Projet - Sudoku

November 9, 2021

Ce document contient les instructions du projet de fin de semestre pour le cours d'IN304 2021-2022. Le programme développé doit être rendu sur un seul fichier Python, `sudoku.py` et déposé sur Moodle avant le 10 décembre 2021. La réalisation de ce projet doit être strictement individuelle. Assurez-vous que votre programme fonctionne correctement avant de le soumettre. Il ne faut pas neécessairement bien répondre à toutes les questions pour avoir la note maximale. Tous les fichiers envoyés pour correction seront analysés par un outil qui détecte le plagiat entre les fichiers soumis et sur internet. Les programmes copiés entre eux sont donc très facilement identifiables, même si le nom des variables est modifié. Mieux vaut ne faire que la moitié du projet soi-même que d'avoir 0 pour triche !

1 Jeu de sudoku

Le sudoku est un jeu de logique visant à remplir une grille de 9x9 carrés avec des nombres de 1 à 9. La grille de sudoku est divisée en 9 régions de taille 3x3, visibles en gras sur la figure ci-dessous. La règle du jeu de sudoku est simple: la grille doit être remplie en utilisant qu'une seule fois chaque chiffre de 1 à 9 sur chaque ligne, colonne et région de la grille. Une grille de sudoku peut avoir plusieurs bonne solution, mais une "bonne" grille ne possède qu'une solution unique.

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

Figure 1: Une grille de sudoku

2 Questions

Le projet consiste à manipuler des grilles de sudoku, et sera de difficulté croissante. La première partie permettra de définir des fonctions utiles concernant la manipulation et la validation de grilles de sudoku. La deuxième partie permettra la résolution et la génération de grilles de sudoku.

2.1 Fonctions de base

Nous allons représenter une grille de sudoku par une liste de listes de taille 9. Les éléments remplis de la grille seront directement représentés par leur valeur de 1 à 9, et les éléments vides par un 0. La fonction `afficher_grille` déjà incluse dans le fichier `sudoku.py` permet d'afficher une grille de sudoku fournie en entrée. Les différentes grilles incluses dans le fichier `sudoku.py` vous permettront de rapidement tester vos fonctions.

1. Ecrire une fonction `unique(x)` qui prend une liste `x` en argument, et qui renvoie `True` si tous les éléments de `x` sont différents et `False` si au moins deux éléments de `x` sont identiques. Attention, la valeur 0 qui représente une case vide, ne doit pas être incluse dans cette vérification.

2. Ecrire trois fonctions, `ligne(x, i)`, `colonne(x, i)` et `region(x, i)` qui renvoie la ligne `i` (facile), la colonne `i` (un peu moins facile) et la région `i` (difficile) d'une grille de sudoku `x` sous forme d'une liste de 9 éléments. Attention, les indices `i` vont de 1 à 9 et non de 0 à 8. Indice: la region `k` associée aux coordonnées `(i, j)` est obtenue avec la formule

$$k = 3 \times ((i-1)//3) + ((j-1)//3) + 1$$

3. Ecrire une fonction `ajouter(x, i, j, v)` qui ajoute la valeur `v` aux coordonnées (`i`, `j`) sur la grille `x`. Utiliser les fonctions `unique` en combinaison avec les fonctions `ligne`, `colonne` et `region` afin de vérifier que cet ajout est compatible avec les règles du sudoku. Si l'ajout est interdit, restaurer la valeur qui se trouvait dans la grille `x` aux coordonnées (`i`, `j`).

4. Ecrire une fonction `verifier(x)` qui vérifie que la grille de sudoku a été correctement remplie. Cette fonction renvoie `True` si toutes les lignes, colonnes et régions de la grille sont valides et si la grille ne contient pas de 0, et `False` sinon.

5. Finalement, écrire une fonction `jouer(x)` qui prend en entrée une grille `x` et qui demande à l'utilisateur un triplet de valeurs `(i, j, v)` représentant une valeur `v` à placer aux coordonnées `(i, j)` sur la grille jusqu'à ce que la grille `x` soit entièrement remplie. La fonction doit ré-afficher la grille après chaque ajout de l'utilisateur.

2.2 Génération et résolution de grilles

Une grille de sudoku peut être résolue à l'aide d'un algorithme récursif relativement simple. Pour chaque case vide de la grille, on liste ses valeurs potentielles en éliminant les valeurs déjà prises dans sa ligne, colonne et région. On choisit alors une case vide avec le moins de valeurs potentielles possibles. Si toutes les cases ont une valeur, le puzzle est résolu. Si une case est vide et n'a pas de valeur potentielle, la grille n'a pas de solution valide. En connaissant ces conditions d'arrêt, on peut alors tester recursivement chacune des cases d'une grille jusqu'à renvoyer sa solution.

1. Ecrire une fonction `solutions` qui prend une grille `x` en entrée et qui renvoie un dictionnaire contenant les valeurs potentielles de chaque case vide de `x`. La fonction se construit de la manière suivante:

    1. Créer un dictionnaire contenant 10 clés numérotées de 0 à 9. Chacune de ces clés est associée à une liste vide dans le dictionnaire.
    2. Parcourir toutes les cases de `x` selon leur coordonées ligne et colonnes `i` et `j`. Pour chaque case vide de `x`, calculer les valeurs potentielles de cette case en éliminant les valeurs déja présentes sur le même colonne, ligne et région (réutiliser la formule qui permet de calculer la région d'une case selon ses coordonnées vu dans la partie précédente).
    3. Ajouter la case au dictionnaire sous la forme d'un tuple (`i`, `j`, `l`) où `i` et `j` sont les coordonnées de la case et `l` la liste de ses valeurs potentielles. Ce tuple est ajouté au dictionnaire dans la liste associée à la clé égale au nombre de valeurs potentielles de la case (e.g. si une case a deux valeurs potentielles, on l'ajoute à la liste de la clé 2).

4. Ecrire une fonction recursive `resoudre(x)` qui permet de résoudre une grille de sudoku `x` selon l'algorithme de résolution récursif décrit ci-dessus. La fonction se contruit de la manière suivante:

    1. Commencer par calculer le dictionnaire des valeurs potentielles de la grille `x` à l'aide de la fonction `solutions(x)`. Construire une liste `l` qui contient toutes les cases vides et ses valeurs potentielles (sous la forme des triplets vus à la question précédente) à partir du résultat de la fonction `solutions(x)`. Indice: utiliser la fonction `values()` d'un dictionnaire et une compréhension de liste pour la constuire rapidement.
    2. Ajouter ensuite les conditions d'arrêt de la fonction: on renvoie `False` si au moins une case de la grille n'a pas de solutions possibles (i.e. la liste des cases a au moins une entrée dans le dictionnaire des solutions à la clé 0). À l'inverse, si la liste `l` est vide, cela veut dire que toutes les cases sont remplies, et on renvoie alors directement la grille `x`.
    3. On écrit alors enfin la partie récursive de la fonction. On itère sur tous les tuples `(i, j, v)` de `l`. Pour chacune des valeurs potentielles `t` de la liste de `v`, on ajoute la valeur `t` à la grille `x` et on appelle récursivement `resoudre(x)` sur cette grille. Si le résultat de cet appel est une grille valide, on la remonte directement à l'aide d'un `return`. Si cet appel renvoie `False`, la valeur choisie n'était pas correcte: on réinitialise alors la valeur de la case à 0. Si aucune des valeurs `t` de la liste `v` testées ne renvoie une grille, cela veut dire qu'un choix antérieur était incorrect: on renvoie alors également `False`.

4. On cherche finalement à génerer une grille de sudoku valide. Est-il possible d'en générer une à l'aide de la fonction `resoudre` sur une grille vide ? Copier la fonction `resoudre` en une fonction `generer` qui remplit une grille vide en une grille pleine. On remarque qu'un appel à cette nouvelle fonction génère toujours la même grille. Utiliser la fonction `shuffle` de la librairie `random` dans cette nouvelle fonction pour générer une grille différente à chaque appel.

5. Finalement, créer une fonction `nouvelle()` qui crée une grille pleine à l'aide de la fonction `generer`, et qui retire ensuite aléatoirement des valeurs de cette grille pleine avant de la renvoyer en tant que nouveau puzzle. Utiliser la fonction `randint` de la librairie `random` pour choisir les indices des cases à effacer aléatoirement. Il faut au moins laisser 17 cases remplies pour s'assurer que la grille puisse être résolue sans algorithme.