Correction du DS 4

Informatique pour tous, première année

Julien Reichert

Exercice 1

Question 1.1

Une version du cours (compatible avec les listes de listes ainsi que les tableaux) est recopiée ici pour les besoins de la comparaison avec la question suivante, et adaptée pour supporter d'autant mieux la « factorisation de code ».

```
def cherche_pivot(M, i):
    if M[i][i] == 0.:
        j = i+1
        while (j < len(M) and M[j][i] == 0.):
            j += 1
        return j
    else:
        return i

def echange(M, i, j):
    for ind in range(len(M[i])):
        M[i][ind], M[j][ind] = M[j][ind], M[i][ind]

def division(M, i, rapport):
    for k in range(len(M[i])):
        M[i][k] = M[i][k] / rapport

def transvection(M, i, j, rapport):
    for k in range(len(M[j])):
        M[j][k] = M[j][k] - rapport*M[i][k]

def tour_de_pivot(M, i, indice):
    echange(M, i, indice)
    rapport = M[i][i]
    division(M, i, rapport)
    for j in range(len(M)):
        if j != i:
            rapport = M[j][i]
            transvection(M, i, j, rapport)

def pivot(M):
    n = len(M)
    for i in range(n):
        j = cherche_pivot(M, i)
        assert j < n, "Le système n'est pas de Cramer"
        tour_de_pivot(M, i, j)
```

Question 1.2

Il suffit de modifier la fonction principale en gérant les cas où la recherche du pivot échoue, mais en faisant cette recherche sur la ligne et la colonne (à ce niveau, l'algorithme du pivot total est plus facile à adapter, et son inconvénient au niveau de la répercussion des échanges de colonnes sur le second membre n'intervient pas ici) :

```
def echangecolonne(M, i, j):
    for ind in range(len(M)):
        M[ind][i], M[ind][j] = M[ind][j], M[ind][i]

def rang(M): # D'abord copy.deepcopy si on ne veut pas modifier M
    n = len(M)
    rang = n # n ne doit pas être modifié, vu la comparaison
    for i in range(n):
        j = cherche_pivot(M, i)
        if j == n:
            testcolonne = i+1
            while testcolonne < n and M[i][testcolonne] == 0.:
                testcolonne += 1
            if testcolonne == n:
                rang -= 1 # la ligne i est définitivement nulle
            else:
                echangecolonne(M, i, testcolonne)
                j = i # pour le tour de pivot à venir
        else:
            tour_de_pivot(M, i, j)
    return rang
```

Une autre possibilité que de chercher la première colonne avec autre chose que `0` en ligne `i` est d'échanger avec la dernière colonne non connue comme nulle (initialement la dernière), en reculant d'un cran la localisation de ladite colonne et d'une unité le rang. Dans ce cas, la boucle principale devra être conditionnelle pour ne pas étudier deux fois la même colonne (condition d'arrêt : atteindre le rang théorique).

Exercice 2

Question 2.1

```
SELECT DISTINCT(Pays) FROM Spectacle
```

Question 2.2

```
SELECT COUNT(*) FROM Spectacle WHERE Type="Opéra"
```

Question 2.3

```
SELECT Personne FROM Distribution JOIN Spectacle ON Distribution.Id_spectacle=Spectacle.Id_spectacle
WHERE Role="Chef d'orchestre" AND Sur_scene=0 AND Titre="La Traviata"
```

Question 2.4

Requête facile pour donner un titre arbitraire :

```
SELECT Titre FROM Spectacle JOIN Distribution ON Spectacle.Id_spectacle=Distribution.Id_spectacle
GROUP BY Spectacle.Id_spectacle ORDER BY COUNT(DISTINCT Personne) DESC LIMIT 1
```

(Pas de pénalité en cas de `COUNT(*)`.)

Requête compliquée pour tous les donner :

```
SELECT Titre FROM Spectacle JOIN Distribution ON Spectacle.Id_spectacle=Distribution.Id_spectacle
GROUP BY Spectacle.Id_spectacle HAVING COUNT(DISTINCT Personne) =
(SELECT COUNT(DISTINCT Personne) FROM Spectacle JOIN Distribution
ON Spectacle.Id_spectacle=Distribution.Id_spectacle GROUP BY Spectacle.Id_spectacle
ORDER BY COUNT(DISTINCT Personne) DESC LIMIT 1)
```

Une autre version utilise une table dérivée et la fonction `MAX` pour éviter la combinaison `ORDER BY` et `LIMIT`, mais elle ne s'inspire pas de l'énoncé donc ne vient pas à l'esprit aussi bien (et l'avis général serait sans doute qu'elle est encore plus compliquée).

Question 2.5

Plutôt que de s'embêter à copier-coller, il suffit de glisser avant chaque `GROUP BY` la condition `WHERE Sur_scene`, assorti ou non d'une comparaison à `1`, identique à la comparaison d'un booléen à `True` en Python.

Question 2.6

Requête facile, la version compliquée s'obtenant par une adaptation analogue.

```
SELECT Ville FROM Spectacle GROUP BY Ville ORDER BY COUNT(DISTINCT Salle) DESC LIMIT 1
```

Remarque intéressante : j'ai pu observer que `COUNT(DISTINCT Salle)` éliminait automatiquement les cas où `Salle` valait `NULL`, ce qui est pratique car la gestion de cette valeur n'est pas au programme (et donc le barème ignorera des tentatives malheureuses).

Question 2.7

La première requête donne la liste des chefs d'orchestre (avec une vérification qu'il ne s'agit pas d'un nom de personnage, une vérification inutile dans la mesure où chaque spectacle a un chef d'orchestre et que la clé de la table exclut un doublon avec le même nom) qui ont dirigé au moins deux spectacles auxquels j'ai assisté, avec le nombre en question, celui-ci faisant office de critère de tri décroissant pour la présentation du résultat.

La deuxième requête donne la liste des artistes ayant figuré sur scène dans des spectacles auxquels j'ai assisté dans au moins deux villes différentes, ainsi que le nombre en question, avec le même tri que précédemment.

La troisième requête détermine le ou les compositeur(s) dont j'ai assisté au nombre maximal de représentations d'un opéra, ainsi que le nombre en question. Les opéras vus plusieurs fois comptent pour autant d'occurrences, et le nombre d'œuvres distinctes aurait été pris en compte si j'avais remplacé dans la requête toutes les étoiles dans les `COUNT` par `DISTINCT Titre`.