TD IPT^2 n° 4: Récursivité 2

Exemples approfondis

Exercice n°1: Résolution de Sudoku par "backtracking"

Définition Wikipedia: *Le retour sur trace, ou backtracking en anglais, est un algorithme qui consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage. La méthode des essais et erreurs constitue un exemple simple de backtracking. Le terme est surtout utilisé en programmation, où il désigne une stratégie pour trouver des solutions à des problèmes de satisfaction de contraintes..*

On propose d'exploiter la technique *retour sur trace* dans la résolution récursive d'une grille du jeu logique **Sudoku** dont on rappelle brièvement le principe:

- On dispose d'une grille $9 \times 9 = 81$ cases, divisée en 9 régions de 9 cases chacune.
- Initialement, seulement certaines cases de la grille sont remplies par un chiffre (1 à 9), les autres étant vides.
- Le joueur doit remplir chaque case vide par un chiffre $k \in [\![1, 9]\!]$ en respectant 3 contraintes:
  - aucun chiffre n'a le droit d'apparaître deux fois dans la même ligne.
  - aucun chiffre n'a le droit d'apparaître deux fois dans la même colonne.
  - aucun chiffre n'a le droit d'apparaître deux fois dans la même région.

On se donne par exemple à résoudre la grille suivante:

	8							
7		6	4		5			
5				6		7		2
			1	3			9	
8	2						1	3
	3			4	6			
1		9		8				7
			5		4	8		9
							2	

Région 1 (0,0)	Région 2 (0,1)	Région 3 (0,2)
Région 4 (1,0)	Région 5 (1,1)	Région 6 (1,2)
Région 7 (2,0)	Région 8 (2,1)	Région 9 (2,2)

Diverses méthodes algorithmiques de résolution de Sudoku existent, parmi lesquelles certaines simulent les stratégies humaines (techniques euristiques). Compte tenu de la puissance des ordinateurs modernes, le backtracking qui consiste à explorer toutes les possibilités et à revenir en arrière si nécesaire est encore la plus simple à programmer,

et donnera pleine satisfaction.

Mise en oeuvre du backtracking:

On identifie les cases vides, puis la première d'entre-elles est remplie avec un 1. On teste ensuite si cette valeur convient aux 3 règles du jeu énoncées précédemment (test de satisfaction des contraintes); si c'est le cas alors on teste la case vide suivante; sinon, on vérifie si 2 convient, puis 3, etc... En somme, dès qu'une incompatibilité est constatée, il est nécessaire de revenir en arrière et d'augmenter d'une unité la valeur du chiffre précédemment essayé. Ce retour en arrière persiste tant qu'il subsiste une incompatibilité concernant le choix d'un chiffre. Cette méthode permet de tester toutes les configurations possibles et finit donc par trouver la bonne solution.

En outre, on se propose de mettre à profit la récursivité dans la rédaction de la solution.

❶ Ecrire une fonction itérative `test(M,i,j,chiffre)` qui teste la validité du chiffre `chiffre` à la position (i, j) dans la matrice M de Sudoku. La fonction renverra `True` si les conditions de validité sont vérifiées, et `False` sinon. Pour la vérification de l'appartenance de `chiffre` à une région, on pourra remarquer que si (i, j) sont les indices de l'élément testé, alors $(i_r = \lfloor i/3 \rfloor, j_r = \lfloor j/3 \rfloor)$ sont les indices de cette région, et $\left(i_p = 3\lfloor i/3 \rfloor, j_p = 3\lfloor j/3 \rfloor\right)$ les indices du premier élément de cette région.

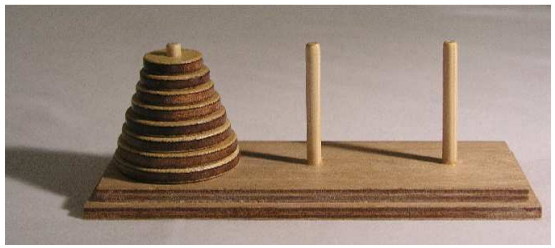
❷ Ecrire une fonction itérative `rechzero(M)` qui recherche la première occurrence d'une case vide c'est à dire un 0 dans la matrice et renvoie sa position sous forme d'un tuple (i, j) le cas échéant, ou bien (10,10) sinon.

❸ Enfin, compléter le code suivant pour la fonction de résolution de Sudoku en faisant appel à la récursivité:

Listing 1:

```
def sudoku(M):
    (i,j)=rechzero(M)
    if i==10:
        print M
    else:
        for k in range(1,10):
            if .......
               .......
               .......
        .......
```

Exercice n°2: Les tours de Hanoï



Le jeu des `tours de Hanoi` est un casse-tête inventé par la mathématicien français Edouard Lucas en 1883 lors d'exercices "récréatifs" inspirés par un prétendu ami, un certain N. Claus de Siam, plus probablement un simple anagramme de Lucas d'Amiens (ville d'origine d'Edouard Lucas).

Le jeu consiste à déplacer des disques de diamètres différents d'une tour de départ nommée A (celle de gauche) à une tour d'arrivée nommée C (celle de droite) en passant par une tour auxiliaire nommée B, ceci en respectant deux règles:

- on ne peut déplacer qu'un disque à la fois.
- on ne peut poser sur un disque qu'un disque de diamètre inférieur.

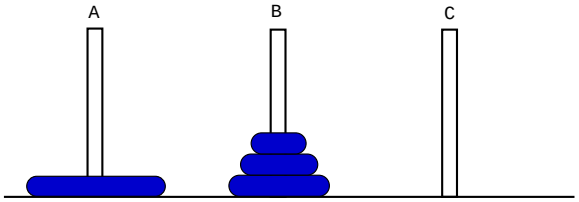
NB: nous partirons d'une situation de départ dans laquelle cette dernière règle est vérifiée, à savoir tous les disques sont correctement empilés sur la tour de gauche (photographie ci-dessus ou schéma plus bas).

L'objectif de cet exercice est de programmer une **fonction** qui donne la suite des manipulations à effectuer pour terminer le jeu c'est à dire obtenir tous les disques correctement empilés sur la tour de droite.

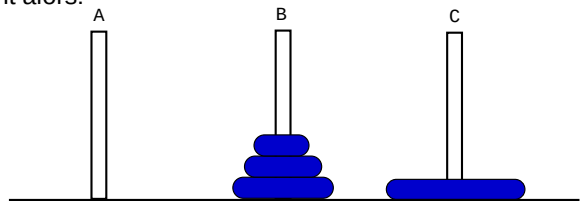
La démarche de déplacement est la suivante:

- **s'il n'y a qu'un seul disque, on le déplace de la tige A à la tige C**
- **si on suppose être capable de déplacer une pyramide de $n-1$ disques d'une tige sur une autre, alors pour déplacer n disques de la tige A vers la tige C, il faut:**

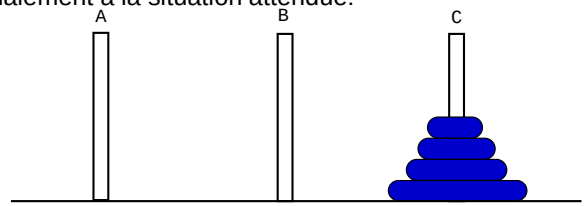
– déplacer la pyramide des $n-1$ disques de la tige A vers la tige B, sans toucher au disque n (le plus grand), la situation étant alors par exemple avec n=4 disques:



– puis de déplacer le disque n de la tige A vers la tige C, cette situation intermédiaire étant alors:



– et enfin de déplacer la pyramide de la tige intermédiaire B sur la tige C pour aboutir finalement à la situation attendue:



Résumons-nous; 3 étapes essentielles:

- Etape 1: déplacer $n-1$ disques de A vers B (en passant forcément par C)
- Etape 2: déplacer le plus grand disque (n) de A vers C.
- Etape 3: déplacer $n-1$ disques de B vers C (en passant forcément par A

Naturellement, les règles du jeu imposent de réaliser les étapes 1 et 3 en plusieur sous-étapes.

❶ Décomposer l'étape 1 en 3 sous étapes, ceci afin de déplacer $n-1$ (ici 3) disques de la tige A vers la tige B selon le même protocole

❷ De même, décomposer l'étape 3 en 3 sous-étapes. On pourra naturellement reprendre les sous étapes du ❶ en modifiant correctement le nom des tiges.

❸ A partir de cette analyse, programmer une fonction Python récursive `Hanoi(n,init, final, aux)` qui prend en arguments:

- `n`: le nombre de disques
- `init`: la chaine de caractères désignant la tour de départ, soit "A" lors du premier appel à la fonction.
- `final`: la chaine de caractères désignant la tour d'arrivée, soit "C" lors du premier appel à la fonction.
- `aux`: la chaine de caractères désignant la tour auxiliaire, soit "B" lors du premier appel à la fonction.

et **qui affiche tous les déplacements de disques nécessaires pour résoudre le jeu**.

❹ Déterminer le nombre de déplacements nécessaires pour terminer le jeu. On pourra confirmer cela avec un compteur et le faire afficher en fin de jeu.

❺ Cette récursivité est-elle terminale?