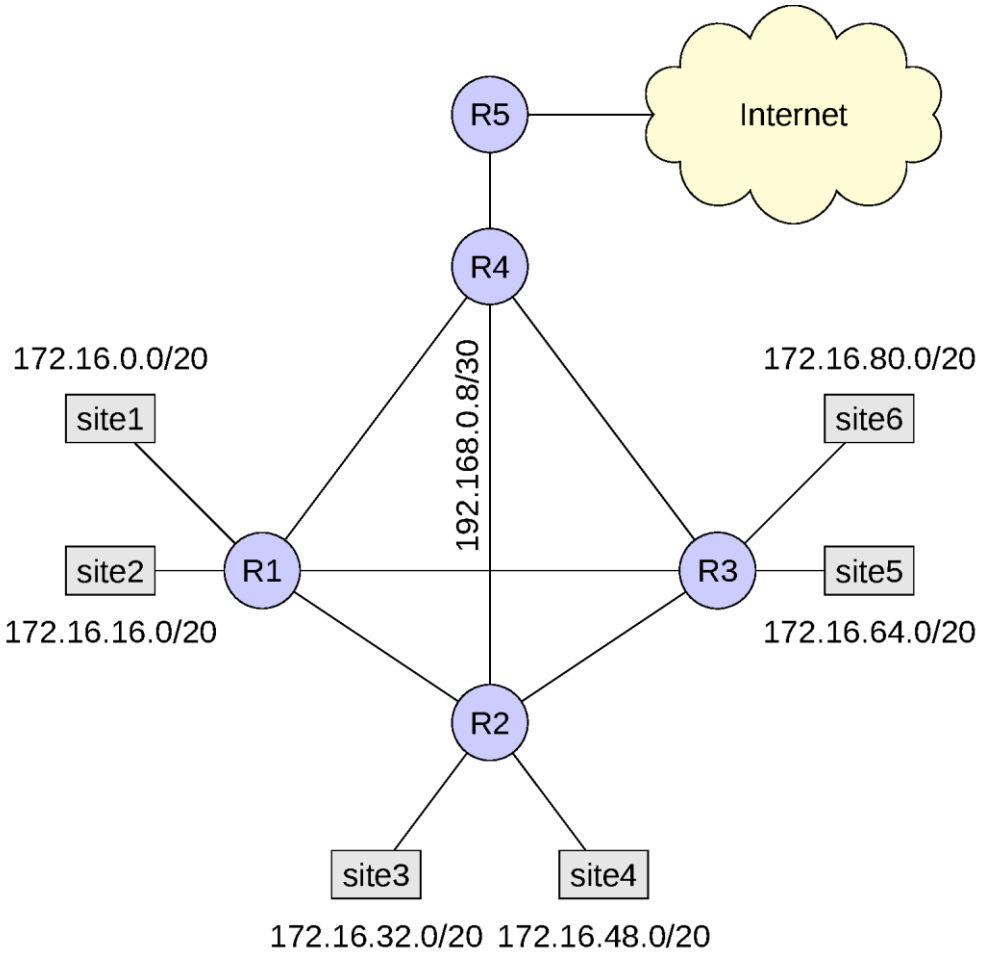

Figure 1. Schéma du réseau du lycée

Chaque site possède :

- un **sous-réseau dédié**, avec un adressage IP propre ;
- un **routeur** permettant la communication entre sites et vers Internet.

On utilise la notation CIDR pour définir l'adressage d'un réseau : la notation `a.b.c.d/n` signifie que les `n` bits de poids fort (à gauche de l'adresse IP) désignent la *partie réseau* de l'adresse IP et que les bits suivants de poids faible (à droite de l'adresse IP) désignent la *partie machine*. Dans un même sous-réseau, toutes les machines utilisent des adresses IP ayant la même partie réseau.

L’administrateur réseau a établi la convention d’adressage suivante :

- la **passerelle** (*gateway*) de chaque site correspond à la **dernière adresse IP disponible** du sous-réseau ;
- elle est affectée à l’interface du **routeur connecté au site**.

Exemple :

Pour le réseau du **site6** (`172.16.80.0/20`) :

- Adresse réseau : `172.16.80.0`
- Adresse de passerelle : `172.16.95.254`
- Adresse de l’interface du routeur `R3` dans le **site6** : `172.16.95.254`

Partie A : Configuration IP du site3

Un hôte situé dans le **site3** possède la configuration suivante dans son fichier `/etc/network/interfaces` :

```
iface eth0 inet static
   address 172.16.32.15
   netmask 255.255.240.0
   gateway 172.16.48.254
```

1. Convertir les adresses suivantes en **binaire sur 32 bits** :
   - Adresse IP : `172.16.32.15`
   - Masque de sous-réseau : `255.255.240.0`

   Donner le résultat sous la forme :

   ```
   Adresse IP : XXXXXXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX
   Masque     : XXXXXXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX
   ```

2. Indiquer le nombre d’hôtes que le réseau peut accueillir, ainsi que la première et la dernière adresse IP utilisables par ces hôtes.

L’administrateur a remarqué que depuis cet hôte, il pouvait accéder aux serveurs ou imprimantes du **site3** mais ne pouvait pas accéder à Internet.

Il effectue un test de connectivité vers l’adresse `172.16.47.254` : ce test réussit.

3. Identifier dans la configuration de cet hôte le paramètre mal configuré et proposer une correction.

Le lien entre les routeurs **R2** et **R4** est configuré avec des adresses appartenant au réseau `192.168.0.8/30`.

4. Proposer deux adresses IP à attribuer aux interfaces des routeurs **R2** et **R4** pour établir ce lien.

Partie B

On s'intéresse ici au routage dynamique appliqué dans le réseau du lycée.

L'administrateur lance la commande suivante dans un terminal depuis un hôte du **site3**. Il observe le résultat :

```
$ traceroute education.gouv.fr

Détermination de l'itinéraire vers education.gouv.fr
  1     4 ms     7 ms     9 ms  R2 [172.16.47.254]
  2    22 ms     6 ms     8 ms  R3 [192.168.0.14]
  3    22 ms     7 ms     9 ms  R4 [192.168.0.6]
  4    21 ms     6 ms     8 ms  R5 [192.168.0.1]
  5    19 ms     7 ms     9 ms  internet-router [...]
  6     ...
```

Le protocole de routage dynamique activé sur les routeurs est **RIP**. On rappelle que dans le protocole RIP, le coût d'une route est donnée par le nombre de liens réseaux empruntés sur cette route.

5. D'après le résultat de la commande passée par l'administrateur, donner le chemin suivi par l'information et son coût pour aller de **R2** à **R5**.

6. Expliquer en quoi le réseau présente un dysfonctionnement et formuler une cause possible de ce problème.

Le protocole de routage dynamique activé sur les routeurs est maintenant **OSPF**.

On rappelle la relation suivante entre le débit et le coût d'une liaison entre deux routeurs :

$$cout = \frac{10^8}{debit}$$

7. Sur un réseau informatique, pour chacun des deux termes 'débit' et 'coût' d'une liaison, préciser si l'on souhaite le minimiser ou le maximiser.

On donne les informations suivantes sur les types de liaisons utilisées entre les routeurs :

- Liaison R1-R2 : Fast Ethernet
- Liaison R1-R3 : Fibre optique
- Liaison R1-R4 : Fast Ethernet
- Liaison R2-R3 : Ethernet
- Liaison R2-R4 : Ethernet
- Liaison R3-R4 : Fibre optique
- Liaison R4-R5 : Fibre optique

On donne également les débits des différentes liaisons utilisées sur le réseau :

- Ethernet : 10 Mbits/s
- Fast Ethernet : 100 Mbit/s
- Fibre optique : 4 Gbit/s

8. Donner, en précisant son coût, le chemin emprunté par un paquet de données depuis le **site4** jusqu'à Internet.

Partie C

On s'intéresse maintenant à la sécurisation des échanges de données entre les ordinateurs du réseau du lycée.

Un élève de l'internat se situant dans le **site2** a ouvert un logiciel malveillant ayant infecté son ordinateur. Le pirate a maintenant pris le contrôle de son ordinateur et peut donc maintenant accéder au réseau du lycée.

Alice se trouve dans le **site4** et Bob dans le **site5**, et entament une conversation privée en utilisant le réseau du lycée. On suppose que l'échange débute après l'arrivée du pirate sur le réseau.

9. Entre le chiffrement symétrique ou asymétrique, donner en justifiant lequel des deux Alice et Bob doivent préférer utiliser pour éviter que le pirate ne puisse connaître le contenu de leurs échanges.

10. Expliquer en quoi le protocole HTTPS est plus sécurisé que le protocole HTTP pour les échanges entre Alice et Bob.

Exercice 2 (6 points)

Cet exercice porte sur le binaire, les graphes, la récursivité.

Le jeu du *Recto-Verso*, de la famille des solitaires, se compose de neuf jetons disposés sur un plateau carré. Le plateau a 3 lignes et 3 colonnes. Chaque jeton possède une face *Recto*, dessinée en couleur noire sur les figures, et une face *Verso*, dessinée en blanc.

Partie A : représentation binaire des configurations

On appelle *configuration* l'état du plateau. La figure 1 illustre un exemple de configuration.

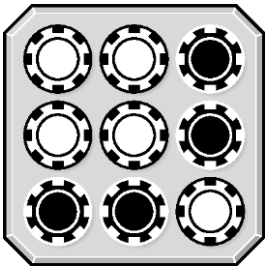

Figure 1. Illustration d'une configuration

1. Justifier que le nombre total de configurations possibles est 2^9.

Le but du jeu est, depuis une configuration quelconque, de retourner tous les jetons sur leur face Recto (noire). Pour ce faire, le seul type d'opération autorisé est de choisir une ligne ou une colonne ou une diagonale, et d'en retourner tous les jetons. On appelle une telle opération un *coup*. La figure 2 illustre les huit différents coups possibles à partir d'une configuration donnée.

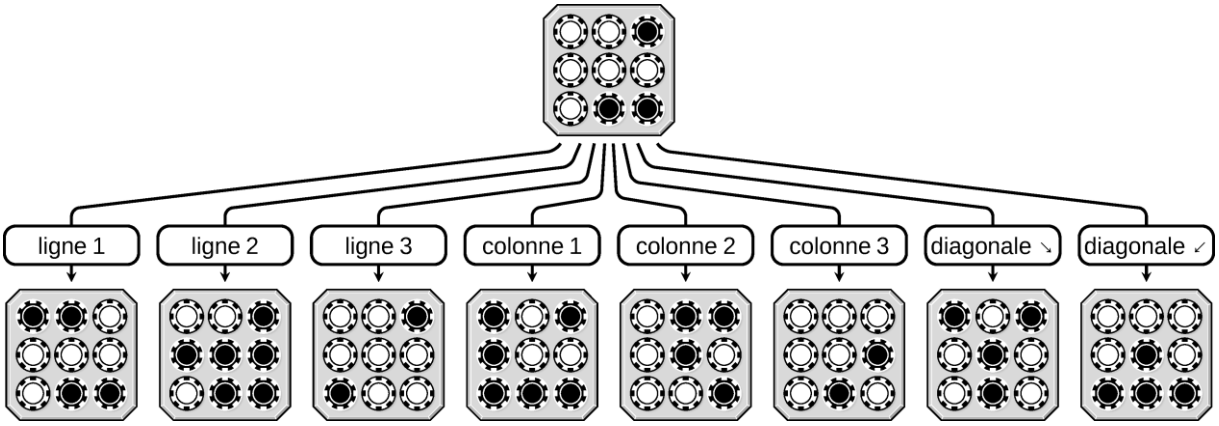

Figure 2. Les coups possibles depuis la configuration 67

2. Indiquer en justifiant si la séquence de coups suivante, jouée depuis la configuration donnée en **figure 1**, permet de mener à la victoire : ligne 1, ligne 2, colonne 2.

Une configuration est représentée, ligne par ligne et de haut en bas, par une liste de bits en considérant les jetons Recto comme des `1` et les jetons Verso comme des `0`. Ainsi on lit la configuration initiale de la figure 2 comme la liste `[0, 0, 1, 0, 0, 0, 0, 1, 1]`.

On peut aussi lire la liste de bits d'une configuration comme l'écriture binaire d'un nombre entier. Pour l'exemple de la figure 2, la configuration initiale est donc aussi représentée par l'entier `67`.

3. En détaillant le calcul, donner l'entier représentant la configuration illustrée en figure 1.

4. Implémenter une fonction `representant` prenant en paramètre une liste de `0` et de `1` et renvoyant l'entier correspondant.

Dans la suite on considère à disposition une fonction `binaire` prenant réciproquement en paramètre un entier et renvoyant son écriture en binaire sous forme d'une liste de **neuf** `0` ou `1`.

L'opérateur *ou exclusif*, noté $\oplus$, a pour table de vérité :

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

La fonction `xor` définie ci-dessous prend en paramètres deux bits `b1`, `b2` (de type `int`) et renvoie le résultat de `b1`⊕`b2`.

```
def xor(b1, b2):
    if b1 == b2:
        return 0
    else:
        return 1
```

L'opérateur $\oplus$ peut être appliqué sur deux listes de même longueur `l_x` et `l_y` contenant plusieurs bits : pour chaque indice de ces listes, on applique le $\oplus$ aux deux bits positionnés à cet indice. Par exemple, si `l_x = [1, 1]` et `l_y = [0, 1]`, alors `l_x`⊕`l_y` vaut `[1, 0]`, puisque `1`⊕`0` vaut `1`, et `1`⊕`1` vaut `0`.

5. Implémenter une fonction `xor_etendu` qui prend en paramètre deux listes de bits de même longueur `l_x` et `l_y` et renvoie la liste `l_x`⊕`l_y`.

On peut observer que pour un bit `X` donné, l'opération `X⊕1` a pour résultat l'inverse de `X` et l'opération `X⊕0` a pour résultat `X`.

Si `X` représente un jeton, alors `X⊕1` le retourne et `X⊕0` le laisse inchangé.

Appliquer un coup à une configuration revient à lister les jetons retournés et les jetons non-retournés. On peut donc représenter chaque coup par une liste de bits. Par exemple le coup "ligne 1" est représenté par la liste `[1, 1, 1, 0, 0, 0, 0, 0, 0]` et l'appliquer à la configuration `67` revient à calculer `[0, 0, 1, 0, 0, 0, 0, 1, 1]⊕[1, 1, 1, 0, 0, 0, 0, 0, 0]`.

La figure 3 illustre les 8 coups possibles de la même manière au moyen du ⊕.

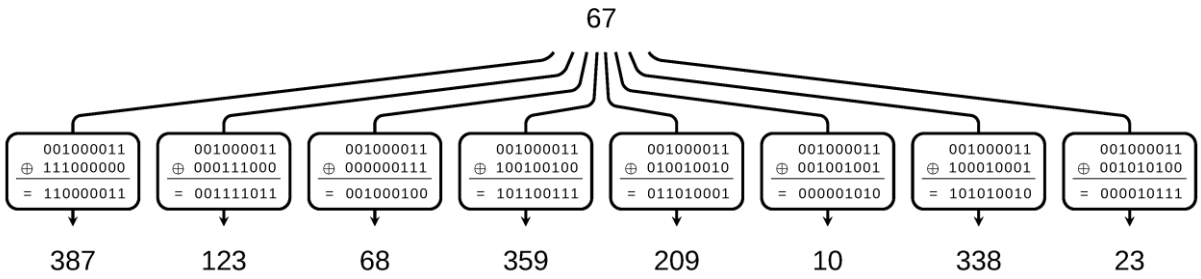

Figure 3. Les coups possibles depuis la configuration `67` appliqués par un `xor`. Par souci de lisibilité la notation en liste est simplifiée.

On suppose à disposition le dictionnaire `coups_possibles` qui énumère chaque coup représenté de la sorte :

```
coups_possibles = {
    "ligne 1"     : [1, 1, 1, 0, 0, 0, 0, 0, 0],
    "ligne 2"     : [0, 0, 0, 1, 1, 1, 0, 0, 0],
    "ligne 3"     : [0, 0, 0, 0, 0, 0, 1, 1, 1],
    "colonne 1"   : [1, 0, 0, 1, 0, 0, 1, 0, 0],
    "colonne 2"   : [0, 1, 0, 0, 1, 0, 0, 1, 0],
    "colonne 3"   : [0, 0, 1, 0, 0, 1, 0, 0, 1],
    "diagonale 1" : [1, 0, 0, 0, 1, 0, 0, 0, 1],
    "diagonale 2" : [0, 0, 1, 0, 1, 0, 1, 0, 0]
}
```

On souhaite implémenter une fonction qui prend en paramètre un entier `config` représentant une configuration et qui renvoie la liste des entiers correspondant à toutes les configurations accessibles en un seul coup.

```
def configurations_suivantes(config):
    config_bin = ...
    voisins = []
    for ... in ...:
        ... # ligne facultative
        ... = xor_etendu(..., ...)
```

```
        voisins.append(representant(...))
    return voisins
```

6. Recopier et compléter la fonction `configurations_suivantes`.

Partie B : graphe des configurations

On constate qu'il semble impossible d'atteindre la victoire depuis la configuration représentée par l'entier 4. On appelle *configuration perdante* toute configuration pour laquelle il n'existe aucune suite de coups menant à la victoire (c'est-à-dire à tous les jetons côté Recto). Inversement, on appelle *configuration gagnante* toute configuration qui n'est pas perdante.

La configuration représentée par l'entier 4 est effectivement perdante. Pour s'en convaincre on construit le graphe des configurations du jeu Recto-Verso.

Les sommets de ce graphe sont les différentes configurations possibles et il existe une arête $s_1 \to s_2$ d'un sommet s_1 vers un sommet s_2 si et seulement s'il est possible de passer de la configuration s_1 à la configuration s_2 en jouant un coup autorisé.

7. Montrer que s'il existe une arête $x \to y$ dans le graphe, alors il existe également une arête $y \to x$.

En Python, on choisit de représenter ce graphe par un dictionnaire construit grâce à la fonction `configurations_suivantes` :

```
graphe = {
  0 : [...],
  ...
  67 : [387, 123, 68, 359, 209, 10, 338, 23],
  ...
}
```

8. Une représentation par matrice d'adjacence aurait aussi pu être envisagée. Justifier le choix du dictionnaire en comparant pour les deux structures le nombre d'entiers nécessaires pour stocker en mémoire le graphe des configurations. On pourra laisser ce nombre écrit sous la forme d'une puissance de 2.

La fonction `parcours_profondeur` ci-dessous implémente un parcours en profondeur d'abord. Elle prend en paramètres la représentation d'un graphe, un entier `configuration` qui indique le sommet que l'on est en train de visiter, et une liste `vus` de sommets déjà visités par le parcours. Cette fonction effectue le parcours en rajoutant les sommets visités à la liste `vus`.

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

9. Recopier et compléter la fonction `parcours_profondeur`.

10. En utilisant la fonction `parcours_profondeur`, écrire en Python une suite d'instructions permettant de vérifier que la configuration représentée par l'entier 4 est bien perdante.

11. Indiquer et justifier quel parcours de graphe (en largeur ou en profondeur) est le plus approprié pour trouver un chemin partant d'une configuration gagnante et menant à la victoire en un minimum de coups.

Exercice 3 (8 points)

Cet exercice porte sur les arbres, l'algorithmique des arbres, et les bases de données.

On s'intéresse au fonctionnement interne d'une plateforme en ligne dédiée spécifiquement aux débats. Plutôt que de lister des messages des utilisateurs dans l'ordre chronologique, les différentes contributions sont structurées de façon arborescente afin de mieux comprendre les liens entre les arguments. Chaque contribution vient se rattacher à une **affirmation**, soit afin d'appuyer cette **affirmation** avec un argument **pour**, soit afin de l'attaquer avec un argument **contre**.

Cet exercice contient deux parties indépendantes l'une de l'autre.

Partie A : structure arborescente des arguments

Dans cette partie on modélise un arbre de débat pour un sujet de débat. La figure 1 montre un exemple d'arbre de débat.

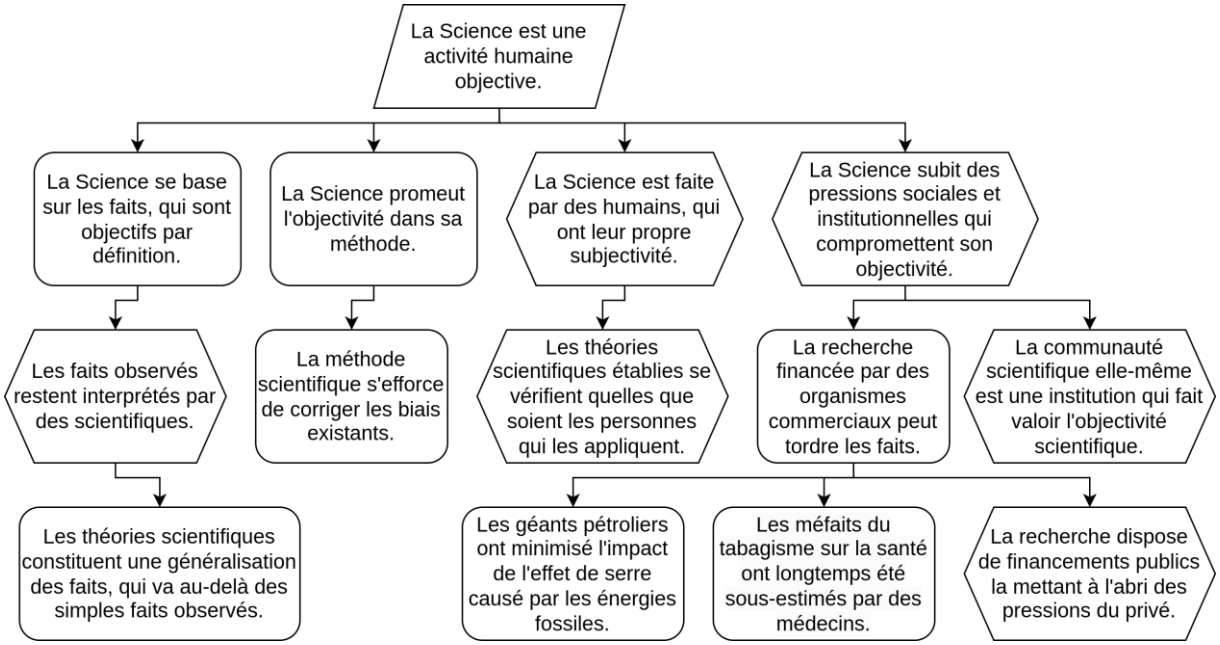

Figure 1. Arbre de débat sur l'objectivité de la science

Dans un arbre de débat, chaque nœud est une **affirmation**, et il y en a de trois sortes :

- le **sujet** du débat (dans le parallélogramme) est une **affirmation**, c'est le point de départ du débat ;
- un argument **pour** (dans un rectangle aux coins arrondis) peut venir soutenir une **affirmation**, cet argument est lui-même une **affirmation** ;
- un argument **contre** (dans un hexagone) peut venir attaquer une **affirmation**, cet argument est lui-même une **affirmation**.

De plus, chaque **affirmation** peut être *likée* (j'aime) ou *dislikée* (je n'aime pas) par les utilisateurs.

1. Indiquer le nom donné au nœud "point de départ du débat" dans le vocabulaire des arbres.

2. Indiquer en justifiant si un arbre de débat est un arbre binaire ou non.

On modélise la structure d'arbre de débat à l'aide de la classe suivante :

```
 1 class Affirmation:
 2     def __init__(self, phrase):
 3         """
 4         Création d'un objet Affirmation à partir d'une
 5         phrase (str).
 6         """
 7         self.contenu = phrase
 8         self.sorte = "" # vaudra "sujet", "pour" ou "contre"
 9                         # vaut "" avant ajout dans l'arbre
10         self.arguments = []
11         self.nb_likes = 0
12         self.nb_dislikes = 0
```

3. Donner le nom et le type de chaque attribut de la classe `Affirmation`.

On souhaite ajouter à cette classe une méthode `soutenir` qui permet d'ajouter un argument **pour** à l'affirmation.

On suppose à disposition une méthode `contrer` similaire qui permet d'ajouter un argument **contre** à l'affirmation.

```
 1 def soutenir(self, argument):
 2     """
 3     Ajoute l'Affirmation `argument` comme argument pour de
 4     l'Affirmation `self`.
 5
 6     Précondition : l'Affirmation ne doit pas avoir déjà été
 7     utilisée, ni comme sujet, ni comme pour, ni comme
 8     contre, ce qu'on vérifie grâce à son attribut `sorte`.
 9     """
10     assert argument.sorte == ...
11     ... .append(argument)
12     argument.sorte = "..."
```

4. Recopier et compléter les lignes 10 à 12 de la méthode `soutenir`, en respectant sa documentation.

Pour mesurer les débats les plus vifs, la plateforme de débats dispose de plusieurs mesures.

5. Écrire la méthode `nb_contre` qui permet d'obtenir le nombre d'arguments **contre** *directement* rattachés à cette `Affirmation`.

On s'intéresse à la méthode `mystere` suivante, qui utilise la méthode `nb_contre` :

```
1 def mystere(self):
2     m = self.nb_contre()
3     for arg in self.arguments:
4         candidat = arg.mystere()
5         if candidat > m:
6             m = candidat
7     return m
```

6. Expliquer par une phrase ce que renvoie la méthode `mystere`.

Afin de trancher le débat dans sa globalité, on souhaite évaluer une `Affirmation`, en tenant compte de la totalité des *likes* et des *dislikes* de chaque argument, et en tenant compte du fait que ces arguments soient **pour** ou **contre**. Si l'évaluation est positive le **pour** l'emporte, si elle est négative le **contre** l'emporte.

La formule pour l'**évaluation** d'une **affirmation** correspond à la somme de ses *likes* et des **évaluations** de ses arguments **pour**, à laquelle on soustrait ses *dislikes* et les **évaluations** de ses arguments **contre**.

7. Écrire, en utilisant uniquement des lignes de code choisies parmi les lignes suivantes, une méthode `evaluation` qui renvoie le résultat du calcul décrit dans la formule ci-dessus. Les lignes ne sont pas toutes à utiliser et l'indentation est à adapter.

```
def evaluation(self):
def evaluation():
total = self.nb_likes + self.nb_dislikes
total = self.nb_likes - self.nb_dislikes
total = self.nb_dislikes - self.nb_likes
total = self.nb_likes * self.nb_dislikes
for arg in self.arguments:
for arg in self.arguments_pour:
for arg in self.arguments_contre:
if arg.sorte == "sujet":
if arg.sorte == "pour":
if arg.sorte == "contre":
else:
total = arg.evaluation()
total = total + arg.evaluation()
total = total - arg.evaluation()
total = total * arg.evaluation()
return total
return 1 + total
```

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

Dans cette partie, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND`, `OR`) et `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT` et `DELETE`.

Pour manipuler toutes les informations sur plusieurs sujets de débats, la plateforme dispose d'un Système de Gestion de Bases de Données (SGBD).

8. Parmi les propositions suivantes, indiquer celles qui font partie des rôles d'un SGBD :

   a) sécuriser les accès à la base de données ;
   b) assurer l'alimentation électrique des serveurs ;
   c) assurer la persistance des données même en cas de panne matérielle ;
   d) gérer les accès en parallèle de plusieurs utilisateurs ;
   e) assurer des connexions en HTTPS au serveur.

Pour gérer les données, le schéma relationnel utilisé est représenté sur la figure 2.

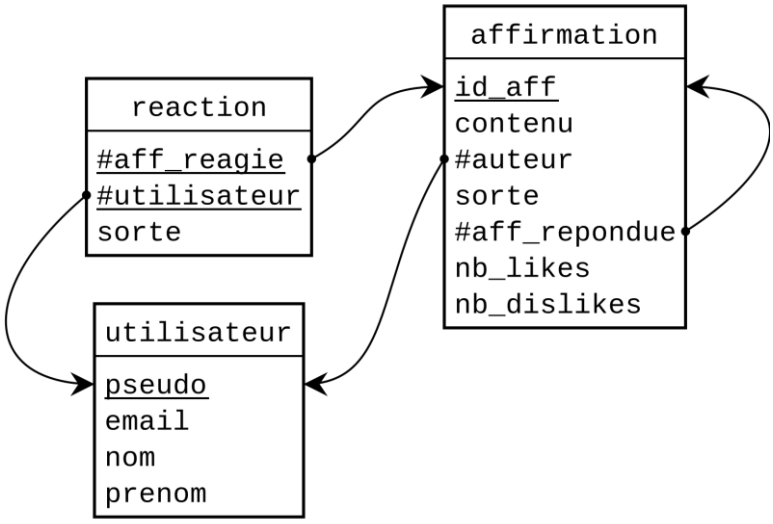

Figure 2. Schéma relationnel de la plateforme de débat

La clé primaire de chaque table correspond à l'ensemble de ses attributs soulignés. Chaque attribut précédé d'un "`#`" est une clé étrangère.

L'attribut `sorte` de la table `affirmation` peut valoir `'sujet'` quand c'est l'affirmation initiale d'un sujet de débat, ou bien `'pour'` ou `'contre'` quand c'est une réponse à une autre affirmation (référencée par l'attribut `aff_repondue`). L'attribut `contenu` correspond à la phrase elle-même de l'affirmation.

L'attribut `sorte` de la table `reaction` peut valoir `'like'` ou `'dislike'`, et s'applique à l'affirmation référencée par l'attribut `aff_reagie`.

Voici des exemples d'extraits des tables `affirmation` et `reaction`. On a omis, pour ces exemples, les attributs `auteur` et `contenu` de la table `affirmation` :

affirmation				
id_aff	sorte	aff_repondue	nb_likes	nb_dislikes
0	'sujet'	NULL	42	17
1	'pour'	0	20	20
2	'pour'	0	40	10
3	'contre'	0	13	12
4	'contre'	0	18	6
7	'contre'	1	15	16

reaction		
aff_reagie	utilisateur	sorte
0	'alice'	'like'
1	'alice'	'dislike'
0	'bob'	'dislike'
1	'bob'	'dislike'

9. Indiquer en justifiant quel autre attribut de la table `utilisateur` aurait aussi pu servir de clé primaire.

On rappelle que la fonction d'agrégation `COUNT` permet de compter les éléments du résultat d'une requête en plaçant `COUNT(*)` dans la clause `SELECT`.

10. Écrire une requête SQL permettant d'obtenir le nombre d'affirmations qui ont 50 *likes* ou plus.

11. Écrire une requête qui permet d'obtenir les contenus **des sujets** et leur nombre de *likes*, créés par les utilisateurs dont les prénom et nom sont respectivement `'Pierre'` et `'Durand'`.

Le mot-clé `AS` permet de donner un autre nom à une table au sein d'une requête. Il est notamment nécessaire lorsqu'on a besoin de faire la jointure d'une table avec elle-même. On considère la requête SQL suivante :

```
SELECT aff.contenu, rep.contenu
FROM affirmation AS aff JOIN affirmation AS rep
  ON rep.affirmation_repondue = aff.id_affirmation
WHERE rep.nb_likes >= 2 * aff.nb_likes
  AND rep.sorte = 'contre'
```

12. Donner une interprétation en français de ce que permet d'obtenir la requête ci-dessus.

L'utilisateur au pseudo `'i<3descartes'` approuve par un *like* l'affirmation `'Cogito ergo sum'` qui a l'`id_aff` 108.

13. Recopier et compléter les deux requêtes SQL suivantes, permettant de mettre à jour les informations de la base de données suite à cette action.

```
... reaction
VALUES (..., ..., ...);;

... affirmation
SET nb_likes = ...
WHERE ... = ...;;
```

L'utilisateur au pseudo `'i<3rgpd'` souhaite faire valoir son *droit à l'effacement*, et supprimer son compte utilisateur et toutes les données qui y sont liées. Dans cette situation, la plateforme de débat, qui souhaite maintenir la qualité des échanges, suit la procédure suivante :

- elle anonymise toutes les affirmations de l'utilisateur (en remplaçant l'attribut `auteur` par la valeur `NULL`), considérant que les affirmations elle-mêmes ne sont pas des données personnelles ;
- elle supprime toutes les réactions de l'utilisateur (mais sans toucher au nombres de likes et de dislikes des affirmations correspondantes) ;
- et enfin elle supprime l'utilisateur lui-même.

14. Expliquer pourquoi il est nécessaire de supprimer les réactions de l'utilisateur avant de supprimer l'utilisateur.

15. Écrire les trois requêtes qui permettent la suppression des données du compte de `'i<3rgpd'` en suivant la procédure décrite ci-dessus.