BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 2

Durée de l'épreuve : **3 heures 30**

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

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

Ce sujet comporte 19 pages numérotées de 1/19 à 19/19.

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, les requêtes SQL, la programmation en Python et la manipulation de listes.

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` et `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`.

Le but de cet exercice est d'établir une prédiction de la météo du jour en utilisant les observations du jour précédent de plusieurs stations météorologiques voisines.

Partie A

Une version simplifiée des observations peut être représentée sous forme de tables dont la description est donnée ci-dessous. Les clés primaires ont été soulignées et les clés étrangères sont indiquées par un # :

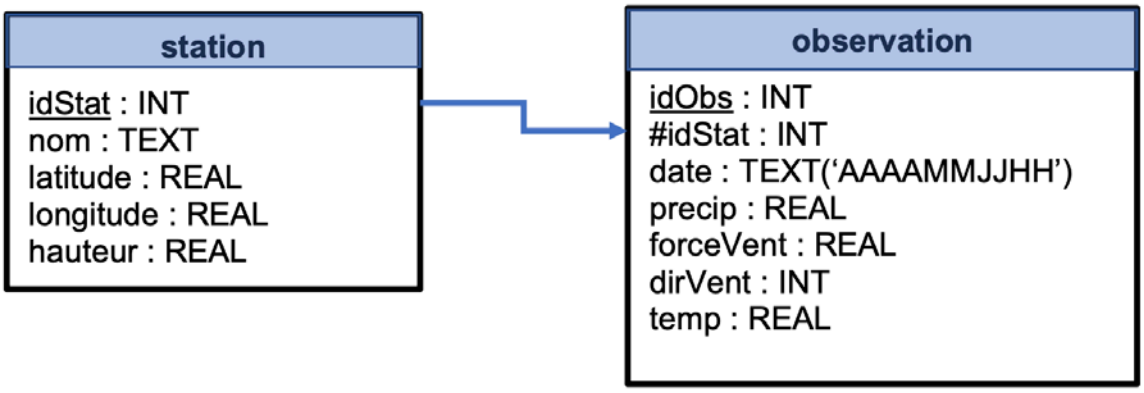

Figure 1. Tables

Dans cette partie, on considère les observations météorologiques de la Nouvelle Calédonie.

La table `station` contient l'identifiant `idStat`, le nom `nom` et les coordonnées géographiques de toutes les stations météorologiques.
La table `observation` contient l'identifiant `idStat` de l'observation, la date de l'observation `date`, la hauteur de précipitation `precip`, la force du vent `forceVent`, la direction du vent `dirVent` et la température `temp` heure par heure de toutes les stations.

Extrait table `station`				
`idStat`	`nom`	`latitude`	`longitude`	`hauteur`
…	…	`...`	`...`	…
98818001	NOUMEA	-22.276000	166.452833	69
98818002	MAGENTA	-22.260333	166.473667	3
…	…	`...`	`...`	…

Extrait de la table `observation`						
`idObs`	`idStat`	`date`	`precip`	`forceVent`	`dirVent`	`temp`
…	…	`...`	…	`...`	…	…
123456	98818001	2023123121	0.0	`5.7`	260	24.4
123457	98818001	2023123122	0.0	`5.5`	260	24.4
123458	98818001	2023123123	0.2	`5.5`	250	24.1
123459	98818002	2023010100	0.0	`4.7`	260	24.1
123460	98818002	2023010101	1.4	`3.5`	80	23.5
123461	98818002	2023010102	0.4	`2.1`	190	23.4
123462	98818002	2023010103	0.2	`1.7`	330	23.4
123463	98818002	2023123122	0.1	`1.8`	310	22.7
…	…	`...`	…	`...`	…	…

1. Donner le résultat de la requête ci-dessous en considérant les extraits de table fournis.

```
SELECT nom
FROM station
WHERE latitude = -22.276000 AND longitude = 166.452833
```

2. Écrire une requête permettant d'obtenir le nom de toutes les stations météorologiques triées par ordre alphabétique.

En SQL, la fonction d'agrégation COUNT permet de compter le nombre d'enregistrements dans une table.
Pour connaître le nombre de lignes totales dans une colonne, la syntaxe est la suivante :

```
SELECT COUNT(nom_colonne)
FROM table
```

Par exemple pour compter le nombre de stations météorologiques de la Nouvelle Calédonie, la requête est la suivante :

```
SELECT COUNT(idStat)
FROM station
```

Dans la table `observation`, les relevés météorologiques sont effectués au même moment pour toutes les stations (date identique). Ainsi, chaque station a le même nombre de relevés.

3. Écrire une requête permettant d'obtenir la force et la direction du vent à BOURAKE le 2 janvier 2023 à 14h.

4. Écrire une requête permettant d'obtenir le nombre total de relevés en Nouvelle Calédonie.

On souhaite regrouper toutes les informations dans une seule table `meteo`.

5. Écrire le schéma relationnel de la table `meteo` en supprimant les données `hauteur`, `precip`, `forceVent` et `dirVent`.

Partie B

Les données collectées sont stockées dans un unique fichier texte au format csv (*Comma Separated Values*, valeurs séparées par des virgules). Le module Python `csv` implémente des classes pour lire et écrire des données tabulaires au format csv.

On fournit ci-dessous un extrait du fichier `observations.csv` qui donne heure par heure les précipitations en millimètre, la force du vent en mètre par seconde et la direction du vent en degré (de 0 à 360 degrés) ainsi que la température en degré Celsius de la journée du 01/01/2024 pour toutes les stations météorologiques de Nouvelle Calédonie :

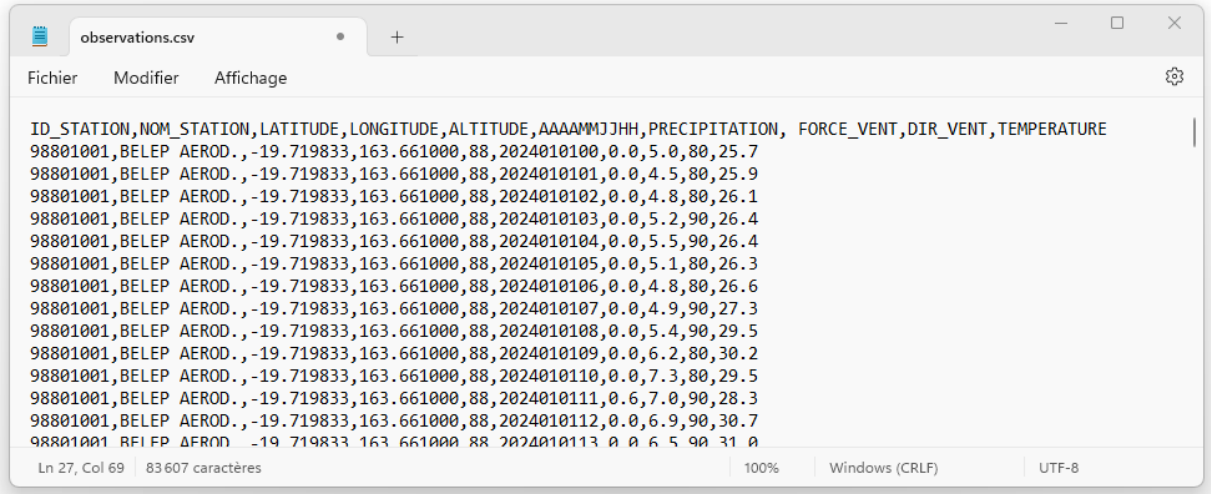

Figure 2. Extrait fichier observations.csv

Source : d'après meteo.data.gouv

Pour la suite de l'exercice, on dispose du code Python donné en annexe ainsi que de la documentation suivante :

DOCUMENTATION :

- `with open('mon_fichier.csv', 'r') as csvfile`, ouvre le fichier `mon_fichier.csv` en mode lecture (`r`) ;
- `csv.reader(csvfile, delimiter=',')`, renvoie un objet lecteur, qui itérera sur les lignes de l'objet `csvfile` donné. Chaque ligne lue depuis le fichier csv est renvoyée comme une liste de chaînes de caractères.

Dans la console, on saisit la suite d'instructions suivante :

```
>>>liste_obs = creation_liste_obs('observations.csv')
>>>liste_obs = supp_champs(liste_obs)
>>>transtype(liste_obs)
>>>liste_obs[0]
[98801001, 'BELEP AEROD.', -19.719833, 163.661, 88, 2024010100,
0.0, 5.0, 80, 25.7]
```

6. Expliquer cette liste de commande et le résultat obtenu.

Dans la suite de l'exercice, la variable `liste_obs` est initialisée avec les valeurs du fichier `observations.csv`.

La fonction `distance` renvoie la distance entre deux points définis par leur latitude et leur longitude. Cette fonction utilise des fonctions du module Python `math`.

7. Donner la ligne de commande nécessaire à l'utilisation du module Python `math`.

On rappelle les informations relatives à une observations sont données dans l'ordre suivant :

```
ID_STATION,NOM_STATION,LATITUDE,LONGITUDE,ALTITUDE,AAAAMMJJHH,PRECIPITATION,FORCE_VENT,DIR_VENT,TEMPERATURE
```

8. Compléter les lignes 40 et 41 de la fonction `coord`, qui prend en paramètres une liste d'observations `l_obs` et un nom de station `stat_ref`, et qui renvoie un tuple composé de sa latitude et sa longitude.

On considère la fonction `liste_stations` qui prend en paramètres une liste d'observations `l_obs`, un nom de station `stat_ref` et un flottant `dist` et qui renvoie la liste des identifiants `ID_STATION` des stations données dans la liste `l_obs` situées à une distance inférieure à `dist` de la station de référence `stat_ref`.

9. Écrire un algorithme en pseudo-code de la fonction `liste_stations`.

10. Écrire une fonction `nettoyage` qui prend en paramètres une liste d'observations `l_obs` et station de référence `stat_ref` (nom de la station), et qui renvoie la liste des températures des stations données dans la liste d'observations `l_obs` situées à une distance inférieure à 2000 unités de la station de référence `stat_ref`.

11. Écrire la fonction `moyenne` qui calcule et renvoie la moyenne de toutes les valeurs de type `float` contenues dans la liste passée en paramètre.

On considère maintenant le fichier `observations2.csv` donnant heure par heure les observations de la journée du 01/01/2024 pour toutes les stations météorologiques de France.

12. Donner les commandes permettant d'obtenir la moyenne des températures des stations situées à moins de 2000 unités de la station Paris_11 le 1er janvier 2024.

ANNEXE

```
 1 import csv
 2
 3 def creation_liste_obs(fichier) :
 4   liste_obs=[]
 5   with open(fichier,'r') as csvfile:
 6     fic=csv.reader(csvfile,delimiter=',')
 7     for ligne in fic:
 8       liste_obs.append(ligne)
 9   return liste_obs
10
11 def supp_champs(L) :
12   res = []
13   for i in range(1,len(L)):
14     res.append(L[i])
15   return res
16
17 def transtype(L):
```

```
18   i=0
19   while i < len(L):
20     L[i] = [int(L[i][0]),
21     L[i][1],
22     float(L[i][2]),
23     float(L[i][3]),
24     int(L[i][4]),
25     int(L[i][5]),
26     float(L[i][6]),
27     float(L[i][7]),
28     int(L[i][8]),
29     float(L[i][9])]
30     i = i + 1
31
32 def distance(p1, p2):
33   """Renvoie la distance entre deux points définis par leur
latitude et leur longitude. p1 et p2 sont des tuples
(latitude,longitude)"""
34   # Cette fonction n'est pas à compléter
35
36 def coord(l_obs , stat_ref):
37   """Renvoie la latitude et la longitude données dans la
liste d'observation l_obs de la station stat_ref"""
38   # Cette fonction est à compléter à la question 8.
39   for obs in l_obs :
40     if ... :
41       return ...,...
42
43 def liste_stations(l_obs, stat_ref, dist):
44   """Renvoie la liste des identifiants ID_STATION des
stations données dans la liste l_obs situées à une distance
inférieure à dist de la station de référence stat_ref"""
45   # Cette fonction n'est pas à compléter
46
47 def nettoyage(l_obs, stat_ref):
48   """Renvoie la liste des températures des stations données
dans la liste d'observations  l_obs situées à une distance
inférieure à 2000 unités de la station de référence
stat_ref."""
49   # Cette fonction est à compléter à la question 10.
50
51 def moyenne(L):
52   """Calcule et renvoie la moyenne de tous les nombres
contenus dans la liste passée en paramètre. L est une liste de
flottants."""
53   # Cette fonction est à compléter à la question 11.
```

Exercice 2 (6 points)

Cet exercice porte sur la structure de pile, la programmation objet et l'algorithmique.

Défi Tubes est un jeu à un joueur. Le joueur dispose de 4 tubes. Chaque tube peut contenir de 0 à 3 phases. Chaque phase possède une couleur. Il y a 3 couleurs possibles. On peut s'imaginer ces phases comme des palets de couleur dans le tube. Pour modéliser les couleurs, on utilisera les entiers 1, 2 et 3. Lorsqu'un tube contient 0 phase, on dit que le tube est vide. Lorsqu'il en a 3, on dit qu'il est plein. Lorsqu'un tube n'est pas vide, sa **dernière couleur** est la couleur de sa phase supérieure.

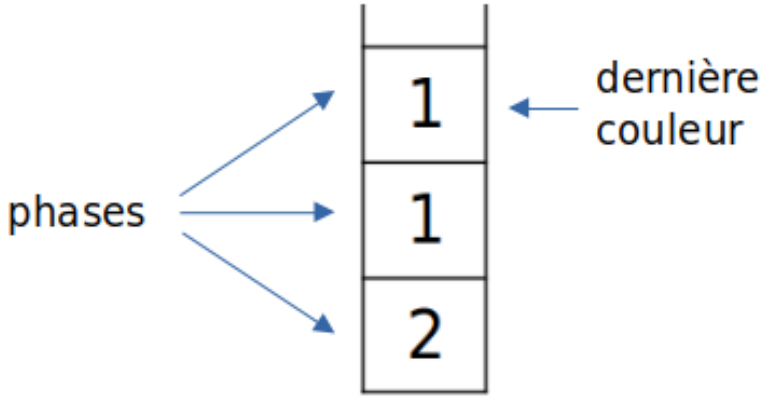

Figure 1. Exemple de tube.

Le jeu **Défi Tube** consiste à verser successivement la dernière couleur des tubes dans les autres tubes avec les contraintes suivantes :

- on ne peut rien verser dans un tube plein ;
- pour verser un **tube 1** dans un **tube 2**, il faut que la dernière couleur du **tube 1** soit la même que celle du **tube 2** ou que le **tube 2** soit vide. Dans ces deux cas, on retire la dernière couleur du **tube 1** pour qu'elle devienne la dernière couleur du **tube 2**. On réitère cela tant que la dernière couleur du **tube 1** est la même et que le **tube 2** n'est pas plein.

Le jeu se termine lorsque 3 des 4 tubes sont pleins et que leurs 3 phases sont de même couleur.

Les figures 2, 3, 4 et 5 ci-après représentent un exemple de partie du jeu **Défi Tube**.

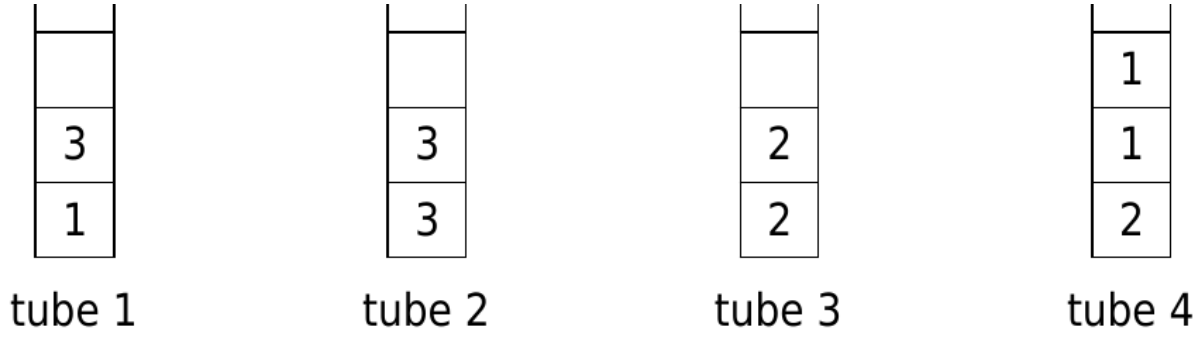

Figure 2. État initial du jeu.

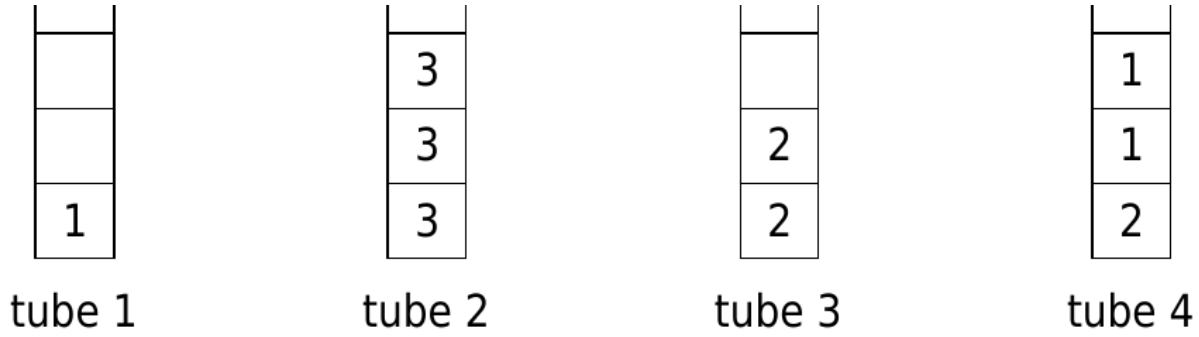

Figure 3. On a versé le tube 1 dans le tube 2.

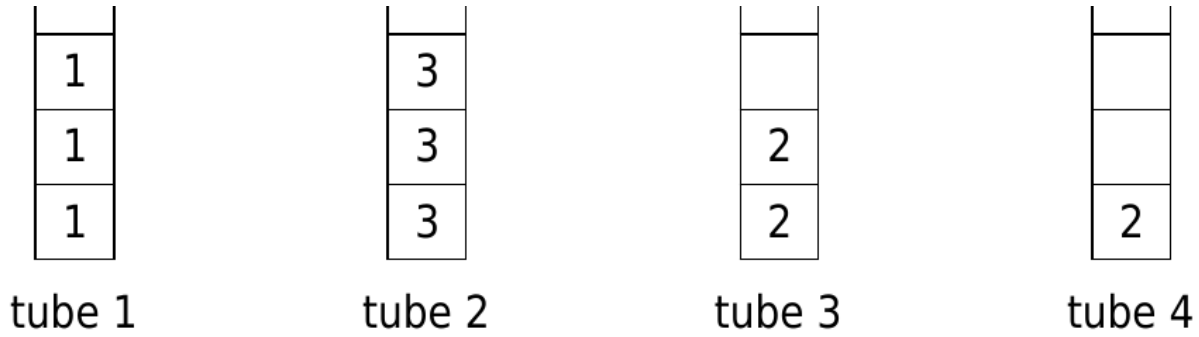

Figure 4. On a versé le tube 4 dans le tube 1.

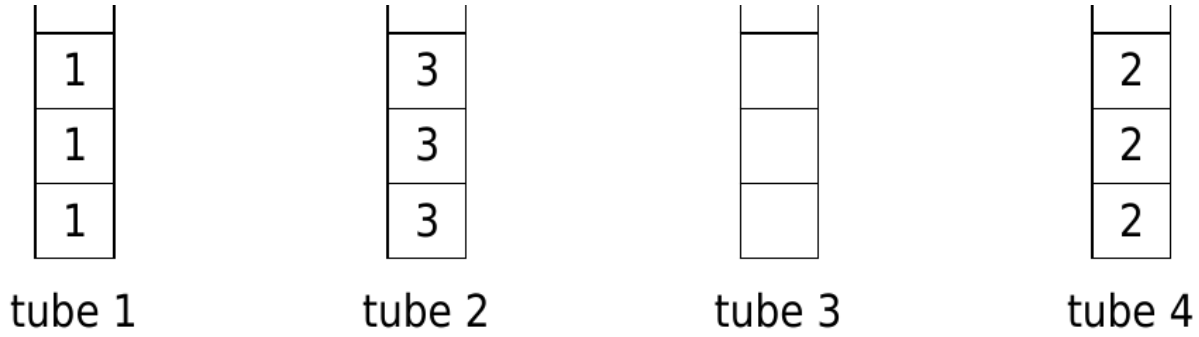

Figure 5. On a versé le tube 3 dans le tube 4.

À la figure 5, la partie est terminée.

1. Donner un exemple d'une autre séquence de versements qui aurait permis de terminer le jeu en partant de la situation de la figure 4.

Ainsi le déroulement du jeu n'est pas unique.

Partie A : Les tubes

Pour modéliser le jeu **Défi Tube**, chaque tube sera représenté par une pile finie de taille maximale 3. Les tubes sont modélisés par des objets de la classe `tube` dont le code est donné ci-dessous.

```
class tube:
    def __init__(self):
        self.taille = 0
        self.contenu = [0, 0, 0]

    def est_vide(self):
        return self.taille == 0

    def empiler(self, couleur):
        if self.taille < 3:
            self.contenu[self.taille] = couleur
            self.taille = self.taille + 1

    def depiler(self):
        if self.taille > 0:
            self.taille = self.taille - 1
            couleur = self.contenu[...]
            self.contenu[self.taille] = 0
            return ...
        else:
            return ...
```

Chaque instance de la classe `tube` a deux attributs :

- l'attribut `taille` représente le nombre d'éléments non nuls dans le tube;
- l'attribut `contenu` représente la liste (de taille 3) des éléments du tube. Lorsqu'une phase n'est pas vide, elle contiendra une couleur 1, 2, ou 3. Lorsqu'une phase est vide, sa valeur est 0.

Par exemple, le tube suivant :

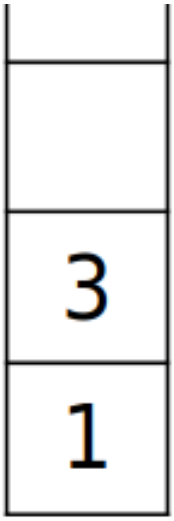

Figure 6. tube1

sera modélisé avec la classe `tube` par le code :

```
1 t = tube()
2 t.taille = 2
3 t.contenu = [1, 3, 0]
```

2. Expliquer ce qu'est la structure de pile en précisant ce que sont les méthodes `empiler` et `depiler`.

3. Expliquer les lignes 11 et 12 du code de la classe `tube`.

4. Recopier et compléter le code de la méthode `depiler` précédente. Lorsque le tube est vide, la méthode `depiler` doit renvoyer -1.

5. Écrire une méthode `est_plein` de la classe `tube`. Cette méthode renvoie `True` si le tube est plein et `False` si le tube n'est pas plein.

6. Écrire une méthode `est_homogene` de la classe `tube` qui renvoie `True` si le tube est plein et si son contenu est composé de trois fois la même couleur, et qui renvoie `False` sinon.

7. Écrire une méthode `derniere_couleur` de la classe `tube` qui renvoie le numéro de la dernière couleur du tube. Si le tube est vide, la méthode renverra la valeur `-1`.

Le code incomplet d'une méthode `verser` de la classe `tube` est donné ci-dessous :

```
1 def verser(self, other):
2     while ...
3         couleur = self.depiler()
4         other.empiler(couleur)
```

8. Recopier et compléter le code de cette méthode `verser` afin de verser l'instance `self` de la classe `tube` dans l'instance `other`. On veillera à vérifier toutes les conditions nécessaires au bon déroulement de cette opération.

Partie B : Le jeu

Pour modéliser le jeu, on appellera **état** du jeu une liste de 4 tubes. Le code suivant permet de représenter l'**état** de la figure 2.

```
 1 tube1 = tube()
 2 tube1.contenu = [1, 3, 0]
 3 tube1.taille = 2
 4 tube2 = tube()
 5 tube2.contenu = [3, 3, 0]
 6 tube2.taille = 2
 7 tube3 = tube()
 8 tube3.contenu = [2, 2, 0]
 9 tube3.taille = 2
10 tube4 = tube()
11 tube4.contenu = [1, 1, 2]
12 tube4.taille = 3
13 etat = [tube1, tube2, tube3, tube4]
```

9. En utilisant la méthode `verser` et la variable `etat` représentant la figure 2, écrire un code permettant de faire passer la variable `etat` de la représentation en figure 2 à celle de la figure 3.

10. Écrire une fonction `gagne` qui prend comme argument un état et qui renvoie `True` si la partie est terminée et `False` sinon.

Exercice 3 (8 points)

Cet exercice porte sur la programmation Python, les graphes et les réseaux.

Partie A

On considère un réseau d'antennes radios, représenté dans la figure 1, où les disques représentent la zone d'émission de chaque antenne. Pour éviter toute interférence, deux antennes "proches" géographiquement doivent émettre à des fréquences différentes.

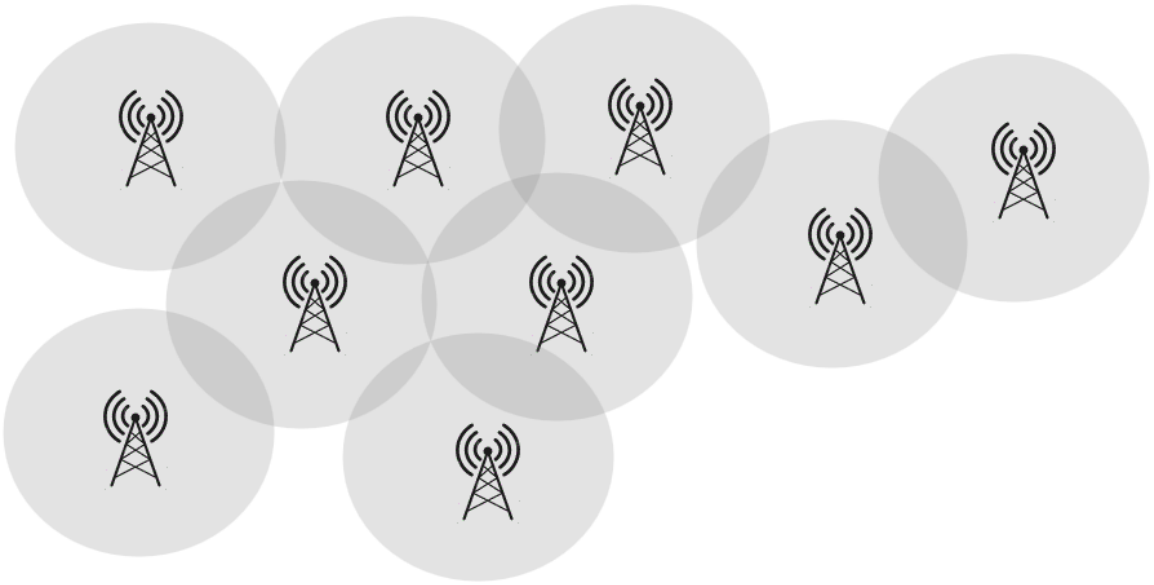

Figure 1. Réseau d'antennes

On modélise ainsi le réseau d'antennes par un graphe non orienté, appelé graphe d'interférences, dont les sommets sont les antennes numérotées de 1 à n, n étant un entier naturel supérieur ou égal à 1, et les sommets sont reliés par une arête si leurs zones d'émission s'intersectent.

Soit `G` le graphe associé au réseau d'antennes précédent :

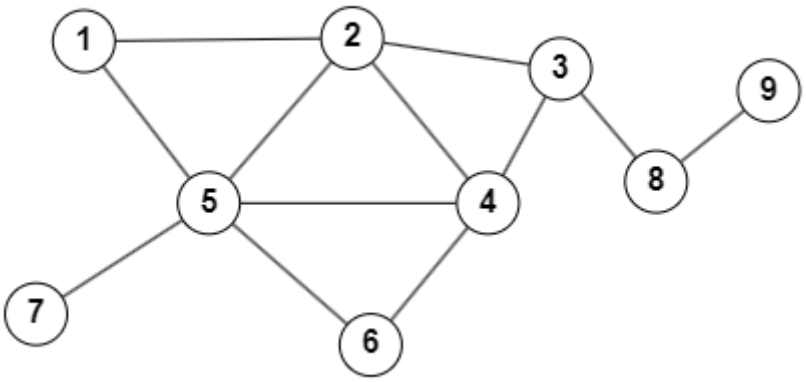

Figure 2. Modélisation du réseau sous la forme d'un graphe G

Les fréquences à allouer sont associées à des couleurs comme rouge, vert, jaune, bleu, etc.
Pour éviter les interférences, la coloration doit être une coloration propre : deux sommets adjacents ne peuvent recevoir la même couleur.

Dans cet exercice on représente le graphe `G` par un dictionnaire de listes d'adjacence dont les clefs sont les sommets de type `int` et les valeurs sont des listes de voisins du sommet clef, chaque liste contenant des éléments de type `int`.

1. Donner la valeur associée à la clé `1` dans ce dictionnaire.

2. Écrire une fonction `voisins`, qui prend en paramètres un dictionnaire et un entier, telle que `voisins(graphe, k)` renvoie une liste contenant les voisins du sommet `k` dans le graphe qui est modélisé par le dictionnaire de listes d'adjacence `graphe`.

   Exemple :

   ```
   >>> voisins(G, 2)
   [1, 3, 4, 5]
   ```

L'algorithme de Welsh et Powell consiste à colorer séquentiellement le graphe en visitant les sommets par ordre de degrés décroissants. Le degré d'un sommet d'un graphe non orienté est le nombre d'arêtes dont le sommet est une extrémité. L'idée est que les sommets ayant beaucoup de voisins sont plus difficiles à colorer : il faut les colorier en premier.

3. Écrire la fonction `degre_du_sommet` qui prend en paramètres un graphe modélisé par le dictionnaire de listes d'adjacence `graphe` et un sommet `sommet` et qui renvoie le degré du sommet `sommet`.

Exemple :

```
>>> degre_du_sommet(G, 2)
4
```

4. Écrire la fonction `degre_sommets` qui prend en paramètre un graphe modélisé par le dictionnaire de listes d'adjacence `graphe` et qui renvoie la liste des tuples `(sommet,degre)` de chaque sommet du graphe.

   Exemple :

```
>>> degre_sommets(G)
[(1, 2), (2, 4), (3, 3), (4, 4), (5, 5), (6, 2), (7, 1),
(8, 2), (9, 1)]
```

On définit la fonction `tri_liste` ci-après :

```
 1 def tri_liste(l_deg):
 2   """l_deg : liste de tuples (sommets,degré).
 3   Trie la liste l_deg par degrés décroissants"""
 4   for i in range(len(l_deg)+1):
 5     som_max = i
 6     deg_max = l_deg[i][1]
 7
 8     for j in range(i+1, len(l_deg)):
 9       if deg_max < l_deg[j][1]:
10         som_max = j
11         deg_max = l_deg[j][1]
12     temp = l_deg[i]
13     l_deg[i] = l_deg[som_max]
14     l_deg[som_max] = temp
15   return l_deg
```

À l'exécution, `tri_liste([(1, 2), (2, 2), (3, 3)])` renvoie l'erreur suivante :

```
IndexError: list index out of range
```

5. Commenter puis corriger cette erreur.

6. Choisir parmi les tris proposés celui qui correspond à la fonction `tri_liste` : tri par insertion, tri par sélection, tri fusion, tri bulle.

7. Écrire une fonction `tri_sommets` qui prend en paramètre un graphe `graphe` et qui ne renvoie que la liste des sommets du graphe `graphe` triés par degré décroissant. On pourra utiliser les fonctions définies dans les questions précédentes.

Exemple :

```
>>> tri_sommets(G)
[5, 2, 4, 3, 1, 6, 8, 7,9]
```

On suppose que le graphe est planaire, c'est-à-dire qu'il existe une représentation de ce graphe dans un plan pour laquelle les arêtes ne se croisent pas, et on définit la fonction coloration ci-après.

```
 1 def coloration(g):
 2     """Renvoie une coloration du graphe g"""
 3     # Algorithme de Welsh-Powell, limité à 4 couleurs
 4
 5     couleur = ['Rouge', 'Bleu', 'Vert', 'Jaune']
 6     coloration_sommets = {}
 7     for s_i in g:
 8         coloration_sommets[s_i] = None
 9     for s_i in tri_sommets(g):
10         couleurs_voisins_s_i = [coloration_sommets[s_j] for
s_j in voisins(g, s_i)]
11         k = 0
12         while couleur[k] in couleurs_voisins_s_i :
13             k = k + 1
14         coloration_sommets[s_i] = couleur[k]
15     return coloration_sommets
```

8. Donner le type et le contenu de la variable `coloration_sommets` de la fonction `coloration_graphe` ci-dessus pour le graphe `G`, après execution de la boucle des lignes 7 et 8.

9. Recopier et compléter le retour de la fonction `coloration` appliquée au graphe `G` donné plus haut.

   ```
   {1: 'Vert', 2: ..., ...}
   ```

Partie B

On s'intéresse maintenant à un réseau informatique.

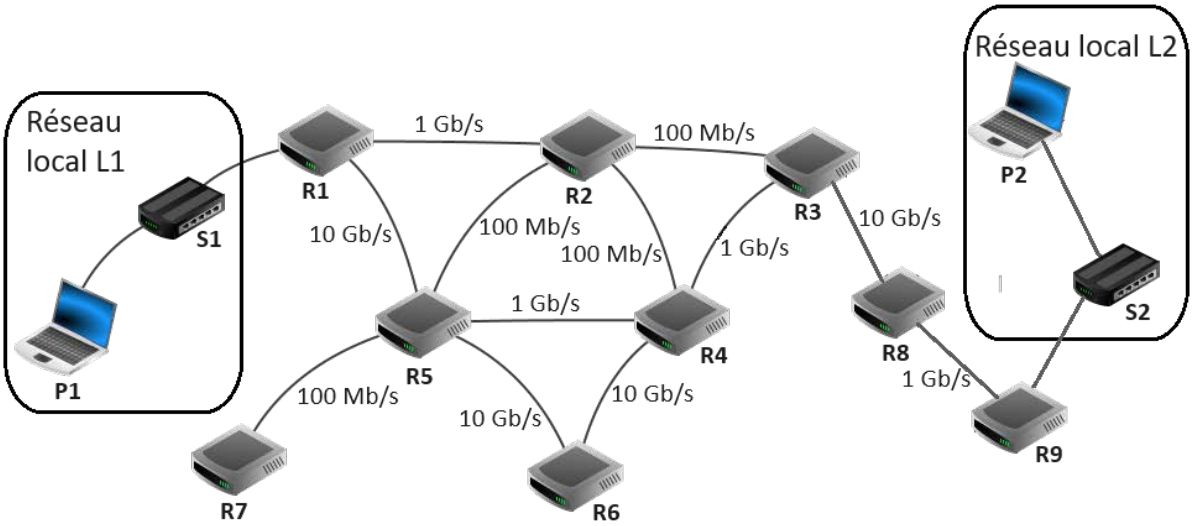

Figure 3. Réseau entreprise

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 représentations décimales des 4 octets. La notation X1.X2.X3.X4/n signifie que les n premiers bits de l'adresse IP représentent la partie « réseau », les bits suivants représentent la partie « hôte ».

On fournit les données suivantes concernant le réseau de cette entreprise.

Réseau local L1 :
- Adresse IP de l'ordinateur P1 : 190.12.10.25/24
- S1 : switch

Réseau local L2 :
- Adresse réseau : 12.128.0.0
- Masque de sous réseau : 255.255.0.0
- S2 : switch
- P2 : ordinateur

Extrait de l'arborescence du système de fichiers de l'ordinateur P2 :

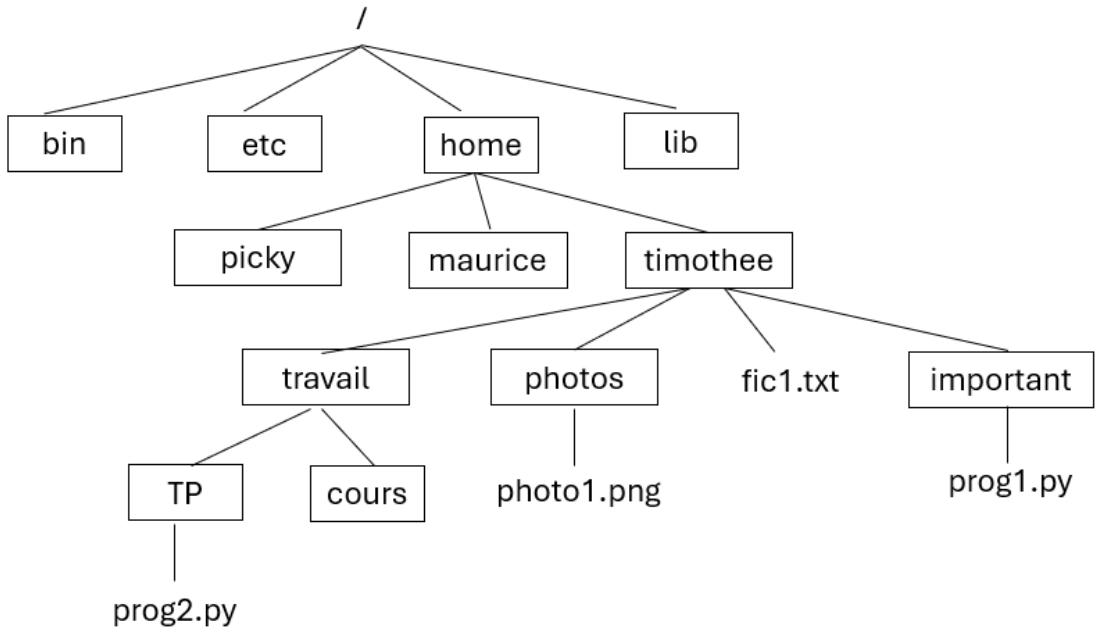

Figure 4. Arborescence

Extrait du manuel de la commande cp :

```
[root@localhost ~]# man cp
CP(1)                          User Commands                          CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
```

Figure 5. Manuel de la commande cp

10. Donner une commande en ligne qui permet de copier le fichier `prog1.py` dans le répertoire `TP` lorsqu'on se trouve dans le répertoire nommé `important`.

11. Donner la commande qui permet de vérifier si l'ordinateur P1 est accessible lorsque l'on travaille sur l'ordinateur P2.

12. Donner une adresse possible pour l'ordinateur P2 du réseau local L2.

Dans le cadre du protocole RIP, le chemin emprunté par les informations est celui qui aura la distance la plus petite en nombre de sauts.
Dans le cadre du protocole OSPF, le chemin emprunté par les informations est celui qui aura le coût total minimal.

Extraits des tables de routage :

Routeur	Destination	Passerelle
R1	R9	R2
R2	R9	R3
R3	R9	R8
R4	R9	R3
R5	R9	R4
R6	R9	R4
R7	R9	R5
R8	R9	R9
R9	R9	LOCALHOST

13. Donner le chemin emprunté par un paquet de données allant de l'ordinateur P1 à l'ordinateur P2, en utilisant l'extrait de la table de routage.

14. Donner le nom du protocole de routage qui semble être utilisé.

Dans les questions suivantes, on utilise le protocole de routage OSPF.

Pour calculer le coût C d'une liaison, on utilise la formule : $C = \frac{10^8}{BP}$ où BP est la bande passante en bits par seconde.

15. Calculer les coûts pour des liaisons de 100 Mbits/s, 1 Gbits/s et 10 Gbits/s.

16. Déterminer la route qui sera empruntée par le paquet de données envoyé de l'ordinateur P1 à l'ordinateur P2, en respectant le protocole OSPF.