Centres étrangers – 2024 – sujet2

Exercice 1 (6 points)

Cet exercice porte sur la programmation en Python en général, la programmation orientée objet et la récursivité.

On se déplace dans une grille rectangulaire. On s'intéresse aux chemins dont le départ est sur la case en haut à gauche et l'arrivée en bas à droite. Les seuls déplacements autorisés sont composés de déplacements élémentaires d'une case vers le bas ou d'une case vers la droite.

Un itinéraire est noté sous la forme d'une suite de lettres : • `D` pour

un déplacement vers la droite d'une case ;

• `B` pour un déplacement vers le bas d'une case.

Le nombre de caractères `D` est la longueur de l'itinéraire. Le nombre de caractères `B` est sa largeur.

Ainsi l'itinéraire `'DDBDBBDDDDB'` a pour longueur 7 et pour largeur 4. Sa représentation graphique est :

```
S * *
    * *
      *
      * * * * *
              E
```

- `S` représente la case de départ (start). Ses coordonnées sont $(0 ; 0)$;
- `*` représente les cases visitées ;
- `E` représente la case d'arrivée (end).

Partie A – Programmation orientée objet

On représente un itinéraire avec la classe `Chemin` suivante :

```
1  class Chemin:
2
3      def __init__(self, itineraire):
4          self.itineraire = itineraire
5          longueur, largeur = 0, 0
6          for direction in self.itineraire:
7              if direction == "D":
8                  longueur = longueur + 1
9              if direction == "B":
10                 largeur = largeur +1
11         self.longueur = longueur
12         self.largeur = largeur
13         self.grille = [['.' for i in range(longueur+1)] for j in
                          range(largeur+1)]
14
15     def remplir_grille(self):
16         i, j = 0, 0                # Position initiale
17         self.grille[0][0] = 'S'    # Case de départ marquée d'un S
18         for direction in ...:
19             if direction == 'D':
20                 ... = ...          # Déplacement vers la droite
21             elif direction == 'B':
22                 ... = ...          # Déplacement vers le bas
23             self.grille[i][j] = '*'  # Marquer le chemin avec '*'
24         self.grille[self.largeur][self.longueur] = 'E' # Case
                                                  d'arrivée marquée d'un E
```

1. Donner un attribut et une méthode de la classe `Chemin`.

On exécute le code ci-dessous dans la console Python :

```
chemin_1 = Chemin("DDBDBBDDDDB")
a = chemin_1.largeur
b = chemin_1.longueur
```

2. Préciser les valeurs contenues dans chacune des variables `a` et `b`.

3. Recopier et compléter la méthode `remplir_grille` qui remplace les `'.'` par des `'*'` pour signifier que le déplacement est passé par cette cellule du tableau.

4. Écrire une méthode `get_dimensions` de la classe `Chemin` qui renvoie la longueur et la largeur de l'itinéraire sous la forme d'un `tuple`.

5. Écrire une méthode `tracer_chemin` de la classe `Chemin` qui affiche une représentation graphique d'un itinéraire.

Partie B – Génération aléatoire d'itinéraires

On souhaite créer des chemins de façon aléatoire. Pour cela, on utilise la méthode `choice` de la bibliothèque `random` dont on fournit ci-dessous la documentation.

```
`random.choice(sequence : list)`
Renvoie un élément choisi dans une liste non vide.
Si la population est vide, lève `IndexError`.
```

On rappelle que l'opérateur `*` permet de répéter une chaîne de caractères. Par exemple, on a :

```
>>> "Hello world ! " * 3
'Hello world ! Hello world ! Hello world ! '
```

L'algorithme proposé est le suivant :

- on initialise :
  - une variable `itineraire` comme une chaîne de caractères vide,
  - les variables `i` et `j` à 0 ;
- tant que l'on n'est pas sur la dernière ligne ou la dernière colonne du tableau :
  - on tire au sort entre un déplacement à droite ou en bas,
  - le déplacement est concaténé à la chaîne de caractères `itineraire`,
  - si le déplacement est vers la droite, alors `j` est incrémenté de 1,
  - si le déplacement est vers le bas, alors `i` est incrémenté de 1 ;
- il reste à terminer le chemin en complétant par des déplacements afin d'atteindre la cellule en bas à droite.

6. Écrire les lignes manquantes dans le code ci-dessous. Le nombre de lignes effacées dans le code n'est pas indicatif.

```
def itineraire_aleatoire(m, n):
    itineraire = ''
    i, j = 0, 0
    while i != m and j != n
      ... # il y a plusieurs lignes
    if i == m:
        itineraire = itineraire + 'D'*(n-j)
    if j == n:
        itineraire = itineraire + 'B'*(m-i)
    return itineraire
```

Partie C – Calcul du nombre de chemins possibles

Soit m et n deux entiers naturels non nuls. On se place dans dans le contexte d'un itinéraire de longueur m et de largeur n de dimension $m \times n$.

On note `N(m, n)` le nombre de chemins distincts respectant les contraintes de l'exercice.

7. Pour un itinéraire de dimension $1 \times n$ justifier, éventuellement à l'aide d'un exemple, qu'il y a un seul chemin, c'est-à-dire que, quel que soit n entier naturel, on a $N(1, n) = 1$.

De même, $N(m, 1) = 1$.

8. Justifier que $N(m, n) = N(m-1, n) + N(m, n-1)$.

9. En utilisant les questions précédentes, écrire une fonction récursive `nombre_chemins(m, n)` qui renvoie le nombre de chemins possibles dans une grille rectangulaire de dimension $m \times n$.