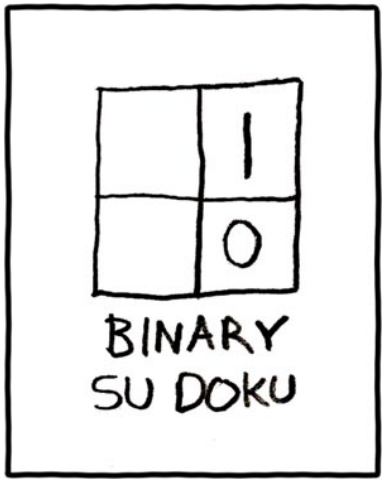
TP Info : Sudoku



1 Principe du jeu et de la résolution

Le but de ce TD est de résoudre des grilles de *Sudoku*. Le principe de ce jeu est de remplir une grille 9×9 avec les chiffres de 1 à 9 en respectant les trois contraintes suivantes :

— chaque chiffre n'apparaît qu'une fois dans chaque ligne ;
— chaque chiffre n'apparaît qu'une fois dans chaque colonne ;
— chaque chiffre n'apparaît qu'une fois dans chacun des 9 carrés 3×3.

Un certain nombre de chiffres sont placés au départ (normalement de manière à ce qu'il existe une unique solution), les autres cases sont laissées vides (ou remplies par un zéro que nous interpréterons comme une absence d'information). Un exemple de grille incomplète est donné ci-dessous, avec à droite la solution correspondante.

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

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

Pour résoudre une grille, un humain utilise normalement deux types de méthodes :

— des méthodes déductives qui consistent à trouver une case vide pour laquelle on est sûr qu'un seul chiffre est possible, la remplir, et continuer avec les informations supplémentaires liées au chiffre que l'on vient de placer ;
— quand ces méthodes échouent (on n'arrive pas à trouver de case pour laquelle il n'y a qu'une seule possibilité), on peut essayer de placer un chiffre dans une case et continuer la résolution. Si l'on arrive au bout, très bien (on ne peut en revanche pas affirmer que la solution trouvée est unique), si on obtient une contradiction (après un certain nombre d'étapes de résolution), on est sûr que le chiffre placé était faux. On peut alors recommencer en essayant de placer un autre chiffre dans la case dont on avait tenté de «deviner» le contenu.

Nous allons programmer la résolution en utilisant uniquement la deuxième méthode. Le principe de l'algorithme est le suivant :

— On crée une liste `trous` contenant les indices des cases initialement vides.
— On se déplace dans les cases initialement vides, en commençant par la première :
  — si la case sur laquelle on est contient un zéro (qui signifie qu'on n'a encore rien essayé) ou un chiffre autre que 9 (qui signifie qu'il reste des possibilités à tester), on essaie d'augmenter le contenu de 1.
    — si cela crée un conflit dans la ligne, la colonne ou le carré actuel, on teste le chiffre suivant (on ré-incrémente de 1) ;
    — sinon, on passe à la case (initialement) vide suivante.
  — si la case contient un 9, c'est qu'on a essayé toutes les possibilités sans trouver de solution. Une des suppositions faites auparavant est donc erronée. On remet la case à 0 et l'on revient à la case (initialement) vide précédente.
— Au bout d'un moment, on va arriver dans l'une des situations suivantes :
  — on sort de la liste `trous` «par la droite», ce qui signifie qu'on a réussi à remplir toutes les cases vides sans créer de conflit : on a trouvé une solution et on s'arrête, en renvoyant cette solution.
  — on sort «par la gauche» : cela signifie qu'on a testé toutes les possibilités et qu'aucune ne convient. La grille n'a pas de solution, on le signale et on s'arrête.

2 Implémentation

2.1 Numérotation des cases

Les grilles de Sudoku sont traditionnellement de taille 9×9 mais peuvent en fait être de taille $n^2 \times n^2$ pour tout entier $n \geqslant 1$. Il sera plus clair de définir

```
1 n = 3
2 cote = n * n
```

Ces variables sont globales, on pourra donc s'y référer à l'intérieur des fonctions sans avoir besoin de les passer en paramètre. On choisit de représenter une grille «à plat», c'est-à-dire par une seule liste de longueur `cote * cote`. Pour n = 3, la numérotation des cases est alors la suivante :

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80

On aura donc une numérotation implicite des *blocs*, c'est-à-dire des lignes, des colonnes et des carrés :

— les lignes sont numérotées de 0 à 8 de haut en bas ;
— les colonnes sont numérotées de 0 à 8 de gauche à droite ;
— les carrés sont numérotés de 0 à 8 dans l'ordre

0	1	2
3	4	5
6	7	8

Soit `k` un numéro de case (entre 0 et `cote * cote - 1`) et `i`, `j` et `pk` les numéros de ligne, de colonne et de carré correspondants.

1. Exprimer `k` en fonction de `i` et `j`.
2. Exprimer `i` et `j` en fonction de `k`.
3. Exprimer `pk` en fonction de `i` et `j`.

2.2 Extraction des blocs

1. Écrire trois fonctions `ligne(grille, i)`, `colonne(grille, j)` et `carre(grille, pk)` prenant en entrée une grille (sous forme de liste de longueur `cote * cote`) et un numéro de bloc (un entier entre 0 et `cote - 1`) et renvoyant le contenu du bloc en question dans la grille. Avec la grille donnée en exemple, on doit avoir :

```
>>> ligne(grille_exemple, 5)
  [0, 0, 6, 7, 0, 8, 2, 0, 0]
>>> colonne(grille_exemple, 5)
  [0, 5, 6, 2, 0, 8, 9, 3, 0]
>>> carre(grille_exemple, 5)
  [9, 0, 0, 0, 0, 8, 2, 0, 0]
```

2. Écrire une fonction `blocs(grille, k)` prenant en entrée une grille et un numéro de case et renvoyant les trois listes correspondant à la ligne, à la colonne et au carré contenant la case numéro `num_case`. Sur la grille exemple :

```
>>> blocs(grille_exemple, 30)
  ([0, 0, 8, 1, 0, 2, 9, 0, 0],
   [0, 3, 8, 1, 0, 7, 6, 2, 0],
   [1, 0, 2, 0, 0, 0, 7, 0, 8])
```

2.3 Détection des conflits

1. Écrire une fonction `possible(bloc)` qui prend en entrée une liste `bloc` d'entiers entre 0 et `cote` et renvoie :
   — `False` si au moins un entier compris entre 1 et `cote` apparaît plusieurs fois dans la liste ;
   — `True` sinon.
   Il est possible (et souhaitable) de faire cette vérification en un seul parcours de la liste en utilisant une liste auxiliaire.
2. Écrire une fonction `compatible(grille, k)` qui renvoie `False` ssi le contenu de la case `k` crée un conflit dans la grille passée en argument. On vérifiera uniquement les conflits potentiels dans la ligne, la colonne et le carré contenant `num_case`.

```
# On place un 1 dans la première case
>>> grille_exemple[0] = 1
# Il y a un conflit (il y a déjà un 1 dans le carré)
>>> compatible(grille_exemple, 0)
  False
# On place un 5 dans la première case
>>> grille_exemple[0] = 5
# Pas de conflit (pour l'instant, on s'apercevrait si
# l'on continuait la résolution que ce 5 est en fait
# incorrect).
>>> compatible(grille_exemple, 0)
  True
```

2.4 Fonction de résolution

1. Écrire une fonction `a_remplir(grille)` qui renvoie la liste des numéros des cases vides de `grille` (c'est-à-dire des cases contenant un zéro).

On dispose maintenant de tous les outils nécessaires pour procéder à la résolution.

2. Écrire une fonction `resout(grille)` qui prend en entrée une grille partiellement remplie et renvoie la grille complétée. On ne vérifiera pas l'unicité de la solution renvoyée, et l'on affichera un message signalant l'absence de solution le cas échéant.
   Cette fonction devrait logiquement faire une quinzaine de lignes.
3. Qu'obtient-on si l'on appelle `resout` sur une grille initialement vide ?

2.5 Lecture d'un fichier

1. Le fichier `grilles.txt` est disponible à l'adresse `https://tinyurl.com/tnhtlym` et contient 50 grilles de Sudoku dans un format assez simple (ouvrez-le avec le bloc-notes Windows pour voir). Si l'on résout toutes ces grilles, que l'on lit à chaque fois le nombre formé par les trois chiffres contenus dans les trois premières cases (483 pour la grille exemple) et que l'on somme tous ces nombres, combien obtient-on ?