BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU MARDI 17 JUIN 2025

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 17 pages numérotées de 1/17 à 17/17.

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 les requêtes SQL.

Dans cet exercice, 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`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT`, `DELETE` ;
- affiner les recherches à l'aide de `DISTINCT`, `ORDER BY`.

Dans un schéma relationnel, on utilisera les conventions suivantes :

- la clé primaire d'une relation est définie par son attribut souligné ;
- les attributs précédés de # sont les clés étrangères.

Le guitariste Slash possède une incroyable collection de guitares. Maud est une grande fan de Slash. Elle décide de faire un inventaire de la collection de guitares sous la forme d'une base de données relationnelle.

Partie A

Dans cette partie, Maud utilise la relation suivante :

`inventaire (id, marque, modele, annee, num_ser, prix)`

`num_ser` représente le numéro de série d'une guitare. Il est unique pour chaque guitare d'une même marque. Le `prix` est en euro.

Voici un extrait de la table `inventaire`.

inventaire					
id	marque	modele	annee	num_ser	prix
1	Gibson	Les Paul Goldtop	1956	@70562	100000
2	Gibson	Les Paul Goldtop	1988	81738349	20000
3	Gibson	Les Paul Standard	1959	@90663	250000
4	Gibson	Les Paul Standard	1987	81757532	25000
5	Fender	Telecaster	1952	000230	150000
6	Fender	Telecaster	1965	81345673	10000
7	Fender	Stratocaster	1956	001359	200000
8	Fender	Stratocaster	1965	81757532	15000

1. Expliquer pourquoi l'attribut `num_ser` ne peut pas être une clé primaire de la relation `inventaire`.

2. Donner, sous forme de tableau, le résultat de la requête suivante appliquée à l'extrait de table précédent.

```
SELECT marque, modele
FROM inventaire
WHERE annee = 1956
```

3. Écrire une requête SQL permettant d'obtenir toutes les années du modèle Les Paul Standard dans la collection.

4. Écrire une requête SQL permettant d'obtenir tous les modèles de guitares de la marque Gibson par ordre croissant de l'année dans la collection.

5. Maud a fait une erreur de saisie pour la guitare d'identifiant `id=1`. L'année est en réalité 1957. Écrire une requête SQL permettant de corriger cette erreur de saisie.

Partie B

Maud change de représentation pour l'inventaire de la collection. Dans cette partie, Maud utilise maintenant les trois relations suivantes :

`marque (id, nom)`

`modele (id, nom, #id_marque)`

`guitare (id, #id_modele, annee, num_ser, prix)`

Dans la relation `modele`, `#id_marque` est une clé étrangère reliée à la clé primaire `id` de la relation `marque`. Dans la relation `guitare`, `#id_modele` est une clé étrangère reliée à la clé primaire `id` de la relation `modele`.

Voici des extraits des trois tables `marque`, `modele`, `guitare`.

marque	
id	nom
1	Gibson
2	Fender

modele		
id	nom	id_marque
1	Les Paul Goldtop	1
2	Les Paul Standard	1
3	Telecaster	2
4	Stratocaster	2

guitare				
id	id_modele	annee	num_ser	prix
1	1	1956	@70562	100000
2	1	1988	81738349	20000
3	2	1959	@90663	250000
4	2	1987	81757532	25000
5	3	1952	000230	150000
6	3	1965	81345673	10000
7	4	1956	001359	200000
8	4	1965	81757532	15000

6. Expliquer brièvement, en justifiant, dans quel ordre les trois tables doivent être créées.

7. Écrire une requête SQL permettant d'obtenir le numéro de série et l'année de toutes les guitares Les Paul Standard de la collection.

Maud vient d'apprendre que Slash a fait cadeau d'une de ses guitares à un ami. Elle doit donc la retirer de sa base de données.

8. Écrire une requête SQL permettant de retirer de la collection la guitare d'identifiant `id=3`.

Slash a aussi acheté une guitare d'une marque qu'il n'avait pas encore dans sa collection. Maud décide de la rajouter.

9. Écrire l'ensemble des requêtes SQL permettant d'ajouter la guitare suivante :

   – marque : BC Rich
   – modèle : Mockingbird
   – année : 1992
   – numéro de série : 92R
   – prix : 5000.

   On supposera que l'on peut attribuer la valeur 3 pour l'attribut `id` dans la table `marque` pour la marque BC Rich, que l'on peut attribuer la valeur 5 pour l'attribut `id` dans la table `modele` pour le modèle Mockingbird et que l'on peut attribuer la valeur 9 pour l'attribut `id` dans la table `guitare` pour cette guitare.

Maud souhaite connaître la valeur totale des modèles Stratocaster de la collection. Son ami David lui conseille de regarder la fonction `SUM`. La syntaxe pour utiliser cette fonction SQL peut être similaire à celle-ci :

```
SELECT SUM(nom_colonne)
FROM tab
```

Cette requête SQL permet de calculer la somme des valeurs contenues dans la colonne `nom_colonne` de la table `tab`.

10. Écrire une requête SQL permettant de calculer la valeur totale des modèles Stratocaster de la collection de Slash.

Exercice 2 (6 points)

Cet exercice porte sur l'algorithmique, les structures de données, et la gestion de processus.

On cherche à créer une application de type *liste de tâches à faire* pour aider Alice à planifier sa journée. Pour cela Alice saisit les informations concernant chacune des tâches qu'elle doit effectuer : elle indique un nom pour la tâche, ainsi que la durée qu'elle estime nécessaire afin de la réaliser. On représente une tâche saisie par Alice à l'aide d'un objet de type `Tache`, muni de quatre attributs :

- le `numero` de la tâche, saisi par Alice ;
- le `nom` de la tâche, saisi par Alice ;
- la `duree` (un entier exprimé en minute) nécessaire à la réalisation de la tâche saisie par Alice ;
- la `duree_restante` (un entier exprimé en minute) avant la fin de la tâche. Cet attribut sera initialisé avec la durée totale nécessaire à la réalisation de la tâche.

Avancer de n minutes (n entier positif) dans une tâche consiste à diminuer de n la durée restante de cette tâche. Une tâche est terminée si la durée restante est négative ou nulle.

Lors de la phase de planification de ses tâches (aucune d'entre elles n'est commencée), Alice liste les tâches suivantes qui doivent être effectuées :

Numéro	Nom	Durée	Durée restante
1	Répondre aux e-mails	45	45
2	Ranger ma chambre	60	60
3	Réviser la NSI	90	90
4	S'entraîner aux échecs	30	30
5	Apprendre le vocabulaire de chinois	30	30
6	Lire Fondation	60	60
7	Écrire ma lettre au Père Noël	20	20

On dispose de la classe `Tache` ci-dessous pour représenter les tâches :

```
1 class Tache:
2     def __init__(self, numero, nom, duree):
3         self.numero = numero
4         self.nom = nom
5         self.duree_initiale = duree
6         self.duree_restante = duree
7
8     def __repr__(self):
9         return '<t'+str(self.numero)+'>'
```

1. Donner le code Python qui permet d'instancier deux variables `tache1` et `tache2` représentant les tâches :

   – tâche numéro 1 : Répondre aux e-mails. Durée estimée : 45 minutes.

   – tâche numéro 2 : Ranger ma chambre. Durée estimée : 60 minutes.

On supposera dans la suite que les variables `tache1`, `tache2`, ..., `tache7` représentent les tâches établies par Alice lors de la phase de planification.

La méthode `__repr__` renvoie une représentation de l'instance sous forme d'une chaîne de caractères. La fonction `print` utilise cette méthode. Ainsi on a :

```
>>> print(tache1)
<t1>
```

2. Recopier et compléter le code de la méthode `avancer` de la classe `Tache` qui permet d'avancer la tâche `self` de n minutes.

```
1     def avancer(self, n):
2         ...
```

3. Recopier et compléter le code de la méthode `est_terminee` de la classe `Tache` qui renvoie `True` si la tâche est terminée, ou `False` sinon.

```
1     def est_terminee(self):
2         ...
```

Afin d'aider Alice à planifier sa journée, on lui propose d'associer à chacune des tâches une priorité. La priorité d'une tâche est représentée par un entier de la manière suivante : 1 est la priorité minimale et, plus le nombre est grand, plus la tâche associée est prioritaire.

Pour stocker toutes les tâches à effectuer, on utilise une file, dans laquelle les éléments sont des tuples `(tache, priorite)`. Les éléments stockés dans la file doivent respecter les deux conditions ci-après.

- Condition 1 : les éléments sont rangés par ordre décroissant de priorité. L'élément de priorité maximale se trouve au début de la file, l'élément le moins prioritaire se trouve à la fin de la file.

- Condition 2 : parmi les éléments de même priorité, les éléments sont rangés dans l'ordre dans lequel ils ont été insérés dans la file. Ainsi, le premier élément de priorité p inséré se trouve devant les éléments de même priorité p insérés plus tard.

Par exemple, si la file de tâches `f` est la file :

```
[début] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin]
```

Cela signifie que :

- la tâche de priorité maximale est la tâche numéro 3 ;
- les deux tâches à exécuter en priorité après la tâche numéro 3 sont les tâches numéro 1 et numéro 2. La tâche numéro 1 a été ajoutée à la file des tâches à traiter avant la tâche numéro 2 ;
- il n'y a pas de tâche de priorité 2 ;
- les tâches les moins prioritaires de la file sont les tâches numéro 4 et numéro 5. La tâche numéro 4 a été ajoutée avant la tâche numéro 5.

4. Représenter l'état de la file `f` lorsqu'on lui ajoute successivement la tâche numéro 6 avec la priorité 2, puis la tâche numéro 7 avec la priorité 4 en respectant les conditions 1 et 2 décrites ci-dessus.

On suppose déjà définies les méthodes suivantes pour la classe `File` :

- `File()` : crée et renvoie un objet de type `File`, vide.
- `enfiler(self, e)` : ajoute l'élément e à la fin de la file `f`.
- `defiler(self)` : renvoie, en le supprimant de la file, le premier élément de la file si cela est possible.
- `examiner(self)` : renvoie, sans le supprimer de la file, le premier élément de la file si cela est possible.
- `est_vide(self)` : renvoie `True` si la file est vide, ou `False` sinon.

5. En repartant de la file `f` suivante :

   ```
   [début](<t3>, 4)(<t1>, 3)(<t2>, 3)(<t4>, 1)(<t5>, 1)[fin]
   ```

   donner la valeur de `f.defiler()[0]`, et représenter le contenu de la file `f` après l'exécution de cette instruction.

6. En repartant de la file `f` suivante :

   ```
   [début](<t3>, 4)(<t1>, 3)(<t2>, 3)(<t4>, 1)(<t5>, 1)[fin]
   ```

   donner la valeur de `f.examiner()[1]`, et représenter le contenu de la file `f` après l'exécution de cette instruction.

On souhaite écrire une fonction `ajouter_file_prio` qui prend en paramètres :

- une file `f` dont les éléments sont des tuples `(tache, priorite)` respectant les deux conditions de l'énoncé ;
- une tâche `t` ;
- la priorité `p` de la tâche `t` ;

et qui ajoute le tuple `(t, p)` à la bonne position dans la file `f`.

On utilise une file auxiliaire `f_aux` que l'on remplit en défilant les éléments en début de file `f` tant que la priorité du premier élément de la file est supérieure ou égale à `p`. Puis on enfile l'élément `(t, p)` dans la file auxiliaire. On défile ensuite tous les éléments restants de `f` dans `f_aux` et enfin on enfile dans `f` tous les éléments de `f_aux`.

7. Recopier et compléter le code de la fonction `ajouter_file_prio`.

```
1 def ajouter_file_prio(f, t, p):
2     f_aux = File()
3     while ...:
4         ...
5     ...enfiler(...)
6     while not ...:
7         ...
8     while not ...:
9         ...
```

8. Donner le coût d'exécution temporel dans le pire des cas de la fonction `ajouter_file_prio`, en fonction du nombre `m` d'éléments de la file `f`.

Une fois qu'Alice a entré les tâches qu'elle doit effectuer, leur durée estimée, ainsi que la priorité à laquelle elle doit les effectuer, l'application lui propose un planning en utilisant la technique dite Pomodoro :

- la tâche à effectuer est la tâche qui se trouve en tête de file ;
- on défile cette tâche de la file des tâches à effectuer ;
- on avance cette tâche de 25 minutes ;
- si cette tâche n'est pas terminée, on rajoute cette tâche dans la file des tâches à effectuer, avec la même priorité qu'initialement (en utilisant la fonction `ajouter_file_prio`) ;
- si cette tâche se termine au cours des 25 minutes, alors Alice attend la fin des 25 minutes en se reposant ;
- on continue ces étapes tant que la file des tâches à effectuer n'est pas vide.

On rappelle les tâches à effectuer ci-dessous, classées par ordre de priorité. On considérera que les tâches sont ajoutées à la file de priorité dans l'ordre du tableau ci-dessous :

Numéro	Nom	Durée	Priorité
3	Réviser la NSI	90	4
7	Écrire ma lettre au Père Noël	20	4
1	Répondre aux e-mails	45	3
2	Ranger ma chambre	60	3
6	Lire Fondation	60	2
4	S'entraîner aux échecs	30	1
5	Apprendre le vocabulaire de chinois	30	1

9. Indiquer pour chaque bloc de 25 minutes la tâche qui avance, en suivant le modèle proposé, jusqu'à la fin de toutes les tâches.

   On fera particulièrement attention au cas où la tâche n'est pas terminée : celle-ci est rajoutée à la file des tâches à effectuer (dont elle avait été supprimée) avec la même priorité qu'initialement, en respectant les conditions 1 et 2 de l'énoncé.

10. Écrire le code d'une fonction `planning` qui prend en paramètre une file de priorité `f` dont les éléments sont des tuples `(tache, prio)`, et qui renvoie une liste de tâches, dans l'ordre dans lequel elles vont être effectuées par tranche de 25 minutes avec la méthode Pomodoro.

    Par exemple, si `tache1`, `tache2` et `tache3` sont les tâches numéro 1, numéro 2 et numéro 3, alors le programme suivant :

```
1 file = File()
2 for t, p in [(tache1, 3), (tache2, 3), (tache3, 4)]:
3     ajouter_file_prio(file, t, p)
4 print(planning(file))
```

    produit l'affichage :

```
[<t3>, <t3>, <t3>, <t3>, <t1>, <t2>, <t1>, <t2>, <t2>]
```

Exercice 3 (8 points)

Cet exercice porte sur l'architecture matérielle (réseau), les arbres binaires de recherche et la programmation Python.

L'entreprise CaféNet possède plusieurs cafés répartis dans différentes villes. Le réseau de la chaîne de cafés est représenté en Figure 1.

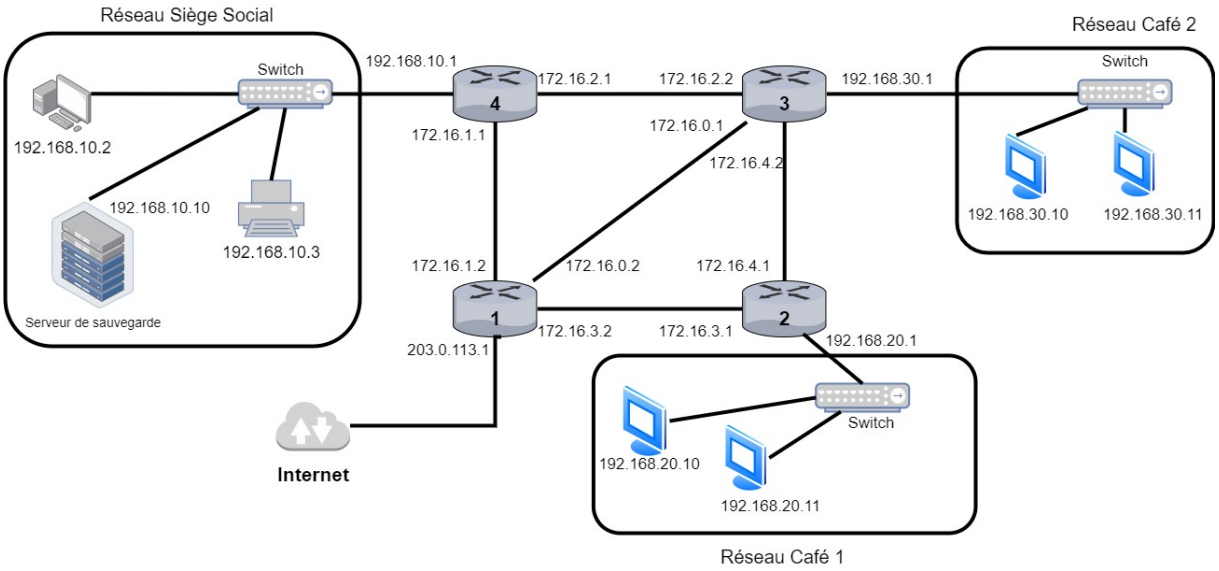


Figure 1. Schéma d'une partie du réseau

Sur le schéma sont représentés 4 routeurs, le réseau du siège social, le réseau du café 1, le réseau du café 2. Dans les réseaux du café 1 et du café 2, des bornes de commandes sont connectées à des switchs (ce sont des boitiers de connexion qui n'ont pas eux-mêmes d'adresse IP). Les 4 routeurs représentés sont composés d'au moins 3 interfaces réseau capable de relier des réseaux ensemble. Chaque interface possède donc une adresse IPV4 sur le réseau auquel elle est reliée.

Les masques des sous-réseaux sont tous 255.255.255.0. Avec ce masque, les trois premiers octets des adresses IP codent l'adresse réseau. Le dernier octet, c'est-à-dire les 8 derniers bits, code l'adresse des machines à l'intérieur de chaque sous-réseau.

Partie A

Le gérant veut faire installer une troisième borne de commande dans le café 1.

1. Indiquer les deux seules adresses IP valides pour cette nouvelle borne, parmi les quatre adresses IP proposées.

   (a) 192.168.20.2 (c) 192.168.20.261

   (b) 192.168.20.157 (d) 192.168.24.10

L’adresse de diffusion, appelée aussi adresse de broadcast, est la dernière adresse disponible à l’intérieur d’un réseau local.

2. Déterminer l’adresse de diffusion du réseau du café 1.

3. Déterminer combien de machines informatiques il est encore possible de connecter au réseau du café 1 après l’installation de la troisième borne de commande.

Le réseau local du café 1 n’a pas besoin de plus de 8 adresses IP différentes. Ce décompte d’adresses IP inclut les adresses IP réservées (à savoir l’adresse de diffusion et l’adresse du réseau). Il est rappelé que la longueur du masque de sous-réseau est actuellement de 24 bits (c’est-à-dire 3 octets).

4. Expliquer quelle est la longueur maximale du masque de sous-réseau que l’on pourrait choisir pour le réseau local du café 1.

Partie B

RIP (Routing Information Protocol) est un protocole de routage utilisé dans les réseaux IP. Il est conçu pour réduire le nombre de sauts entre deux réseaux. Un “saut” correspond au transfert des données d’un routeur à un autre. Le protocole RIP utilise le nombre de sauts comme critère principal pour évaluer le coût d’un chemin. Autrement dit, il considère que le chemin le plus optimal est celui qui traverse le moins de routeurs.

La table de routage du routeur 2 de la Figure 1 est représentée ci-dessous :

Routeur 2			
Réseau destination	Interface de sortie	Prochain routeur	Nombre de sauts
192.168.20.0	192.168.20.1	aucun	0
172.16.3.0	172.16.3.1	aucun	0
172.16.4.0	172.16.4.1	aucun	0
192.168.10.0	172.16.3.1	172.16.3.2	2
172.16.0.0	172.16.4.1	172.16.4.2	1
172.16.2.0	172.16.4.1	172.16.4.2	1
192.168.30.0	…	…	…
172.16.1.0	…	…	…

5. Recopier et compléter les deux dernières lignes de la table de routage du routeur 2.

La table de routage du routeur 2 contient un réseau de destination pour lequel deux routes différentes sont possibles. La ligne correspondante dans la table de routage aurait donc pu être remplie différemment tout en respectant le protocole RIP.

6. Identifier, dans la table de routage du routeur 2, le réseau de destination que l'on peut atteindre d'une autre façon et indiquer comment cette ligne de la table de routage pourrait être modifiée.

Une adresse IP qui n'est pas référencée dans la table de routage doit être routée par défaut vers Internet.

7. Recopier et compléter la ligne à ajouter à la table de routage du routeur 2.

Réseau destination	Interface de sortie	Prochain routeur
autre	…	…

Partie C

OSPF est également un protocole d'échanges de données entre les routeurs qui prend en compte le coût des routes. Le coût est lié au débit des liaisons entre les routeurs par la formule suivante :

$cout = \frac{10^9}{debit}$ avec le débit en $bit.s^{-1}$.

8. Recopier et compléter la dernière colonne du tableau ci-dessous :

Tableau des coûts		
Type de connexion	Débit en $bit.s^{-1}$	coût
Ethernet	$10\ Mbit.s^{-1} = 10^7\ bit.s^{-1}$	100
Fast Ethernet	$100\ Mbit.s^{-1} = 10^8\ bit.s^{-1}$	…
Fibre optique	$1\ Gbit.s^{-1} = 10^9\ bit.s^{-1}$	…

Le schéma ci-dessous met en évidence les types de connexion qui relient les routeurs.

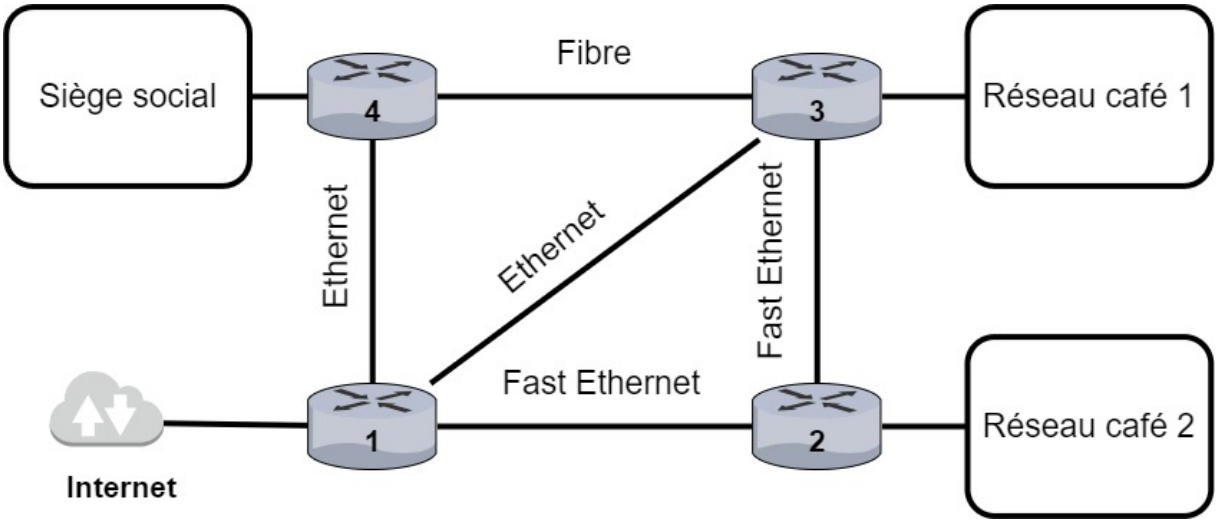


Figure 2. Schéma des types de connexion

9. Déterminer la route dont le coût est minimal pour aller du routeur 1 jusqu'au routeur 4 et calculer son coût au sens du protocole OSPF.

Partie D

Le but de cette partie est de classer les adresses IP des différents réseaux afin de faciliter leur recherche.

La fonction `ip_bin` prend en argument une chaîne de caractères décrivant une adresse IP en notation décimale, et renvoie une chaîne de caractères, de longueur 35 (32 bits et les 3 points), décrivant l'adresse IP en notation binaire.

Exemple :

```
>>> ip_bin('192.168.10.1')
'11000000.10101000.00001010.00000001'
```

10. Donner la chaîne de caractères renvoyée par `ip_bin('192.168.20.12')`.

La fonction `precede` prend en paramètres deux adresses IP en notation binaire, sous forme de chaînes de caractères identiques à celles renvoyées par la fonction `ip_bin`. La fonction `precede` renvoie un booléen qui vaut `True` si la première adresse IP en paramètre précède la seconde adresse IP.

Exemple :

```
>>> a = '11000000.10101000.00001010.00000001'
>>> b = '11000000.10101000.00001111.00000001'
>>> precede(a, b)
True
```

L'algorithme compare bit à bit les deux chaînes binaires, en lisant les chaînes de caractères dans le sens usuel (de gauche à droite). Dans l'exemple ci-dessus, tous les caractères sont identiques jusqu'au sixième caractère du troisième octet. Comme le bit de l'adresse a est inférieur à celui de l'adresse b, on en déduit que l'adresse IP a précède l'adresse IP b.

Si la première adresse IP ne précède pas la seconde, la fonction doit renvoyer `False`.

L'algorithme de comparaison est traduit dans le langage Python sous la forme suivante :

```
1 def precede(ip_1, ip_2):
2     for i in range(35):
3         if ip_1[i] < ip_2[i]:
4             return ...
5         elif ip_1[i] > ip_2[i]:
6             return ...
7     return ...
```

11. Expliquer dans quel cas la fonction `precede` exécutera la dernière instruction `return` de la ligne 7.

12. Recopier et compléter les lignes 4, 6 et 7 du code de la fonction `precede`.

Les tables de routage de chaque routeur sont implémentées sous la forme d'arbre binaire de recherche avec la classe `Abr`.

```
 1 class Abr:
 2     def __init__(self, adresse_ip,
 3                  interface, passerelle,
 4                  cout):
 5         self.adresse_ip = adresse_ip
 6         self.interface  = interface
 7         self.passerelle = passerelle
 8         self.cout       = cout
 9         if adresse_ip != '':
10             self.gauche = Abr('','','',0)
11             self.droite = Abr('','','',0)
12
13     def est_vide(self):
14         return ...
```

Dans cette représentation :

- `adresse_ip` désigne l'adresse IP de la destination ;
- `interface` désigne l'interface réseau ;
- `passerelle` désigne l'adresse IP du prochain routeur ;
- `cout` désigne le nombre de sauts pour atteindre la destination.

- par convention, l'arbre binaire vide est une instance de `Abr` pour laquelle `adresse_ip` est une chaîne de caractères vide ;
- un arbre binaire de recherche non vide possède nécessairement un sous-arbre gauche et un sous-arbre droit, éventuellement vides, qui sont tous les deux des arbres binaires de recherche. Ces sous-arbres sont désignés par `gauche` et `droite` dans la classe `Abr` ;
- si elle n'est pas vide, l'adresse IP du sous-arbre gauche précède l'adresse IP de l'instance parent ;
- si le sous-arbre droit n'est pas vide, alors l'adresse IP de l'instance parent précède l'adresse IP du sous-arbre droit.

13. Citer un attribut et citer une méthode de la classe `Abr`.

14. Recopier et compléter la ligne 14 du code de la classe `Abr`.

15. Justifier, en mobilisant des connaissances de cours, l'intérêt qu'il peut y avoir à représenter la table de routage par un arbre binaire de recherche.

La section de code qui définit `modifie` est incluse dans la classe `Abr`.

```
16      def modifie(self, adresse_ip,
17                  interface, passerelle,
18                  cout):
19          if self.est_vide():
20              self.adresse_ip = adresse_ip
21              self.interface  = interface
22              self.passerelle = passerelle
23              self.cout       = cout
24              self.gauche = Abr('','','',0)
25              self.droite = Abr('','','',0)
26          else:
27              self.adresse_ip = adresse_ip
28              self.interface  = interface
29              self.passerelle = passerelle
30              self.cout       = cout
```

Les lignes 20 à 23 sont exactement les mêmes que les lignes 27 à 30.

16. Réécrire le code de la fonction `modifie` en évitant cette répétition.

La classe `Abr` est complétée afin de permettre l'ajout de nouvelles lignes à la table de routage, tout en conservant les propriétés que doit posséder un arbre binaire de recherche.

```
32      def rechercher(self, adresse_ip):
33          if self.est_vide() or adresse_ip==self.adresse_ip:
34              return self
35          elif precede(...):
36              return self.gauche.rechercher(adresse_ip)
37          else:
38              return self.droite.rechercher(adresse_ip)
39
40      def inserer(self, adresse_ip,
41                  interface, passerelle,
42                  cout):
43          destination = self.rechercher(adresse_ip)
44          destination.modifie(adresse_ip,
45                              interface, passerelle,
46                              cout)
```

On rappelle que la fonction `precede` prend en arguments des adresses IP écrites sous forme binaire.

17. Recopier et compléter la ligne 35 du code de la fonction `rechercher`.

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU MERCREDI 18 JUIN 2025

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 15 pages numérotées de 1/15 à 15/15.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

Exercice 1 (6 points)

Cet exercice porte sur les arbres binaires et la programmation Python.

Le codage de Shannon-Fano est un système de codage utilisé pour la compression sans pertes de données. Il a été mis au point par Robert Fano d'après une idée de Claude Shannon.

Partie A

Dans cette partie, on va étudier l'utilisation des arbres de codage.

Un arbre de codage est un arbre binaire où chaque feuille contient un symbole du texte que l'on souhaite coder. Le code binaire d'un symbole s'obtient alors en concaténant les `0` et les `1` sur les branches qui mènent de la racine à la feuille contenant ce symbole. Par exemple, pour l'arbre de codage donné en Figure 1, le symbole c est codé par le mot binaire `1101`, tandis que d est codé par le mot binaire `11000`. Les codes binaires des symboles ne sont donc pas tous de la même taille. Pour décoder un mot binaire, il suffit de descendre dans l'arbre, depuis la racine, selon les `0` et les `1` qu'on lit jusqu'à trouver une feuille (et donc un symbole), puis de recommencer avec la suite du mot binaire pour décoder les symboles suivants.

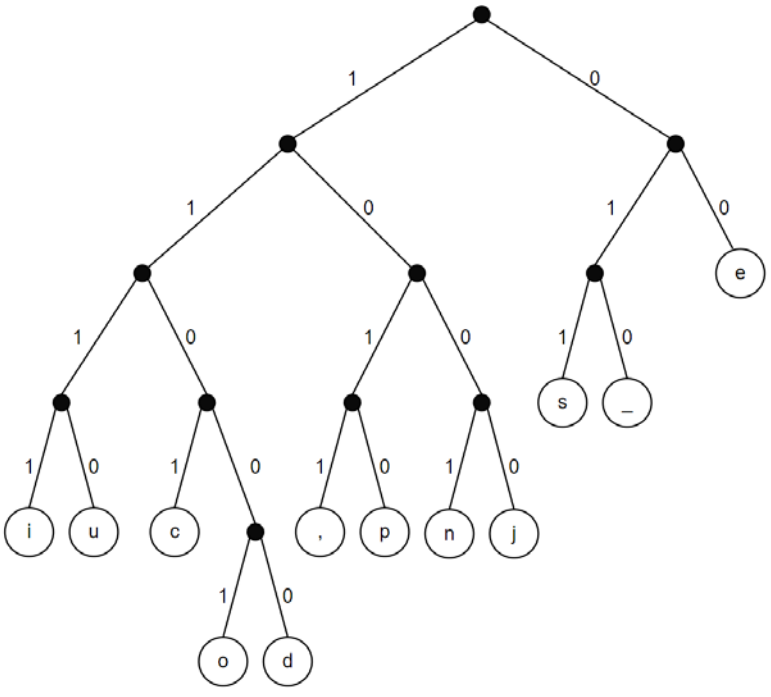


Figure 1. Exemple d'arbre de codage

1. Écrire le mot binaire qui sera utilisé pour encoder le caractère espace, représenté par le symbole _ dans l'arbre.

2. Déterminer le texte codé par le mot binaire `000111010111111001100 1`.

3. Citer le type de parcours de l'arbre qui permettrait d'obtenir les symboles classés par taille d'encodage croissante.

Partie B

Dans cette partie, on va utiliser le codage de Shannon-Fano pour encoder le texte :

```
je pense, donc je suis
```

Dans la méthode de Shannon-Fano, l'arbre de codage est calculé pour un texte donné par l'algorithme suivant.

- Étape 1 : classer les symboles du texte par nombre d'occurrences croissant ;
- Étape 2 : en gardant le classement obtenu, séparer les symboles en deux sous-groupes de sorte que les totaux des nombres d'occurrences soient les plus proches possibles dans les deux sous-groupes ;
- Étape 3 : placer tous les symboles du premier groupe dans le fils gauche (côté étiqueté par `1`), et ceux du second groupe dans le fils droit (côté étiqueté par `0`) ;
- Étape 4 : recommencer récursivement pour chacun des sous-groupes jusqu'à ce qu'ils n'aient plus qu'un seul symbole ; on a alors une feuille étiquetée par ce symbole.

Après avoir classé les symboles par nombre d'occurrences croissant (étape 1), on obtient le tableau suivant :

symbole	i	u	c	o	d	,	p	n	j	s	_	e
nombre d'occurrences	1	1	1	1	1	1	1	2	2	3	4	4

4. Justifier par le calcul que l'étape 2 mène à la situation illustrée par la Figure 2.

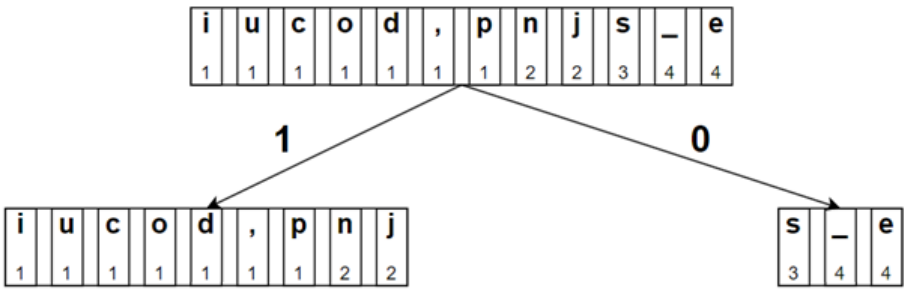


Figure 2. Le résultat de l'étape 2

En appliquant l'algorithme de Shannon-Fano, on peut obtenir l'arbre de la Figure 3.

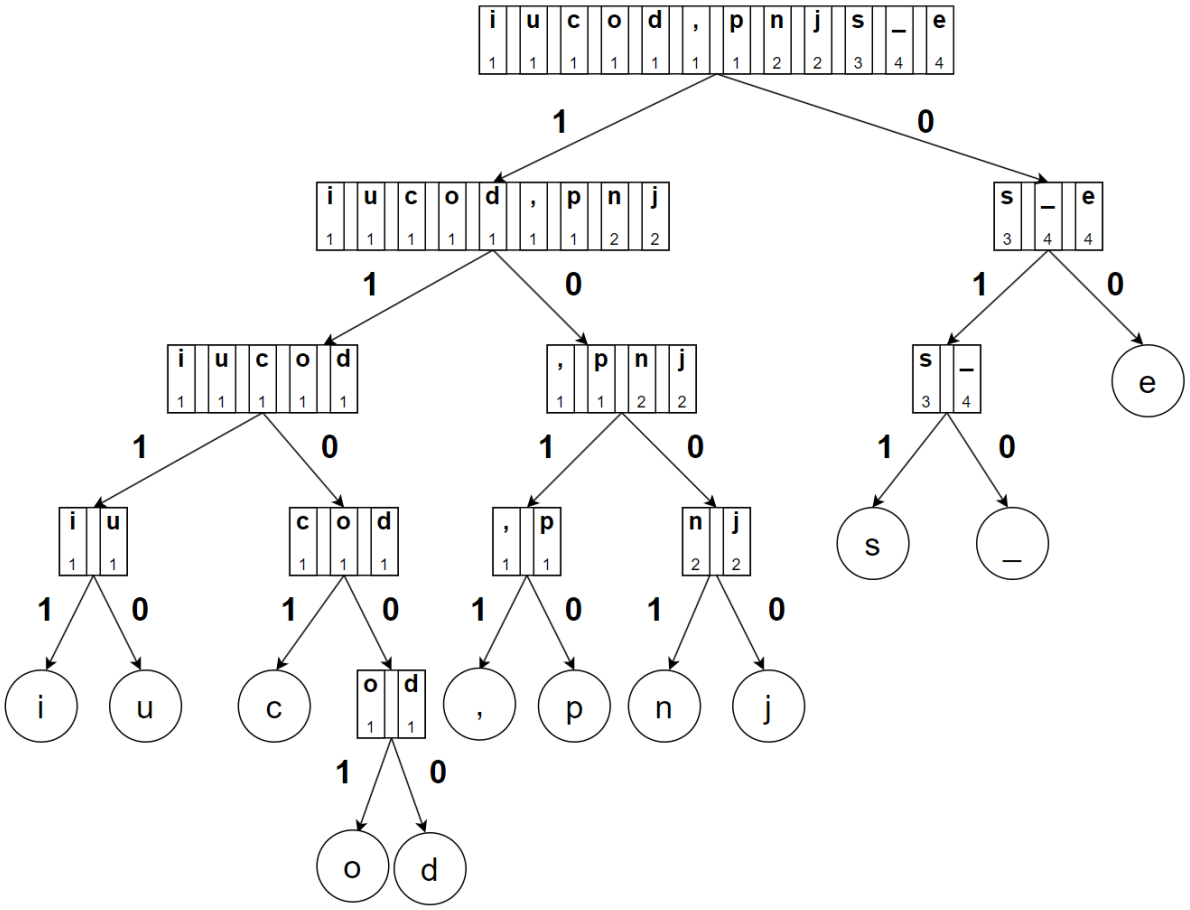


Figure 3. Arbre de codage obtenu par l'algorithme de Shannon-Fano

On rappelle qu'un arbre réduit à un seul nœud, c'est-à-dire réduit à une feuille, est de hauteur 0.

5. Donner la hauteur de l'arbre de la Figure 3 et préciser dans le contexte de l'exercice ce qu'elle représente.

On rappelle que dans le code ASCII, chaque symbole est codé sur un octet.

6. Justifier, en comparant le codage ASCII et le codage de Shannon-Fano, que ce second codage permet d'utiliser environ deux fois moins d'octets pour le texte :

   `je pense, donc je suis`

7. Dessiner, en vous inspirant de l'arbre de la Figure 1, un arbre de codage qui permettrait d'encoder le mot « `chiffrer` » en utilisant l'algorithme de Shannon-Fano.

Partie C

Dans cette partie, on souhaite écrire une fonction Python qui donnera le mot binaire obtenu pour coder un texte avec l'algorithme de Shannon-Fano. On commence par la fonction `creer_dico_occ` :

```
 1 def creer_dico_occ(texte):
 2     """renvoie un dictionnaire dont les clés sont les
 3     symboles de texte et les valeurs associées leur
 4     nombre d'occurences dans texte"""
 5     dico = {}
 6     for symbole in texte:
 7         if symbole in dico:
 8             dico[symbole] = ...
 9         else:
10             dico[symbole] = ...
11     return dico
```

8. Recopier et compléter les lignes 8 et 10 du code de la fonction `creer_dico_occ`.

On dispose d'une fonction `creer_tab_trie` qui prend en paramètre un dictionnaire construit avec la fonction `creer_dico_occ` et qui renvoie une liste de tuples classés dans l'ordre croissant d'occurrences des symboles.

Par exemple :

```
>>> texte = 'je pense, donc je suis'
>>> dico = creer_dico_occ(texte)
>>> creer_tab_trie(dico)
[('i', 1), ('u', 1), ('c', 1), ('o', 1), ('d', 1), (',', 1),
('p', 1), ('n', 2), ('j', 2), ('s', 3), (' ', 4), ('e', 4)]
```

9. Écrire une fonction `somme_occ` qui prend en paramètres un tableau `tab` de tuples `(symbole, nb_occ)` et qui renvoie la somme des nombres d'occurrences des symboles du tableau. Les tuples utilisés sont de même structure que l'élément renvoyé dans l'exemple précédent.

On suppose pour la suite qu'on dispose d'une fonction `separe` qui sépare un tableau trié en deux sous-tableaux de manière à ce que les sommes de ces derniers soient les plus proches possible :

```
 1 def separe(tab):
 2     moitie = somme_occ(tab) // 2
 3     somme = 0
 4     i = 0
 5     while moitie > somme:
 6         somme = somme + tab[i][1]
 7         i = i + 1
 8     tab1 = [tab[k] for k in range(0, i)]
 9     tab2 = [tab[k] for k in range(i, len(tab))]
10     return tab1, tab2
```

10. Recopier et compléter les lignes 9 et 11 du code de la fonction récursive `shannon` qui prend en paramètres un caractère `symbole` et un tableau trié `tab` et qui renvoie l'écriture binaire associée à `symbole` dans le tableau `tab`.

```
 1 def shannon(symbole, tab):
 2     """renvoie l'écriture binaire associée à symbole
 3     dans le tableau trié tab"""
 4     if len(tab) == 1:
 5         return ""
 6     else:
 7         t1, t2 = separe(tab)
 8         if symbole in [elt[0] for elt in t1]:
 9             return "1" + ...
10         else:
11             return "0" + ...
```

11. Décrire ce qui garantit la terminaison de la fonction récursive `shannon`.

12. Écrire une fonction `encode_shannon` qui prend en paramètre un texte de type `str` et renvoie un mot binaire de type `str` obtenu après encodage par l'algorithme de Shannon-Fano.
    On pourra utiliser les fonctions vues précédemment qui sont recensées ci-après.

```
creer_dico_occ(texte)
  renvoie un dictionnaire dont les clés sont les symboles
  du texte et les valeurs associées leur nombre
  d'occurrences

creer_tab_trie(dico)
  renvoie la liste crée à partir d'un dictionnaire
  de couples (symbole, nb_occ)

separe(tab)
  renvoie le tuple composé des 2 sous-tableaux triés
  avec des sommes d'occurences proches

shannon(symbole, tab)
  renvoie l'écriture binaire associée au symbole dans le
  tableau trié tab
```

Exercice 2 (6 points)

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

Une ludothèque municipale a décidé de moderniser sa gestion en créant une base de données informatique. Cette base de données permettra de suivre les jeux disponibles, les emprunts effectués par les adhérents, ainsi que les avis laissés sur les différents jeux. Pour commencer, quatre tables principales ont été identifiées : `jeu`, `adhérent`, `emprunt` et `avis`. Ces tables et leurs relations vont permettre de stocker toutes les informations essentielles au bon fonctionnement de la ludothèque. On va considérer que la ludothèque n'a **qu'un exemplaire** de chaque jeu (deux jeux de la ludothèque ne peuvent donc pas avoir le même nom).

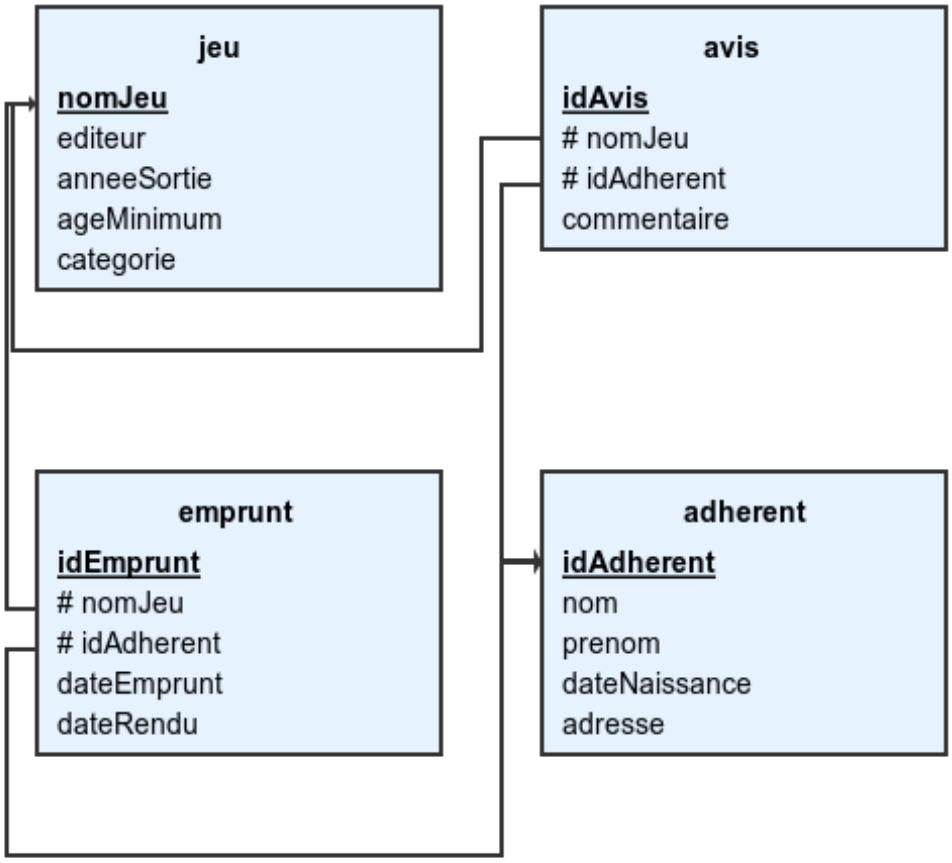


Figure 1. La base de données de la ludothèque

Dans la figure ci-dessus, les clés primaires de chacune des tables sont soulignées et les clés étrangères sont précédées du symbole #.

Dans cet exercice, 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` ;
- affiner les recherches à l'aide de `DISTINCT` et `ORDER BY` ;
- réaliser des agrégations à l'aide de `COUNT`.

Par exemple, l'instruction SQL :

```
SELECT COUNT(nomJeu) FROM jeu;
```

donne le nombre de jeux présents dans la table `jeu`.

1. Expliquer pourquoi on ne peut pas prendre l'attribut `nom` comme clé primaire pour la relation `adherent`.
2. Décrire ce que donne la requête SQL suivante :

```
SELECT nomJeu, editeur
FROM jeu
ORDER BY nomJeu;
```

Lorsque qu'un jeu est emprunté et n'a pas encore été rendu, la valeur de l'attribut `dateRendu` de la table `emprunt` est à `NULL`.

3. Écrire une requête permettant de connaitre le nom de tous les jeux qui sont en cours d'emprunt.
4. Écrire une requête SQL pour afficher le nom et le prénom de tous les adhérents qui ont emprunté le jeu "Catan".
5. Claire VOYANT, adhérente de longue date à cette ludothèque, a emprunté le jeu "Catan" et l'a rendu le 3 juin 2025. Lors de l'emprunt, la valeur de `id_emprunt` était 1538.

   Écrire une requête SQL qui a permis de mettre à jour la base de données afin qu'elle prenne en compte que ce jeu a été rendu. Toutes les dates de la base de données sont écrites sous le format `'AAAA-MM-JJ'`.
6. Écrire une requête SQL qui permet de trouver le nom et la catégorie de tous les jeux de la ludothèque sortis à partir de 2010 et dont l'âge minimum est strictement inférieur à 10 ans.

La ludothèque décide d'organiser des événements. Pour cela, elle ajoute une relation `evenement` à sa base de données. En outre, pour chaque événement, elle souhaite garder en mémoire une trace des adhérents qui y ont participé. À cette fin, elle complète sa base avec une relation `participation`.

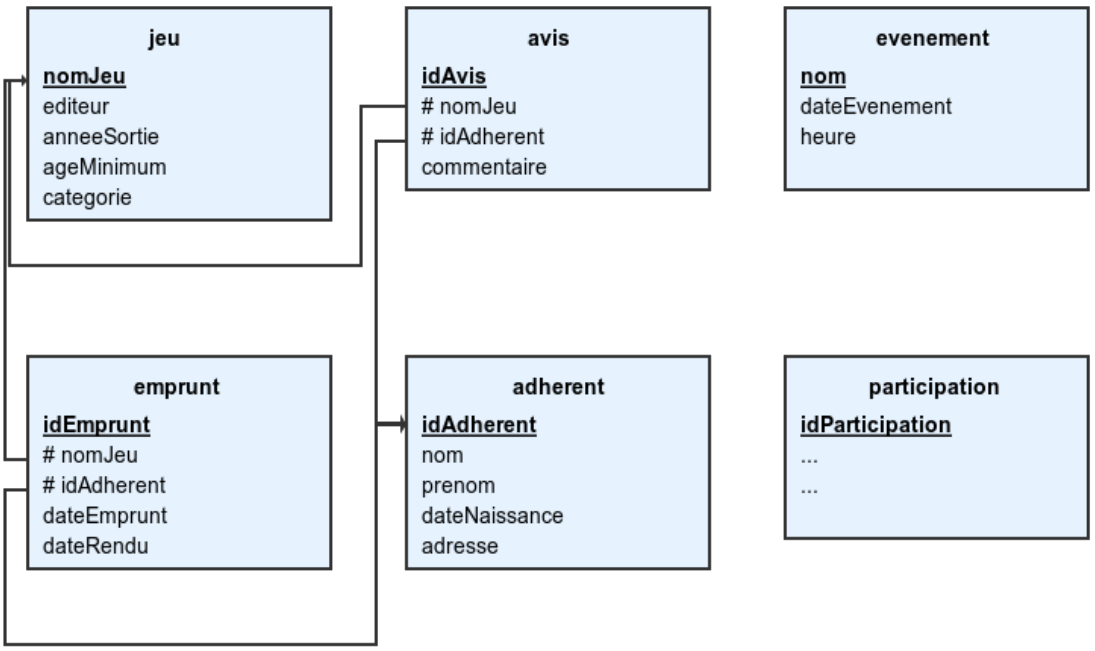


Figure 2. La base de données de la ludothèque actualisée

7. Proposer les clés étrangères de la table `participation` en précisant le nom des attributs auxquels elles font référence.

Le programme Python suivant permet de créer la liste de tous les jeux empruntés, sachant que, dans celle-ci, un jeu va apparaître autant de fois qu'il a été emprunté.

```
 1 import sqlite3
 2
 3 # Connexion à la base de données
 4 connection = sqlite3.connect("ludotheque.db")
 5 curseur = connection.cursor()
 6
 7 # Exécution de la requête
 8 curseur.execute("SELECT nomJeu FROM emprunt")
 9
10 # Récupération des résultats
11 jeux = curseur.fetchall()
12
13 liste = []
14 # Création de la liste des jeux empruntés
15 for jeu in jeux:
16     liste.append(jeu[0])
17
18 # Fermeture de la connexion
```

```
19 curseur.close()
20 connection.close()
```

8. Écrire un script Python permettant de créer le dictionnaire `dict_emprunts` qui, à chaque jeu emprunté, associe le nombre de fois où il a été emprunté.

On veut créer un podium des jeux les plus souvent empruntés. Comme il peut y avoir des égalités à la première, deuxième ou troisième place, il peut y avoir plus de trois jeux sélectionnés sur le podium.

Par exemple, si le dictionnaire des emprunts est :

```
1 dict_emprunts = {
2     "Terraforming Mars": 25,
3     "Codenames": 22,
4     "Agricola": 18,
5     "Puerto Rico": 18,
6     "Caylus": 18,
7     "Dominion": 22,
8     "Dixit": 12
9 }
```

il y aura sur le podium les jeux “Agricola”, “Puerto Rico” et “Caylus” puis les jeux “Dominion” et “Codenames” et enfin le jeu “Terraforming Mars”.

Pour modéliser ce podium en Python, on va utiliser une liste de trois listes.

Pour l’exemple précédent, cette liste sera :

`[["Agricola", "Puerto Rico", "Caylus"], ["Dominion", "Codenames"], ["Terraforming Mars"]]`.

9. Proposer un script Python permettant de générer ce podium.

Exercice 3 (8 points)

Cet exercice porte sur la programmation de base en Python, la sécurisation des communications et les réseaux.

Partie A - La méthode du *masque jetable*

Dans cette partie, on s’intéresse à une méthode de chiffrement dite du *masque jetable*. Voici ce que l’on peut lire sur le site Wikipédia :

Le chiffrement par la méthode du masque jetable consiste à combiner le message en clair avec une clé présentant les caractéristiques très particulières suivantes :

- *la clé doit être une suite de caractères au moins aussi longue que le message à chiffrer ;*

- *les caractères composant la clé doivent être choisis de façon totalement aléatoire ;*
- *chaque clé, ou « masque », ne doit être utilisée qu'une seule fois (d'où le nom de masque jetable).*

*Illustrons cette méthode par un exemple : on souhaite chiffrer le message **HELLO** avec la clé aléatoire, ou « masque », **WMCKL**.*

Pour cela, on attribue un nombre à chaque lettre, par exemple le rang dans l'alphabet, de 0 à 25.

Tableau de correspondance													
Lettre	A	B	C	D	E	F	G	H	I	J	K	L	M
Rang	0	1	2	3	4	5	6	7	8	9	10	11	12

Tableau de correspondance													
Lettre	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Rang	13	14	15	16	17	18	19	20	21	22	23	24	25

Ensuite, on additionne la valeur du rang de chaque lettre du message avec la valeur du rang correspondante dans le masque.

Enfin, si le résultat est supérieur à 25 on soustrait 26 (calcul dit « modulo 26 »).

*Ainsi, le chiffrement du message **HELLO** avec la clé **WMCKL** donne le message chiffré **DQNVZ** comme le montre l'illustration suivante.*

```
   7 (H)   4 (E)  11 (L)  11 (L)  14 (O) message
+ 22 (W)  12 (M)   2 (C)  10 (K)  11 (L) masque
= 29      16      13      21      25     masque + message
=  3 (D)  16 (Q)  13 (N)  21 (V)  25 (Z) masque + message  modulo 26
```

Figure 1. Exemple de chiffrement par la méthode du masque jetable

Source : d'après l'article Masque jetable de Wikipédia en français (https://fr.wikipedia.org/wiki/Masque_jetable)

Dans cet exercice, on ne travaillera que sur des chaînes de caractères écrites en majuscules non accentuées (les 26 caractères allant de 'A' à 'Z').

1. Chiffrer, par la méthode du *masque jetable*, le message **LIBRE** à l'aide de la clé **EYQMT**.

En Python, on crée une fois pour toute la variable `alphabet` qui sera accessible et utilisable dans toutes les fonctions. Celle-ci contient la liste des 26 lettres de l'alphabet rangées dans l'ordre alphabétique :

```
alphabet = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
'W', 'X', 'Y', 'Z']
```

2. Écrire une fonction Python `indice` qui prend pour paramètre une liste `L` et renvoie l'indice de `element` dans la liste `L`.

   On supposera que chaque élément de la liste `L` n'y apparaît qu'une seule fois et que `element` est bien présent dans la liste `L`.

   Par exemple, l'appel `indice(alphabet, 'K')` renvoie l'entier `10`.

3. Écrire une fonction Python `lettres_vers_indices` qui prend pour paramètre une chaîne de caractères et renvoie, dans l'ordre, la liste des indices de ces caractères dans l'alphabet.

   Par exemple, l'appel `lettres_vers_indices('HELLO')` renvoie la liste d'entiers `[7, 4, 11, 11, 14]`.

On dispose également d'une fonction `indices_vers_lettres`, qu'on ne demande pas d'écrire, permettant de convertir une liste d'entiers, compris entre 0 et 25, en une chaîne de caractères.

Par exemple, l'appel `indices_vers_lettres([3, 16, 13, 21, 25])` renvoie la chaîne de caractères `'DQNVZ'`.

Ci-après, on donne une fonction Python `chiffrement` incomplète, qui, à partir d'un message `msg` et d'une clé `cle` entrés en paramètres, renvoie la chaîne de caractères représentant le message chiffré par la méthode du *masque jetable*.

```
 1 def chiffrement(msg, cle):
 2     assert len(cle) >= len(msg), 'impossible'
 3     indices_msg = lettres_vers_indices(msg)
 4     indices_cle = lettres_vers_indices(cle)
 5     n = len(msg)
 6     indices_msg_chiffre = []
 7     for k in range(n):
 8         ind = ...
 9         if ind >= 26:
10             ind = ...
11         indices_msg_chiffre.append(ind)
12     msg_chiffre = indices_vers_lettres(...)
13     return msg_chiffre
```

4. Recopier et compléter les lignes 7 à 13 de la fonction `chiffrement`.

5. Indiquer, en justifiant, ce que l'on observe lors de l'appel `chiffrement('RESEAU', 'GFTZ')`.

On s'intéresse maintenant au déchiffrement d'un message chiffré par la méthode du *masque jetable*.

Par exemple, le déchiffrement du message **DQNVZ** avec la clé **WMCKL** donne le message **HELLO**.

6. Déchiffrer le message **GMEDH** avec la clé **FVEIT**.
7. Expliquer comment procéder pour déchiffrer un message lorsqu'on connaît la clé.

On souhaite maintenant écrire, en Python, une fonction `dechiffrement` qui permet de déchiffrer un message chiffré par la méthode du *masque jetable*.

Pour cela, on s'inspire de la fonction `chiffrement` dans laquelle les paramètres ainsi que les lignes 2 à 5 sont inchangées. On décide cependant de remplacer, ligne 6, le nom de la variable `indices_msg_chiffre` par le nom plus explicite `indices_msg_dechiffre`.

8. Adapter les lignes 6 à 13 de la fonction `chiffrement` pour obtenir la nouvelle fonction `dechiffrement`.

Partie B - Sécurisation des communications

9. Expliquer la différence entre un algorithme de chiffrement symétrique et un algorithme de chiffrement asymétrique.

Alice souhaite envoyer un message à Bob par l'intermédiaire d'un réseau informatique en utilisant un algorithme de chiffrement asymétrique.

Pour cela, Bob envoie à Alice sa clé publique. Alice chiffre ensuite le message à l'aide de la clé publique de Bob qu'elle vient de recevoir, puis elle envoie ce message chiffré à Bob.

10. Indiquer comment Bob peut déchiffrer le message que lui envoie Alice.
11. Expliquer comment une tierce personne pourrait se faire passer pour Alice sans que Bob ne s'en aperçoive.
12. Expliquer brièvement le fonctionnement du protocole HTTPS.
13. Expliquer pourquoi, pour sécuriser intégralement les communications sur Internet, on utilise le protocole HTTPS plutôt qu'un chiffrement asymétrique.

Partie C - Réseaux

Bob et Marc travaillent pour une petite compagnie d'assurances.

Leurs postes de travail font partie d'un même réseau local géré par l'administratrice système qui dispose du bloc d'adresses IPv4 192.168.110.0/24.

La notation /24 situé à la suite de l'adresse 192.168.110.0 signifie que le masque de sous-réseau du réseau de cette entreprise est 255.255.255.0 : les trois premiers octets d'une adresse IP sur ce réseau permettent donc d'identifier la partie réseau de l'adresse, alors que le dernier octet permet d'identifier la partie hôte et est propre à chaque machine sur le réseau. Ce sous-réseau permet donc d'attribuer 256 adresses IPv4 différentes.

L'administratrice choisit alors d'attribuer, en représentation décimale, l'identifiant 115 pour la partie hôte du poste de travail de Bob et l'identifiant 153 pour celui de Marc.

Depuis son poste de travail, Marc souhaite tester la communication avec celui de Bob. Pour cela, il exécute la commande `ping 192.168.100.115` et obtient l'affichage suivant :

```
--- 192.168.100.115 ping statistics ---

4 packets transmitted, 0 received, 100% packet loss, time
3060ms
```

14. Expliquer l'affichage obtenu et corriger l'erreur de Marc.

Afin d'améliorer les performances et la sécurité du réseau de l'entreprise, l'administratrice système décide de séparer le réseau local en plusieurs sous-réseaux et de les relier entre eux par des routeurs. Pour cela, elle modifie le masque de sous-réseau qui devient 11111111.11111111.11111111.11100000, donné ici en représentation binaire.

15. Donner la représentation décimale de ce masque de sous-réseau.

Pour obtenir l'adresse IPv4 du sous-réseau auquel appartient une machine, il suffit d'appliquer l'opérateur binaire **ET**, bit à bit, entre le masque de sous-réseau et l'adresse IPv4 de la machine.

Par exemple, prenons le dernier octet de l'adresse IPv4 de Bob dont la représentation binaire est 01110011 : en appliquant bit à bit l'opérateur binaire **ET** entre cet octet et l'octet correspondant dans le masque, on obtient le dernier octet de l'adresse du sous-réseau, soit 01100000.

```
   1 1 1 0 0 0 0 0 (224)
ET 0 1 1 1 0 0 1 1 (115)
-----------------
   0 1 1 0 0 0 0 0 (96)
```

Le poste de travail de Bob est donc sur le sous-réseau d'adresse 192.168.110.96.

16. Indiquer le nombre total d’adresses IPv4 pouvant être attribuées sur le sous-réseau d’adresse 192.168.110.96 sur lequel se trouve Bob.

L’administratrice système attribue maintenant l’adresse IPv4 192.168.110.134 au poste de travail de Zoé, nouvelle employée de la compagnie d’assurances.

17. Donner la représentation binaire du nombre 134.

Depuis son poste de travail, Zoé exécute les deux commandes suivantes :

- **commande n°1** : `ping 192.168.110.115` ;
- **commande n°2** : `ping 192.168.110.153`.

18. Indiquer, en justifiant, laquelle de ces deux commandes a produit l’affichage :

```
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
```

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU MERCREDI 19 JUIN 2024

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 15 pages numérotées de 1 / 15 à 15 / 15

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

EXERCICE 1 (6 points)

Cet exercice porte sur la programmation objet en Python et les graphes.

Nous avons représenté sous la forme d'un graphe les liens entre cinq différents sites Web :

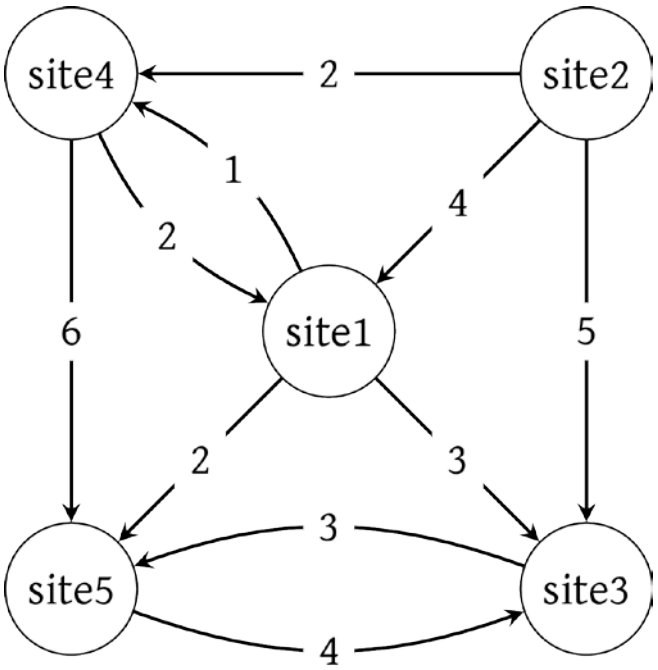


Figure 1. Graphe avec 5 sites

La valeur de chaque arête représente le nombre de citations (de liens hypertextes) d'un site vers un autre. Ainsi, le site **site4** contient 6 liens hypertextes qui renvoient vers le site **site5**.

Les sites sont représentés par des objets de la classe `Site` dont le code est partiellement donné ci-dessous. La complétion de la méthode `calculPopularite` fera l'objet d'une question ultérieure

```
1  class Site:
2
3      def __init__(self, nom):
4          self.nom = nom
5          self.predecesseurs = []
6          self.successeurs = []
7          self.popularite = 0
8          self.couleur = 'blanche'
9
10     def calculPopularite(self):
11         ...
```

Le graphe précédent peut alors être représenté ainsi :

```
1   # Description du graphe
2   s1, s2, s3, s4, s5 = Site('site1'), Site('site2'),
Site('site3'), Site('site4'), Site('site5')
3   s1.successeurs   = [(s3,3), (s4,1), (s5,3)]
4   s2.successeurs   = [(s1,4), (s3,5), (s4,2)]
5   s3.successeurs   = [(s5, 3)]
6   s4.successeurs   = [(s1,2), (s5,6)]
7   s5.successeurs   = [(s3,4)]
8   s1.predecesseurs = [(s2,4), (s4,2)]
9   s2.predecesseurs = []
10  s3.predecesseurs = [(s1,3), (s2,5), (s5,4)]
11  s4.predecesseurs = ...
12  s5.predecesseurs = ...
```

1. Expliquer la ligne 9 de ce code.

2. Les lignes 11 et 12 de cette description du graphe ne sont pas complètes. Recopier et compléter le code des lignes 11 et 12.

3. Donner et expliquer la valeur de l'expression suivante :

```
s2.successeurs[1][1]
```

Pour mesurer la pertinence d'un site, on commence par lui attribuer un nombre appelé valeur de popularité qui correspond au nombre de fois qu'il est cité dans les autres sites, c'est-à-dire le nombre de liens hypertextes qui renvoient sur lui. Par exemple, la valeur de popularité du site **site4** est 3.

4. Donner, selon cette définition, la valeur de popularité du site **site1**.

5. Écrire sur votre copie le code de la méthode `calculPopularite` de la classe `Site` qui affecte à l'attribut `popularite` la valeur de popularité correspondante et renvoie cet attribut.

Afin de calculer cette valeur de popularité pour chacun des sites, nous allons faire un parcours dans le graphe de façon à exécuter la méthode `calculPopularite` pour chacun des objets.

Voici le code de la fonction qui permet le parcours du graphe :

```
1  def parcoursGraphe(sommetDepart):
2      parcours = []
3      sommetDepart.couleur = 'noire'
4      listeS = []
5      listeS.append(sommetDepart)
6      while len(listeS) != 0:
7          site = listeS.pop(0)
8          site.calculPopularite()
9          parcours.append(site)
10         for successeur in site.successeurs:
11             if successeur[0].couleur == 'blanche':
12                 successeur[0].couleur = 'noire'
13                 listeS.append( successeur[0] )
14     return parcours
```

On rappelle les points suivants :

- la méthode `append` ajoute un élément à une liste Python ;

  par exemple, `tab.append(el)` permet d'ajouter l'élément `el` à la liste Python `tab` ;

- la méthode `pop` enlève de la liste l'élément situé à la position indiquée et le renvoie en valeur de retour;

  par exemple, `tab.pop(2)` enlève l'élément à l'indice 2 et le renvoie.

Dans ce parcours, les sites non encore traités sont de couleur `'blanche'` (valeur par défaut à la création de l'objet) et ceux qui sont traités de couleur `'noire'`.

6. Dans ce parcours, on manipule la liste Python nommée `listeS` uniquement à l'aide d'appels de la forme `listeS.append(sommet)` et `listeS.pop(0)`. Donner la structure de données correspondant à ces manipulations.

7. Donner le nom de ce parcours de graphe.

8. La fonction `parcoursGraphe` renvoie une liste `parcours`. Indiquer la valeur renvoyée par l'appel de fonction :

   ```
   parcoursGraphe(s1)
   ```

On cherche maintenant le site le plus populaire, celui dont la valeur de popularité est la plus grande.

Voici le code de la fonction qui renvoie le site le plus populaire, elle prend comme argument une liste non vide contenant des instances de la classe `Site`.

```
1  def lePlusPopulaire(listeSites):
2      maxPopularite = 0
3      siteLePlusPopulaire=listeSites[0]
4      for site in listeSites:
5          if site.popularite > maxPopularite:
6              ...
7              ...
8      return siteLePlusPopulaire
```

9. Copier et compléter les lignes 6 et 7 de cette fonction.

10. Donner ce que renvoie la ligne de code suivante :

```
lePlusPopulaire(parcoursGraphe(s1)).nom
```

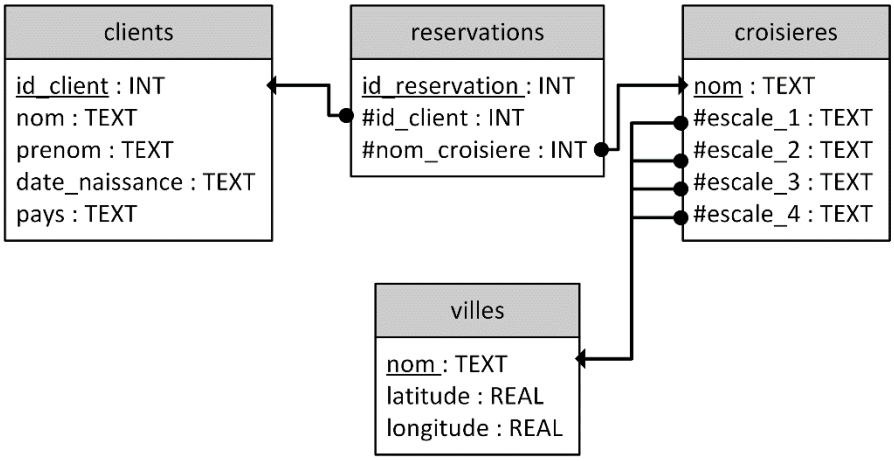
11. On envisage d'utiliser l'ensemble des fonctions proposées ci-dessus pour rechercher le site le plus populaire parmi un très grand nombre de sites (quelques milliers de sites). Expliquer si ce code est adapté à une telle quantité de sites à traiter. Justifier votre réponse.

EXERCICE 2 (6 points)

Cet exercice traite de protocoles de routage, de sécurité des communications et de base de données relationnelle.

Une agence de voyage propose des croisières en bateau. Chaque croisière a un nom unique et passe par quatre escales correspondant à des villes qui ont elles aussi des noms différents.

Pour gérer les réservations de ses clients, l'agence utilise une base de données. Voici la description des trois relations de cette base dont les clés primaires ont été soulignées et les clés étrangères indiquées par un # :



Remarque : l'énoncé de cet exercice utilise tout ou une partie des mots suivants du langage SQL : `SELECT`, `FROM`, `WHERE`, `JOIN ON`, `INSERT INTO`, `VALUES`, `UPDATE SET`, `OR`, `AND`.

Partie A

L'agence de voyage possède deux bureaux distincts.

Elle passe par un prestataire de service qui héberge sa base de données et utilise un système de gestion de base de données relationnelle.

Vous trouverez ci-après un schéma du réseau entre les deux bureaux de l'agence de voyage et le prestataire.

On peut y voir les différents routeurs (nommés de A à I) ainsi que le coût des liaisons entre eux.

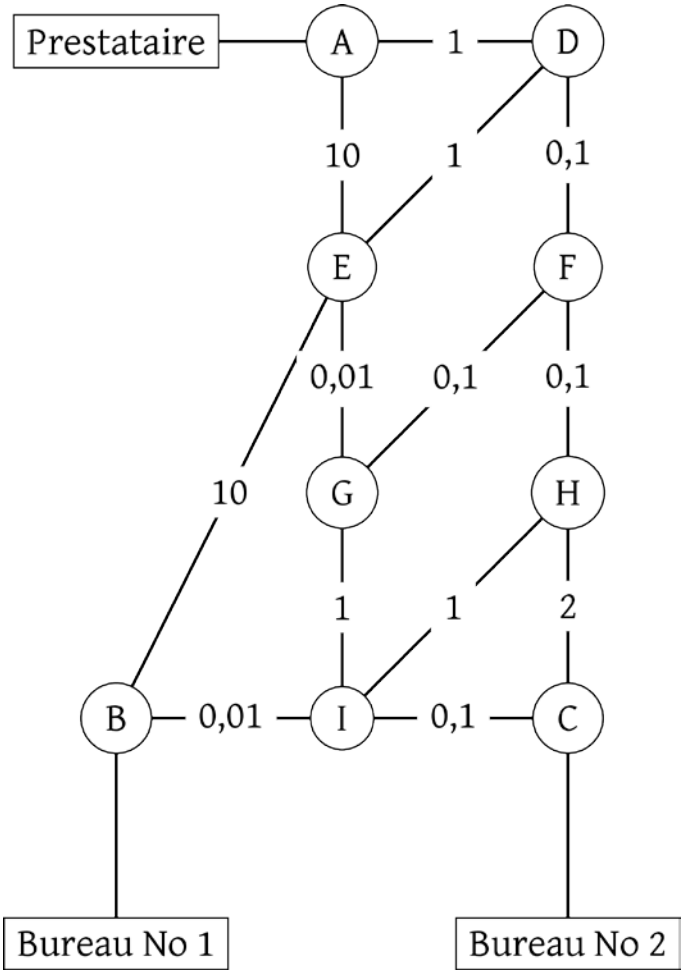


Figure 1. Topologie du réseau

1. Donner deux services rendus par un système de gestion de bases de données relationnelles.

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

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

2. Donner la route suivie par une requête issue du bureau numéro 1 jusqu'au prestataire si on utilise le protocole RIP.

3. Donner les deux routes que pourrait suivre une requête issue du bureau numéro 2 jusqu'au prestataire si on utilise le protocole OSPF. Donner le coût de chaque route.

Partie B

4. Expliquer pourquoi l'attribut `id_client` a été choisi comme clé primaire dans la relation `clients`.

5. Définir ce qu'est une clé étrangère. Donner la ou les clés étrangères de chaque relation qui en a en précisant la clé primaire qu'elles référencent.

L'agence a obtenu l'autorisation de faire escale dans quatre nouvelles villes : *Puerto saibo*, *Puerto kifecho*, *Puerto kifebo* et *Puerto repo*. Elle souhaite créer une nouvelle croisière qui passera par ces quatre villes. Un stagiaire de l'agence demande de l'aide à une *Intelligence Artificielle* (IA) :

Pour ajouter la nouvelle croisière nommée 'Croisière Puerto' avec ses escales correspondantes, vous pouvez utiliser la requête suivante :

sql Copy code

```
INSERT INTO croisieres (nom, escale_1, escale_2, escale_3,
escale_4)
VALUES ('Croisière Puerto',
'Puerto sebo',
'Puerto kifecho',
'Puerto kifebo',
'Puerto repo');
```

Figure 2 - Réponse de l'IA

Il tape alors la requête proposée mais obtient le message d'erreur suivant du SGBD (*Systèmes de Gestion de Bases de Données*) : *FOREIGN KEY constraint failed*.

6. Expliquer l'erreur commise et proposer une solution.

Partie C

7. Jean Barc, un allemand né le 29 juin 1972, demande un geste commercial en raison de sa fidélité à l'agence. Expliquer les requêtes SQL suivantes saisies par le gestionnaire :

```
SELECT id FROM clients
WHERE nom = 'Barc' AND prenom = 'Jean' AND date_naissance =
'1972/06/29' AND pays = 'Allemagne';
```

```
SELECT id_reservation FROM reservations
WHERE id_client = 1243;
```

8. Écrire les deux requêtes de la question 7. sous la forme d'une requête unique.

9. Un client souhaite modifier sa réservation d'identifiant 20456. Il souhaite remplacer la croisière de cette réservation par la toute dernière offre de l'agence : la *Croisière Puerto*. Écrire une requête SQL qui permet de mettre à jour la base de données pour lui donner satisfaction.

10. Donner une requête SQL permettant d'obtenir les noms, prénoms et dates de naissance des clients ayant choisi la croisière nommée *Croisière Piano* ou celle nommée *Croisière Puerto.*

EXERCICE 3 (8 points)

Cet exercice porte sur la programmation orientée objet, sur les arbres binaires de recherche et la récursivité.

Chaque année, plusieurs courses de chiens de traîneaux sont organisées sur les terrains enneigés. L'une d'elle, *La Traversée Blanche*, est une course se déroulant en 9 étapes. L'organisateur de cette course est chargé de créer un programme Python pour aider à la bonne gestion de l'événement.

Partie A : la classe `Chien`

Afin de caractériser un chien, l'organisateur décide de créer une classe `Chien` avec les attributs suivants :

- `id_chien`, un nombre entier correspondant au numéro attribué au chien lors de son inscription à la course ;
- `nom`, une chaîne de caractères correspondant au nom du chien ;
- `role`, une chaîne de caractères correspondant au poste occupé par le chien : en fonction de sa place dans l'attelage, un chien a un rôle bien défini et peut être `'leader'`, `'swing dog'`, `'wheel dog'` ou `'team dog'`.
- `id_proprietaire`, un nombre entier correspondant au numéro de l'équipe.

Le code Python incomplet de la classe `Chien` est donné ci-dessous.

```
1 class Chien:
2     def __init__(self, id_chien, nom, role, id_prop):
3         self.id_chien = id_chien
4         self.nom = nom
5         self.role = role
6         self.id_proprietaire = id_prop

7     def changer_role(self, nouveau_role):
8         """Change le rôle du chien avec la valeur passée en
             paramètre."""
9         ...
```

Voici un extrait des informations dont on dispose sur les chiens inscrits à la course.

Chiens inscrits à la course			
`id_chien`	`nom`	`role`	`id_proprietraire`
40	Duke	wheel dog	10
41	Sadie	team dog	10
42	Zeus	swing dog	11
43	Roxie	swing dog	11
44	Scout	team dog	11
45	Ginger	team dog	11
46	Helka	team dog	11

Suite aux inscriptions, l'organisateur procède à la création de tous les objets de type `Chien` et les stocke dans des variables en choisissant un nom explicite. Ainsi, l'objet dont l'attribut `id_chien` a pour valeur 40 est stocké dans la variable `chien40`.

1. Écrire l'instruction permettant d'instancier l'objet `chien40` caractérisant le chien ayant le numéro d'inscription 40.

2. Selon l'état de fatigue de ses chiens ou du profil de l'étape, le *musher* (nom donné à la personne qui conduit le traîneau) peut décider de changer le rôle des chiens dans l'attelage.
   Recopier et compléter la méthode `changer_role` de la classe `Chien`.

3. Le propriétaire de Duke décide de lui attribuer le rôle de `'leader'`. Écrire l'instruction permettant d'effectuer cette modification.

Partie B : la classe `Equipe`

On souhaite à présent créer une classe `Equipe` ayant les attributs suivants :

- `num_dossard`, un nombre entier correspondant au numéro inscrit sur le dossard du *musher* ;
- `nom_equipe`, une chaîne de caractères correspondant au nom de l'équipe ;
- `liste_chiens`, une liste d'objets de type `Chien` dont chaque élément correspond à un chien au départ de l'étape du jour ;
- `temps_etape`, une chaîne de caractères (par exemple `'2h34'`) représentant le temps mis par l'équipe pour parcourir l'étape du jour ;
- `liste_temps`, une liste de chaînes de caractères permettant de stocker les temps de l'équipe pour chacune des 9 étapes. Cet attribut peut, par exemple, contenir la liste : `['4h36', '3h57', '3h09', '5h49', '4h45', '3h26', '4h57', '5h52', '4h31']`.

On donne le code Python suivant de la classe `Equipe`.

```
1  class Equipe:
2      def __init__(self, num_dossard, nom_equipe):
3          self.num_dossard = num_dossard
4          self.nom_equipe = nom_equipe
5          self.liste_chiens = []
6          self.temps_etape = ''
7          self.liste_temps = []
8
9      def ajouter_chien(self, chien):
10         self.liste_chiens.append(chien)
11
12     def retirer_chien(self, numero):
13         ...
14
15     def ajouter_temps_etape(self, temps):
16         self.liste_temps.append(temps)
```

Pour la première étape, le *musher* de l'équipe numéro 11, représentée en Python par l'objet `eq11`, décide de constituer une équipe avec les quatre chiens identifiés par les numéros 42, 44, 45 et 46. On donne ci-dessous les instructions Python permettant de créer l'équipe `eq11` et l'attelage constitué des 4 chiens précédents.

```
1  eq11 = Equipe(11, 'Malamutes Endurants')
2  eq11.ajouter_chien(chien42)
3  eq11.ajouter_chien(chien44)
4  eq11.ajouter_chien(chien45)
5  eq11.ajouter_chien(chien46)
```

Malheureusement, le *musher* s'aperçoit que sa chienne Helka, chien numéro 46, n'est pas au mieux de sa forme et il décide de la retirer de l'attelage.

4. Recopier et compléter la méthode `retirer_chien` ayant pour paramètre `numero`, un entier correspondant au numéro attribué au chien lors de l'inscription, et permettant de mettre à jour l'attribut `liste_chiens` après retrait du chien dont la valeur de l'attribut `id_chien` est `numero`.

5. En vous aidant de la fonction précédente, écrire l'instruction qui permet de retirer Helka de l'attelage de l'équipe `eq11`.

On donne à présent le code Python d'une fonction `convert` prenant pour paramètre `chaine`, une chaîne de caractères représentant une durée, donnée en heure et minute. On supposera que cette durée est toujours strictement inférieure à 10 heures, temps maximal fixé par le règlement pour terminer une étape.

```
1 def convert(chaine):
2     heure_dec = int(chaine[0]) + int(chaine[2] + chaine[3])/60
3     return heure_dec
```

6. Indiquer le résultat renvoyé par l'appel `convert('4h36')`.

7. Écrire une fonction `temps_course` qui prend pour paramètre `equipe` de type `Equipe` et qui renvoie un nombre flottant correspondant au cumul des temps de l'équipe `equipe` à l'issue des 9 étapes de la course.
   On rappelle que la classe `Equipe` dispose d'un attribut `liste_temps`.

Partie C : classement à l'issue d'une étape

Chaque jour, à la fin de l'étape, on décide de construire un Arbre Binaire de Recherche (ABR) afin d'établir le classement des équipes. Chaque nœud de cet arbre est un objet de type `Equipe`.

Dans cet arbre binaire de recherche, en tout nœud :

- toutes les équipes du sous-arbre gauche sont strictement plus rapides que ce nœud ;
- toutes les équipes du sous-arbre droit sont moins rapides ou sont à égalité avec ce nœud.

Voici les temps, en heure et minute, relevés à l'issue de la première étape :

Temps à l'arrivée de la première étape											
Equipe	`eq1`	`eq2`	`eq3`	`eq4`	`eq5`	`eq6`	`eq7`	`eq8`	`eq9`	`eq10`	`eq11`
Temps	4h36	3h57	3h09	5h49	4h45	3h26	4h51	5h52	4h31	3h44	4h26

Dans l'arbre binaire de recherche initialement vide, on ajoute successivement, dans cet ordre, les équipes `eq1`, `eq2`, `eq3`, …, `eq11`, 11 objets de la classe `Equipe` tous construits sur le même modèle que l'objet `eq11` précédent.

8. Dans l'arbre binaire de recherche ci-dessous, les nœuds `eq1` et `eq2` ont été insérés. Recopier et compléter cet arbre en insérant les 9 nœuds manquants.

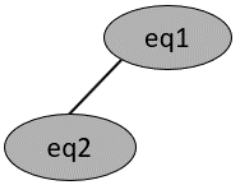


Figure 1. Premiers éléments de l'ABR

9. Indiquer quel parcours d'arbre permet d'obtenir la liste des équipes classées de la plus rapide à la plus lente.

On donne ci-dessous la classe `Noeud`, permettant de définir les arbres binaires :

```
1 class Noeud:
2     def __init__(self, equipe, gauche = None, droit = None):
3         self.racine = equipe
4         self.gauche = gauche
5         self.droit = droit
```

On donne ci-dessous le code d'une fonction `construction_arbre` qui, à partir d'une liste d'éléments de type `Noeud` permet d'insérer successivement chaque nœud à sa place dans l'ABR.

```
1 def construction_arbre(liste):
2     a = Noeud(liste[0])
3     for i in range(1,len(liste)):
4         inserer(a, liste[i])
5     return a
```

La fonction `construction_arbre` fait appel à la fonction `inserer` qui prend pour paramètre `arb`, de type `Noeud`, et `eq`, de type `Equipe`. Cette fonction construit le nœud à partir de `eq` et l'insère à sa place dans l'ABR.

```
1  def inserer(arb, eq):
2      """ Insertion d'une équipe à sa place dans un ABR contenant
3          au moins un noeud."""
4      if convert(eq.temps_etape) < convert(arb.racine.temps_etape):
5          if arb.gauche is None:
6              arb.gauche = ...
7          else:
8              inserer(..., eq)
9      else:
10          if arb.droit is None:
11             arb.droit = Noeud(eq)
12         else:
13             ...
```

10. Expliquer en quoi la fonction `inserer` est une fonction récursive.

11. Recopier et compléter les lignes 6, 8 et 13 de la fonction `inserer`.

12. Recopier et compléter les lignes 3 et 5 de la fonction `est_gagnante` ci-dessous qui prend en paramètre un ABR `arbre`, de type `Noeud`, et qui renvoie le nom de l'équipe ayant gagné l'étape.

```
1 def est_gagnante(arbre):
2     if arbre.gauche == None:
3         return ...
4     else:
5         return ...
```

Partie D : classement général

On décide d'établir un classement général obtenu à partir du cumul des temps mis par chaque équipe pour parcourir l'ensemble des 9 étapes.

Sur le même principe que l'arbre de la partie précédente, on construit l'ABR ci-dessous qui permet, grâce au parcours d'arbre approprié, d'établir ce classement général des équipes.

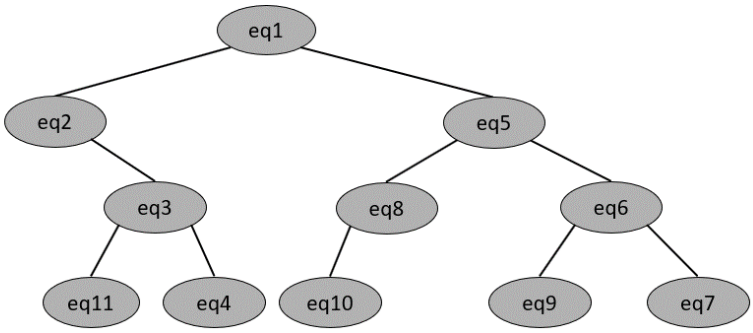


Figure 2. ABR du classement général

Le règlement prévoit la disqualification d'une équipe en cas de non-respect de celui-ci. Il s'avère que l'équipe 2 et l'équipe 5 doivent être disqualifiées pour manquement au règlement. Les nœuds `eq2` et `eq5` doivent donc être supprimés de l'ABR précédent.

Pour supprimer un nœud `N` dans un ABR, trois possibilités se présentent :

- le nœud `N` à supprimer est une feuille : il suffit de le retirer de l'arbre ;
- le nœud `N` à supprimer n'a qu'un seul fils : on relie le fils de `N` au père de `N` et on supprime le nœud `N` ;
- le nœud `N` à supprimer possède deux fils : on le remplace par son successeur (l'équipe qui a le temps immédiatement supérieur) qui est toujours le minimum de ses descendants droits.

13. Dessiner le nouvel arbre de recherche `a_final` obtenu après suppression des équipes `eq2` et `eq5` dans l'ABR correspondant au classement général.

L'organisateur souhaite disposer d'une fonction `rechercher` permettant de savoir si une équipe a été disqualifiée ou non. On donne les spécifications de la fonction `rechercher`, prenant en paramètre `arbre` et `equipe`.

```
1  def rechercher(arbre, equipe):
2      """
3      Paramètres
4      ---------
5      arbre : un ABR, non vide, de type Noeud, représentant le
6              classement général.
7      equipe : un élément, de type Equipe, dont on veut déterminer
8               l'appartenance ou non à l'ABR arbre.
9      Résultat
10     ---------
11     Cette fonction renvoie True si equipe est un nœud de arbre,
12     False sinon.
13     """
14     ...
```

Pour cette fonction (`a_final` désigne l'arbre obtenu à la question 13, après suppression des équipes 2 et 5) :

- l'appel `rechercher(a_final, eq1)` renvoie `True` ;
- l'appel `rechercher(a_final, eq2)` renvoie `False`.

14. Écrire le code de la fonction `rechercher`.

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU JEUDI 20 JUIN 2024

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 15 pages numérotées de 1/15 à 15/15.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

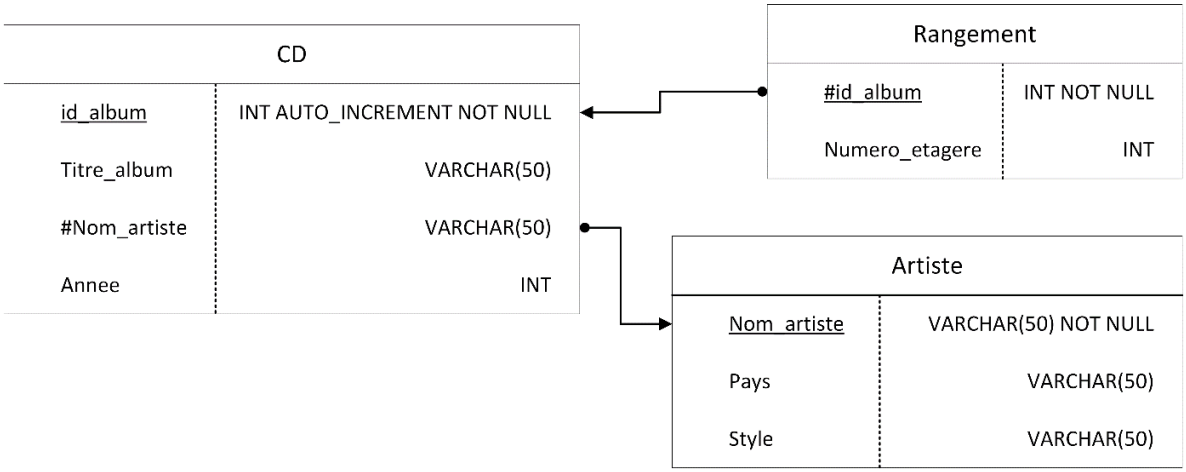
EXERCICE 1 (6 points)

Cet exercice porte sur la notion de bases de données relationnelles, le langage SQL et les protocoles de sécurisation.

Partie A – Bases de données

L'énoncé de cette partie utilise les mots du langage SQL suivants : SELECT, FROM, WHERE, JOIN, UPDATE, SET, DELETE. L'attribut AUTO_INCREMENT permet d'incrémenter automatiquement un entier dans une table à l 'insertion d'un nouvel élément.

Bob, qui dispose d'une très grande collection de CDs rangés sur plusieurs étagères numérotées, a mis en place une base de données. Voici la description des trois relations de cette base dont les clés primaires ont été soulignées et les clés étrangères indiquées par un # :



1. Indiquer, avec justification, s'il aurait été possible de choisir l'attribut Nom_artiste comme clé primaire dans la relation CD.

Dans la suite, on considère les clés étrangères suivantes :

- CD.Nom_artiste qui référence l'attribut Artiste.Nom_artiste ;
- Rangement.id_album qui référence l'attribut CD.id_album.

Voici un extrait des enregistrements des relations CD, Artiste et Rangement définies plus haut :

CD			
id_album	Titre_album	Nom_artiste	Annee
1	‘Master of Puppets’	‘Metallica’	1986
2	‘The Marshall Mathers LP’	‘Eminem’	2000
3	‘Wasting Light’	‘Foo Fighters’	2011
4	‘Wishmaster’	‘Nightwish’	2001
5	‘Dead Letters’	‘The Rasmus’	2003
6	‘Somewhere in Time’	‘Iron Maiden’	1986

Artiste		
Nom_artiste	Pays	Style
‘Nightwish’	‘Finlande’	‘Metal’
‘Foo Fighters’	‘Etats-Unis’	‘Rock’
‘Metallica’	‘Etats-Unis’	‘Metal’
‘Iron Maiden’	‘Royaume-Uni’	‘Metal’
‘Eminem’	‘Etats-Unis’	‘Rap’
‘The Rasmus’	‘Finlande’	‘Rock’

Rangement	
id_album	Numero_etagere
1	2
2	1
3	1
4	3
5	3
6	2

2. Écrire ce que renvoie la requête suivante lorsqu'on l'applique aux extraits ci-dessus.

```
SELECT Nom_artiste
FROM Artiste
WHERE Pays = "Finlande";
```

3. Écrire à présent ce que renvoie la requête suivante.

```
SELECT CD.Annee
FROM CD
JOIN Artiste
ON CD.Nom_artiste = Artiste.Nom_artiste
WHERE Artiste.Style = "Metal";
```

Bob se rend compte que l'album Wishmaster est en réalité sorti en 2000.

4. Donner la requête qu'il doit écrire pour mettre à jour sa base de données.

5. Donner la requête qu'il doit écrire pour afficher les titres de tous les albums de "Metal" rangés sur l'étagère dont le numéro est 1.

Bob a vendu l'album Dead Letters du groupe The Rasmus. Puisqu'il s'agissait du seul album de ce groupe qu'il possédait, il veut supprimer tous les enregistrements qui sont à présent inutiles dans les trois relations.

6. Donner l'ordre dans lequel il doit les supprimer en expliquant pourquoi, puis écrire la requête correspondant à la suppression de l'album dans la relation CD.

Partie B – Sécurisation

La base de données de Bob est hébergée sur un serveur auquel il accède depuis un client sur son ordinateur personnel. Pour sécuriser la connexion, un algorithme de chiffrement symétrique est utilisé.

7. Expliquer brièvement ce qu'est un algorithme de chiffrement symétrique.

La clé de chiffrement, notée C dans la suite, est choisie aléatoirement par le serveur à chaque connexion depuis un client. Afin que le chiffrement et le déchiffrement puisse se faire sans problème, le serveur doit envoyer au client la clé C de façon sécurisée.

8. Rappeler brièvement ce qu'est un algorithme de chiffrement asymétrique.

On suppose à présent que Bob possède une clé publique et une clé privée. La clé publique de Bob est supposée connue par le serveur.

9. Proposer alors une solution pour que le serveur puisse envoyer la clé C à l'ordinateur de Bob de façon sécurisée, c'est-à-dire pour que seul Bob puisse déchiffrer la clé envoyée.

EXERCICE 2 (6 points)

Cet exercice porte sur la programmation orientée objets, les tris, les algorithmes gloutons, la récursivité et les assertions.

Cet exercice est composé de trois parties dont les deux dernières sont indépendantes entre elles.

Dans cet exercice, l'entête des fonctions est décrit avec le type des objets en paramètre et le type de l'objet renvoyé. Ainsi la fonction puissance qui prend un paramètre flottant `x` et un entier `n` puis qui renvoie le flottant `x**n`, a pour entête `puissance(x: float, n: int) -> float`

Une entreprise transporte des marchandises. Elle souhaite maximiser son profit en optimisant le remplissage de ses moyens de transport. On considère qu'un moyen de transport est limité par son volume (exprimé en litres). Chaque marchandise est caractérisée par son prix (en euros) et son volume indivisible (en litres).

Supposons qu'on ait trois marchandises caractérisées par les couples (prix, volume) suivants : $m_1 = (100, 10)$, $m_2 = (100, 10)$ et $m_3 = (250, 20)$. Si le moyen de transport peut encore charger 25 litres, il vaut mieux charger la marchandise numéro 3 qui rapporte 250 € à l'entreprise plutôt que charger les marchandises numéros 2 et 3 qui rapportent 200 € au total pour le même espace utilisé.

Partie A – Quelques outils

Nous souhaitons définir une classe `Marchandise` dont chaque instance définit une marchandise possédant deux attributs entiers `prix` et `volume`.

1. Compléter le constructeur qui renvoie un objet `Marchandise`. Utiliser le mot-clé `assert` afin qu'une exception soit levée si le paramètre `v` n'est pas strictement positif.
   On rappelle que si `condition` est une expression Python booléenne s'évaluant à `True` ou `False`, l'instruction `assert condition` déclenche une exception quand la condition s'évalue à `False`.

```
class Marchandise:
    def __init__(self, p: int, v: int) -> 'Marchandise':
        ...
```

2. Donner une instruction qui permet de créer une variable `m1` représentant une marchandise d'un volume de 7 litres coûtant 20 €.

3. Proposer une méthode `ratio(self) -> float` qui renvoie le ratio prix/volume d'une marchandise.

4. Proposer une fonction `prixListe(tab: list) -> int` qui renvoie le prix cumulé de l'ensemble des marchandises formant le tableau `tab`.

Partie B – Première approche de rangement

Le transporteur souhaite maximiser son profit. On considère que nous avons les quatre marchandises définies par les couples (prix, volume) suivants :

$$m_1 = (40, 20),\ m_2 = (210, 70),\ m_3 = (160, 40) \text{ et } m_4 = (50, 50).$$

5. Préciser toutes les combinaisons de marchandises possibles si on ne dépasse pas un volume de 100 litres et le prix associé. En déduire la combinaison de marchandises qui maximise le prix.

Une première méthode appelée `ChargementGlouton` consiste à trier les marchandises dans l'ordre décroissant de leur prix volumique (ratio prix/volume), puis transporter en priorité les marchandises avec le plus grand prix volumique. Si une marchandise est trop volumineuse pour être transportée, on essaie avec la marchandise ayant le prix volumique juste inférieur, ce jusqu'à ce qu'aucune marchandise ne puisse rentrer. Ainsi, en notant `v_restant` le volume disponible et `m_i` la $(i+1)^{\text{e}}$ marchandise une fois les marchandises triées, l'algorithme peut s'écrire :

`ChargementGlouton`

```
n = nombre de marchandises
POUR i ALLANT de 0 à n-1 FAIRE
| SI volume de m_i <= v_restant ALORS
|  | charger m_i
|  | v_restant = v_restant - volume de m_i
TRANSPORTER le chargement prévu
```

Le tri dans l'ordre décroissant des prix volumiques donne m_3 , m_2, m_1, m_4. Si le moyen de transport accepte 100 litres de chargement, l'algorithme charge m_3 et m_1 pour un prix de 200 € (à comparer avec la combinaison trouvée précédemment pour maximiser le prix).

Par la suite, on rappelle que dans l'implémentation Python, les marchandises sont définies par des instances de la classe `Marchandise`.

On donne quelques qualificatifs : dichotomique, glouton, graphique, insertion, maximum, récursif, tri.

6. Indiquer, sans justification, le qualificatif qui s'applique le mieux à l'algorithme précédent.

7. Recopier et compléter la fonction `tri(tab: list) -> None` ci-dessous afin qu'elle trie en place un tableau contenant des objets de type `Marchandise` selon l'ordre décroissant des ratios. Ainsi, `tab[0]` doit contenir la marchandise avec le plus haut ratio prix/volume après l'appel `tri(tab)`.

```
def tri(tab: list) -> None:
    n = len(tab)
    for i in range(1, n):
        marchandise = tab[i]
        j = i-1
        while ... and ... > ... :
            tab[j+1] = ...
            j = ...
        tab[j+1] = marchandise
```

8. Sans justifier, préciser le nom de ce tri, ainsi que son coût temporel dans le pire des cas (constant, logarithmique, linéaire, quasi-linéaire ($n \log_2 n$), quadratique, cubique ou exponentiel).

9. Recopier et compléter la fonction `charge` suivante qui applique l'algorithme `ChargementGlouton` décrit plus haut.

```
def charge(tab: list, volume: int) -> list:
    tri(tab)
    chargement = []
    n = len(tab)
    for ...
        if ...
           ...
           ...
    return ...
```

Partie C – Rangement optimisé par récursivité

L'algorithme précédent ne renvoie pas toujours une solution optimale. On peut donc suivre un algorithme récursif. On note `n` le nombre de marchandises et on souhaite implémenter la fonction récursive `chargeOptimale` d'entête :

```
chargeOptimale(tab: list, v_restant: int, i: int) -> list
```

Un appel à cette fonction doit permettre de calculer la charge optimale pour un transport de volume `v_restant` utilisant les marchandises à partir de l'indice `i` :

- si `i >= n`, toutes les marchandises ont été essayées et il n'en reste plus d'autres disponibles. L'appel récursif renvoie la liste vide ;
- si `i < n` et la marchandise d'indice `i` est de volume strictement supérieur au volume restant, l'appel récursif renvoie le résultat de l'appel effectué avec le même volume restant mais avec la marchandise suivante, c'est-à-dire `chargeOptimale(tab, v_restant, i+1)` ;

- si `i < n` et la marchandise d'indice `i` est de volume inférieur ou égal au volume restant, il existe deux options possibles :
  - Option 1 soit on utilise la marchandise `i`, auquel cas le chargement contiendra cette marchandise et celles du résultat de l'appel récursif à partir de la prochaine marchandise et d'un volume restant strictement inférieur,
  - Option 2 soit on n'utilise pas la marchandise `i`, auquel cas le chargement sera le résultat de l'appel récursif avec le même volume restant mais à partir de la marchandise suivante.

  On garde l'option de chargement qui maximise le prix transporté.

10. Compléter le code de la fonction `chargeOptimale` dont le principe a été décrit ci-avant.

```
def chargeOptimale(tab: list, v_restant: int, i: int) -> list:
    if i>=...:
        return ...
    else:
        if tab[i].volume > v_restant:
            return chargeOptimale(tab, v_restant, i+1)
        else:
            option1 = chargeOptimale(tab, ..., ...)
            option2 = [tab[i]] + chargeOptimale(tab, ..., ...)
            if prixListe(option1) > prixListe(option2):
                return ...
            else:
                return ...
```

EXERCICE 3 (8 points)

Cet exercice porte sur la programmation orientée objet, les graphes et utilise la structure de données dictionnaire.

La direction de la station de ski *Le Lièvre Blanc*, spécialisée dans la pratique du ski de fond, souhaite disposer d'un logiciel lui permettant de gérer au mieux son domaine skiable. Elle confie à un développeur informatique la mission de concevoir ce logiciel. Celui-ci décide de caractériser les pistes de ski à l'aide d'une classe `Piste` et le domaine de ski par une classe `Domaine`.

Le code Python de ces deux classes est donné en **Annexe**.

Partie A – Analyse des classes `Piste` et `Domaine`

1. Lister les attributs de la classe `Piste` en précisant leur type.

La difficulté des pistes de ski de fond est représentée par 4 couleurs : verte, bleue, rouge et noire. La piste verte est considérée comme très facile, la piste bleue comme facile, la piste rouge de difficulté moyenne et la piste noire difficile. Dans la station de ski *Le Lièvre blanc*, l'équipe de direction décide de s'appuyer uniquement sur le dénivelé pour attribuer la couleur d'une piste de ski.

Ainsi, une piste de ski sera de couleur :

- `'noire'` si son dénivelé est supérieur ou égal à 100 mètres ;
- `'rouge'` si son dénivelé est strictement inférieur à 100 mètres, mais supérieur ou égal à 70 mètres ;
- `'bleue'` si son dénivelé est strictement inférieur à 70 mètres, mais supérieur ou égal à 40 mètres ;
- `'verte'` si son dénivelé est strictement inférieur à 40 mètres.

2. Écrire la méthode `set_couleur` de la classe `Piste` qui permet d'affecter à l'attribut `couleur` la chaîne de caractères correspondant à la couleur de la piste.

On exécute à présent le programme suivant afin d'attribuer la couleur adéquate à chacune des pistes du domaine skiable *Le Lièvre Blanc*.

```
1 for piste in lievre_blanc.get_pistes():
2     piste.set_couleur()
```

3. Indiquer, parmi les 4 propositions ci-dessous, le type de l'élément renvoyé par l'instruction Python `lievre_blanc.get_pistes()`.

    - Proposition A : une chaîne de caractères ;
    - Proposition B : un objet de type `Piste` ;
    - Proposition C : une liste de chaînes de caractères ;
    - Proposition D : une liste d'objets de type `Piste`.

En raison d'un manque d'enneigement, la direction de la station est souvent contrainte de fermer toutes les pistes vertes car elles sont situées généralement en bas du domaine.

4. Écrire un programme Python dont l'exécution permet de procéder à la fermeture de toutes les pistes vertes en affectant la valeur `False` à l'attribut `ouverte` des pistes concernées.

5. Écrire une fonction `pistes_de_couleur` prenant pour paramètres une chaîne de caractères `couleur` représentant la difficulté d'une piste et une liste `lst` de pistes de ski de fond. Cette fonction renvoie la liste des noms des pistes dont `couleur` est le niveau de difficulté.

    Exemple : l'instruction `pistes_de_couleur(lievre_blanc.get_pistes(), 'noire')` renvoie la liste `['Petit Bonheur', 'Forêt', 'Duvallon']`.

Un skieur de bon niveau se prépare assidûment pour le prochain semi-marathon, d'une distance de 21,1 kilomètres. À chaque entraînement, il note la liste des noms des pistes qu'il a parcourues et il souhaite disposer d'un outil lui indiquant si la distance totale parcourue est au moins égale à la distance qu'il devra parcourir le jour du semi-marathon.

La fonction `semi_marathon` donnée ci-dessous répond aux attentes du skieur : cette fonction prend en paramètre une liste `L` de noms de pistes et renvoie un booléen égal à `True` si la distance totale parcourue est strictement supérieure à 21,1 kilomètres, `False` sinon.

```
1  def semi_marathon(L):
2      distance = ...
3      liste_pistes = lievre_blanc.get_pistes()
4      for nom in L:
5          for piste in liste_pistes:
6              if piste.get_nom() == ...:
7                  distance = distance + ...
8      return ...
```

On donne ci-dessous deux exemples d'appels à cette fonction :

```
>>> entrainement1 = ['Verneys', 'Chateau enneigé', 'Rois mages',
'Diablotin']
>>> semi_marathon(entrainement1)
True
>>> entrainement2 = ['Esseillon', 'Aigle Royal', 'Duvallon']
>>> semi_marathon(entrainement2)
False
```

6. Recopier et compléter la fonction `semi_marathon`.

Partie B – Recherche par force brute

Le plan des pistes du domaine *Le Lièvre Blanc* peut être représenté par le graphe suivant :

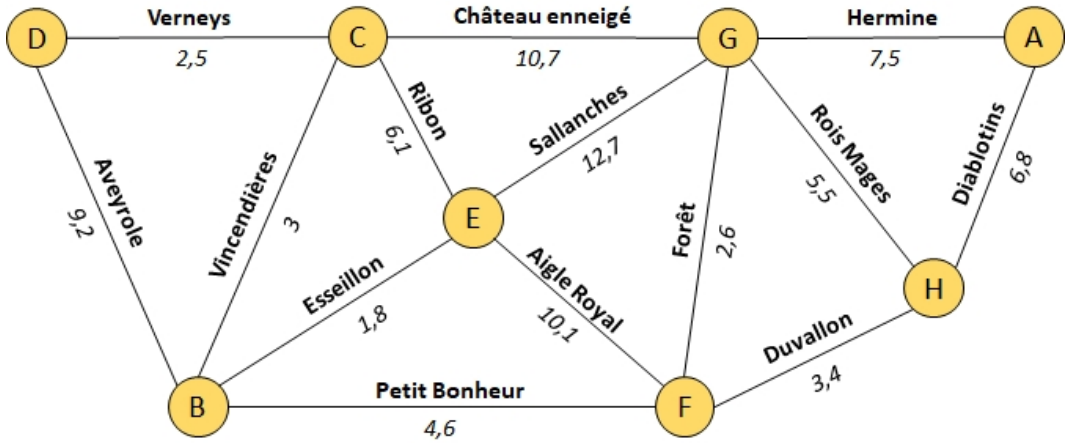


Figure 1. Graphe du domaine *Le Lièvre Blanc*

Sur chaque arête, on a indiqué le nom de la piste et sa longueur en kilomètres. Les sommets correspondent à des postes de secours.

Un pisteur-secouriste de permanence au point de secours D est appelé pour une intervention en urgence au point de secours A. La motoneige de la station étant en panne, il ne peut s'y rendre qu'en skis de fond. Il décide de minimiser la distance parcourue et cherche à savoir quel est le meilleur parcours possible. Pour l'aider à répondre à ce problème, on décide d'implémenter le graphe ci-dessus grâce au dictionnaire de dictionnaires suivant :

```
domaine = {'A' : {'G' : 7.5, 'H' : 6.8},
           'B' : {'C' : 3.0, 'D' : 9.2, 'E' : 1.8, 'F' : 4.6},
           'C' : {'B' : 3.0, 'D' : 2.5, 'E' : 6.1, 'G' : 10.7},
           'D' : {'B' : 9.2, 'C' : 2.5},
           'E' : {'B' : 1.8, 'C' : 6.1, 'F' : 10.1, 'G' : 12.7},
           'F' : {'B' : 4.6, 'E' : 10.1, 'G' : 2.6, 'H' : 3.4},
           'G' : {'A' : 7.5, 'C' : 10.7, 'E' : 12.7, 'F' : 2.6,
                  'H' : 5.5},
           'H' : {'A' : 6.8, 'F' : 3.4, 'G' : 5.5} }
```

7. Écrire une instruction Python permettant d'afficher la longueur de la piste allant du sommet `'E'` au sommet `'F'`.

8. Écrire une fonction `voisins` qui prend en paramètres un graphe `G` et un sommet `s` du graphe `G` et qui renvoie la liste des voisins du sommet `s`.

   Exemple : l'instruction `voisins(domaine, 'B')` renvoie la liste `['C', 'D', 'E', 'F']`.

9. Recopier et compléter la fonction `longueur_chemin` donnée ci-dessous : cette fonction prend en paramètres un graphe `G` et un chemin du graphe `G` sous la forme d'une liste de sommets et renvoie sa longueur en kilomètres.

   Exemple : l'instruction `longueur_chemin(domaine, ['B', 'E', 'F', 'H'])` renvoie le nombre flottant `15.3`.

```
def longueur_chemin(G, chemin):
    precedent = ...
    longueur = 0
    for i in range(1, len(chemin)):
        longueur = longueur + ...
        precedent = ...
    return ...
```

On donne ci-dessous une fonction `parcours` qui renvoie la liste de tous les chemins du graphe `G` partant du sommet `depart` et parcourant les sommets de façon unique, c'est-à-dire qu'un sommet est atteint au plus une fois dans un chemin.

Par exemple, l'appel `parcours(domaine, 'A')` renvoie la liste de tous les chemins partant du sommet A dans le graphe `domaine` sans se soucier ni de la longueur du chemin, ni du sommet d'arrivée. Ainsi, `['A', 'G', 'C']` est un chemin possible, tout comme `['A', 'G', 'C', 'B', 'E', 'F', 'H']`.

```
def parcours(G, depart, chemin = [], lst_chemins = []):
    if chemin == []:
        chemin = [depart]
    for sommet in voisins(G, depart):
        if sommet not in chemin:
            lst_chemins.append(chemin + [sommet])
            parcours(G, sommet, chemin + [sommet])
    return lst_chemins
```

10. Expliquer en quoi la fonction `parcours` est une fonction récursive.

Un appel à la fonction `parcours` précédente renvoie une liste de chemins dans laquelle figurent des doublons.

11. Recopier et compléter la fonction `parcours_dep_arr` ci-après qui renvoie la liste des chemins partant du sommet `depart` et se terminant par le sommet `arrivee` dans le graphe `G` entrés en paramètres. La liste renvoyée ne doit pas comporter de doublons. Attention, plusieurs lignes de code sont nécessaires.

```
def parcours_dep_arr(G, depart, arrivee):
    liste = parcours(G, depart)
    ...
```

12. Recopier et compléter la fonction `plus_court` donnée ci-dessous. La fonction `plus_court` prend pour paramètres un graphe `G`, un sommet de départ `depart` et un sommet d'arrivée `arrivee` ; elle renvoie un des chemins les plus courts sous la forme d'une liste de sommets.

```
def plus_court(G, depart, arrivee):
    liste_chemins = parcours_dep_arr(G, depart, arrivee)
    chemin_plus_court = ...
    minimum = longueur_chemin(G, chemin_plus_court)
    for chemin in liste_chemins:
        longueur = longueur_chemin(G, chemin)
        if ...:
            minimum = ...
            chemin_plus_court = ...
    return chemin_plus_court
```

13. Expliquer en quoi le choix fait par le pisteur-secouriste de choisir la distance minimale pour arriver le plus rapidement possible sur le lieu de l'incident est discutable. Proposer un meilleur critère de choix.

Annexe

```
# Pistes
class Piste:
    def __init__(self, nom, denivele, longueur):
        self.nom = nom
        self.denivele = denivele   # en mètres
        self.longueur = longueur   # en kilomètres
        self.couleur = ''
        self.ouverte = True

    def get_nom(self):
        return self.nom

    def get_longueur(self):
        return self.longueur

    def set_couleur(self):
        # À compléter

    def get_couleur(self):
        return self.couleur

# Domaine skiable
class Domaine:
    def __init__(self, a):
        self.nom = a
        self.pistes = []

    def ajouter_piste(self, nom_piste, denivele, longueur):
        self.pistes.append(Piste(nom_piste, denivele, longueur))

    def get_pistes(self):
        return self.pistes

# Programme principal
lievre_blanc = Domaine("Le Lièvre Blanc")
lievre_blanc.ajouter_piste('Aveyrole', 62, 9.2)
lievre_blanc.ajouter_piste('Verneys', 10, 2.5)
lievre_blanc.ajouter_piste('Vincendières', 45, 3)
lievre_blanc.ajouter_piste('Ribon', 70, 6.1)
lievre_blanc.ajouter_piste('Esseillon', 8, 1.8)
lievre_blanc.ajouter_piste('Petit Bonheur', 310, 4.6)
lievre_blanc.ajouter_piste('Aigle Royal', 85, 10.1)
lievre_blanc.ajouter_piste('Château enneigé', 54, 10.7)
lievre_blanc.ajouter_piste('Sallanches', 78, 12.7)
lievre_blanc.ajouter_piste('Forêt', 145, 2.6)
```

```
lievre_blanc.ajouter_piste('Hermine', 27, 7.5)
lievre_blanc.ajouter_piste('Rois mages', 42, 5.5)
lievre_blanc.ajouter_piste('Diablotin', 76, 6.8)
lievre_blanc.ajouter_piste('Duvallon', 200, 3.4)
```

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU MERCREDI 11 SEPTEMBRE 2024

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.

L'annexe page 16 est à rendre avec la copie

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

EXERCICE 1 (6 points)

Cet exercice porte sur la programmation Python, les bases de données relationnelles et les requêtes SQL.

En particulier, les mots-clés suivants peuvent être utilisés :

`SELECT`, `CREATE TABLE`, `FROM`, `WHERE`, `JOIN ON`, `INSERT INTO`, `VALUES`, `UPDATE`, `SET`, `COUNT`, `DELETE`, `DISTINCT`, `AND`, `OR`, `AS`.

Partie A

Dans cette partie, on utilise une base de données relationnelle.

Une entreprise de location de voitures propose à ses clients la possibilité de rapporter le véhicule dans une autre agence. Les informations correspondantes sont rangées dans une base de données. Voici les extraits de deux tables utilisées.

Agences					
Agence	Ville	CP	Departement	Adresse	Telephone
Licorne	Paris	75001	Paris	123 Rue de la Révolution	0123456789
Le Carosse	Paris	75001	Paris	456 Virtual Street	0156789012
Vroum	Lyon	69001	Rhône	789 Virtual Lane	0456789034
Rapide	Toulouse	31000	Haute Garonne	321 Virtual Avenue	0567890123
Deep Place	Bordeaux	33000	Gironde	987 Virtual Road	0567890145

Voitures						
id_voiture	marque	modele	kilometrage	nombre_places	type	carburant
1	Renault	Clio	64022	5	Berline	Essence
2	Renault	Clio	50350	5	Berline	Essence
3	Dacia	Sandero	62031	5	Berline	Essence
4	Dacia	Sandero	58955	5	Berline	Essence
5	Dacia	Sandero	65779	5	Berline	Essence
6	Dacia	Sandero	56253	5	Berline	Essence
7	Renault	Clio	49660	5	Berline	Essence
8	Fiat	500	2545	4	Citadine	Electrique
9	Fiat	500	1953	4	Citadine	Electrique
10	Fiat	500	549	4	Citadine	Electrique

1. Donner pour la table `Agences` un type possible pour l'attribut `CP` qui indique le code postal.

La fonction `COUNT` permet de compter le nombre d'enregistrements et le mot-clé `DISTINCT` permet de ne pas prendre en compte les doublons.

2. Donner le résultat de la requête suivante pour les extraits des tables données :

```
SELECT COUNT(DISTINCT 'Telephone')
FROM 'Agences';
```

3. Expliquer à quelle condition l'attribut `Telephone` pourrait servir de clé primaire pour la table `Agences`.

Certaines agences ont décidé de partager un même standard téléphonique. On ajoute à la table `Agences` l'attribut `id_agence` qui est utilisé comme clé primaire. Voici la nouvelle table obtenue :

Agences						
id_agence	Agence	Ville	CP	Departement	Adresse	Telephone
1	Licorne	Paris	75001	Paris	123 Rue de la Révolution	0123456789
2	Le Carosse	Paris	75001	Paris	456 Virtual Street	0123456789
3	Vroum	Lyon	69001	Rhône	789 Virtual Lane	0456789034
4	Rapide	Toulouse	31000	Haute Garonne	321 Virtual Avenue	0567890123
5	Deep Place	Bordeaux	33000	Gironde	987 Virtual Road	0567890145

Nous souhaitons créer une table `couple_voitures_agences` qui permettra d'associer la table `Agences` à la table `Voitures`. On précise que la table `Voitures` a pour clé primaire `id_voiture`.

4. Décrire les attributs de cette nouvelle table ; une clé primaire sera soulignée et une clé étrangère commencera par un dièse (#).

5. Écrire une requête qui permet d'enregistrer dans la table `couple_voitures_agences` le fait que le véhicule `2` est associé à l'agence `Deep Place`.

6. Écrire une requête qui permet d'actualiser la table pour indiquer que le véhicule `2` se trouve maintenant à l'agence `Le Carrosse`.

7. Écrire une requête qui permet d'afficher le type, la marque et le nom de l'agence pour l'ensemble des véhicules.

Partie B

Pour cette seconde partie, on admet que pour chacune des trois tables, la clé primaire est auto-incrémentée. On utilise par la suite la programmation en Python.

On dispose de la fonction `execute_requete_insert` :

```
 1  def execute_requete_insert(requete):
 2      """
 3      Exécute la requête d'insertion passée en paramètre.
 4      Paramètre :
 5      - requete : STR, la chaîne formatée de la requête
 6      Résultat :
 7      - BOOL, booléen de contrôle : Vrai si la requête s'est bien
passée, Faux sinon
 8      """
       ...
```

Exemple d'utilisation :

```
>>> execute_requete_insert('INSERT INTO couple_voitures_agences (id,
id_agence, id_voiture) VALUES (9, 3, 3)')
>>> True
```

On souhaite disposer d'une fonction Python qui insère un nouveau véhicule dans une agence. Cette fonction prend en paramètres les valeurs des attributs de la voiture à insérer et l'entier `id_agence`. Elle renvoie un booléen de contrôle.

```
   def insert_voiture(liste_valeurs, id_agence):
       ...
```

Exemple d'utilisation pour une Fiat dans l'agence Vroum :

```
>>> insert_voiture((‘Fiat’, ’500’, 18, 4, ’Citadine’,
‘Electrique’),3)

>>> True
```

8. Écrire en Python la fonction `insert_voiture`.

9. Préciser les conditions que doivent vérifier les paramètres transmis à cette fonction.

EXERCICE 2 (6 points)

Cet exercice porte sur les réseaux, les protocoles de routage et les graphes.

Partie A

Le réseau informatique d'une société est constitué d'un ensemble de routeurs interconnectés à l'aide de fibres optiques.

La figure ci-dessous représente le schéma de ce réseau. Il est composé de deux réseaux locaux L1 et L2. Le réseau local L1 est relié au routeur R1 et le réseau local L2 est relié au routeur R9.

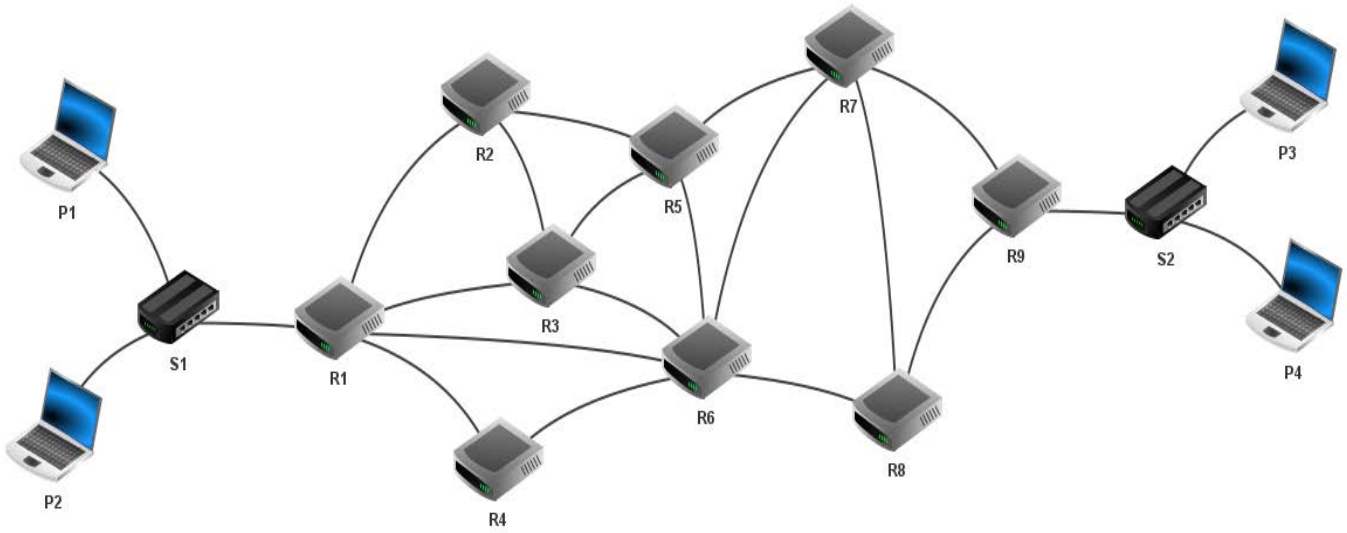


Figure 1. Réseau

Dans cette partie, les adresses IP sont composées de 4 octets, soit 32 bits. Elles sont notées X1.X2.X3.X4, où X1, X2, X3 et X4 sont les valeurs des 4 octets, converties en notation décimale. La notation X1.X2.X3.X4/n signifie que les n premiers bits de poids forts de l'adresse IP représentent la partie « réseau », les bits suivants représentent la partie « hôte ».

Toutes les adresses des machines connectées à un réseau local ont la même partie réseau.

Le tableau suivant indique les adresses IPv4 des machines constituant le réseau de la société.

NOM	TYPE	ADRESSE IPV4
R1	Routeur	Interface 1 :192.168.1.1/24 Interface 2 :192.168.2.1/24 Interface 3 :192.168.3.1/24 Interface 4 :192.168.4.1/24 Interface 5 :192.168.5.1/24

NOM	TYPE	ADRESSE IPV4
R2	Routeur	Interface 1 :192.168.2.2/24 Interface 2 :192.168.7.1/24 Interface 3 :192.168.8.1/24
R3	Routeur	Interface 1 :192.168.3.2/24 Interface 2 :192.168.7.2/24 Interface 3 :192.168.9.1/24 Interface 4 :192.168.10.1/24
R4	Routeur	Interface 1 :192.168.5.2/24 Interface 2 :192.168.6.1/24
R5	Routeur	Interface 1 :192.168.8.2/24 Interface 2 :192.168.9.2/24 Interface 3 :192.168.11.1/24 Interface 4 :192.168.12.1/24
R6	Routeur	Interface 1 :192.168.4.2/24 Interface 2 :192.168.6.2/24 Interface 3 :192.168.10.2/24 Interface 4 :192.168.11.2/24 Interface 5 :192.168.13.1/24 Interface 6 :192.168.14.1/24
R7	Routeur	Interface 1 :192.168.12.2/24 Interface 2 :192.168.13.2/24 Interface 3 :192.168.15.1/24 Interface 4 :192.168.16.1/24
R8	Routeur	Interface 1 :192.168.14.2/24 Interface 2 :192.168.15.2/24 Interface 3 :192.168.17.1/24
R9	Routeur	Interface 1 :192.168.16.2/24 Interface 2 :192.168.17.2/24 Interface 3 :192.168.18.1/24
P1	Portable	192.168.1.10
P2	Portable	Non fourni
P3	Portable	Non fourni
P4	Portable	Non fourni

1. En utilisant les adresses IP des différentes interfaces et des ordinateurs portables, en déduire une adresse possible pour le portable P2.

2. Donner l'adresse du réseau local L2 ainsi que le nombre d'adresses possibles pour les ordinateurs portables P3 et P4.

Partie B

Le graphe G, représenté ci-dessous, schématise l'architecture du réseau de la société. Les sommets représentent les routeurs et les arêtes représentent les liaisons.

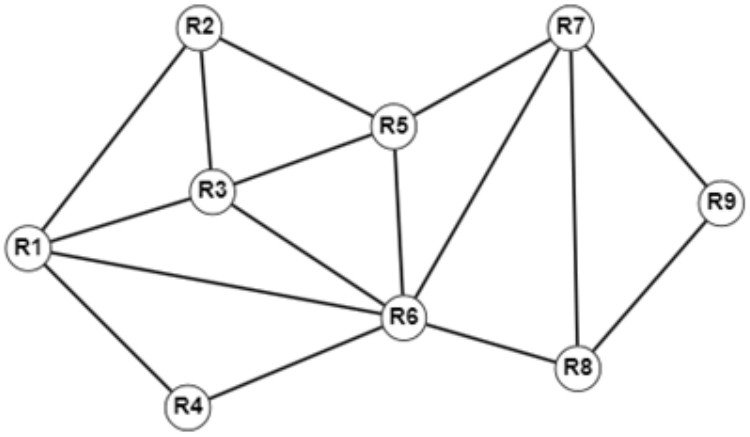


Figure 2. Graphe non pondéré

3. Donner l'implémentation Python des listes d'adjacence de ce graphe à l'aide d'un dictionnaire dont les clés sont les sommets et les valeurs la liste des sommets adjacents du sommet clé. On nomme G ce dictionnaire.

   Afin de faciliter la notation, on s'autorise à écrire chaque couple clé/valeur sur une nouvelle ligne.

On suppose que le protocole de routage RIP est utilisé.

4. Recopier et compléter, en rajoutant autant de lignes que nécessaire, la table de routage simplifiée suivante du routeur R1.

Destination	Suivant	Nombre de sauts
R2	R2	1
R3		

L'ordinateur P1 envoie un paquet de données à l'ordinateur P3.

5. Donner l'un des chemins empruntés par le paquet ainsi que le nombre de sauts.

La société doit vérifier l'état physique de la fibre optique installée sur le réseau. Un robot inspecte toute la longueur de la fibre optique afin de s'assurer qu'elle ne présente pas de détérioration apparente.

On appelle M la matrice d'adjacence du graphe de la figure 2. Les sommets sont rangés par ordre croissant des numéros des routeurs (R1, R2, …, R9).

6. Donner l'écriture en Python de cette matrice d'adjacence sous la forme d'une liste de listes.

Le degré d'un sommet est le nombre d'arêtes dont ce sommet est une extrémité.

7. Recopier et compléter les lignes 3, 5 et 6 de le fonction `degre` qui prend en paramètre la matrice d'adjacence d'un graphe donné sous forme d'une liste de listes et qui renvoie la liste des degrés de tous les sommets du graphe rangés dans le même ordre que les sommets de la matrice d'adjacence.

```
1 def degre(MATRICE):
2     d = []
3     for ... in ...:
4         cpt = 0
5         for ... in ...:
6             cpt = cpt +...
7         d.append(cpt)
8     return d
```

8. Donner la liste renvoyée par `degre(M)`.

On appelle chaîne eulérienne d'un graphe non orienté un chemin qui passe une et une seule fois par toutes les arêtes du graphe. Un graphe connexe admet une chaîne eulérienne si et seulement si le graphe possède, au plus, deux sommets de degré impair.

9. En utilisant le résultat de la question précédente et en admettant que le graphe est connexe, indiquer si le robot peut parcourir l'ensemble du réseau en suivant les fibres optiques et en empruntant chaque fibre optique une et une seule fois.

Partie C

Le poids sur chaque arête représente la bande passante en megabits par seconde (Mb/s) de chaque liaison.

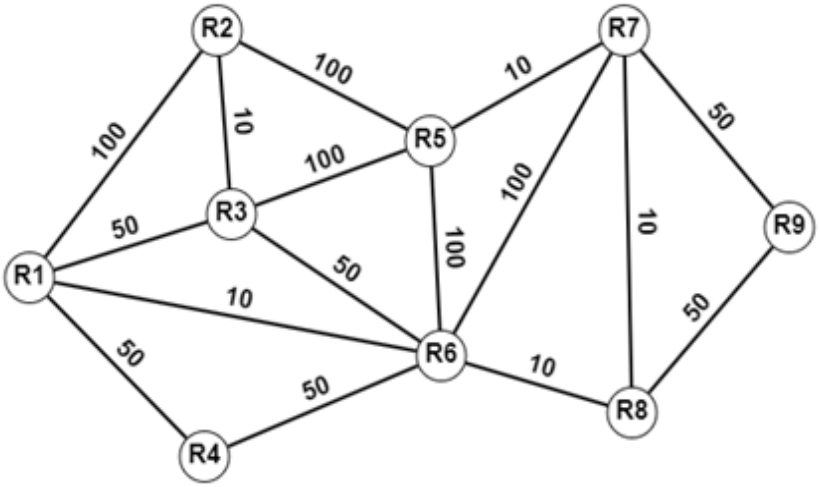


Figure 3. Graphe pondéré

Dans cette partie, on utilise le protocole de routage OSPF. Pour calculer le coût d'une liaison, on utilise la formule :

$$C = \frac{10^8}{BP}$$

où BP est la bande passante en bits par seconde.

10. Déterminer la route qui sera empruntée par le paquet pour aller de l'ordinateur P1 à l'ordinateur P3. Préciser le coût de ce trajet.

EXERCICE 3 (8 points)

Cet exercice porte sur les bases de numération, la structure de données PILE et la POO.

La civilisation *Maya* est une ancienne civilisation de Mésoamérique principalement connue pour ses avancées dans les domaines de l'écriture, de l'art, de l'architecture, de l'agriculture, des mathématiques et de l'astronomie.

La numération *Maya* est une numération positionnelle de base 20 (dite vigésimale) utilisant trois symboles pour former les “chiffres” :

- une coquille pour le zéro  ,
- un point pour l’unité ●,
- un trait pour la valeur 5 ▬.

Les “chiffres” sont les suivants. Ils utilisent une numération additive :

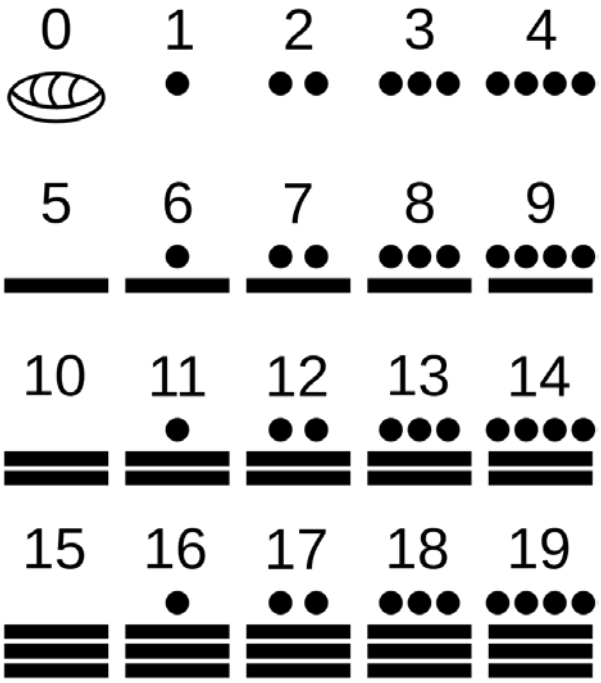
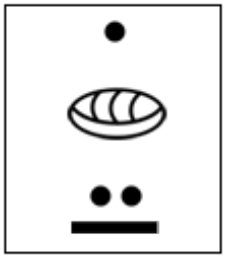


Figure 1. Table des “chiffres” et valeurs correspondantes (source : Wikipédia)

Dans une version simplifiée de ce système, l’écriture d’un nombre se fait par empilement de “chiffres”. Chaque étage correspond à un chiffre de poids 20 fois supérieur au poids du chiffre de l’étage inférieur.

Ainsi la valeur du chiffre de l’étage le plus bas est multipliée par 20^0 soit 1, du second étage par 20^1, du troisième étage par 20^2, et ainsi de suite.

Exemple : la représentation *Maya* de l'entier 407 est la suivante.



1. Compléter le tableau donné en **annexe à rendre avec la copie**.

2. Justifier que l'écriture *Maya* de l'entier 3435 est :

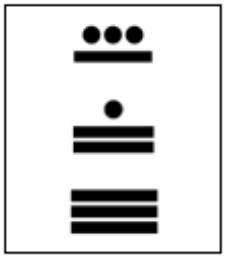


Figure 2. Ecriture *Maya* de l'entier 3435

On **modélise l'écriture d'un entier** dans sa représentation *Maya* par une pile formée de listes. Chacune de ces listes est composée de trois entiers et modélise le "chiffre" d'un étage :

- le premier entier vaut 0 ou 1 suivant s'il y a ou non une coquille ;
- le deuxième représente le nombre de points ;
- le troisième représente le nombre de traits.

Ainsi, la modélisation *Maya* de l'entier 3435 est `[[0, 0, 3], [0, 1, 2], [0, 3, 1]]` et celle de l'entier 407 est `[[0, 2, 1], [1, 0, 0], [0, 1, 0]]`.

Le sommet de la pile se situe en fin de liste.

On dispose de la classe suivante :

```
class Maya:
    def __init__(self):
        self.nombre = []

    def ajouter(self, chiffre):
        """ chiffre est une liste de longueur 3.
        La méthode empile le chiffre au sommet de la pile """
        self.nombre.append(chiffre)

    def retirer(self):
        """ depile et renvoie le chiffre qui etait au sommet de

          la pile """
        if not self.estVide():
            return self.nombre.pop()

    def estVide(self):
        return self.nombre == []

    def nbEtages(self) :
        """ renvoie le nombre de chiffres de la pile """
        ...

    def MayaToDec(self):
        """ renvoie le nombre entier correspondant a la
        modelisation Maya de l'instance courante """
        coeff = 20**...
        ch_Dec = 0
        while ... :
            ch_Maya = ...
            ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
            coeff = ...
        return ch_Dec

    def multiplie(self):
        """ renvoie le resultat de la multiplication par 20 d'un
        nombre en modelisation Maya. """
        ...

    def somme(self, maya2):
        """ ajoute maya2 à l'instance courante et renvoie le
        resultat en modelisation Maya """
        if self.nbEtages() == maya2.nbEtages()
        ...
```

3. Écrire une suite d'instructions permettant de créer une instance, nommée `M`, de la classe `Maya` qui modélise le nombre entier 3435.

4. Écrire la méthode `nbEtages` de la classe `Maya`. Celle-ci renvoie le nombre de "chiffres" utilisés pour écrire le nombre correspondant en écriture *Maya*.

De l'écriture *Maya* à l'écriture décimale

5. Écrire une fonction `valeurChiffre` ayant pour paramètre une liste `L`. Celle-ci renvoie la valeur de l'entier associé à la liste `L = [c, p, t]` où `c` (de valeur 0 ou 1) indique la présence d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant un "chiffre" *Maya*.

Exemple :

```
>>> valeurChiffre([0, 2, 3])
>>> 17
>>> valeurChiffre([1, 0, 0])
>>> 0
```

6. Recopier et compléter les lignes 2, 4, 5 et 7 de la méthode `MayaToDec` suivante de la classe `Maya`. Cette méthode renvoie la valeur de l'entier associé à l'objet `Maya`. On pourra utiliser les méthodes `estVide`, `nbEtages` et `retirer`.

```
1 def MayaToDec(self):
2     coeff = 20**...
3     ch_Dec = 0
4     while ... :
5         ch_Maya = ...
6         ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
7         coeff = ...
8     return ch_Dec
```

De l'écriture décimale vers sa modélisation *Maya*

On considère que la fonction `DecToVige` est déjà écrite. Celle-ci prend en paramètre un entier `n` et renvoie la décomposition en base 20 de celui-ci sous la forme d'une liste $[\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_p]$ telle que :

$$n = \mathbf{a}_0 \times 20^0 + \mathbf{a}_1 \times 20^1 + \mathbf{a}_2 \times 20^2 + \ \ldots + \mathbf{a}_p \times 20^p$$

Exemple :

```
>>> DecToVige(3435)
[15, 11, 8]
>>> DecToVige(407)
[7, 0, 1]
```

7. Écrire la fonction `decompChiffre` qui prend en paramètre un entier `n` compris entre 0 et 19 et renvoie la liste `[c, p, t]` où `c` vaut 0 ou 1 et indique la présence ou non d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant le "chiffre" *Maya* correspondant.

Exemple :

```
>>>decompChiffre(17)
[0, 2, 3]
```

```
>>>decompChiffre(0)
[1, 0, 0]
```

8. Écrire la fonction `DecToMaya` qui prend en paramètre un entier `n` et renvoie la modélisation *Maya* d'un objet M de la classe `Maya` correspondant.

Exemple :

```
>>>DecToMaya(3435).nombre
[[0, 0, 3], [0, 1, 2], [0, 3, 1]]
>>>DecToMaya(407).nombre
[[0, 2, 1], [1, 0, 0], [0, 1, 0]]
```

Opérations sur les nombres en modélisation *Maya*

On souhaite additionner des nombres directement à partir de leur modélisation *Maya*.

9. Écrire la méthode `multiplie` de la classe `Maya` qui renvoie le résultat de la multiplication par 20 d'un nombre en modélisation *Maya*.

Exemple :

```
>>> M = Maya()
>>> M.ajouter([0 ,0, 3])
>>> M.ajouter([0, 1, 2])
>>> M.multiplie().nombre
[[1, 0, 0], [0, 0, 3], [0, 1, 2]]
```

On donne la fonction `mystere` suivante :

```
1 def mystere(m1, m2, ret):
2     c = 0
3     p = (m1[1] + m2[1] + ret)%5
4         if m1[1] + m2[1] + ret >= 5:
5             ret = 1
6         else:
7             ret = 0
8     t = (m1[2] + m2[2] + ret)%4
9     if m1[2] + m2[2] + ret < 4:
10        ret = 0
11    else:
12        ret = 1
13    if (m1[0] == 0 and m2[0] == 1) or (p + t = 0 and ret == 1):
14        c = 1
15    return ([c, p, t], ret)
```

10. Donner les résultats renvoyés par les deux appels suivants :

- `mystere([0, 1, 1], [0, 3, 1], 0)`
- `mystere([0, 1, 1], [0, 4, 2], 0)`

11. Écrire une méthode `somme` de la classe `Maya` permettant d'ajouter à l'instance courante un autre nombre `maya2` de même taille en modélisation Maya.

ANNEXE À RENDRE AVEC LA COPIE

Exercice 3 – Question 1.

Étage	Écriture *Maya*	Valeur du “chiffre” de l’étage	Valeur dans la conversion
3		$1 \times 5 + 3 \times 1 = 8$	$8 \times 20^2 = 3200$
2			
1			

Modèle CCYC : ©DNE

NOM DE FAMILLE (naissance) **:**
(en majuscules)

PRENOM :
(en majuscules)

N° candidat :

N° d'inscription :

(Les numéros figurent sur la convocation, si besoin demander à un surveillant.)

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Né(e) le : ⬜⬜ / ⬜⬜ / ⬜⬜⬜⬜

1.2