
ITC 2° ANNÉE, TP 1

RÉVISIONS : RÉCURSIVITÉ, DICTIONNAIRES

Exercice 1 (Fibonacci)

On appelle *suite de Fibonacci* la suite $(u_n)_{n \in \mathbb{N}}$ définie par

$$\begin{cases} u_0 = 1 \\ u_1 = 1 \\ \forall n > 1, u_n = u_{n-1} + u_{n-2} \end{cases}$$

1. Codez une fonction récursive `fibonacci(n)` qui prend en entrée un entier positif n et renvoie la valeur de u_n .
2. Jusqu'à quel n pouvez-vous appeler votre fonction sans qu'elle mette plus d'une seconde à renvoyer une valeur ?

Exercice 2 (Rendu de monnaie)

À une époque au Royaume-Uni, les différents types de pièces qui existaient étaient les pièces de 1, 3, 6, 12, 24 et 30 centimes.

Si on voulait rendre à quelqu'un 49 centimes en utilisant le moins de pièces possibles, on pouvait le faire en utilisant 3 pièces : 2 pièces de 24 centimes et une pièce de 1 centime.

On généralise cette anecdote par le problème algorithmique suivant :

Entrées :

- Un nombre r de centimes à rendre.
- Un système monétaire P qui décrit l'ensemble des valeurs possibles pour chaque pièce.

Sortie : Le nombre minimum n de pièces $(p_i)_{i \in \llbracket 1, n \rrbracket}$ permettant de rendre r centimes (c'est à dire telles que $\sum_{i=1}^n p_i = r$ et $\forall i \in \llbracket 1, n \rrbracket, p_i \in P$).

1. Écrire une fonction récursive `rendu_monnaie(a_rendre, valeurs_possibles)` qui prend en entrée un nombre positif de centimes à rendre (`a_rendre`) et une liste des valeurs possibles de chaque pièce (`valeurs_possibles`) et renvoie le nombre minimal de pièces qu'il faut pour rendre la monnaie.

Par exemple, l'appel `rendu_monnaie(49, [1, 3, 6, 12, 24, 30])` doit renvoyer 3.

2. (Question bonus) On considère désormais le système de rendu de monnaie européen (1, 2, 5, 10, 20, 50, 100 et 200 centimes). Connaissez-vous un algorithme plus simple que celui de la question précédente pour rendre la monnaie ?
3. (Question bonus) Si on essaye de rendre 49 centimes avec le système à 1, 3, 6, 12, 24 et 30 centimes et l'algorithme de la question 2, combien de pièces rend-t-on ? Est-ce correct ?

Exercice 3 (Bonus : Labyrinthe)

J'ai sur mon ordinateur un labyrinthe carré de 100 cases. Il s'agit d'un labyrinthe à chemin unique, sans embranchements. Je note chacune des cases du labyrinthe par ses coordonnées (ligne, colonne) où ligne et colonne sont chacune comprise entre 0 inclus et 10 exclu. Je parcours intégralement ce labyrinthe. Je note mon parcours dans le dictionnaire ci-dessous, où à chaque case par laquelle je suis passé, je note depuis quelle case je suis arrivé. Le parcours commence à la case (0, 0), qui n'est pas une clef du dictionnaire.

```
parcours = {(0, 1): (0, 0), (5, 7): (6, 7), (6, 7): (6, 8),
            (5, 2): (5, 3), (9, 1): (8, 1), (4, 7): (5, 7),
            (9, 2): (9, 1), (8, 6): (9, 6), (9, 5): (9, 4),
            (7, 8): (8, 8), (5, 4): (4, 4), (9, 3): (9, 2),
            (3, 9): (4, 9), (9, 6): (9, 5), (4, 4): (3, 4),
            (9, 4): (9, 3), (8, 8): (8, 7), (1, 2): (0, 2),
            (2, 2): (1, 2), (5, 1): (5, 2), (5, 3): (5, 4),
            (4, 9): (4, 8), (3, 2): (2, 2), (6, 8): (7, 8),
            (6, 1): (5, 1), (8, 7): (8, 6), (3, 4): (3, 3),
            (7, 1): (6, 1), (8, 1): (7, 1), (3, 3): (3, 2),
            (0, 2): (0, 1), (4, 8): (4, 7)}
```

1. Écrire une fonction `ordre_parcours` qui prend en entrée un tel dictionnaire, et renvoi en sortie la liste des cases, dans l'ordre où elles ont été parcourues.
2. Écrire une procédure `dessine_parcours` qui prend en paramètre un tel dictionnaire, et dessine le labyrinthe correspondant (une aide au dessin est donnée ci-dessous).

Pour dessiner, on pourra par exemple utiliser la méthode `spy` de `pyplot` qui prend en argument une matrice à deux dimensions et qui dessine les cases non-nulles.

Par exemple, le code ci-dessous dessine un smiley.

```
import numpy as np
import matplotlib.pyplot as plt

# Une matrice de taille 5*7
canevas = np.zeros([5,7])

# Pixels a colorier
coords_pixels_noirs = [
    (0, 2), (1, 2), # oeil gauche
    (0, 4), (1, 4), # oeil droit
    (2, 0), (3, 1), (4, 2),
    (4, 3), (4, 4), (3, 5), (2, 6) ]

# Coloriage
for coord in coords_pixels_noirs:
    canevas[coord] = 1

# Affichage
plt.spy(canevas)
plt.show()
```

