BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 1

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 11 pages numérotées de 1 / 11 à 11 / 11.

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, la programmation dynamique, les graphes et les réseaux.

On cherche à lutter contre un virus informatique qui essaie de contourner les protocoles de sécurité en migrant régulièrement vers un autre ordinateur, en choisissant à chaque fois au hasard sa nouvelle cible parmi les ordinateurs accessibles. On cherche à savoir quels ordinateurs protéger afin de lutter de manière la plus efficace possible avec des ressources limitées.

On considère le réseau informatique suivant, composé de 5 ordinateurs numérotés : 0, 1, 2, 3 et 4.

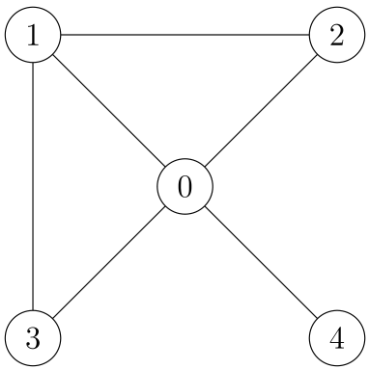


Figure 1. Réseau informatique

1. On représente ce réseau informatique par un graphe que l'on stocke sous forme de listes de voisins. Compléter la définition de la variable `voisins`.

```
voisins = [[1, 2, 3, 4],
           [0, 2, 3],
           [0, 1],
           [...],
           [...]]
```

On ajoute au réseau actuel un sixième ordinateur, numéroté 5. Cet ordinateur n'est accessible que des ordinateurs numérotés 0 et 2.

2. Dessiner le nouveau graphe.

3. Donner la nouvelle définition de la variable `voisins`.

4. Compléter la fonction `voisin_alea` qui prend en paramètre un graphe `voisins` sous forme de listes de voisins et un entier `s` représentant un sommet et qui renvoie un entier représentant un voisin de `s` choisi aléatoirement. On pourra utiliser la fonction `random.randrange(n)` qui renvoie un nombre aléatoire entre `0` inclus et `n` exclus.

```
def voisin_alea(voisins, s):
    ...
```

On donne la fonction `marche_alea` suivante :

```
1 def marche_alea(voisins, i, n):
2     if n == 0:
3         return i
4     return marche_alea(voisins, voisin_alea(voisins, i), n-1)
```

5. Justifier que la fonction `marche_alea` est une fonction récursive.

6. Décrire ce que modélise cette fonction, en rapport avec le contexte de l'exercice.

7. Compléter la fonction `simule` qui simule `n_tests` fois le déplacement d'un virus pendant `n_pas` étapes, démarrant au sommet `i`, et qui renvoie une liste contenant en position `j` le nombre de fois que le virus a terminé son parcours au sommet `j`, divisé par `n_tests`.

```
1 def simule(voisins, i, n_tests, n_pas):
2     results = [0] * len(voisins)
3     ...
```

8. L'appel `simule(voisins, 4, 1000, 1000)` renvoie la valeur suivante : `[0.328, 0.195, 0.18, 0.12, 0.059, 0.118]`. Déduire de ce résultat l'ordinateur du réseau qu'il est le plus rentable de protéger.

Au début, on suppose que le virus n'est présent que sur un ordinateur. À chaque étape, il contamine tous ses voisins non déjà contaminés. On cherche à savoir combien de temps prend ce virus pour se propager à tout le réseau.

9. Un graphe `voisins` représente un réseau, et `s` représente un sommet de départ. Proposer un algorithme pour déterminer le temps, en étape, que met un virus à se propager dans l'intégralité d'un réseau.

EXERCICE 2 (6 points)

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

Rappels :

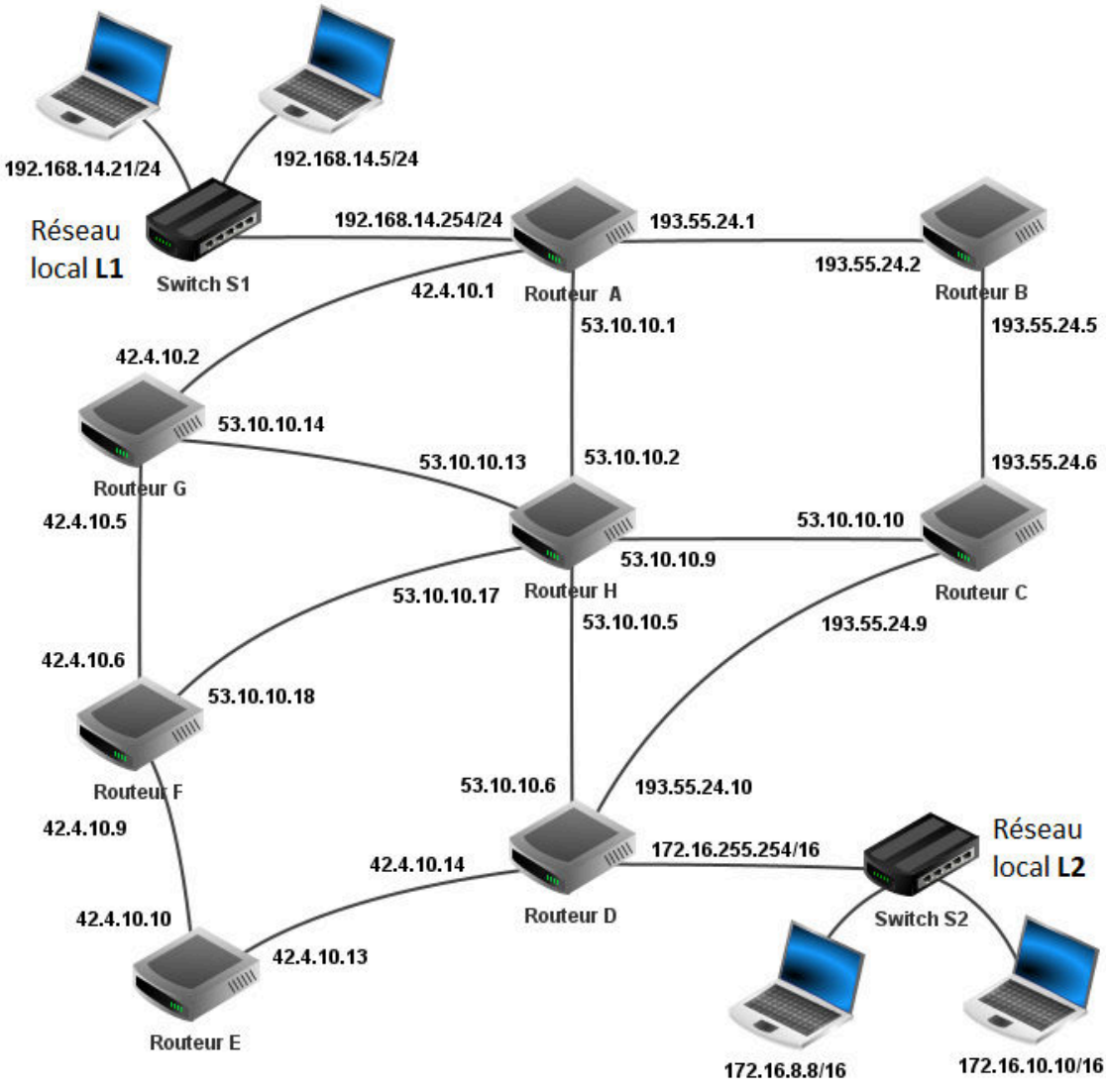
Une adresse IPv4 est composée de 4 octets, soit 32 bits. Elle est notée a.b.c.d, où a, b, c et d sont les valeurs décimales des 4 octets et nommée « notation décimale pointée ».

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

L'adresse IPv4 dont tous les bits de la partie « machine » sont à 0 est appelée « adresse du réseau ».

L'adresse IPv4 dont tous les bits de la partie « machine » sont à 1 est appelée « adresse de diffusion ».

On considère le réseau représenté ci-dessous :



Partie A : Adresses IP

1. Les machines du réseau local L1 indiquent un masque de sous réseau sur 24 bits en notation CIDR, soit 255.255.255.0 en notation décimale pointée.

   Donner le masque de sous réseau en notation décimale pointée des machines du réseau L2 (masque de sous réseau de 16 bits).

Concernant le réseau local L2 :

2. Donner l'adresse du réseau.

3. Donner l'adresse de diffusion.

4. Donner le nombre maximum de machines pouvant être connectées à ce réseau.

Partie B : Protocoles de routage

On donne ci-dessous des extraits des tables de routage des routeurs :

Routeur	Réseau destinataire	Passerelle	Interface
A	L2	53.10.10.2	53.10.10.1
B	L2	193.55.24.6	193.55.24.5
C	L2	193.55.24.10	193.55.24.9
D	L2	Connecté	172.16.255.254
E	L2	42.4.10.14	42.4.10.13
F	L2	42.4.10.10	42.4.10.9
G	L2	53.10.10.13	53.10.10.14
H	L2	53.10.10.6	53.10.10.5

5. À l'aide des extraits des tables de routage ci-dessus, donner un chemin (c'est-à-dire nommer les routeurs traversés) suivi par un message envoyé du réseau L1 vers le réseau L2.

La liaison entre les routeurs H et D est rompue :

6. Sachant que le protocole de routage RIP est utilisé (distance en nombre de sauts), donner les nouveaux chemins que pourra suivre un message allant de L1 vers L2.

7. Choisir un des chemins de la question précédente.

   Donner les routeurs dont la règle de routage à destination de L2 est obligatoirement modifiée. Après avoir examiné tous les routeurs, écrire sur votre copie les règles de routage modifiées en conséquence.

La liaison entre les routeurs H et D est rétablie.

Pour tenir compte du débit des liaisons, on décide d'utiliser le protocole OSPF (distance liée au coût des liaisons) pour effectuer le routage.

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

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

Les valeurs des bandes passantes de chaque liaison entre les routeurs sont données ci-dessous :

Liaison	**Bande passante**
A-B	1 Gbit/s
A-H	1 Gbit/s
A-G	1 Gbit/s
B-C	1 Gbit/s
C-H	100 Mbit/s
C-D	1 Gbit/s

Liaison	**Bande passante**
D-H	100 Mbit/s
D-E	10 Gbit/s
E-F	10 Gbit/s
F-H	1 Gbit/s
F-G	10 Gbit/s
G-H	1 Gbit/s

8. Calculer le coût des liaisons pour les 3 valeurs de bande passante qui apparaissent dans le tableau ci-dessus.

9. Déterminer alors le chemin que suivra un message allant de L1 vers L2 et donner son coût.

10. La liaison entre les routeurs G et F est rompue. Déterminer le nouveau chemin suivi par un message allant de L1 vers L2 et donner son coût.

EXERCICE 3 (8 points)

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

L'objectif est de faciliter la gestion du système d'information d'un camping municipal. Les informations nécessaires sont stockées dans une base de données relationnelle composée de trois relations. On pourra utiliser les mots-clés SQL suivants : `AND`, `FROM`, `INSERT`, `INTO`, `JOIN`, `ON`, `SELECT`, `SET`, `UPDATE`, `VALUES`, `WHERE`.

Voici le schéma des deux premières relations :

`Client (` <u>`id client`</u> `, nom, prenom, adresse, ville, pays, telephone)`

`Reservation (` <u>`id reservation`</u> `, #id_client, #id_emplacement, nombre_personne, date_arrivee, date_depart)`

Dans ce schéma :

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

La troisième relation est appelée `Emplacement` et elle contient tous les emplacements du camping. Le tableau ci-dessous en donne un extrait.

Emplacement			
id_emplacement	nom	localisation	tarif_journalier
1	myrtille	A4	25
2	mirabelle	D1	35
3	mangue	B2	29.90
4	mandarine	B1	25
5	mûre	C3	29.90
6	melon	A2	25

Partie A

1. Citer deux avantages à utiliser une base de données relationnelle plutôt qu'un fichier texte ou un fichier tableur.
2. Quelle doit être la caractéristique d'un attribut pour pouvoir être utilisé en tant que clé primaire ?
3. Dans la relation `Reservation`, quel est le rôle des clés étrangères `id_client` et `id_emplacement` ?

4. Donner le schéma relationnel de la relation `Emplacement` en précisant la clé primaire et le type de chacun des attributs.
5. À partir de l'extrait du contenu de la relation `Emplacement` donner le résultat de la requête ci-dessous :

```
SELECT id_emplacement, nom, localisation
FROM Emplacement
WHERE tarif_journalier = 25;
```

6. Écrire une requête permettant de donner le `nom` et le `prenom` de tous les clients habitant à 'Strasbourg'.
7. Écrire une requête permettant d'ajouter un nouveau client :
   - id_client 42 ;
   - nom 'CODD' ;
   - prénom 'Edgar' ;
   - adresse '28 rue des Capucines' ;
   - ville 'Lyon' ;
   - pays 'France' ;
   - numéro de téléphone '0555555555'.
8. Écrire une requête SQL permettant de récupérer les informations ci-dessous concernant la réservation dont l'identifiant `id_reservation` est 18 :
   - `Client.nom`
   - `Client.prenom`
   - `Reservation.nombre_personne`
   - `Reservation.date_arrivee`
   - `Reservation.date_depart`
   - `Emplacement.tarif_journalier`

Partie B

Dans cette partie, on souhaite éditer une facture correspondant au séjour d'un client. Pour cela, on dispose d'une fonction Python qui récupère auprès de la base de données, à la manière de la question 8, les informations concernant la réservation voulue et renvoie le résultat sous forme d'un tuple contenant trois objets respectivement des classes `Client`, `Reservation` et `Emplacement`.

```
1 from datetime import datetime
2 class Client:
3     def __init__(self, nom, prenom, adresse, ville, pays,
                   telephone):
4         self.nom = nom
5         self.prenom = prenom
6         self.adresse = adresse
7         self.ville = ville
```

```
 8         self.pays = pays
 9         self.telephone = telephone
10
11 class Reservation:
12     def __init__(self, id_reservation, nombre_personne,
                    date_arrivee, date_depart):
13         self.id_reservation = id_reservation
14         self.nombre_personne = nombre_personne
15         self.date_arrivee = date_arrivee
16         self.date_depart = date_depart
17     def nb_jours(self):
18         """ renvoie, à l'aide de l'attribut days de la classe
19             timedelta, un entier correspondant
20             au nombre de jours passés au camping."""
21         return (self.date_depart - self.date_arrivee).days
22
23 class Emplacement:
24     def __init__(self, nom, tarif_journalier):
25         self.nom = nom
26         self.tarif_journalier = tarif_journalier
```

9. Expliquer pourquoi le terme `self` est utilisé comme paramètre pour les méthodes des classes `Client`, `Reservation` et `Emplacement`.
10. Instancier une variable `client01` de la classe `Client` représentant un client se nommant CODD Edgar habitant au 28 rue des Capucines à Lyon, France, ayant pour numéro de téléphone le 0555555555.

On considère un tuple constitué de trois objets, respectivement dans cet ordre, des classes `Client`, `Reservation` et `Emplacement`. On souhaite écrire une fonction qui renvoie le montant dû par ce client pour cet emplacement et pour cette durée de séjour. Sachant qu'au tarif journalier de location de l'emplacement il faut ajouter une taxe de séjour de 2,20 € par jour et par personne.

Exemple de calcul du montant à régler pour un client ayant réservé pour 4 personnes pendant 12 jours un emplacement à 30 € la journée :

```
>>> 30 * 12 + 4 * 2.20 * 12
465.6
```

11. Compléter la ligne 5 de la fonction `montant_a_regler`.

```
1 def montant_a_regler(triplet):
2     """ renvoie le montant en euros
3         à régler pour cette réservation """
4     client, reservation, emplacement = triplet
5     return ..............................
```

Chaque facture doit posséder ce que l'on appelle communément un numéro de facture unique. En réalité il s'agit d'une chaîne de caractères. Pour ses factures, depuis 2018,

le camping a adopté le format `'AAAA-MMM-xxx'` composé des trois chaînes de caractères ci-dessous :

- `'AAAA'` une année comprise entre 2018 et 2024 ;
- `'MMM'` les trois premières lettres du mois en anglais ;
- `'xxx'` désigne trois chiffres.

On décide d'écrire une fonction `facture_est_valide` pour tester si une chaîne de caractères représente un numéro de facture valide ou non. Voici quelques exemples du comportement attendu de la fonction `facture_est_valide`.

```
>>> facture_est_valide('2024-MAY-230')
True
>>> facture_est_valide('2012-MAY-230')
False
>>> facture_est_valide('2024-MAI-230')
False
>>> facture_est_valide('2024-JUN-23')
False
```

On considère le programme suivant :

```
1  calendrier = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN',
                 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC']
2
3  def separe(chaine):
4      """renvoie une liste constituée de chaînes qui étaient
5         séparées par le caractère - """
6      return chaine.split('-')
7
8  def que_des_chiffres(chaine):
9      """renvoie vrai si chaine n'est constituée que
10         des caractères de 0 à 9 faux sinon"""
11        for car in chaine:
12            if not(car in "012345789"):
13               return False
14        return True
15
16 def facture_est_valide(chaine):
17     """renvoie vrai si chaine est une chaîne de
18        caractères conforme au modèle de facture"""
19     partie = separe(chaine)
20     if not(len(partie) == 3):
21        return False
22     annee, mois, numero = partie[0], partie[1], partie[2]
23     if not(que_des_chiffres(annee)):
24         return False
25     if not(len(annee) == 4) or not(2018 <= annee <= 2024):
26         return False
27     # Reste à faire vérifier les mois MMM
28     ...
29     # Reste à faire vérifier le numéro xxx
```

```
30      ...
31      return True
```

On rappelle que la fonction `split` en Python divise une chaîne de caractères en une liste de sous-chaînes en fonction d'un séparateur spécifié.

Par exemple;

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

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

12. Expliquer pourquoi une erreur se produit à l'exécution de la fonction `facture_est_valide` donnée ci-dessus.
13. Proposer une correction du code pour que cette erreur ne se produise plus.
14. Compléter le code afin de vérifier les mois (ligne 28) et le numéro (ligne 30) dans la fonction `facture_est_valide`. Pour chaque vérification, il est possible d'insérer une ou plusieurs lignes.