TP 15. Pour finir : Résolution d'un sudoku.

Présentation

Le sudoku est un jeu de logique de la famille des carrés latins. L'origine de son nom vient des deux mots japonais « su » qui signifie chiffre et « doku » qui signifie seul.

Règles du jeu
On dispose d'une grille de $9 \times 9 = 81$ cases divisées en $3 \times 3 = 9$ régions. Initialement la grille est pré remplie par un certain nombre de chiffres de 1 à 9. Le but du jeu est de remplir les cases vides de 1 à 9 en respectant les contraintes suivantes :

- ★ chaque chiffre doit apparaître une seule fois dans chaque ligne
- ★ chaque chiffre doit apparaître une seule fois dans chaque colonne
- ★ chaque chiffre doit apparaître une seule fois dans chaque région

La grille que l'on va compléter avec Python est la suivante :

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

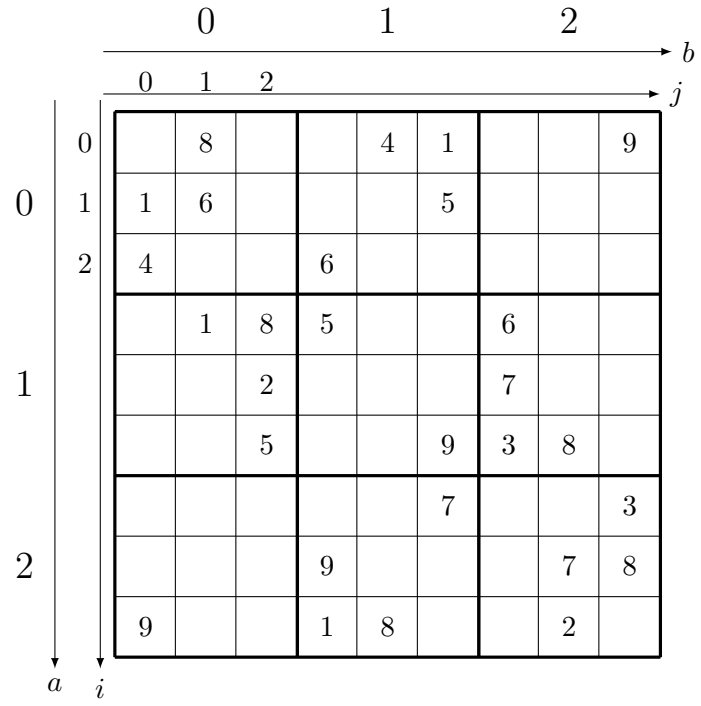
Modélisation

La grille de sudoku est transformée en une matrice carrée d'ordre 9 vue comme une liste de listes.

1°) Récupérer le fichier `grille.py` sur **moodle**. Dans ce fichier, vous trouverez la matrice `M` correspondant à la grille à traiter. Les cases vides sont remplies par des 0. Copier cette matrice dans votre script.

2°) Quelle doit être la réponse de `M[3][2]` ? Vérifier sur machine.

Afin de repérer une case dans la grille, on utilise ses coordonnées `i` et `j`, définies comme dans le schéma qui suit. L'indexation commence à 0. Une région est, quant à elle, définie par ses coordonnées `a` et `b`.



	0		8			4	1			9
0	1	1	6				5			
	2	4			6					
			1	8	5			6		
1				2				7		
				5			9	3	8	
							7			3
2					9				7	8
		9			1	8			2	

Définition de fonctions utiles à la résolution

3°) Écrire une fonction `cases_vides(M:list)` d'argument une matrice `M` et renvoyant la liste des coordonnées des cases vides (les coordonnées seront définies comme des listes à 2 éléments).

Le parcours de la matrice se fera ligne par ligne.

Afficher la liste `les_cases_vides` de `M`. Vérifier le résultat obtenu.

4°) Écrire une fonction `test_ligne(nb:int, i:int, M:list)` d'argument un chiffre `nb` entre 1 et 9, un indice de ligne `i`, une matrice `M`.

Cette fonction renvoie :

- `True` si `nb` est différent de toutes les valeurs des cases de la ligne i.
- `False` dans le cas contraire.

5°) Écrire une fonction analogue `test_colonne(nb:int, j:int, M:list)` pour les colonnes.

6°) Écrire une fonction `test_region(nb:int, a:int, b:int, M:list)` d'arguments un chiffre `nb` entre 1 et 9, des indices de région `a` et `b`, une matrice `M`. Cette fonction renvoie :

- `True` si `nb` est différent de toutes les valeurs des cases de la région d'indices a, b.
- `False` dans le cas contraire.

7°) En utilisant la notion de division euclidienne, écrire une fonction `region(i:int, j:int)` d'arguments les coordonnées `i` et `j` d'une case et renvoyant les coordonnées `a` et `b` de la région dans laquelle se trouve la case repérée par `i` et `j`.

8°) Déduire des questions précédentes une fonction `test(nb:int, i:int, j:int, M:list)` d'arguments un chiffre `nb` entre 1 et 9, des coordonnées `i` et `j` et une matrice `M`. Cette fonction renvoie `True` si et seulement si : `nb` est différent de toutes les valeurs des cases de la ligne `i`, des cases de la colonne `j` et des cases de la région où se trouve la case repérée par `i` et `j`.

Le backtracking

Il existe de nombreuses méthodes de résolution du sudoku. Les méthodes simulant le raisonnement humain sont efficaces mais difficiles à programmer. La méthode du *retour arrière systématique* (« backtracking ») est préférable : elle teste toutes les possibilités de remplissage. Étant donnée la puissance des ordinateurs actuels, une grille solution est généralement donnée en moins de 5 secondes.

Principe de la méthode
Les cases vides sont identifiées puis la première case vide est remplie par un 1. Si cette valeur convient, c'est-à-dire si elle obéit à toutes les règles du jeu, alors la case vide suivante est testée. Sinon, on vérifie si 2 convient, puis 3...
Dès qu'une incompatibilité survient (c'est-à-dire si aucun chiffre de 1 à 9 ne convient), il est alors nécessaire de revenir en arrière et d'augmenter le chiffre de la case vide précédente. Ce retour en arrière a lieu tant qu'il subsiste une incompatibilité concernant le choix d'un chiffre.

Concrètement
Supposons qu'on a déjà rempli les `k - 1` premières cases vides de la liste `les_cases_vides`. La case vide numéro `k` (numérotation de `k` à partir de `0`) contient une certaine valeur :

- `0` si c'est la première fois qu'on la traite,
- un autre nombre si on l'a déjà traitée mais que le nombre choisi est trop petit et implique des incompatibilités dans la suite. On est alors en train de faire un retour en arrière.

Dans les deux cas, il faut incrémenter la case numéro `k` de 1 autant de fois que nécessaire pour obtenir un (nouveau) nombre compatible avec le reste déjà construit de la matrice.

- Si l'on trouve un tel nombre compatible, on initialise la case vide à ce nombre trouvé puis on passe à la case vide numéro `k + 1`.
- Si jamais on ne trouve aucun nombre compatible, c'est qu'il faut réinitialiser : la case vide numéro `k` à `0` et reprendre la case vide numéro `k - 1` de la même manière.

9°) Créer une fonction `remplit_CV` d'arguments la liste `les_cases_vides`, un entier `k` et une matrice `M`. Cette fonction incrémente la case vide numéro `k` de la matrice `M` ou bien réinitialise cette case vide à `0`. De plus, cette fonction doit renvoyer le numéro de la case vide qui doit être traitée par la suite.

Résolution de la grille

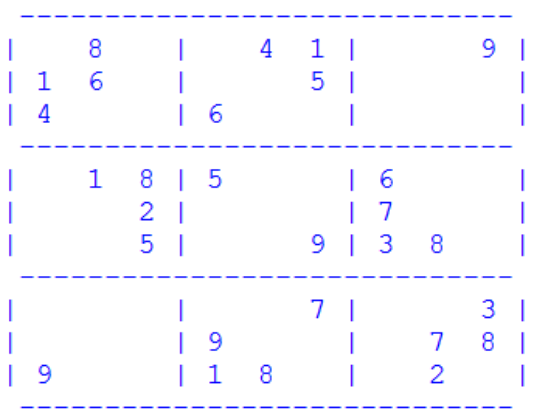
10°) Ecrire le programme entier et résoudre la grille de sudoku proposée.

11°) A l'aide de la fonction `perf_counter` de la bibliothèque `time`, déterminer la durée de résolution.

Partie facultative

1°) Écrire une fonction `grille` d'argument une matrice `M` qui renvoie une chaîne de caractères représentant la grille de sudoku associée à `M`.

Par exemple, en écrivant `print(grille(M))` dans le script avec la matrice `M` initiale (non remplie), on voit s'afficher après exécution :



2°) Écrire une fonction `est_un_sudoku` d'arguments une matrice `M` (représentant une grille de sudoku remplie). Cette fonction renverra un booléen permettant de savoir si `M` remplit bien les conditions pour être un sudoku.

Indication : On pourra utiliser des fonctions intermédiaires.