Numérique et Sciences Informatiques – EXERCICE I (20 points)

Les quatre premières questions de cet exercice portent sur les bases de données relationnelles et le langage SQL ; les mots clés du langage SQL utilisés sont : `DISTINCT` *(dont le sens est rappelé),* `SELECT`, `FROM`, `WHERE`, `JOIN…ON`, `INSERT INTO…VALUES`, `ORDER BY`. *La dernière question de cet exercice porte sur la programmation en Python, la manipulation des dictionnaires et le parcours de liste.*

On souhaite stocker la composition de plats et leur analyse nutritionnelle moyenne (% de glucides, % de lipides et % de protéines) dans une base de données relationnelle. Le schéma relationnel de cette base de données comporte trois relations :

- `plat(id_plat, nom)`
- `ingredient(#id_plat, #id_ingr, quantite)`
- `analysenutri(id_ingr, nom, glucides, lipides, proteines)`

Dans ce schéma relationnel :

- tous les attributs dont le nom commencent par `id_` ont des valeurs entières ;
- les attributs dont le nom est `nom` ont pour valeurs des chaînes de caractères ;
- les autres attributs ont des valeurs réelles avec un chiffre après la virgule ;
- les clés primaires sont soulignées ;
- les clés étrangères sont précédée d'un `#` .

Ainsi

- le couple `(ingredient.id_plat, ingredient.id_ingr)` est la clé primaire de la relation `ingredient` ;
- `ingredient.id_plat` est une clé étrangère faisant référence à `plat.id_plat` ;
- `ingredient.id_ingr` est une clé étrangère faisant référence à `analysenutri.id_ingr`.

I-1. On suppose que la base de données est encore vide ; les trois relations/tables (`plat`, `ingredient` et `analysenutri`) ne contiennent donc aucun enregistrement. On considère les cinq requêtes d'insertion ci-dessous numérotées de (r1) à (r5). Pour chaque ordre d'exécution proposé dans le document réponse, indiquez si une erreur se produit.

```
(r1) INSERT INTO plat VALUES (1, 'pâtes aux beurre') ;
(r2) INSERT INTO ingredient VALUES (1, 1, 85.0) ;
(r3) INSERT INTO ingredient VALUES (1, 2, 5.0) ;
(r4) INSERT INTO analysenutri VALUES (1, 'penne', 71.7, 1.6, 12.6 ) ;
(r5) INSERT INTO analysenutri VALUES (2, 'beurre', 1.0, 80.8, 0.7) ;
```

I-2. On suppose que les tables `plat` et `ingredient` contiennent chacune au moins un enregistrement.

(a) Le résultat de la requête ci-dessous peut-il être vide ? Justifiez brièvement.

```
SELECT *
FROM plat
JOIN ingredient ON plat.id_plat = ingredient.id_plat ;
```

(b) Le résultat de la requête ci-dessous peut-il être vide ? Justifiez brièvement.

```
SELECT *
FROM ingredient
JOIN plat ON plat.id_plat = ingredient.id_plat ;
```

I-3. On rappelle que le mot clé `DISTINCT` du langage SQL permet d'éliminer les doublons du résultat d'une requête ; dans cette question, un identifiant de plat apparaîtra donc au plus une fois dans le résultat de d'une requête. Complétez les requêtes ci-dessous de sorte que leur résultat indique la liste des identifiants de plats qui contiennent

(a) l'ingrédient dont l'identifiant est 7, celui dont l'identifiant est 9, ou les deux.

```
SELECT DISTINCT id_plat
FROM …①…
WHERE id_ingr …②… 7 …③… id_ingr …②… 9 ;
```

(b) au moins un ingrédient dont la part des lipides est strictement supérieure à `50%`.

```
SELECT DISTINCT id_plat
FROM …①…
…④…
WHERE lipides > 0.5 ;
```

I-4. Complétez (sans ajouter de mot clé du langage SQL) la requête ci-dessous de sorte que son résultat indique la composition de chaque plat sous la forme d'une liste d'enregistrements (`id_plat, plat.nom, analysenutri.nom`); les enregistrements sont ordonnés suivants l'ordre alphabétique des noms de plat puis des noms d'ingrédients.

```
SELECT plat.id_plat, plat.nom, analysenutri.nom
FROM plat
JOIN …①… ON plat.id_plat = …②…
JOIN …③… ON …④…
ORDER BY plat.nom, analysenutri.nom ;
```

La fonction `composition` définie en Python renvoie un dictionnaire dont les clés sont des identifiants d'ingrédients (entier) et dont les valeurs sont les quantités (valeurs réelles) de ces ingrédients dans une portion individuelle du plat dont l'identifiant (entier) a été transmis à la fonction. En prenant comme exemple les *pâtes au beurre* de la question I-1, l'appel `composition(1)` renverrait `{1: 85.0, 2: 5.0}`.

La fonction `analyse_nutri` définie en Python reçoit en argument un identifiant d'ingrédient (entier) dont elle renvoie l'analyse nutritionnelle sous la forme d'un dictionnaire dont les clés sont les noms des attributs (`'glucides'`, `'lipides'` et `'proteines'`) de la relation `analysenutri` de la base de données et dont les valeurs sont des réels. En prenant comme exemple le *beurre* de la question I-1, l'appel `analyse_nutri(2)` renverrait `{'glucides': 1.0, 'lipides': 80.8, 'proteines': 0.7}`.

I-5. Complétez la définition en Python de la fonction `nutri` de sorte qu'elle renvoie le poids de glucides, de lipides et de protéines d'une portion individuelle du plat dont l'identifiant (entier) lui a été transmis ; le résultat est donné sous la forme d'un dictionnaire dont les clés sont `'glucides'`, `'lipides'` et `'proteines'`.

```
def nutri(id_plat)
    res = …①…
    comp = composition(id_plat)
    for i in comp.keys():
        anutri = analyse_nutri(i)
        for k in anutri.keys():
            …②… = …②… + …③…/100 * …④… [i]
    return res
```

Numérique et Sciences Informatiques – EXERCICE II (20 points)

Cet exercice porte sur les graphes, les algorithmes de graphes, la récursivité, la programmation en Python, la manipulation des dictionnaires et le parcours de liste.

Un guide touristique décide de baliser des sentiers reliant différents sites olympiques et de publier une application permettant de planifier des itinéraires ne passant que par ces sentiers. Le guide donne un nom à chacun des sentiers et il mesure leur longueur (en mètres). Il regroupe ces informations dans une liste de sentiers où chaque sentier est représenté par un quadruplet *(nom du sentier, extrémité 1, extrémité 2, longueur)* où les extrémités 1 et 2 sont les noms des deux sites olympiques reliés par le sentier. Sa liste de sentiers commence ainsi :

```
sentiers = [('Briana', 'La Concorde',  'Grand Palais',  100),
            ('Teddy',  'Grand Palais', 'Trocadéro',     550),
            ('Elaine', 'La Concorde',  'Trocadéro',     200),
            ('Lucie',  'Grand Palais', 'Champ de Mars', 512),
            ('Sydney', 'Trocadéro',    'Champ de Mars', 100), ...
```

L'objectif de cet exercice est de réaliser une fonction en langage Python calculant l'itinéraire le plus court entre deux sites olympiques. Le problème est modélisé à l'aide d'un graphe orienté dont les sommets correspondent aux sites olympiques et dont les arcs correspondent aux sentiers permettant d'aller d'un site à l'autre. Un itinéraire correspond à un chemin dans le graphe dont la longueur est la somme des longueurs des arcs qui le composent (c'est-à-dire la somme des longueurs des sentiers empruntés).

On représente ce graphe à l'aide d'un dictionnaire dont les clés sont les sommets du graphe (nom d'un site olympique) auxquels on associe comme valeurs la liste de leurs arcs sortants. Chaque arc sortant est défini sous la forme d'un triplet *(nom du site à l'autre extrémité du sentier, nom du sentier, longueur du sentier)*. On donne ci-dessous un exemple d'extrait du dictionnaire :

```
{'La Concorde': [('Grand Palais', 'Briana', 100),
                 ('Trocadéro', 'Elaine', 200)],
 'Champ de Mars': [('Grand Palais', 'Lucie', 512),
                   ('Trocadéro', 'Sydney', 100)], ...
```

II-1. Complétez la définition de la fonction `creer_dico_arcs_sortants` qui produit un tel dictionnaire à partir de la liste des sentiers établie par le guide touristique. Notez qu'un sentier peut être emprunté dans les deux sens ; pour un sentier permettant d'aller du point `p1` au point `p2`, il faut ajouter au dictionnaire les arcs (`p1`, `p2`) et (`p2`, `p1`) qui sont de même longueur.

```
def creer_dico_arcs_sortants(sentiers):
    arcs_sortants = …①…
    for (n,p1,p2,d) in sentiers:
        if …②… not in arcs_sortants:
            arcs_sortants[ …②… ] = []
        arcs_sortants[ …②… ].append((p1,n,d))
        if …③… not in …④… :
            arcs_sortants[ …③… ] = []
        arcs_sortants[ …③… ].append( …⑤… )
    return arcs_sortants
```

Pour rechercher le plus court chemin entre deux sommets du graphe, on utilise l'algorithme de Dijkstra qui prend en entrées un graphe orienté pondéré par des réels positifs et un sommet de départ. Il construit la liste des sommets du graphe et de leur distance minimale au sommet de départ, progressivement, par ordre croissant de leur distance. Au cours de l'exécution de cet algorithme, on utilise deux dictionnaires :

- `visites` : contient les sommets dont le poids minimal est définitivement établi (à la fin de l'exécution, ce dictionnaire contient la liste ordonnée des sommets du graphe et de leur distance minimale au sommet de départ) ;
- `a_visiter` : contient les sommets dont le poids est en cours de calcul.

Les clés de ces dictionnaires sont des sommets (noms de sites olympiques) et leurs valeurs sont, pour chaque sommet, le triplet : `(poids, precedent, sentier)` avec

- `poids` : longueur du chemin le plus court connu depuis le sommet de départ ;
- `precedent` : nom du sommet qui le précède sur ce chemin ;
- `sentier` : nom de l'arc utilisé depuis ce prédécesseur (à des fins d'affichage).

L'algorithme de Dijkstra procède ainsi :

- Initialisation : tous les sommets sont placés dans le dictionnaire `a_visiter`, le sommet de départ ayant pour poids zéro (sa distance à lui-même), les autres sommets ayant un poids infini. Les prédécesseurs et les noms de sentiers sont tous initialisés comme des chaînes de caractères vides, le dictionnaire `visites` est vide.
- Tant qu'il reste des sommets à visiter, itérer les étapes 1 à 3 ci-dessous :
  1. choisir le sommet N de poids le plus faible parmi les sommets `a_visiter` ;
  2. mettre à jour les voisins de N qui restent à visiter : si le poids actuel d'un voisin V est supérieur à la somme du poids de N et de la longueur du sentier S liant N à V, alors le poids de V devient cette somme, le prédécesseur de V devient N et le nom du dernier sentier menant à V devient S ;

3. retirer N de l'ensemble des sommets `a_visiter` et ajouter N à l'ensemble des sommets `visites`.

Une fois que tous les sommets ont été visités, on peut construire le chemin entre le sommet de départ et le sommet d'arrivée en partant de la fin, grâce à l'enregistrement des prédécesseurs.

II-2. Complétez la définition de la fonction `plus_proche`, qui prend en argument le dictionnaire des sommets `a_visiter` et qui renvoie le nom du sommet de plus faible poids. Cette fonction sera appelée par la fonction `meilleur_chemin` définie à la question suivante.

```
from math import inf # la constante inf représente (+ infini)
def plus_proche(a_visiter):
    min = …①…
    proche = ''
    for (p, (poids, pred, sent)) in a_visiter.items():
        if …②… <= …③… :
            ( …④… , …③… ) = ( …⑤… , …②… )
    return …⑥…
```

II-3. Complétez la définition de la fonction `meilleur_chemin` (voir document réponse), qui prend en entrée la liste de sentiers établie par le guide touristique, le nom d'un site olympique de `depart` et le nom d'un site olympique d'`arrivee`, qui calcule le plus court chemin par l'algorithme de Dijkstra puis appelle la fonction d'affichage définie à la question suivante.

II-4. La définition de la fonction `affichage` ci-dessous est incomplète ; les instructions `pass` utilisées sont des instructions vides qui n'ont aucun effet et dont certaines doivent être remplacées par les instructions suivantes :

```
①  print('aucun chemin du point', depart, 'au point', arrivee)
②  print('étape au point', arrivee, '(distance', dist, ')')
③  print('en passant par le chemin', nom)
④  affichage(visites, depart, depuis)
```

À quelles lignes faut-il placer les instructions ①, ②, ③ et ④ en remplacement de l'instruction `pass` dans la définition suivante :

```
1.  def affichage(visites,depart,arrivee):
2.      pass
3.      (dist,depuis,nom) = visites[arrivee]
4.      pass
5.      if dist == inf:
6.          pass
7.          return
8.      pass
9.      if depart != arrivee:
10.         pass
11.         pass
12.     pass
```

pour que l'appel `meilleur_chemin(sentiers_olympiques,'Champ de Mars', 'Grand Palais')` produise l'affichage suivant :

```
étape au point Champ de Mars (distance 0)
en passant par le chemin Sydney
étape au point Trocadéro (distance 100)
en passant par le chemin Elaine
étape au point La Concorde (distance 300)
en passant par le chemin Briana
étape au point Grand Palais (distance 400)
```