Exercice : Plus grand carré dans une matrice – Eléments de correction

Soit une matrice carrée remplie de 0 ou 1.
On souhaite connaître la taille du plus grand carré de 1 dans cette matrice.
Par exemple, ce nombre est 2 pour la matrice suivante (carré en pointillé) :

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

La case de coordonnées (0, 0) est celle en haut à gauche.
La case de coordonnés (x,y) est celle sur la ligne x, colonne y.
On supposera que les indices en arguments des fonctions ne dépassent pas des tableaux ou matrices correspondants.

1. Définir en Python la matrice M précédente qui servira à faire les tests.

Solution

```
M = [[1, 0, 0, 0], [0, 0, 1, 1], [0, 1, 1, 1], [0, 1, 0, 1]]
```

Méthode naïve

2. Écrire une **fonction estCarree** telle que estCarree(M, x, y, k) détermine si la sous-matrice de m de taille k x k et dont la case en haut à gauche a pour coordonnées (x, y) ne contient que des 1.

Solution :

```
def estCarree(M, x, y, k):
    for i in range(x, x + k):
        for j in range(y, y + k):
            if M[i][j] != 1:
                return False
    return True
assert(estCarree(M, 1, 2, 2) and not estCarree(M, 1, 1, 2))
```

3. Écrire une **fonction contientCarre** telle que contientCarre(M, k) renvoie True si M contient un carré de 1 de taille k x k , False sinon.

Solution :

```
def contientCarre(M, k):
    n = len(M)
    for i in range(n - k + 1):
        for j in range(n - k + 1):
            if estCarree(M, i, j, k):
                return True
    return False
assert(contientCarre(M, 2) and not contientCarre(M, 3))
```

4. Écrire une **fonction PlusGrandCarre_v1** telle que PlusGrandCarre_v1(M) renvoie la taille maximale d'un carré de 1 contenu dans m.

Solution :

```
def PlusGrandCarre_v1(M):
    n = len(M)
    for k in range(n, 0, -1):
        if contientCarre(M, k):
            return k
    return 0
PlusGrandCarre_v1(M)   # 2
```

ITC $2^{ème}$ année S3 – Programmation dynamique –Plus grand carré dans une matrice

5. Quelle est la **complexité de PlusGrandCarre_v1(M) dans le pire cas** ?

Solution

- *estCarree(M, x, y, k)* est en $O(k^2)$
- *contientCarre(M, k)* appelle *estCarree* O(n) fois, donc est en $O(n^2k^2)$
- *PlusGrandCarre_v1(M)* appelle *contientCarre* pour k=1,2,…,n , donc est de complexité

$\sum_{k=1}^{n} O(n^2k^2) = O(n^3 \sum_{k=1}^{n} k^2)$
Or $\sum_{k=1}^{n} O(k^2) = n(n+1)(2n+1)/6 = O(n^3)$
La complexité totale est donc $O(n^6)$.

Amélioration en programmation dynamique

On construit une matrice c telle que c[x][y] est la taille maximale d'un carré de 1 dans M dont la case en bas à droite est M[x][y] (c'est à dire un carré de 1 qui contient M[x][y] mais aucun M[i][j] si i>x ou j>y)
Exemples pour la matrice M précédente :

- c[1][2] = 1
- c[2][3] = 2.

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

6. Que vaut **c[0][y] et c[x][0]** ?

Solution
c[0][y] = 0 si M[0][y] = 0 et c[0][y] = 1 sinon.
De même pour c[x][0].
Remarque : c[0][y] et c[x][0] sont donc les mêmes valeurs que m[0][y] et m[x][0], on peut donc initialiser c comme une copie de m.

7. Que vaut **c[x][y] si M[x][y] = 0** ?

Solution : c[x][y] = 0.

8. Que vaut **c[x][y] si M[x][y] = 1 ?**

Si M[x][y] = 1, c[x][y] = 1 + min(c[x-1][y], c[x][y-1], c[x-1][y-1]).

En déduire une **fonction PlusGrandCarre_v2** telle que PlusGrandCarre_v2(m) renvoie la taille maximum d'un carré de 1 contenu dans M, ainsi que les coordonnées de la case en haut à gauche d'un tel carré.

```
def PlusGrandCarre_v2(M):
    c = M.copy()
    maxi = 0
    indicesMaxi=[]
    for i in range(1,len(M)):
        for j in range(1,len(M[0])):
            if M[ i ][ j ] == 1:
                c[ i ][ j ] = 1 + min(c[ i – 1 ][ j ]  , c[ I ][ j - 1] , c[i - 1][j - 1])
                if c[ i ][ j ]>maxi :
                    maxi = c[ i ][ j ]
                    indicesMaxi=[i-maxi+1,j-maxi+1]
     return maxi, indicesMaxi
print(PlusGrandCarre_v2(M)) # 2, [1,2]
```

9. Quelle est la **complexité de PlusGrandCarre_v2(M),** en fonction des dimensions de M? Comparer avec PlusGrandCarre_v1(M).

Solution
PlusGrandCarre_v2(M) est en $O(n^2)$ du fait des deux boucles for imbriquées.
La complexité temporelle est donc meilleure que PlusGrandCarre_v1(M) qui est en $O(n^6)$.