4. evaluation_a_rendre_matrices

October 28, 2020

On traitera au choix **un seul** des deux exercices ci-dessous, le second étant moins facile (utilisation de la récursivité, pour ceux qui savent faire).

0.1 Cueillette de fraises (⋆)

On considère le problème de la cueillette de fraises mentionnée dans les travaux à faire pendant les vacances d'été (voir le module correspondant sur m@gistere : Evaluations à faire pendant les vacances > algorithmes débranchés, énoncé & correction).

Matrice représentant le champ de fraises Le champ de fraises de nb_lig lignes et nb_col colonnes sera représenté par une matrice `champ` : - `champ[i][j] = 0` indique qu'il n'y a aucune fraise à la case de la ligne `i`, colonne `j`, - `champ[i][j] = 1` indique qu'il y a une fraise à la case de la ligne `i`, colonne `j`.

Point méthode : pour générer un nombre aléatoire choisi au hasard parmi 0 ou 1 avec une probabilité de 0.37 (c-à-d une probabilité de 37 %) qu'il soit égal à 1 il suffit d'utiliser l'expression suivante (un `if ... else` codé sur une seule ligne, ce qui est possible en Python …):

[1]:
```
from random import random
```

[2]:
```
1 if random() < 0.37 else 0
```

[2]:
```
0
```

Matrice représentant les meilleures récoltes possibles On stockera dans une matrice `recoltes_max` - et plus précisément dans `recoltes_max[i][j]` - le nombre maximal de fraises qu'il est possible de cueillir pour le robot lorsque l'arrivée du trajet est à la case de la ligne `i` et de la colonne `j`.

Pour remplir la première ligne de `recoltes_max`, le nombre maximal de fraises est très simple puisque le trajet est imposé. C'est la même chose pour la première colonne. Voici un exemple :

```
champ = [[1, 0, 0, 1, 0, 1, 0, 1, 1],
         [0, 1, 0, 0, 1, 0, 0, 1, 0],
         [1, 0, 1, 1, 1, 0, 0, 1, 0],
         [0, 0, 1, 0, 0, 0, 0, 0, 1],
         [0, 0, 1, 1, 1, 1, 1, 0, 0],
         [1, 0, 1, 0, 1, 0, 1, 0, 0]]

recoltes_max = [[1, 1, 1, 2, 2, 3, 3, 4, 5],
```

```
[1, ...                  ],
[2, ...                  ],
[2, ...                  ],
[2, ...                  ],
[3, ...                  ]]
```

Une fois qu'on connait les récoltes maximales pour la première ligne et la première colonne, on peut alors compléter le reste de la matrice ligne par ligne (ou colonne par colonne) à partir du coin supérieur gauche : pour compléter `recoltes_max[lig][col]` on choisit la plus grande des deux valeurs `recoltes_max[lig][col-1]` et `recoltes_max[lig-1][col]` et on rajoute la valeur de `champ[lig][col]` (c'est à dire 1 s'il y a une fraise et 0 sinon).

Ainsi, pour calculer un élément de `recoltes_max` on se sert des élements du dessus et de gauche ainsi que de l'élément correspondant de `champ`.

Point méthode : pour obtenir le maximum de deux nombres on utilisera la fonction python `max` :

[3]:
```
max(7, 14, 89, 43)
```

[3]: 89

Question 1 :

Ecrire une fonction `creer_champ(nb_lig, nb_col, p)` qui renvoie une matrice de `nb_lig` lignes et `nb_col` colonnes dont chaque élément est choisi aléatoirement parmi 0 ou 1 avec une probabilité `p` d'être égal à 1. On pourra consulter avec profit le II du cours (initialisation de matrices) et utiliser les listes définies par compréhension.

Voici deux exemples d'utilisation de cette fonction :

```
>>> champ = creer_champ(5, 9, 0.8)
>>> champ
[[0, 1, 1, 0, 0, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 0, 1, 0, 1],
 [0, 1, 1, 1, 0, 1, 1, 1, 1],
 [1, 0, 1, 1, 1, 1, 1, 1, 1],
 [0, 1, 0, 1, 1, 1, 1, 1, 0]]

>>> champ = creer_champ(13, 19, 0.2)
>>> champ
[[0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0],
 [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1],
 [0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1],
 [0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1],
 [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
```

```
 [1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]]
```

[ ]:
```
# votre code ici
```

Question 2 :

Ecrire une fonction `calculer_recoltes_max` qui prend en argument une matrice (non vide) représentant un champ de fraises et renvoie la matrice des recoltes maximales possibles associée.

[ ]:
```
# votre code ici
```

Question 3 :

Tester la fonction `calculer_recoltes_max` grâce aux assertions suivantes :

[ ]:
```
champ1 = [[0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 0, 0],
 [0, 0, 1, 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, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 1, 1, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 1, 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]]

rec1 = [[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3],
 [0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4],
 [0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4],
 [0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 4, 4],
 [0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 4, 5],
 [1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 4, 5],
 [1, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 5],
 [1, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5],
 [1, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5],
 [1, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5],
 [1, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5],
 [1, 1, 2, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5],
 [1, 2, 2, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5],
 [1, 2, 2, 4, 4, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6]]

champ2 = [[1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1],
```

```
 [1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1],
 [0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1],
 [0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1],
 [1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1],
 [1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1]]

rec2 = [[1, 2, 3, 3, 4, 5, 5, 6, 7, 7, 8, 9],
 [2, 3, 3, 4, 5, 5, 6, 7, 8, 9, 10, 11],
 [2, 4, 5, 5, 6, 6, 7, 7, 8, 10, 11, 12],
 [2, 5, 6, 7, 8, 9, 10, 11, 11, 12, 13, 13],
 [3, 6, 7, 8, 9, 10, 11, 12, 13, 13, 14, 15],
 [4, 6, 8, 9, 10, 10, 11, 12, 14, 15, 15, 16],
 [4, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18],
 [5, 8, 9, 10, 11, 13, 14, 15, 16, 16, 17, 19],
 [6, 8, 9, 11, 12, 13, 15, 16, 17, 17, 18, 20]]

assert(calculer_recoltes_max(champ1) == rec1)
assert(calculer_recoltes_max(champ2) == rec2)
```

Question 4 :

On suppose maintenant que chaque "case" du champ de fraises peut contenir 0, 1 ou plusieurs fraises.
Y a-t-il des modifications à apporter à la fonction `calculer_recoltes_max` ? Si oui, lesquelles ? Non, aucune modification à apporter.

1 Classement en chaîne (⋆⋆)

On se base sur l'exercice d'algorithmique débranchée "Classement en chaîne" (voir Module "Evaluations à faire pendant les vacances > Algorithmique débranchée").

Question 1 :
Ecrire une fonction `creer_matrice_matches(n)` qui renvoie une matrice `matches` de `n` lignes et `n` colonnes générée aléatoirement et telle que : - `matches[lig][col] = True` lorsque l'équipe numéro `lig` a gagné contre l'équipe numéro `col`, - `matches[lig][col] = False` lorsque l'équipe numéro `lig` a perdu contre l'équipe numéro `col`.

La diagonale de la matrice sera par convention remplie avec des `False`. On fera aussi attention au fait que si une équipe numéro `x` a gagné contre une équipe numéro `y`, l'équipe numéro `y` a perdu contre l'équipe numéro `x`. Autrement dit si `matches[x][y] == True` alors `matches[y][x] == False`.

[ ]:
```
from random import choice

# votre code ici
```

[ ]:
```
matches = creer_matrice_matches(10)
matches
```

Question 2 :
Ecrire une fonction récursive `classement_en_chaine(matches, k)` qui prend en argument une matrice correspondant aux résultats des matches de `n` équipes et renvoie un tableau `chaine` de taille `k+1` correspondant à un classement en chaîne possible des `k+1` premières équipes (de `0` à `k`).

On pourra éventuellement utiliser une fonction auxiliaire `inserer` qui se chargera de l'insertion de l'équipe numéro `k` dans le classement des équipes de numéros `0` à `k-1`.

[Plus formellement, `chaine` comporte tous les entiers de `0` à `k` et pour tout indice `i < k`, l'équipe numéro `chaine[i]` a gagné son match contre l'équipe `chaine[i+1]`.]

[ ]:
```
# votre code ici
```

[ ]:
```
classement = classement_en_chaine(matches, len(matches)-1)
classement
```

[ ]:
```

```