Info2A-PSI - Informatique de Tronc Commun – DS01

Exercice 1

On considère un labyrinthe construit sur une grille rectangulaire $n \times p$ (n lignes et p colonnes).
Chaque case de cette grille est repérée par ses coordonnées, ayant la forme d'un couple (i, j) où i et j désignent, respectivement, la ligne et la colonne à l'intersection desquelles la case se trouve.
Les lignes sont numérotées de 0 à $n-1$, et les colonnes de 0 à $p-1$.
On suppose que n et p sont des entiers naturels supérieurs ou égaux à 1.

Par exemple, pour une grille 3×4 :

i \ j	0	1	2	3
0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)

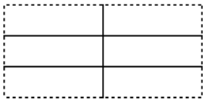
Q1. Définir une fonction `liste_grille(n, p)` renvoyant la liste des couples de coordonnées pour les cases d'une grille $n \times p$.
On devra avoir par exemple :

```
>>> liste_grille(3, 4)
[(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0),
(2, 1), (2, 2), (2, 3)]
```

On suppose dans toute la suite que deux variables globales, `n` et `p` ont été définies, ainsi qu'une liste de liste `grille`, dont les éléments sont les couples de coordonnées des cases d'une grille $n \times p$.
Par exemple, on aura posé :

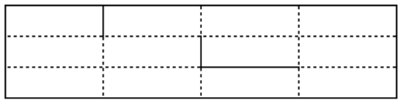
```
n, p = 3, 4
grille = liste_grille(n, p)
```

Chaque case possède entre deux et quatre voisines, situées à droite, au-dessus, à gauche et au-dessous d'elle.
Entre une case et chacune de ses voisines, il existe un mur, qui peut être ouvert ou fermé.
Par exemple, sur une grille 3×2, il existe sept murs, trois verticaux, quatre horizontaux, qui apparaissent ci-dessous en traits pleins :



Q2. Combien existe-t-il de murs verticaux entre cases voisines sur une grille $n \times p$? Combien de murs horizontaux ?

Les bords de la grille sont considérés comme des murs fermés.
Sur la figure suivante, représentant une grille 3×4, trois murs entre cases voisines ont été fermés, les autres ont été laissés ouverts :



Chaque mur est représenté par le couple des coordonnées des cases qu'il sépare :
- les coordonnées des cases à gauche et à droite du mur, dans cet ordre, pour un mur vertical.
  Par exemple $\big((0,0),(0,1)\big)$ et $\big((1,1),(1,2)\big)$ pour les murs verticaux de la figure ci-dessus.

- les coordonnées des cases au-dessus et au-dessous du mur, dans cet ordre, pour un mur horizontal. Par exemple $((1,2),(2,2))$ pour le mur horizontal de la figure ci-dessus

Q3. Construire la liste des couples de coordonnées des murs verticaux, `mursV`, pour une grille $n \times p$.
On aura par exemple, si $n = 2$ et $p = 3$:

```
>>> mursV
[((0, 0), (0, 1)), ((0, 1), (0, 2)), ((1, 0), (1, 1)), ((1, 1), (1, 2))]
```

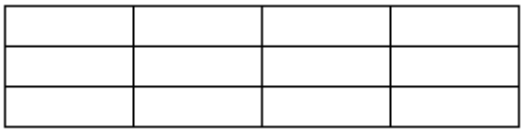
On suppose, les valeurs de n et p ayant été fixées, que la liste des murs horizontaux, `mursH`, a été définie.
Q4. Donner une instruction permettant de définir la liste `murs` de tous les murs de la grille à partir des deux listes `mursV` et `mursH` ?

On souhaite se donner un dictionnaire, `Detat_mur`, dont les clés sont les couples de coordonnées définissant les murs entre cases voisines de la grille, avec pour valeur associée un booléen indiquant si le mur est ouvert ou fermé : `True` si le mur est ouvert, `False` si le mur est fermé.
Initialement, tous les murs entre cases voisines sont considérés comme fermés.
Q5. Définir le dictionnaire `Detat_mur` à partir de la liste `murs`, dans son état initial, où tous les murs entre cases voisines sont fermés.
Pour une grille 2×3 on devra avoir :

```
>>> Detat_mur
{((0, 0), (1, 0)): False, ((0, 1), (1, 1)): False, ((0, 0), (0, 1)): False, ((0,
1), (0, 2)): False, ((1, 0), (1, 1)): False, ((1, 1), (1, 2)): False}
```

Génération de labyrinthe

On souhaite mettre en place un algorithme de génération de labyrinthe.
Pour cela, on commence par définir une numérotation des cases d'une grille $n \times p$, en faisant affectant à une case de coordonnées (i, j) le numéro $a = pi + j$.
Q6. Dessiner sur votre copie une grille de taille 3×4 en écrivant dans chaque case le numéro qui lui est affecté :



Q7. Pour une grille $n \times p$, quels sont les entiers utilisés pour numéroter toutes les cases de la grille ?
Q8. Définir un dictionnaire `Dnumeros`, dont les clés sont les couples de coordonnées des cases de la grille, supposés déjà définis dans la liste `grille`, avec pour valeur associée, le numéro de la case tel que défini ci-avant. On utilisera la liste `grille` pour construire le dictionnaire `Dnumeros`.
Par exemple, pour une grille 2×3, on devra avoir :

```
>>> Dnumeros
{(0, 0): 0, (0, 1): 1, (0, 2): 2, (1, 0): 3, (1, 1): 4, (1, 2): 5}
```

L'algorithme de génération de graphe consiste à répéter les étapes suivantes :

1. choisir aléatoirement un mur fermé entre deux cases voisines
2. si les cases voisines entre lesquelles le mur fermé choisi se trouve portent des numéros différents, a et b, $a < b$:
   - ouvrir le mur
   - affecter le numéro a à toutes les cellules portant le numéro b.

Q9. Sur une grille 2×3, répéter deux fois les étapes ci-dessus à partir d'une grille dont tous les murs sont fermés, en choisissant à votre guise le mur ouvert à chaque étape, et en donnant initialement et après chaque étape les numéros des six cases de la grille. Le contour de la grille et les murs fermés seront figurés par des traits pleins.

Q10. Après k étapes, $k \geq 0$, combien y a-t-il de numéros différents affectés aux cases d'une grille $n \times p$ dont tous les murs sont initialement fermés ?
Q11. Combien d'étapes faut-il pour qu'un seul et même numéro soit affecté à toutes les cases d'une grille $n \times p$?

Afin de mettre en place l'algorithme, on souhaite définir une liste des murs fermés initialement.
Q12. Définir une liste `murs_fermes`, qui soit une copie de la liste `murs` définies à la question **Q4**, à l'aide d'une compréhension de listes.

On rappelle qu'un appel `randrange(a, b)` à la fonction `randrange` du module `random`, a et b entiers, $a < b$, renvoie un entier tiré aléatoirement dans l'intervalle d'entiers $[\![a, b-1]\!]$.
Q13. Par quelle instruction peut-on importer la fonction `randrange` ?
Q14. Définir une fonction `extraire1(L)`, prenant en argument une liste non vide `L`, permettant, par appel à la fonction `randrange`, de choisir au hasard un élément de `L`. La fonction `extraire1`, retirera de la liste `L` l'élément choisi et le renverra.
Q15. Définir une fonction `teste_cases(Dnumeros, mur)`, prenant en argument un dictionnaire, `Dnumeros`, donnant les numéros affectés aux cases d'une grille, et un couple de coordonnées, `mur`, décrivant un mur et renvoyant un booléen indiquant si les cases séparées par le mur `mur`, portent le même numéro ou non.

On souhaite implémenter l'algorithme décrit ci-dessus.
On suppose définies les variables, `n`, `p`, `grille`, `Detat_mur`, `murs_fermes`, initialisés pour une grille $n \times p$.

On donne le squelette d'implémentation suivante :

```
nb_iterations = 0
while [1]:
    mur = extraire1(murs_fermes)
    while [2]:
        mur = extraire1(murs_fermes)
   case1, case2 = mur
   [3]
   [4]
   [5]
   [6]
```

Q16. Quel test écrire en [2] de sorte que le tirage aléatoire d'un mur parmi les murs fermés soit répété jusqu'à avoir tiré un mur fermé séparant des cases portant des numéros distincts ?
Q17. Quelle instruction écrire en [3] pour mettre à jour le dictionnaire `Detat_mur`, afin de prendre en compte l'ouverture du mur `mur`.
Q18. Écrire une séquence d'instructions à écrire en [4] afin de déterminer le plus petit numéro et le plus grand numéro parmi les numéros portés par les cases de la grille, `case1` et `case2`, séparées par le mur fermé choisi, on enregistrera ces numéros dans deux variables `nummin` et `nummax`.
Q19. Écrire une séquence d'instructions à écrire en [5] afin d'affecter dans le dictionnaire `Dnumeros`, le numéro `nummin` à toutes les cases portant le numéro `nummax`.
Q20. Donner le test à écrire en [1] et l'instruction à écrire en [6] pour s'assurer que l'algorithme termine lorsque toutes les cases portent le même numéro.
Q21. [NE PAS FAIRE] Écrire une séquence d'instructions permettant d'afficher à l'aide de matplotlib le labyrinthe généré.

Exercice 2

On cherche à calculer la valeur de $\binom{n}{k}$ de façon récursive, pour tout couple (k, n) tel que $0 \leq k \leq n$.
On rappelle la formule de Pascal, pour tout couple (k, n) tel que $0 \leq k \leq n$:

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}.$$

On rappelle que, pour tout $n \geq 0$:

$$\binom{n}{0} = \binom{n}{n} = 1$$

Q1. Définir une fonction récursive `binrec(n, k)` renvoyant pour tout couple d'entiers naturels (n, k), la valeur de $\binom{n}{k}$.

Q2. Rappeler les défauts de cette implémentation récursive.

Afin de corriger les défauts précédents, on souhaite mettre en place une méthode TOP-DOWN de programmation dynamique.

Q3. Compléter l'implémentation suivante, dans laquelle on définit un dictionnaire `calcules`, dont les clés sont des couples (n, k), pour lesquels la valeur de $\binom{n}{k}$ a déjà été calculée, avec pour valeur associée la valeur calculée de $\binom{n}{k}$.

Dans cette implémentation la fonction auxiliaire `aux` est récursive et un appel `aux(n, k)` renvoie la valeur de $\binom{n}{k}$, pour (k, n) tel que $0 \leq k \leq n$.

```
def bindyn1(n, k):
    calcules = {}
    def aux(k, n):
        if (k, n) in calcules:
            ...
            return ...
        else:
            if ... : # traitement des cas de base (sans appel récursif)
                ...
            else:
                valeur = ... # calcul récursif de k parmi n
                # actualisation du dictionnaire calcules
                ...
                #
                return valeur
    # appel à la fonction aux
    return aux(...)
```

On souhaite maintenant mettre en place une méthode BOTTOM-UP de programmation dynamique pour réaliser le calcul de $\binom{n}{k}$, pour (k, n) tel que $0 \leq k \leq n$.

Pour cela, on définit un tableau `tab` à deux dimensions à $n+1$ lignes et $n+1$ colonnes remplis initialement de zéros.

Q4. Définir en Python le tableau `tab`, en supposant qu'une variable n donnant la valeur de n a été définie.

Q5. À l'aide d'une ou plusieurs boucles `for`, écrire une séquence d'instruction permettant de compléter successivement les lignes d'indice i, du tableau `tab`, pour i variant de 0 à n, dans cet ordre, de sorte que, pour tout i compris entre 0 et n, et pour tout k compris entre 0 et i, `tab[i][k]` ait pour valeur $\binom{n}{i}$.

On rappelle que, par convention, pour tout couple d'entiers naturels tels que $k > n$, on a $\binom{n}{k} = 0$.

Exercice 3

On considère une base de données comportant une seule table nommée **ordinateurs**, décrivant les ordinateurs situés sur un même réseau informatique.
Chaque enregistrement de la table décrit un ordinateur présent sur le réseau.
Les attributs de la table sont

- nom, donnant le nom du poste ;
- ip, donnant l'adresse IP du poste sous la forme d'une chaîne de caractères ;
- marque, donnant le fabricant de la machine ;
- modele, donnant le modèle de la machine.
- ram, donnant la mémoire vive de l'ordinateur en Go

Q1. Suggérer une clé primaire possible pour la relation **ordinateurs**. Justifier.
Q2. Donner le domaine de chacun des attributs.
Q3. Écrire une requête SQL donnant le nombre de postes sur le réseau.
Q4. Écrire une requête SQL affichant le nom de toutes les machines de marque HP.
Q5. Écrire une requête SQL affichant toutes les informations sur les machines de marque HP et possédant au moins 8 Go de RAM. On ordonner les résultats par RAM décroissante.
Q6. Écrire une requête SQL affichant, sans doublons, les différents fabricants de toutes les machines présentes sur le réseau.
Q7. Écrire une requête SQL renvoyant le nombre de fabricants différents représentés au sein du parc de machines.
Q8. Écrire une requête SQL renvoyant la valeur la plus petite quantité de RAM disponible sur une machine du parc
Q9. Écrire une requête SQL, utilisant une sous-requête, et affichant le nom des machines dont la RAM est supérieure à la moyenne des RAM des machines du parc.
Q10. Écrire une requête SQL affichant le nom des différents constructeurs représentés sur le réseau, avec le nombre de machines présentes sur le réseau fabriquées par ce constructeur.
Q11. Écrire une requête SQL affichant le nom des constructeurs représentés sur le réseau, pour lesquels il existe moins de dix machines sur le réseau fabriquées par ce constructeur.

Contrairement à ce qui devrait être, il se trouve qu'il existe sur le réseau au moins deux machines auxquelles la même adresse IP.
Q12. Écrire une requête, utilisant une sous-requête, affichant le nom et l'IP des machines dont l'IP a été attribuée plusieurs fois.