Aix-Marseille Université
L1

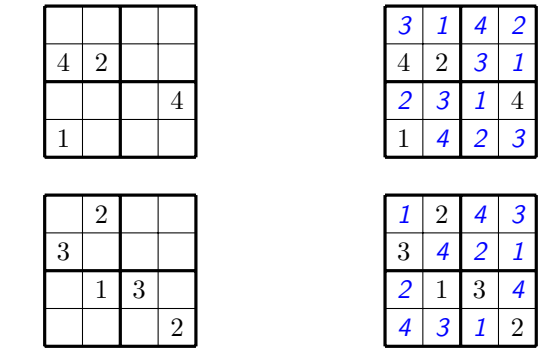

Figure 1: Une grande grille de sudoku et sa solution.

UE Projet informatique

Petit Sudokus - feuille 1

Introduction

Le sudoku est une forme de puzzle très connue. On dispose d'une grille de 3 par 3 blocs, eux-même de dimension 3 par 3. La plupart des 9 par 9 cases sont vides, mais certaines contiennent initialement un chiffre de 1 à 9. Le joueur doit compléter entièrement cette grille tout en respectant les règles suivantes :

— Chaque ligne de la grille ne doit pas contenir le même chiffre 2 fois.
— Chaque colonne de la grille ne doit pas contenir le même chiffre 2 fois.
— Chaque bloc de 3 par 3 ne doit pas contenir le même chiffre 2 fois.

Pour ce projet, nous commencerons par des grilles composées de 2 par 2 blocs, eux même de dimension 2 par 2, et remplies de chiffres allant de 1 à 4.

Votre programme final devra faire les choses suivantes :

— Afficher une grille.
— Vérifier si une grille respecte bien les règles du sudoku.
— Résoudre une grille.
— Générer une grille incomplète pour le joueur.

Comme pour tout projet, il est très important de lire ce sujet jusqu'au bout avant de le commencer.

Méthodes de base : grille et affichage

Il est important de commencer par les éléments les plus simples. Tout d'abord, sous quelle forme votre programme va-t-il encoder la grille du sudoku ? Si cette grille contient des nombres, comment gérer simplement le cas où une case ne contient pas encore de valeur ?

Une fois ces décisions prises, écrire les fonctions suivantes :

— Une fonction qui prend en paramètre la grille et l'indice d'une ligne, et affiche une ligne à l'écran.
— Une fonction qui prend en paramètre la grille et l'affiche à l'écran.

Tentez de rendre votre affichage le plus confortable possible. En particulier, prenez soin de permettre une distinction entre les blocs de la grille.

Pour chacune des fonctions que l'on rédige, il faut les tester. Initialisez une grille de taille 4 par 4 avec des valeurs de test (vous pouvez utiliser les grilles illustrées plus haut, par exemple).

Pour chacune de vos fonctions, faites usage des fonctions précédemment définies si cela est possible. Pour le reste du projet, c'est à vous de déterminer quelles fonctions sont nécessaires. Découpez toute tâche trop complexe en plusieurs fonctions ; cela permet de rendre le code plus facile à lire et à écrire.

Vérification d'une grille

Une fois une grille initialisée et affichable, écrivez une fonction qui prend en paramètre une grille, et vérifie que les règles du sudoku sont bien respectées. Pour cela, nous vous recommandons les étapes suivantes :

— Une fonction qui vérifie qu'une ligne d'indice donné est valide (n'a pas de chiffre en double).
— Une fonction qui vérifie qu'une colonne d'indice donné est valide.
— Une fonction qui vérifie qu'un bloc de coordonnées données est valide.
— Une fonction qui vérifie toutes les lignes.
— Une fonction qui vérifie toutes les colonnes.
— Une fonction qui vérifie tout les blocs.
— Une fonction qui vérifie toute la grille.

Évitez au maximum la redondance et appellez les fonctions déjà définies dès que possible.

Résolution d'une grille

Résoudre le sudoku est la tâche principale de ce projet. L'ensemble des autres tâches (comptage de solutions ou génération de grilles pleines et incomplètes) ne sont que des variations de l'algorithme que vous allez implémenter pour résoudre un sudoku.

Pour cette tâche, vous devez implémenter une fonction qui prend en paramètre une grille (qui peut être déjà partiellement complétée) et la résout, si cela est possible.

L'algorithme exécuté par cette fonction est un algorithme de backtrack, dont le pseudo code est le suivant :

Algorithm 1 Résoudre la grille G

```
if Si G est invalide then
    Retourner False
end if
if Si G est complète then
    Retourner True
end if
(x, y) ← les coordonnées du premier emplacement libre dans G
for value ∈ {1, 2, 3, 4} do
    G[x, y] ← value
    Résoudre récursivement G, et stocker la valeur de retour dans result
    if result = True then
        Retourner True
    end if
    Effacer G[x, y]
end for
Retourner False
```

Cette fonction est récursive, c'est-à-dire qu'elle fait appel à elle même. Ainsi, la séquence des décisions prises se trouve représentée dans le stack des appels de fonctions.

Génération d'une grille aléatoire pleine

La fonction de résolution est parfaite pour trouver la solution à des grilles données. Lorsqu'elle fonctionne sur des grilles vides, elle donnera toujours le même résultat. Faites une variante de cet algorithme, qui génère une solution aléatoire parmis les solutions possibles. Pour ce faire, simplement mélanger l'ordre des valeurs qui sont considérées dans la boucle principale (8ème ligne du pseudo code précédent). En python, on peut mélanger un tableau en important le module random et en appellant random.shuffle(list).

Comptage des solutions

Maintenant que nous pouvons générer une diversité de solutions différentes, on souhaite disposer d'une fonction qui compte le nombre de solutions d'une grille données. Pour ce faire, notre algorithme va passer par toutes les solutions possibles, et compter 1 à chaque solution valide rencontrée.

Cet algorithme est en structure très proche de la résolution d'un sudoku. Voici son pseudo-code :

Algorithm 2 Compter les solutions de la grille G

if Si G possède une incohérence **then**
  Retourner 0
end if
if Si G est complète **then**
  Retourner 1
end if
$(x, y) \leftarrow$ les coordonnées du premier emplacement libre dans G
$count \leftarrow 0$
for $value \in \{1, 2, 3, 4\}$ **do**
  $G[x, y] \leftarrow value$
  Compter récursivement les solutions de G, et stocker la valeur de retour dans $result$
  $count \leftarrow count + result$
  Effacer $G[x, y]$
end for
Retourner $count$

Génération de grilles aléatoires incomplètes

On souhaite pouvoir construire une grille incomplète aléatoire qui ne dispose que d'une solution. Nous allons générer ces grilles de façon à les rendre aussi difficiles que possible. Pour ce faire, on applique l'algorithme suivant :

Algorithm 3 Générer une grille incomplète à une seule solution

$G \leftarrow$ une grille complète générée aléatoirement
$L \leftarrow$ la liste de toutes les coordonnées de la grille
Mélanger L
for $(x, y) \in L$ **do**
  $value \leftarrow G[x, y]$
  Effacer $G[x, y]$
  $count \leftarrow$ le nombre de solutions de G
  if $count > 1$ **then**
    $G[x, y] \leftarrow value$
  end if
end for
Retourner G