3b matrices_exercices_avec_corrige

October 28, 2020

0.0.1 Exercice 1

Ecrire une fonction `somme` qui prend en argument une matrice non vide de nombres entiers et renvoie la somme des nombres de cette matrice. Voici un exemple d'assertion qui doit être vérifiée par votre fonction :

```
assert(somme([[3, 8, 9, 11],
              [6, 8, 1, -4],
              [1, 1, 1, 0 ]]) == 45)
```

[1]:
```
def somme(mat):
    '''Effectue la somme des entiers de la matrice mat supposée non vide'''
    somme = 0
    for lig in range(len(mat)):
        for col in range(len(mat[0])):
            somme = somme + mat[lig][col]
    return somme
```

0.0.2 Exercice 2

Ecrire une fonction `max_somme` qui prend en argument une matrice non vide et renvoie la plus grande somme des différentes colonnes. Voici deux exemples d'assertions qui doivent être vérifiées par votre fonction :

```
assert(max_somme([[3, 8, 9, 11],
              [6, 8, 1, -4],
              [1, 1, 1, 0 ]]) == 17)    #colonne d'indice 1 : 8 + 8 + 1 = 17

assert(max_somme([[-3, -8, -9, -11],
                  [-6, -8, -1, 4  ],
                  [-1, -1, -1, 0  ]]) == -7)    #colonne d'indice 3 : -11 + 4 + 0 = -7
```

[4]:
```
def max_somme(mat):
    '''
    Renvoie la plus grande somme des différentes colonnes de
    la matrice mat supposée non vide
    '''
    # on initialise la somme maximale avec la somme de la colonne d'indice 0
    somme_colonne_zero = 0
```

```
    for lig in range(len(mat)):
        somme_colonne_zero = somme_colonne_zero + mat[lig][0]
    max_somme = somme_colonne_zero

    #  puis on passe en revue les colonnes suivantes (parcours par colonne)
    for col in range(1, len(mat[0])):
        somme_colonne = 0
        for lig in range(len(mat)):
            somme_colonne = somme_colonne + mat[lig][col]
        if somme_colonne > max_somme:
            max_somme = somme_colonne
    return max_somme
```

[5]:
```
assert(max_somme([[3, 8, 9, 11],
                  [6, 8, 1, -4],
                  [1, 1, 1, 0 ]]) == 17)    #colonne d'indice 1 : 8 + 8 + 1 = 17

assert(max_somme([[-3, -8, -9, -11],
                  [-6, -8, -1, 4  ],
                  [-1, -1, -1, 0  ]]) == -7)    #colonne d'indice 3 : -11 + 4 +␣
↪0 = -7
```

0.0.3 Exercice 3

Question 1 :

Ecrire une fonction `compresse` qui prend en argument une matrice non vide dont les éléments sont des entiers et modifie les éléments de la façon suivante : - si l'élément est positif, on diminue l'élément de une unité, - si l'élément est négatif, on augmente l'élément de une unité, - si l'élément est nul, on ne le modifie pas.

La fonction devra **muter** la matrice en place et ne **renverra** donc aucune valeur.

[6]:
```
def compresse(mat):
    for lig in range(len(mat)):
        for col in range(len(mat)):
            if mat[lig][col] < 0:
                mat[lig][col] = mat[lig][col] + 1
            elif mat[lig][col] > 0:
                mat[lig][col] = mat[lig][col] - 1
            else:
                pass
```

[7]:
```
ma_matrice = [[ 0,  1,  2,  3,  4, 5],
              [-1,  0,  1,  2,  3, 4],
              [-2, -1,  0,  1,  2, 3],
              [-3, -2, -1,  0,  1, 2],
              [-4, -3, -2, -1,  0, 1],
```

```
              [-5, -4, -3, -2, -1, 0]]
```

[8]:
```
compresse(ma_matrice)
ma_matrice
```

[8]:
```
[[0, 0, 1, 2, 3, 4],
 [0, 0, 0, 1, 2, 3],
 [-1, 0, 0, 0, 1, 2],
 [-2, -1, 0, 0, 0, 1],
 [-3, -2, -1, 0, 0, 0],
 [-4, -3, -2, -1, 0, 0]]
```

Question 2 :

Modifier la fonction précédente pour **qu'elle ne mute plus la matrice passée en argument qui devra rester intacte** mais qu'elle renvoie **une copie indépendante et compressée** de la matrice passée en argument.

[9]:
```
from copy import deepcopy

def compresse(mat):
    mat2 = deepcopy(mat)
    for lig in range(len(mat2)):
        for col in range(len(mat2)):
            if mat2[lig][col] < 0:
                mat2[lig][col] = mat2[lig][col] + 1
            elif mat2[lig][col] > 0:
                mat2[lig][col] = mat2[lig][col] - 1
            else:
                pass
    return mat2
```

[10]:
```
ma_matrice = [[ 0,  1,  2,  3,  4, 5],
              [-1,  0,  1,  2,  3, 4],
              [-2, -1,  0,  1,  2, 3],
              [-3, -2, -1,  0,  1, 2],
              [-4, -3, -2, -1,  0, 1],
              [-5, -4, -3, -2, -1, 0]]
```

[11]:
```
ma_matrice_compressee = compresse(ma_matrice)
ma_matrice_compressee
```

[11]:
```
[[0, 0, 1, 2, 3, 4],
 [0, 0, 0, 1, 2, 3],
 [-1, 0, 0, 0, 1, 2],
 [-2, -1, 0, 0, 0, 1],
 [-3, -2, -1, 0, 0, 0],
 [-4, -3, -2, -1, 0, 0]]
```

0.0.4 Exercice 4

Programmer une fonction `sablier` qui prend en argument un entier `n` strictement positif et renvoie une matrice de `n` lignes et `n` colonnes dont : - le quart supérieur contient des 8, - le quart inférieur contient des 1, - le reste de la matrice (dont les diagonales) contient des 0.

Voici deux exemples d'assertions vérifiée par la fonction quatre couleurs:

```
assert(sablier(15) ==   [[0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0],
                         [0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0],
                         [0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0],
                         [0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 8, 8, 8, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
                         [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
                         [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
                         [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]]

assert(sablier(8) ==    [[0, 8, 8, 8, 8, 8, 8, 0],
                         [0, 0, 8, 8, 8, 8, 0, 0],
                         [0, 0, 0, 8, 8, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 1, 1, 0, 0, 0],
                         [0, 0, 1, 1, 1, 1, 0, 0],
                         [0, 1, 1, 1, 1, 1, 1, 0]]
```

[18]:
```python
def sablier(n):
    m = [[0 for col in range(n)] for lig in range(n)]
    for lig in range(n//2):
        for col in range(lig+1, n-(lig+1)):
            m[lig][col] = 8

    for lig in range(n//2, n):
        for col in range(n-lig, lig):
            m[lig][col] = 1

    return m
```

[17]:
```python
m = sablier(16)
m
```

[17]:
```
[[0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0],
 [0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0],
 [0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0],
 [0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 8, 8, 8, 8, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 8, 8, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
 [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
 [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]]
```

[14]:
```
m = sablier(8)
m
```

[14]:
```
[[0, 8, 8, 8, 8, 8, 8, 0],
 [0, 0, 8, 8, 8, 8, 0, 0],
 [0, 0, 0, 8, 8, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 1, 0, 0, 0],
 [0, 0, 1, 1, 1, 1, 0, 0],
 [0, 1, 1, 1, 1, 1, 1, 0]]
```