Amérique du sud - septembre 2024 - sujet 2

Exercice 1 (Récursivité et langage SQL - 6 points)
« Le compte est bon » est un jeu dans lequel on dispose de plusieurs valeurs entières et positives et un entier positif appelé *cible*. Il peut se jouer seul, mais on peut aussi y jouer à deux et le but est alors de trouver la cible en premier.

Un joueur peut ajouter, soustraire, multiplier ou diviser deux des entiers disponibles pour en former un nouveau qui remplacera les deux entiers utilisés (pour cela, il faut que le résultat soit un entier positif). Si le joueur parvient à former l'entier cible, il a gagné, sinon il a perdu.

Exemple de partie gagnante
Supposons que les valeurs disponibles au début du jeu sont les entiers 1, 7, 10, 10 et 50, et que la cible est 14. Pour gagner, le joueur peut :

- soustraire 7 à 10 : il lui reste les entiers 1, 3, 10 et 50 ;
- ajouter 1 et 3 : il lui reste les entiers 4, 10 et 50 ;
- ajouter 4 et 10 : il lui reste 14 et 50 et donc il a réussi à obtenir la cible 14 !

Vérification de la faisabilité
Alice et Bob décide d'écrire un programme permettant de savoir s'il est possible d'atteindre la cible.

```
1  def resoudre(cible, valeurs):
2      """
3      cible : valeur qu'on souhaite obtenir
4      valeurs : liste des valeurs restantes pour générer la cible
5      return : True si les valeurs permettent d'obtenir la cible, False sinon
6      """
7      # si la cible a déjà été trouvée,
8      # il est inutile de continuer les calculs :
9      if ... :
10         return True
11
12     # s'il n'y a plus assez de valeurs pour continuer les calculs
13     # il est inutile de continuer les calculs :
14     if ... :
15         return False
16
17     # on va tester tous les calculs possibles :
18     for i in range(len(valeurs))
19         for j in range(i+1, len(valeurs)):
20             for op in [add, mul, sub, div]:
21                 v = op(valeurs[i], valeurs[j])
22                 if v is not False and v != 0: # l'opération était possible
23                     new_vals = [v]
24                     for k in range(len(valeurs)):
25                         if k != i and k != j:
26                             new_vals.append(valeurs[k])
27
28                     if resoudre(cible, new_vals):
29                         return True
30     return False
```

1. Expliquer pourquoi cette fonction est récursive en citant la ou les lignes qui permettent de justifier.
2. Recopier et compléter les lignes 9 et 14.

3. Ecrire les quatre fonctions suivantes :
   - `add(a, b)` qui renvoie la somme `a + b`;
   - `mul(a, b)` qui renvoie le produit `a * b`;
   - `sub(a, b)` qui renvoie la différence positive `a - b` si `a >= b` et `b - a` sinon ;
   - `div(a, b)` qui renvoie `a // b` si `a` est divisible par `b` et `False` sinon.
4. Lors de l'exécution des lignes 18 à 21, déterminer la valeur de `v` lorsque `valeurs = [1, 7, 10, 10, 50]`, `i = 1`, `j = 2` et `op = sub`.

Mémorisation des parties

Dans cette partie, on s'intéresse à la mémorisation des parties réalisées par différents joueurs, en utilisant les deux tables ci-après :

- la table `joueur` permet de mémoriser les informations des joueurs :

joueur			
id	nom	prenom	anne_de_naissance
1	Fabrega	Christine	1965
2	Laffont	Patrice	1972
3	Etienne	Bernard	1984
4	Cabrol	Laurent	1989
5	Meynier	Max	1991
6	Romejko	Laurent	1992
7	Boulin-Prat	Arielle	1986
8	Renard	Bertrand	1975
9	Maire	Blandine	2022

- la table `partie` permet de mémoriser les résultats des joueurs (`gagnant` est l'identifiant de celui qui a gagné et `perdant` celui qui a perdu). L'attribut `idP` est la clé primaire de la table `partie` :

partie		
idP	gagnant	perdant
1	6	8
2	8	9
3	9	3
4	6	4
5	7	1
6	5	4
7	9	3
8	6	9
9	1	3

5. Déterminer la clé primaire de la table `joueur` et la ou les clés étrangères de la table `partie` en précisant les attributs de la table `joueur` référencés.
6. Ecrire une requête SQL permettant d'obtenir le nom du gagnant de la partie dont l'identifiant est 1.
7. Blandine (identifiant 9) ne souhaite plus rester dans la base de données : donner les requêtes SQL permettant de supprimer son nom de la base de données en s'assurant que la base de données reste cohérente.
8. Ecrire une requête SQL permettant d'obtenir le nombre de victoires de Bertrand Renard (on pourra utiliser son identifiant 8).

Exercice 2 (Graphe et POO - 6 points)
On modélise un réseau social sous forme d'un graphe où chaque nœud représente une personne et chaque arête l'amitié entre deux personnes. On considère, dans cet exercice, qu'une amitié est forcément réciproque.

Partie A

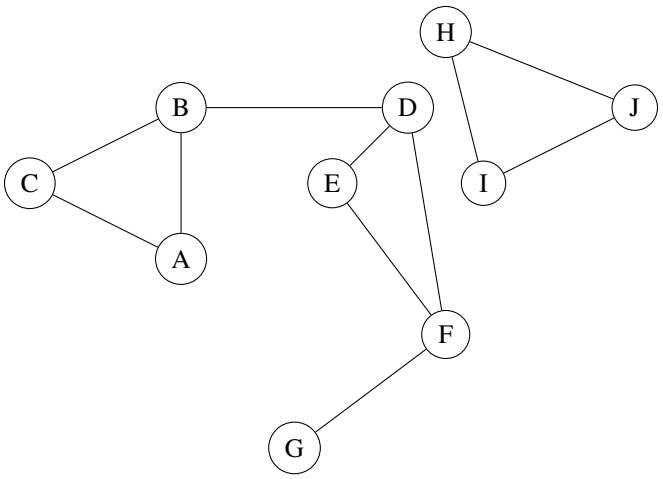


FIGURE 1 – Graphe d'un tout petit réseau social de ce type

Par souci de simplification, chaque personne est représentée dans cette partie par une lettre.

1. Citer deux personnes qui sont amies.
2. Citer deux personnes qui ne sont pas amies, mais qui ont au moins un ami en commun.

Chaque jour, le réseau social fait des recommandations d'amis à chaque membre en lui proposant comme nouveaux amis les amis de ses amis qui ne sont pas encore ses amis.

On suppose que chaque membre ajoute à sa liste d'amis tous les amis qui lui sont proposés chaque jour, et aucun autre.

3. Dessiner le graphe du réseau social le jour suivant la première recommandation.
4. Déterminer combien il faut au minimum de jours de recommandation pour que A et E deviennent amis.
5. Si tous les jours les membres du réseau acceptent systématiquement la proposition d'amis, expliquer au bout de combien de jours plus aucune recommandation ne sera faite sur ce réseau social.

Partie B : on considère maintenant un réseau social comportant de nombreux utilisateurs
On implémente la structure du graphe à l'aide des trois classes suivantes :

⋆ `Noeud` :
- Attributs :
  — `nom` : chaîne de caractères
  — `voisins` : liste d'objets `Noeud`
  — `couleur = 'bleu'` : chaîne de caractères
- Méthode :
  — `ajouter_voisin(self, voisin)` : ajoute le nœud `voisin` à l'attribut `voisins`

⋆ `Arete` :
- Attributs :
  — `noeud1` : objet `Noeud`
  — `noeud2` : objet `Noeud`

⋆ `Graphe` :
- Attributs :
  — `noeuds` : liste d'objets `Noeud`
  — `aretes` : liste d'objets `Arete`
- Méthodes :
  — `ajouter_noeud(self, N` : ajoute le nœud `N` à sa liste de nœuds
  — `ajouter_arete(self, noeud1, noeud2)` : crée un objet `Arete`, l'ajoute à sa liste d'arêtes, ajoute `noeud2` comme voisin de `noeud1` et `noeud1` comme voisin de `noeud2`

Le graphe, objet de la classe `Graphe`, représentant les interactions du réseau social est noté `G`.

6. Dans le contexte de cet exercice, à quoi sert la fonction suivante ?

```
1  def mystere(A, B):
2      '''
3      A et B sont deux instances de la classe Noeud
4      '''
5      if B in A.voisins:
6          return True
7      else:
8          return False
```

7. Recopier et corriger la fonction suivante qui permet de savoir si deux membres du réseau social ont des amis communs :

```
1  def amis_commun(A, B):
2      for S in A.voisins:
3          if B in S:
4              return True
5          else:
6              return False
```

8. On appelle « chemiin d'amis » entre deux membres `A` et `B` du réseau social, un chemin reliant `A` et `B` dans le graphe `G`. Recopier et compléter les lignes 9, 11, 12 et 18 du code suivant qui donne l'existence d'un chemin d'amis entre deux membres et qui utilise le parcours en profondeur du graphe `G` dans sa version récursive.

```
1  def parcours(S, L = []):
2      '''
3      S est un sommet
4      S est le point de départ du parcours du graphe
5      L est une liste dans laquelle on stocke les noms
6      des personnes rencontrées lors de ce parcours
7      '''
8      S.couleur = 'rouge'
9      L.append(...)
10     for E in S.voisins:
11         if E.couleur ... :
12             ...
13     return L
14
15 def chemin_amis(A, B):
16     for N in G.noeuds:
17         N.couleur = 'bleu'
18     if B.nom in ... :
19         return True
20     else:
21         return False
```

9. Ecrire en Python la fonction `parcours_largeur` qui parcourt en largeur le graphe `G` à partir d'un sommet `A` et qui donne le nom de toutes les personnes du réseau social qui ont un chemin d'amis avec la personne `A.nom`. Pour cela, on peut utiliser les classes `Pile` et/ou `File` suivantes :

Méthodes pour la classe `Pile`	Méthodes pour la classe `File`
⋆ `est_videP(self)` : renvoie `True` si la pile est vide, `False` sinon ⋆ `empiler(self, element)` : met `element` dans la pile ⋆ `depiler(self)` : enlève le sommet de la pile et le renvoie ⋆ `lire_sommet(self)` : lit le sommet de la pile sans modifier la pile	⋆ `est_videF(self)` : renvoie `True` si la file est vide, `False` sinon ⋆ `enfiler(self, element)` : met `element` dans la file ⋆ `defiler(self)` : enlève la tête de la file et la renvoie ⋆ `lire_tete(self)` : lit la tête de la file sans modifier la file

Exercice 3 (Réseau, protocoles de routage et algorithmique - 8 points)

Partie A : adressage d'un réseau

Le serveur web d'une entreprise dispose de deux cartes réseau : l'une est reliée au réseau interne de la société, l'autre à Internet.

La commande `ip addr show eth0` permet d'obtenir les caractéristiques de l'interface réseau `eth0` et affiche le résultat suivant :

```
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP>
      mtu 1500 qdisc mq state UP group default qlen 1000
      link/ether f2:3c:92:8f:45:26 brd ff:ff:ff:ff:ff:ff
      inet 172.21.200.11/24 brd 172.21.200.255 scope global no-
      prefixroute dynamic eth0
```

L'adresse IP d'une machine est composée d'une adresse IPv4 et d'une indication sur le masque de réseau. Une adresse IPv4 est composée de quatre octets, soit 32 bits. Elle est notée `a.b.c.d`, où `a`, `b`, `c` et `d` sont les valeurs des quatre octets.

La notation `a.b.c.d/n` signifie que :

- les `n` premiers bits (de poids fort) de l'adresse IP représentent la partie « réseau » ;
- les bits qui suivent représentent la partie « machine ».

Pour cette même adresse `a.b.c.d/n`, le masque de ce réseau, en binaire, est composé des `n` premiers bits à `1` suivis par des `0` pour remplir les quatre octets.

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 ».

Avec un masque de réseau `/24`, la partie réservée pour la partie « machine » est le dernier octet.

Dans l'affichage ci-dessus, `inet` donne des informations sur le réseau local. L'adresse indiquée est `172.21.200.11/24`

1. Déterminer le masque de ce réseau.
2. Déterminer l'adresse de ce réseau.
3. Déterminer l'adresse de diffusion (broadcast) de ce réseau.
4. Déterminer le nombre d'adresses disponibles sur ce réseau pour les équipements.

Partie B : tri par comptage

Afin de faire des statistiques sur l'usage du serveur web par les utilisateurs du réseau de l'entreprise, on dispose des journaux de connexion au serveur web contenant les informations suivantes :

- l'heure de connexion ;
- la page demandée ;
- l'identification du client qui a interrogé le serveur (cet identifiant est un entier unique).

Ces informations sont stockées dans une liste. L'ordre par défaut est l'ordre chronologique. Pour exploiter le fichier, les informations doivent être triées par ordre croissant des identifiants. On utilise pour cela un tri par comptage portant sur les identifiants.

On considère ainsi un tableau `tab` non trié, composé de `n` nombres entiers compris entre `0` et `k` (bornes comprises).

Le principe du tri par comptage est le suivant :

- On compte le nombre de `0`, le nombre de `1`, ..., le nombre de `k` présents dans `tab`. Les résultats sont stockés dans une liste de comptage `liste_compteurs`. Ainsi, `liste_compteurs[i]` est le nombre de fois que le nombre `i` apparaît dans le tableau.
- On construit ensuite le tableau trié en ajoutant `liste_compteurs[0]` fois la valeur `0`, `liste_compteurs[1]` fois la valeur `1`, ..., `liste_compteurs[k]` fois la valeur `k`.

Exemple :

- Le tableau de neuf entiers `[3, 4, 6, 3, 6, 4, 6, 6, 3]` contient trois fois la valeur `3`, deux fois la valeur `4` et quatre fois la valeur `6`.
- La liste de comptage est la suivante : `liste_compteurs = [0, 0, 0, 3, 2, 0, 4]`.
- Le tableau trié est le suivant : `[3, 3, 3, 4, 4, 6, 6, 6, 6]`.

5. Donner la liste de comptage résultant du tableau de onze entiers suivant (`k=9`) :

`[3, 8, 4, 9, 8, 4, 6, 9, 1, 4, 9]`

6. L'exécution de l'algorithme de tri par comptage nécessite de déterminer la valeur maximale des éléments du tableau à trier. Ecrire, en Python, une fonction `maximum` qui prend en paramètre un tableau `tab` non vide d'entiers positifs et renvoie la valeur maximale des éléments du tableau `tab`, sans utiliser la fonction native `max` de Python.

On souhaite écrire une fonction `tri_comptage` en Python qui implémente l'algorithme du tri par comptage. Elle prend en paramètre un tableau `tab` d'entiers positifs à trier et renvoie un nouveau tableau trié et formé des éléments de `tab`.

7. Recopier, en les complétant, les instructions des lignes 9 à 16 du code de la fonction `tri_comptage` ci-dessous :

```
1  def tri_comptage(tab):
2      """ paramètre : tab, tableau non vide d'entiers
3          sortie : un nouveau tableau tab trié
4          remarque : tab n'est pas modifié """
5      val_max = maximum(tab)
6      liste_compteurs = [0]*(val_max + 1)
7      for element in tab:
8          liste_compteurs[element] = liste_compteurs[element] + 1
9      nouv_tab = ...
10     position = 0
11     for i in range(len(liste_compteurs)):
12         val = liste_compteurs[i]
13         for j in range(...):
14             nouv_tab[position] = ...
15             position = position + 1
16     return ...
```

On rappelle que l'instruction `[0]*n` construit un tableau de taille `n` dont la valeur de chaque élément est `0`, et que l'instruction `len(tableau)` renvoie la longueur de `tableau`.

Partie C : protocoles de routage

On considère le réseau représenté sur la figure 1 ci-après :

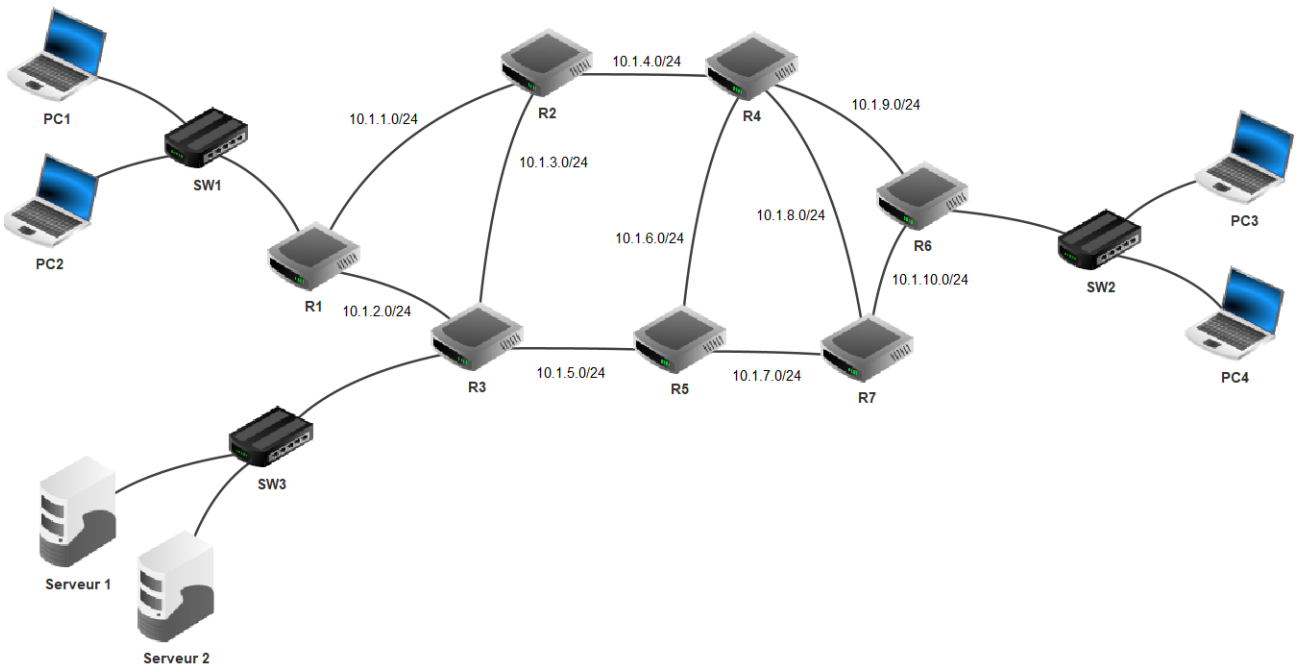


FIGURE 1 – Schéma du réseau

Le schéma est composé de sept routeurs R1, R2, R3, R4, R5, R6 et R7.
Les ordinateurs portables PC1 et PC2 sont reliés au switch SW1.
Les ordinateurs portables PC3 et PC4 sont reliés au switch SW2.
Les serveurs Serveur1 et Serveur2 sont reliés au switch SW3.

8. On donne, dans les tables suivantes, l'adresse des réseaux directement connectés aux routeurs R1, R2 et R3 et on précise à chaque fois l'adresse qu'utilise le routeur pour se connecter à ce réseau.

Routeur R1	
Destination	**Interface**
`192.168.1.0/24`	`192.168.1.1`
`10.1.1.0/24`	`10.1.1.1`
`10.1.2.0/24`	`10.1.2.1`

Routeur R2	
Destination	**Interface**
`10.1.1.0/24`	*Interface 1*
`10.1.3.0/24`	*Interface 2*
`10.1.4.0/24`	`10.1.4.1`

Routeur R3	
Destination	**Interface**
`192.168.3.0/24`	`192.168.3.1`
`10.1.2.0/24`	`10.1.2.2`
`10.1.3.0/24`	`10.1.3.2`
`10.1.5.0/24`	`10.1.5.1`

Proposer une adresse IP pour chacune des interfaces *Interface 1* et *Interface 2* du routeur R2.

L'ordinateur PC2 communique avec l'ordinateur PC3.

Le protocole de routage utilisé est RIP (Routing Information Protocol). Dans ce protocole, le chemin retenu pour transmettre un message est celui traversant le nombre minimum de routeurs.

9. En utilisant le schéma du réseau de la figure 1, donner les noms des routeurs traversés successivement lors de l'envoi d'un paquet entre PC2 et PC3.
10. Suite à des travaux, la liaison entre les routeurs R4 et R6 est coupée. Donner toutes les routes pouvant être empruntées par les paquets de données échangés entre PC2 et PC3 avec le routage dynamique RIP.

On suppose maintenant que le protocole de routage mis en place dans le réseau est OSPF. Ce protocole consiste à minimiser la somme des coûts des liaisons empruntées.

Le coût d'une liaison est défini par la relation

$$\text{coût} = \frac{10^8}{d},$$

où d représente le débit de la liaison en bit/s ; le coût est sans unité.

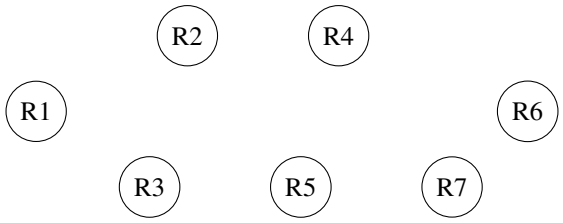
Dans la table 1 suivante, on donne les débits des liaisons entre routeurs :

Liaison	R1-R2	R1-R3	R2-R3	R2-R4	R3-R5	R4-R5	R4-R6	R4-R7	R5-R7	R6-R7
d Mbit/s	100	1 000	10 000	10 000	1 000	10 000	100	1 000	10 000	1 000
Coût	1	0,1	?	?	0,1	?	1	0,1	?	0,1

TABLE 1 – Débit et coût des liaisons entre routeurs

On rappelle qu'un Mbit/s est égal à 10^6 bit/s et qu'un Gbit/s est égal à 10^9 bit/s.

11. Calculer le coût d'une liaison entre deux routeurs dont le débit est 10 Gbit/s.
12. On souhaite représenter le schéma du réseau de routeurs de la figure 1 avec les coûts. Recopier sur votre copie le schéma suivant et tracer les liaisons entre les routeurs en y indiquant le coût de chaque liaison.



13. Déterminer le chemin parcouru par un paquet partant du routeur R1 et arrivant au routeur R6 en utilisant le protocole OSPF et préciser le coût.