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 les graphes et les protocoles réseau.

Les deux parties sont indépendantes.

Partie A : Réseau dans un lycée

Dans un lycée, le réseau contient plusieurs sous-réseaux : pédagogie (noté P), administration (noté AD), vie_scolaire (noté VS).

1. L'adresse IP du réseau pédagogie est 110.217.50.0 et on utilise le masque de sous-réseau 255.255.255.0 (i.e. les trois premiers octets sont réservés au réseau).

   Déterminer le nombre de machines que l'on peut brancher au maximum sur le réseau pédagogie (remarque : l'adresse IP 110.217.50.255 est réservée : c'est l'adresse de diffusion).

2. Déterminer l'écriture binaire du nombre 217.

3. Déterminer l'écriture décimale du nombre binaire 110010.

On scinde finalement le réseau pédagogie en deux sous-réseaux pédagogie 1 (noté P1) et pédagogie 2 (noté P2) et on utilise le masque de sous-réseau 255.255.255.0 pour les deux.

Les adresses IP de ces deux sous-réseaux sont :

- 110.217.50.0 pour le réseau pédagogie 1
- 110.217.52.0 pour le réseau pédagogie 2

4. Indiquer, en justifiant, si une machine ayant l'adresse IP 110.217.53.22 fait partie du réseau pédagogie 2 ou non.

L'adresse IP du réseau administrateur est 110.217.54.0. Celle du réseau vie_scolaire est 110.217.56.0.

Les sous-réseaux sont connectés entre eux par trois routeurs R1, R2 et R3 de la façon suivante :

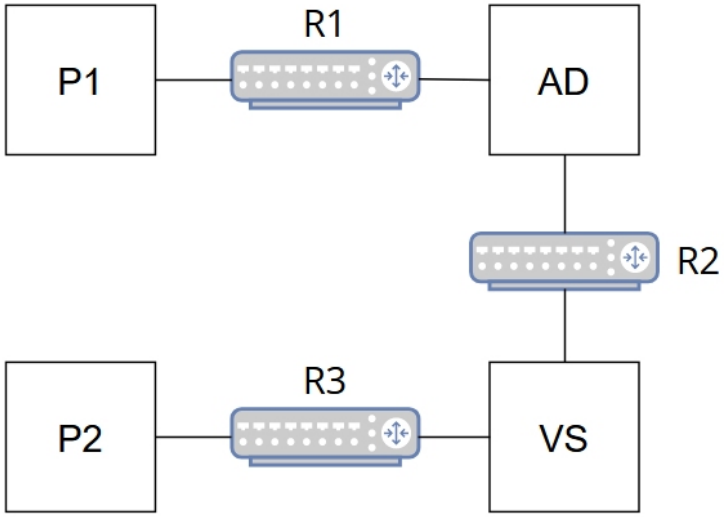

Figure 1. Plan du réseau.

Les adresses IP des routeurs dans chacun des réseaux sont données dans le tableau suivant :

Routeur	adresse dans P1	adresse dans P2	adresse dans AD	adresse dans VS
R1	110.217.50.254		110.217.54.254	
R2			110.217.54.253	110.217.56.254
R3		110.217.52.254		110.217.56.253

5. Recopier et compléter la table de routage du routeur R1 suivante en indiquant les adresses IP des passerelles et des interfaces :

Destination	Passerelle	Interface
110.217.50.0	on-link	110.217.50.254
110.217.52.0	110.217.54.253	
110.217.54.0		
110.217.56.0		

Précision : Si un réseau est directement relié à un routeur, l'adresse IP de la passerelle est remplacée par les mots "on-link".

On ajoute une liaison entre les réseaux pédagogie 1 et pédagogie 2 via un routeur R4 dont les adresses IP sont :

- 110.217.50.253 dans pédagogie 1
- 110.217.52.253 dans pédagogie 2

Le réseau devient alors :

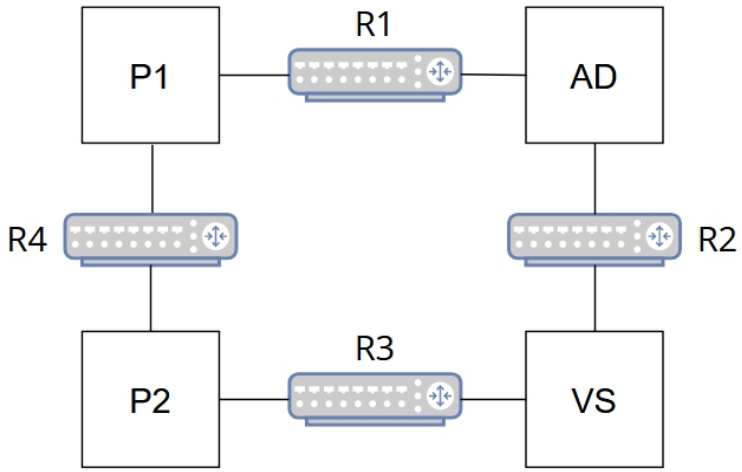

Figure 2. Plan du réseau mis à jour.

On rappelle que le protocole de routage RIP minimise le nombre de routeurs traversés.

6. Indiquer les modifications à apporter à la table de routage de R1, en respectant le protocole RIP sur l'ensemble du réseau du lycée.

7. Indiquer si l'ajout du routeur R4 entraîne des modifications dans la table de routage du routeur R2, toujours en respectant le protocole RIP. Justifier la réponse.

Partie B : Réseaux et graphes

Dans un réseau dont la structure est connue, on veut pouvoir déterminer si deux routeurs peuvent communiquer entre eux, c'est-à-dire s'il existe entre eux une route constituée de routeurs reliés par des sous-réseaux. On représente ce réseau de routeurs par un graphe G dont les routeurs sont les sommets et les sous-réseaux reliant les routeurs sont les arêtes.

On propose la fonction de recherche d'un chemin entre deux sommets R1 et R2 suivante :

```
1 def recherche(R1,R2):
2     if R1 == R2 :
3         return True
4     for S in adjacents(R1,G) :
5          if recherche(S,R2) :
6              return True
7     return False
```

où `adjacents(R1,G)` renvoie la liste des sommets adjacents à R1 dans le graphe G, classés dans l'ordre alphabétique de leur nom.

8. Indiquer ce qui se passera si on utilise cette fonction de recherche entre deux sommets non reliés par un chemin dans le graphe.

9. Proposer une solution pour résoudre ce problème.

EXERCICE 2 (6 points)

Cet exercice porte sur la récursivité.

Au rugby, une équipe peut marquer, pour simplifier :

- soit 3 points (pénalité) ;
- soit 5 points (essai non transformé) ;
- soit 7 points (essai transformé).

Partie A

On souhaite savoir s'il est possible d'obtenir un score donné avec uniquement des pénalités, c'est-à-dire avec une succession de "coups" à 3 points.

1. Écrire une fonction `possible_avec_penalites_seules` qui prend en paramètre un score et qui renvoie `True` si le score passé en paramètre peut être marqué uniquement avec des pénalités.

Exemple:

```
>>> possible_avec_penalites_seules(15)
True
>>> possible_avec_penalites_seules(10)
False
```

2. Recopier et compléter le tableau suivant qui précise, pour les scores de 0 à 10, les évolutions du score menant à un total donné et le nombre de façons différentes d'obtenir ce total. Par exemple, pour obtenir un score de 8, en partant de 0, il y a 2 possibilités :

- Soit l'équipe marque un essai non transformé, atteignant 5 points, puis une pénalité, atteignant 8 points ;
- Soit l'équipe marque une pénalité, atteignant 3 points, puis un essai non transformé, atteignant 8 points.

score	liste des solutions	nombre de solutions
0	[0]	1
1	[]	0
2		
3		
4		
5		
6		
7		
8	[0, 5, 8], [0, 3, 8]	2
9		1
10	[0,3,10]	3

En notant $f(n)$ le nombre de possibilités d'obtenir le score n, on admet que pour $n > 6$ on a la relation suivante :

$$f(n) = f(n-3) + f(n-5) + f(n-7)$$

3. Vérifier cette relation pour $n = 10$ à l'aide du tableau établi à la question 2.

On veut écrire une fonction récursive `nb_solutions`, qui prend en paramètre un entier positif quelconque correspondant à un score, et qui renvoie le nombre de façons d'obtenir ce score donné.

4. Déterminer tous les cas de base, pour chaque entier n de 0 à 6, de cette fonction récursive.

5. Écrire la fonction récursive `nb_solutions`, qui prend en paramètre un entier positif quelconque correspondant à un score, et qui renvoie le nombre de possibilités d'obtenir ce score donné.

6. Lors de l'appel de `nb_solutions(score)`, on se rend compte que le nombre d'appels récursif augmente très rapidement lorsque `score` augmente. Nommer une méthode algorithmique optimisant le nombre d'appels récursifs.

On veut écrire une fonction récursive `solutions_possibles`, qui prend en paramètre un entier positif quelconque correspondant à un score, et qui renvoie la liste composée de toutes les listes représentant les possibilités d'obtenir ce score.

Par exemple :

```
>>> solutions_possibles(8)
[[0, 5, 8], [0, 3, 8]]
```

7. Déterminer quelles lignes du tableau permettent de construire rapidement la liste renvoyée par l'appel `solutions_possibles(11)`.

8. Recopier et compléter la fonction `solutions_possibles` suivante, qui prend en paramètre un score et qui renvoie la liste des possibilités d'obtenir ce score :

```
def solutions_possibles(score):
    if score < 0:
        resultat = []
    elif score == 0:
        resultat = ...
    else:
        resultat = []
        for coup in [..., 5, ...]:
            liste = solutions_possibles(score - coup)
            for solution in liste:
                solution.append(...)
            resultat.append(...)
    return resultat
```

EXERCICE 3 (8 points)

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

Le Tour de France est une course cycliste qui se déroule chaque année. Chaque jour, les coureurs s'affrontent pour remporter l'étape du jour, ce qui détermine un classement d'étape. Le coureur avec le temps cumulé le plus bas sur l'ensemble des étapes mène le classement général. Chaque participant est repéré par un dossard et appartient à une équipe. En 2023, 22 équipes de 8 coureurs, soit 176 cyclistes ont pris le départ du tour.

Partie A

Dans cette partie, nous allons utiliser trois dictionnaires.

Le premier, appelé `participants`, a pour clés les noms complets des coureurs et pour valeurs les numéros de dossard correspondants.

Le deuxième, appelé `temps_etapes`, utilise les numéros de dossard comme clés et contient une liste des temps d'arrivée de chaque étape en seconde.

Le troisième, appelé `classement_general`, utilise également les numéros de dossard comme clés et indique le classement général mis à jour à la fin de chaque nouvelle étape.

Par exemple, à la fin de la quatrième étape, voilà les trois premiers éléments de ces dictionnaires :

```
participants = {"VINGEGAARD Jonas": 1, "BENOOT Tiesj": 2, "KELDERMAN
Wilco": 3, ...}

temps_etapes = {1: [15781, 17199, 16995, 15928], 2: [15960, 17199,
16995, 15928], ...}

classement_general = {1: 6, 2: 30, 3: 13, ...}
```

1. En utilisant ces dictionnaires, écrire une instruction permettant d'obtenir :

- le numéro de dossard de `PHILIPSEN Jasper` ;
- le classement général de `PHILIPSEN Jasper` ;
- le temps, en seconde, mis par le cycliste `PINOT Thibaut` pour courir la quatrième étape.

2. Écrire une fonction `calcul_temps_total` qui a pour paramètre le numéro d'un dossard `d` et qui renvoie le temps total en seconde mis par ce coureur depuis le départ du tour de France.

3. Le dictionnaire `temps_etapes` étant remis à jour après la fin d'une étape, recopier et compléter les lignes 8, 9 et 14 du programme suivant afin que le dictionnaire `classement_general` soit aussi mis à jour.

```
1  classement = []
2
3  for numero_dossard in temps_etapes:
4      element = (numero_dossard,
                  calcul_temps_total(numero_dossard))
5      classement.append(element)
6      pos = len(classement) - 2
7
8      while pos >= 0 and element[...] < classement[pos][...]:
9          classement[pos + 1] = ...
10         pos = pos - 1
11         classement[pos + 1] = element
12
13 for i in range(len(classement)):
14     classement_general[...] = i + 1
```

On suppose qu'on dispose d'un tableau `tableau_temps` composé de tuples contenant le numéro du dossard, le nom, et le temps total en seconde de chaque coureur, trié par ordre croissant de temps. On donne ci-dessous un aperçu du début du tableau :

```
tableau_temps = [(1, "VINGEGAARD Jonas", 65903),
                 (3, "KELDERMAN Wilco", 65987),
                 (2, "BENOOT Tiesj", 66082),
                 ...]
```

On souhaite créer une variable Python `tableau_final` de type liste de listes :

```
[[1, "VINGEGAARD Jonas", 65903],
 [3, "KELDERMAN Wilco", 84],
 [2, "BENOOT Tiesj", 179]]
...
```

Pour le premier, la troisième valeur de la première liste est le temps total mis par le vainqueur. Pour les autres coureurs, la troisième valeur des autres listes est l'écart de temps mis avec le premier.

Par exemple : 84 = 65987 - 65903 et 179 = 66082 - 65903.

4. Recopier et compléter le programme pour qu'il en soit ainsi :

```
tableau_final = []
difference_temps = 0
premier = True
for ligne in tableau_temps:
    coureur = [ligne[0]]
    coureur.append(ligne[1])
    if premier:
        temps_premier = ligne[2]
        coureur.append(temps_premier)
        premier = False
    else:
        difference_temps = ligne[2] - ...
        coureur.append(...)
    tableau_final.append(...)
```

Partie B

Dans cet exercice, on pourra utiliser les mots clés suivants du langage SQL :

`SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `INSERT INTO`, `VALUES`, `MIN`, `MAX`, `OR`, `AND` et `ORDER BY`.

Si `propriete` est un des attributs d'une relation, les fonctions d'agrégation `MIN(propriete)`, `MAX(propriete)`, `SUM(propriete)` renvoient, respectivement, la plus petite, la plus grande valeur et la somme des valeurs des attributs sélectionnés.

On considère la base de données du tour de France 2023 dont le schéma relationnel est donné ci-dessous :

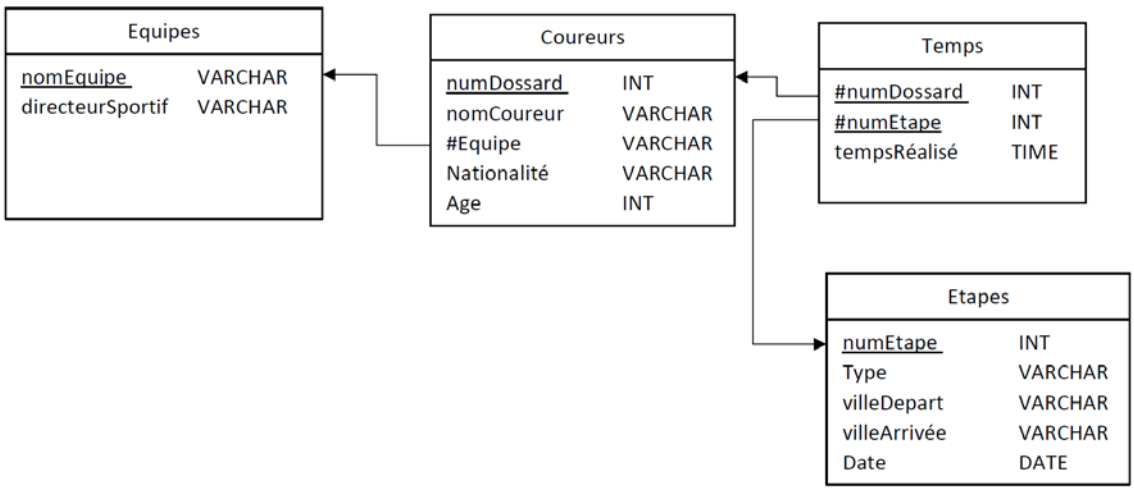

Figure 1. Schéma Relationnel

Dans ce schéma, les clés primaires sont soulignées et les clés étrangères sont précédées du symbole #.

5. Expliquer pourquoi, dans la relation `Temps`, il est nécessaire de prendre le couple (`numDossard`, `NumEtape`) comme clé primaire.

6. Expliquer ce que renvoie la requête SQL suivante :

```
SELECT nomCoureur
FROM Coureurs
WHERE Equipe = 'Cofidis';
```

7. Écrire une requête SQL permettant d'obtenir les dates de toutes les étapes de type `'contre-la-montre'` du tour de France 2023.

8. Écrire une requête SQL permettant d'obtenir le nom du directeur sportif du coureur BARDET Romain.

9. À la fin de la cinquième étape, on veut actualiser la table `Temps` avec les données du jour. Expliquer pourquoi la suite des deux requêtes SQL ci-dessous provoque une erreur.

```
INSERT INTO Temps VALUES (1, 5, 14267);
INSERT INTO Etapes VALUES(5, 'Montagne', 'Pau', 'Laruns',
05/07/2023);
```

10. Expliquer quelle modification est à effectuer pour apporter une solution au problème constaté à la question précédente.

11. Écrire une requête SQL donnant le temps total en course mis par BARDET Romain depuis le départ du tour de France 2023.